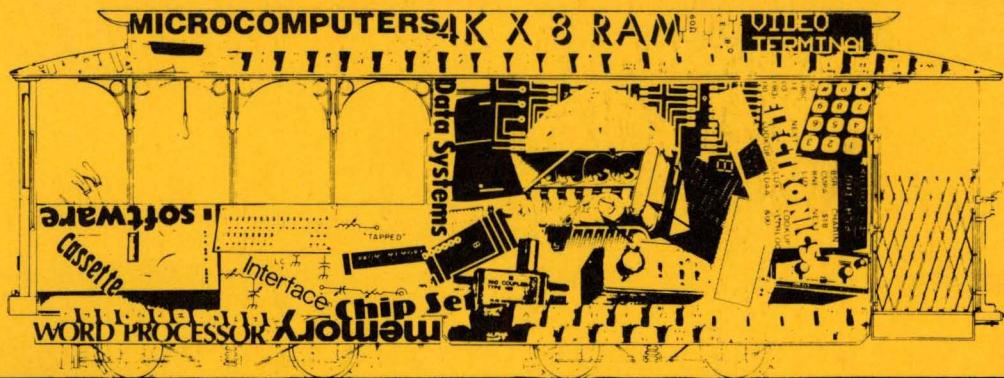


\$12.00
(or, if you
prefer: A
nickel more
than \$11.95)



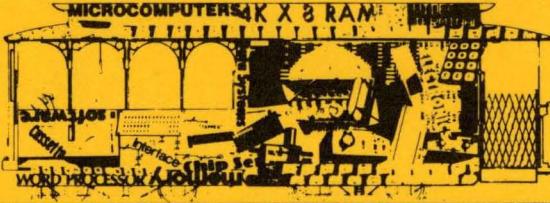
THE FIRST WEST COAST COMPUTER FAIRE

A Conference & Exposition
on
Personal & Home Computers

CONFERENCE PROCEEDINGS

JIM C. WARREN, JR.
EDITOR

april 15-17, 1977 • san francisco



**THE FIRST
WEST COAST
COMPUTER FAIRE**
*A Conference & Exposition
on
Personal & Home Computers*

CONFERENCE PROCEEDINGS

of the largest convention ever held

Exclusively Devoted to Home & Hobby Computing



over 300 pages of conference papers, including:

(Topic headings with approximate page counts)

Friday & Saturday Banquet Speeches (16)
Tutorials for the Computer Novice (16)
People & Computers (13)
Human Aspects of System Design (9)
Computers for Physically Disabled (7)
Legal Aspects of Personal Computing (6)
Heretical Proposals (11)
Computer Art Systems (2)
Music & Computers (43)
Electronic Mail (8)
Computer Networking for Everyone (14)
Personal Computers for Education (38)
Residential Energy & Computers (2)
Systems for Very Small Businesses (5)

Entrepreneurs (6)
Speech Recognition &
Speech Synthesis by Computer (14)
Tutorials on Software Systems Design (11)
Implementation of
Software Systems and Modules (10)
High-Level Languages for Home Computers (15)
Multi-Tasking on Home Computers (10)
Homebrew Hardware (8)
Bus & Interface Standards (17)
Microprogrammable Microprocessors
for Hobbyists (18)
Amateur Radio & Computers (11)
Commercial Hardware (8)

----- plus -----

Names & addresses of the 170+ exhibitors at the Computer Faire

Order now from:
Computer Faire
Box 1579
Palo Alto CA 94302
(415) 851-7664

Proceedings:	\$12.00	(\$11.95, plus a nickel, if you prefer)
Shipping & Handling:	.68	(Write for shipping charges outside U.S.A.)
<u>Outside California:</u>	<u>\$12.68</u>	Payment must accompany the order.
Californians Add:	.72	6% Sales Tax
<u>Inside California:</u>	<u>\$13.40</u>	Payment must accompany the order.

An 8½" x 11" softbound book.

april, 1977 • san francisco

The Digital Group comments on DR. DOBB'S JOURNAL

(We were surprised and pleased recently to find these comments about *Dr. Dobb's Journal* in the latest Digital Group Newsletter.)

the digital group

po box 6528 denver, colorado 80206 (303) 777-7133

flyer

NUMBER 8

It is not very often that there is a journal/newsletter that the Digital Group is able to recommend without some hesitation (and we get them all). However, Dr. Dobb's Journal of Computer Calisthenics & Orthodontia is one pleasant exception. Jim Warren, the editor, has put together a good concept and is managing to follow through very well indeed. There is no advertising in the Journal. It is supported solely on subscriptions. That also means that manufacturers have zero leverage over the content of the magazine. The Journal's primary purpose is to place significant software into the public domain and to provide a communications medium for interested hobbyists. The approach is professional and they are growing quickly.

(In case it might appear otherwise to some people, there is no official link whatsoever between the Digital Group and Dr. Dobb's Journal - we've taken our lumps as appropriate just like everyone else when Jim felt they were justified.)

We think Dr. Dobb's Journal is here to stay and a publication that is a must for everyone in the hobbyist world of computers. Don't miss it!

IN EVERY ISSUE

• UNIQUE SYSTEMS PROJECTS

"Realizable Fantasies" - details of projects that have not yet been done, but are within the limits of current technology, hobbyists' expertise, and the computer enthusiast's budget.

• INDEPENDENT PRODUCT REVIEWS AND CONSUMER ADVOCACY

DDJ staff now includes a three person product & software evaluation group. They perform independent evaluations of products being marketed to hobbyists and publish their findings - good or bad - in this subscriber-supported *Journal*. Note that *Dr. Dobb's* carries no paid advertising; it is responsible only to its readers. It regularly publishes joyful praise and raging complaints about vendor's products and services.

• COMPLETE SYSTEMS & APPLICATIONS SOFTWARE

User documentation, internal specifications, annotated source code. In the first year of publication, *DDJ* carried a large variety of interpreters, editors, debuggers, monitors, graphics games software, floating point routines and software design articles.

• HOT NEWS AND RAGING RUMOR

Unusually fast turn-around on publishing "hot stuff". Typically, an issue will carry information and articles received within two weeks of its distribution. Also, we hear and print lots of rumors belched forth by the nearby "Silicon Valley" rumor mill.

MORE REVIEWS

"THE software source for microcomputers. Highly recommended."

-Philadelphia Area Computer Society
The Data Bus, July, 1976.

"It looks as if it's going to be THE forum of public domain hobbyist software development.

Rating—★ ★ ★ ★ .

Toronto Region Association of Computer Enthusiasts (TRACE), Newsletter, July 9, 1976.

"The best source for Tiny BASIC and other good things. Should be on your shelf."

-The Computer Hobbyist,
North Texas (Dallas) Newsletter,
May 1976

Name _____

Address _____

City/State _____ Zip _____

Please start my one year subscription to *Dr. Dobb's Journal* (10 issues) and bill me for \$12.

In Canada add \$4/year for postage.
European and Japanese rates are available on request.

Mail to: *Dr. Dobb's Journal* • Dept. 51 • 1263 El Camino Real • Box E • Menlo Park, CA 94025

Meet the most powerful μC system available for dedicated work.

Yet it's only \$595.*

*kit price

Here's the muscle you've been telling us you wanted: a powerful Cromemco microcomputer in a style and price range ideal for your dedicated computer jobs—ideal for industrial, business, instrumentation and similar applications.

It's the new Cromemco Z-2 Computer System. Here's some of what you get in the Z-2 for only \$595:

- The industry's fastest μP board (Cromemco's highly regarded 4 MHz, 250-nanosecond cycle time board).
- The power and convenience of the well-known Z-80 μP.
- A power supply you won't believe (+8V @ 30A, +18V and -18V @ 15A — ample power for additional peripherals such as floppy disk drives).
- A full-length shielded motherboard with 21 card slots.
- Power-on-jump circuitry to begin automatic program execution when power is turned on.
- S-100 bus.
- Standard rack-mount style construction.
- All-metal chassis and dust case.
- 110- or 220-volt operation.

DEDICATED APPLICATIONS

The new Z-2 is specifically designed as a powerful but economical dedicated computer for systems work. Notice that the front panel is entirely free of controls or switches of any kind. That makes the Z-2 virtually tamper-proof. No accidental program changes or surprise memory erasures.

FASTEST, MOST POWERFUL μC

Cromemco's microcomputers are the fastest and most powerful available. They use the Z-80 microprocessor which is

Shown with
optional bench cabinet

widely regarded as the standard of the future. So you're in the technical fore with the Z-2.

BROAD SOFTWARE/PERIPHERALS SUPPORT

Since the Z-2 uses the Z-80, your present 8080 software can be used with the Z-2. Also, Cromemco offers broad software support including a monitor, assembler, and a BASIC interpreter.

The Z-2 uses the S-100 bus which is supported by the peripherals of dozens of manufacturers. Naturally, all Cromemco peripherals such as our 7-channel A/D and D/A converter, our well-known BYTESAVER with its built-in PROM programmer, our color graphics interface, etc., will also plug into the S-100 bus.

LOW, LOW PRICE

You'll be impressed with the Z-2's low price, technical excellence and quality. So see it right away at your computer store—or order directly from the factory.

Z-2 COMPUTER SYSTEM KIT (MODEL Z-2K) (includes 4 MHz μP card, full-length 21-card-slot motherboard, power supply, one card socket and card-guide set, and front panel; for rack mounting) \$595.

Z-2 COMPUTER SYSTEM ASSEMBLED (MODEL Z-2W) (includes the above as well as all 21 sockets and card guides and a cooling fan; for rack mounting) \$995.



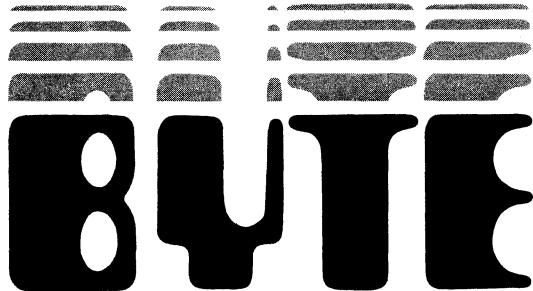
Cromemco

incorporated

Specialists in computers and peripherals

2432 CHARLESTON RD., MOUNTAIN VIEW, CA 94043 • (415) 964-7400

Get updated . . . keep updated with



the leading magazine in the personal computer field

The personal computer age is here.

Join Byte's 100,000 subscribers and catch up on the latest developments in the fast-growing field of microprocessors. Read BYTE, The Small Systems Journal that tells you everything you want to know about personal computers, including how to construct and program your own computer (30,000 BYTE readers have already built, or bought, their own systems and half of these have 8K bytes or more). You'll find our tutorials on hardware and software invaluable reading, also our reports on home applications and evaluative reviews based on experiences with home computer products.



Home computers . . . practical, affordable.

Large scale integration has slashed prices of central processors and other computer components. This has encouraged the development of new, low-cost peripherals resulting in more hardware and software — more applications than you could imagine, more opportunities for you. BYTE brings it all to you. Every issue is packed with stimulating and timely articles by professionals, computer scientists and serious amateurs. BYTE editorials explore the fun of using and applying computers toward personally interesting problems such as electronic music, video games and control of systems for alarms to private information systems.

Subscribe now to BYTE . . . The Small Systems Journal

Read your first copy of BYTE, if it's everything you expected, honor our invoice. If it isn't, just write "CANCEL" across the invoice and mail it back. You won't be billed and the first issue is yours.

Allow 4 to 6 weeks for processing.

BYTE Subscriptions Dept. P.O. Box 361 Arlington, Mass. 02174 U.S.A.

Please enter my subscription for:

- One year U.S. - \$12 Two years U.S. - \$22 Three years U.S. - \$32
 Canada or Mexico - \$17.50 Europe (Air Delivered) - \$25
 Surface delivery to all other countries except Europe, Canada or Mexico - \$25 (Air delivery available on request)

Check enclosed (Bonus: one extra issue)

Bill me.

Bill Master Charge

Please remit in U.S. funds — Card number _____ Expiration date _____

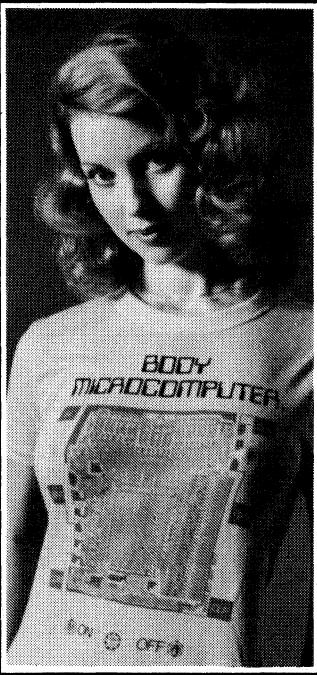
Signature _____ Name (please print) _____

Address _____

City _____ State/Country _____ Code _____

High Technology Fashion™

by George De Silva



© Applied Art 1977

Line: Integrated circuit Tee-Shirts T.M.
Design: Body Microcomputer T.M.

At last! A personal computing system for every body. Affordable, simple, unique.

FEATURES

- Body language software
- Interactive games capability
- Universal interface
- Low power consumption— 10^{-6} EW (excitement watts)
- Wash-dry maintenance

THE COMPLETE GIFT

Packaged in a booksize (10x7x1¼") white gift box with colored microcomputer imprint. Illustrated User's Manual included.

ORDER FORM

Hand screened quality tee-shirt. 50% polyester / 50% cotton. Available in white, yellow, light blue and tan. Microcomputer design in black and red.

Chest Size (Inches)	Quan.	1st Color Choice	2nd Color Choice	Price Each Package
Y O U T H	22-24			5.90
	26-28			plus
	30-32			.85 postage
A D L S	34-36			per package.
	38-40			In California,
	42-44			please add state
	46-48			and local taxes.

Name _____

Address _____

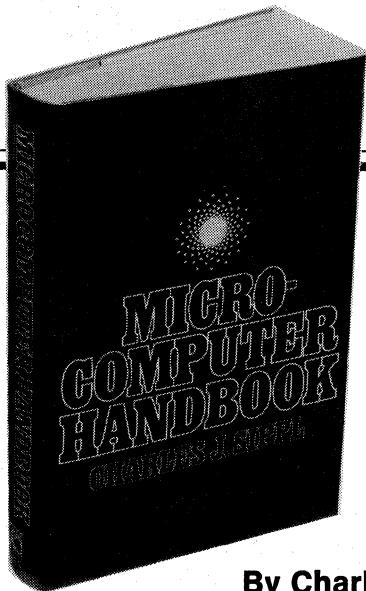
City _____ State _____ Zip _____

Mail your order to:

Applied Art, 852 Madonna Way, Los Altos, California 94022
Tel. (415) 948-0689

Check Money Order
Please allow 10 days for delivery.

NOW— ONE comprehensive reference to microcomputers!



By Charles J. Sippl

(A PETROCELLI/CHARTER BOOK)
480 pages, 65 illus., \$19.95

More than a detailed reference to every aspect of chip technology, *Microcomputer Handbook* is also an important guide to decision-making about specific configurations (4-bit, 8-bit, 16-bit, or even 64-bit "super machines"). Among its many down-to-earth features is a first-rate glossary of terms that runs through every chapter and spotlights everything from ACIA (asynchronous communications interface adapter) to ROM simulator (a general-purpose debugging tool). There's even a detailed analysis of the microcomputer as a hobby item; problem solving; game playing and kit assembling.

For the professional, hobbyist, and student, this book: • contains photos and diagrams providing easy introduction to techniques and concepts • discusses read-only memories (ROM's and RAM's) • examines the role of large-scale interation (LSI) in production-cost reduction • presents cost-performance calculation for microcomputer systems • demonstrates the vast applicability of the microcomputer; its capacity for data storage, data collection, and reducing operating time • and lists the latest "hobby type" microcomputers available, as well as clubs, societies and manufacturers.

10-DAY FREE EXAMINATION



VAN NOSTRAND REINHOLD

7625 Empire Drive • Florence, Kentucky 41042

Please send me the MICROCOMPUTER HANDBOOK by Charles J. Sippl for 10 days' free examination. At the end of that time I will either remit the amount of your invoice (including postage, handling, and my local sales tax) or return the book and owe nothing. (Payment must accompany orders with P.O. box addresses. Offer good in U.S.A. only, and subject to credit department approval.)

(PLEASE PRINT)

Name _____

Address _____

City _____ State _____ Zip _____

SAVE! Enclose payment with order and publisher pays postage and handling. Same return-refund guarantee. Your local sales tax must be included with payment.
0-442-80324-9

C 7010

ENTER THE WORLD OF HOME AND SMALL BUSINESS COMPUTING



EXPLORE

INTERFACE AGE MAGAZINE

★ MONTH AFTER MONTH LOOK TO
INTERFACE AGE MAGAZINE FOR THE LATEST
INFORMATION ON THE DYNAMIC WORLD OF
PERSONAL COMPUTING.

- Use your personal computer for auto repair, work bench controller, teaching machine, central information bank and design test center.
- Control your small business with your own real-time accounting and inventory control system.
- Set your computer to turn sprinklers on and off, manage a household security system, feed your dog.
- Establish a recipe bank to plan daily meals and generate its own shopping list.
- Evaluate the stock market, set up gambling and probability programs. Evaluate odds on sporting events and horse racing.

★ ARTICLES RANGE FROM THE
FUNDAMENTALS OF COMPUTERS TO
LANGUAGES AND SYSTEM DESIGN.
APPLICATIONS INCLUDE BOTH PRO-
FESSIONAL AND NON-TECHNICAL.

★ READ INTERFACE AGE FOR THE LATEST ON NEW
PRODUCT INFORMATION AND TECHNICAL
BREAKTHROUGHS.

- May's issue included inside the FLOPPY ROM™ — a vinyl record which is played on a conventional phonograph to enter this month's program in your computer.

★ ORDER YOUR SUBSCRIPTION NOW!

12 Monthly Issues: \$14 U.S., \$16 Canada/Mexico, \$24 International

Name _____

Address _____

City _____ State _____ Zip _____

Check or M.O. (U.S. Funds drawn on U.S. Bank) Visa Card Master Charge

Acct No. _____ Exp. Date _____

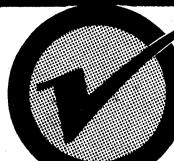
Signature _____

Make checks payable to: **INTERFACE AGE MAGAZINE**, P.O. Box 1234, Cerritos, CA 90701





Our choice line of Personal Computing Books



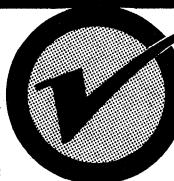
NEW CHOICES

MICROPROCESSOR BASICS (Elphick) Selection and application info on 8 popular micro's, including the 8080, 6800, F8, IMP, and 6100. #5763-6, paper, \$9.95

THE BASIC WORKBOOK: Creative Techniques for Beginning Programmers (Schoman) "Hands-on" learning of problem-solving using a computer. #5104-2, paper, \$4.25

GAME PLAYING WITH BASIC (Spencer) Over 50 easy-to-learn and challenging games and puzzles for your personal computer. #5109-3, paper, \$6.95

TELEPHONE ACCESSORIES YOU CAN BUILD (Gilder) Fully-illustrated, step-by-step instruction on building useful phone accessories at a fraction of the commercial cost. #5748-2, paper, \$3.95



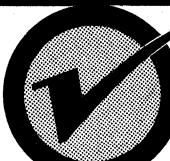
FUTURE CHOICES

STANDARD DICTIONARY OF COMPUTERS AND INFORMATION PROCESSING, Revised Second Edition, #5099-2, Available Oct. '77.

APPLIED COMPUTING: Putting Your Computer to Work, #5761-X, Available Jan. '78.

PROGRAMMING THE PROGRAMMABLE CALCULATOR #5105-0, Available Jan. '78

HAYDEN BOOK COMPANY, INC.



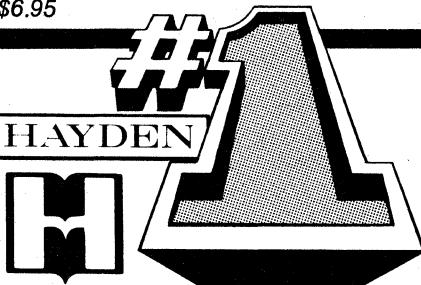
CRITICS' CHOICES

MICROPROCESSORS: New Directions for Designers (Torrero) "... a useful book for the electronics design engineer." BYTE MAGAZINE. #5777-6, paper, \$10.95

FUNDAMENTALS & APPLICATIONS OF DIGITAL LOGIC CIRCUITS (Libes) "A great book for use as a reference by people who are learning digital electronics." PEOPLE'S COMPUTER COMPANY. #5505-6, paper, \$6.95

BASIC BASIC: An Introduction to Computer Programming in BASIC Language (Coan) "... an excellent introduction ... clearly written and well-organized." COMPUTING REVIEWS. #5872-1, paper, \$7.95

ADVANCED BASIC: Applications & Problems (Coan) "This one rates well above average." DATA PROCESSING DIGEST. #5855-1, paper, \$6.95



in personal computing books!

AVAILABLE AT YOUR LOCAL COMPUTER STORE!



MORE CHOICES

GAME PLAYING WITH COMPUTERS, Revised Second Edition (Spencer) #5103-4, cloth, \$16.95

COMPUTERS IN ACTION: How Computers Work (Spencer) #5861-6, paper, \$5.50

COMPUTERS IN SOCIETY: The Wheres, Whys, & Hows of Computer Use (Spencer) #5915-9, paper, \$5.50

PROGRAMMING PROVERBS (Ledgard) #5522-6, paper, \$6.50

PROGRAMMING PROVERBS FOR FORTRAN PROGRAMMERS (Ledgard) #5820-9, paper, \$6.50

COBOL WITH STYLE: Programming Proverbs (Chmura & Ledgard) #5781-4, paper, \$5.45

MINICOMPUTERS: Structure & Programming (Lewis & Doerr) #5642-7, cloth, \$12.95

DIGITAL SIGNAL ANALYSIS (Stearns) #5828-4, cloth, \$19.95

FORTRAN FUNDAMENTALS: A Short Course (Steingraber) #5860-8, paper, \$4.95

DIGITAL TROUBLESHOOTING: Practical Digital Theory and Troubleshooting Tips (Gasperini) #5708-3, paper, \$9.95

DIGITAL EXPERIMENTS: Workbook of IC Experiments (Gasperini) #5713-X, paper, \$8.95

COMPUTER MATHEMATICS (Conrad, Conrad, & Higley) #5095-X, cloth, \$13.95

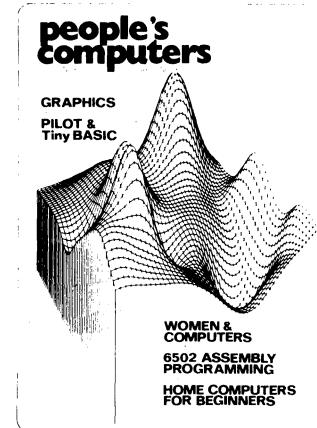
50 Essex Street, Rochelle Park, New Jersey 07662

WE'RE DIFFERENT!

No Ads

No Frills

All Content



People's Computers is for anybody and everybody who wants to learn more about computers, how they work, and how to use them. We aren't padded with color advertisements either; we're wall to wall with articles, listings, reviews, games, letters, and interviews.

Read the whys and wherefors of different computer languages. Discover what a 'chip' is, how a 'gate' works, and what new products are available. Learn to program from our series on easy-to-use computer languages. Find out about the uses of computers and calculators in education. Each issue also contains useful programs, with extensive comments, that you can use on your computer at home, at school, or at work.

There's lots more, and all for only \$8 a year! To subscribe write: People's Computers, Dept 52, 1263 El Camino Real, Menlo Park CA 94025.

people's computers

**S-100
COMPATIBLE
BLANK
BOARDS**

Z-80 CPU	\$35.00
2708/16 EPROM	\$25.00
PROTOBOARD	\$25.00
8k STATIC	\$25.00
16/64 DYNAMIC RAM	\$35.00

PLEASE ADD \$2.00 SHIPPING PER ORDER.

iithaca audio (607)
273-3271
PO BOX 91 ITHACA, N.Y. 14850

**CONVERT YOUR IBM
SELECTRIC INTO A HARD
COPY OUTPUT TERMINAL!**

WITH OUR KIT, YOU CAN
ACTUALLY ADD A FULL SIZE
IMPACT PRINTER TO YOUR
COMPUTER - AT A PRICE
YOU CAN EASILY LIVE WITH!

NO MODIFICATIONS TO YOUR
TYPEWRITER ARE NECESSARY -
AND IF YOU DON'T HAVE ONE,
WE'LL GET IT FOR YOU!

PLEASE CONTACT US AT...

ESCON

1235 TENTH STREET
415/524-8664

BERKELEY
DEPT. J

Now There Are Three

- **northern california
electronics news**

published bi-weekly

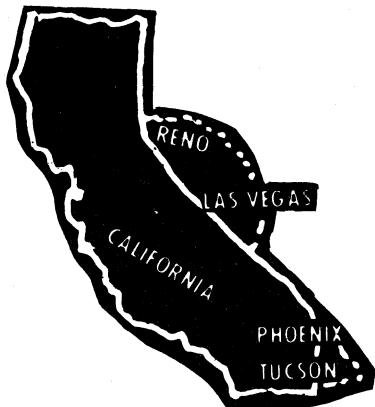
- **southern california
electronics news**

published bi-weekly

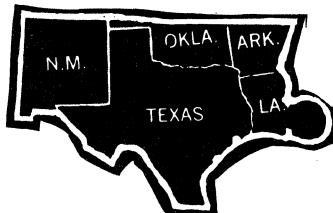
- **southwest
electronics news**

published quarterly

Covering the States of



**CALIFORNIA
TEXAS
OKLAHOMA
NEW MEXICO
ARKANSAS
LOUISIANA**



No Other Publication Emphasizes Local and Regional
News in the above Six (6) States

Each paper Has its own News of People and Events in its own Area
plus National Product News, thus guaranteeing High Readership

published by

BENDER PUBLICATIONS INC.

LOS ANGELES

P.O. Box 3631
L.A. Ca. 90019

(213) 737-6820

PALO ALTO

260 Sheridan
Palo Alto, CA. 94306

(415) 326-8877

DALLAS

6115 Denton Dr.
Dallas, TX 75235

(214) 358-2311

HOUSTON

2990 Richmond
Houston, Tx 77006

(713) 526-5381

Computer Music Journal



Devoted to High Quality Musical Applications of Digital Electronics.

The following topics are covered:

- ★ design of real time playable instruments with controllers like organ keyboards, pressure sensitive surfaces, joysticks, and new designs;
- ★ production of natural sounding quality of tone or timbre by Fourier series like synthesis (with periodic or nonperiodic sine wave summation) or FM synthesis and new methods;
- ★ circuit design of digital oscillators (any waveshape — up to 256 ultra low distortion sine waves with independent control of amplitude and frequency — all from one digital oscillator) controlled by a small computer;
- ★ high speed, high resolution multiplication;
- ★ review of hardware components;
- ★ cost of hardware / quality of sound tradeoffs;
- ★ control of analog synthesizers with a small computer;
- ★ envelope generators (not just exponential or linear attack, sustain, and decay, but any shape);
- ★ digital filtering;
- ★ high resolution, high speed digital to analog converters;
- ★ digital reverberation and movement of spacial location with Doppler shifting;
- ★ analysis of acoustic instruments;
- ★ psychoacoustics;
- ★ generation of different musical scales including meantone, just, and equal tempered (with 5, 7, 12, 19, 22, 29, 31, or 43 notes in each octave) tunings;
- ★ reviews of books about computer music, acoustics of musical instruments, psychoacoustics, music theory and composition, computer design, and electronics.

Computer Music Journal is published every other month by People's Computer Company, a non-profit, educational corporation. For a one year subscription (6 issues) send \$14 to *Computer Music Journal*, PCC, Dept 51, Box E, Menlo Park, CA 94025.

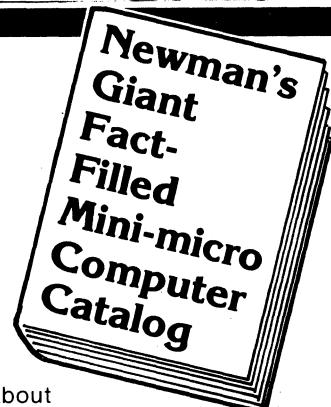
KITS
GALORE and
MORE

Fairchild Tech. Kits™
Clocks — Counters
Power Supply
Micro Computer
Function Generator

Send for our brochure TODAY
E Tronix
Box 321
Issaquah, WA 98027

FREE!

with your
business
card or send
\$1.00 (re-
fundable on
1st order)
Dept. CF



68 pages of News about
the Amazing Technological
Breakthroughs in the
Mini-Micro Computer Field.

Catalog Includes:

- Reproductions of manufacturer's complete catalogs, including Imsai's — normally \$1.00.
- Articles and news about Mini-micro Computers.
- \$2.00 Discount Certificate.
- Discounts up to 90% on Used Equipment.

Catalog offers items like:

- \$279 Complete Computer System for home use. Not a kit!
- Low cost new and used peripherals

LARGEST CATALOG IN ITS FIELD

NEWMAN COMPUTER EXCHANGE
1250 N. Main St. Ann Arbor, Mi. 48104
(313) 994-3200

A veritable revolution of new ideas about programming, new languages, hardware design, robots, video games, analog interfaces and new memory technology (just to name a few) presents the computer experimenter with a pleasant (if sometimes frustrating) dilemma: How do I choose the best books? Where do I get them? What is available?

BYTE's BITS (BYTE Interface Technical Services, Inc) may be the answer. Bookselling is our main business: With the help of BYTE magazine's editors, we screen the scores of computer books that come out every week and offer you only the best. Books like:

Microcomputer Handbook by Charles Sippl
What To Do After You Hit Return . . . -PCC's First Book of Computer Games
Computer Lib-Dream Machines by Ted Nelson
Best of BYTE, Vol. 1
APL—An Interactive Approach, second edition, revised, by Gilman and Rose
Instant BASIC by Jerald Brown
How to Buy & Use Minicomputers & Microcomputers by William Barden
Practical Microcomputer Programming: The Intel 8080 by Weller, Shatzel, Nice
The Acoustical Foundations of Music by John Backus
Computer Power and Human Reason by Joseph Weisenbaum
Fundamentals and Applications of Digital Logic Circuits by Sol Libes
A Dictionary of Microcomputing by Philip E Burton
Understanding Microcomputers and Small Computer Systems from Scelbi
Scelbi's "8080" and "6800" Software Gourmet Guide & Cookbooks
The Don Lancaster Series: **TTL Cookbook**, Active Filter CB
 TV Typewriter, CB,
 CMOS Cookbook
Fingertip Math from Texas Instruments
Build Your Own Working Robot by David L Heiserman
101 BASIC Computer Games by David Ahl
Chess Skill in Man and Machine by Peter W Frey
Electronic Projects for Musicians by Craig Anderton
Digital Computer Fundamentals by Jefferson C Boyce
The Art of Computer Programming, Volumes 1, 2, & 3 by Donald E Knuth
An Introduction to Microcomputers, Volumes 1 & 2 by Adam Osborne
Some Common BASIC Programs by Poole & Borchers
Plus most of the reference DATA books from Texas Instruments

The computer information explosion is here!

If you would like our catalog (a reference book in itself) crammed with information and descriptions of these books, plus many many more, just fill out the coupon below and send it along with \$1 (which incidentally will be applied to your first order) to BITS, Inc, 70 Main St, Peterborough NH 03458.



Send to: **BITS Catalog**
BITS, Inc.
70 Main Street
Peterborough NH 03458

Name _____

Address _____

City _____ State _____ Zip Code _____

Signature _____

Check Payment method:

My check is enclosed
 Bill my MC No. _____ Exp. date _____
 Bill my BAC No. _____ Exp. date _____

_____ BITS Catalog \$1.00

In unusual cases, processing may exceed 30 days.

MINI-MICRO SYSTEMS is the only monthly publication devoted exclusively to the fast-paced world of mini and microcomputers:

- TECHNOLOGY AND PRODUCTS
- PERIPHERALS AND SOFTWARE
- BUSINESS AND FINANCIAL
- APPLICATIONS AND SYSTEMS
- INDUSTRY AND CORPORATE



If you are not currently receiving MINI-MICRO SYSTEMS magazine, write today
for a sample issue and a qualifying form to receive a FREE subscription.

MINI-MICRO SYSTEMS / 5 Kane Industrial Drive / Hudson, MA 01749

CAREER AND SPECIAL ASSIGNMENT OPPORTUNITIES

Rapid growth has opened up editorial opportunities at Mini-Micro Systems magazine — full time staff positions, regional correspondents, and specialized subject assignments. If you are interested in any of this work, please send a resume, along with a brief statement on how you would like to associate with Mini-Micro Systems, to:

Stanley Klein, Editor-in-Chief, Mini-Micro Systems, 5 Kane Industrial Drive, Hudson, MA 01749

For the microcomputer enthusiast ...



YOUR HOME COMPUTER — James White

For the pre-hobbyist and the microcomputer novice, this book provides a complete introduction to the world of home computing. Answers to your many questions about hardware, software, and the personal computing scene today.

1977, 220 pages. \$6.00

INSTANT BASIC — Jerald Brown

Written for the inexperienced, this activity-oriented book will help you teach yourself microcomputer BASIC and the similar DEC BASIC PLUS, for programming your personal computer. There's never a dull page and plenty of activities... so have fun while you learn!!!

1977, 180 pages. \$6.00

AND DON'T FORGET THESE...

My Computer Likes Me — Albrecht. In an easy-going, conversational style, this 64 page workbook introduces BASIC to young or old. This classroom favorite is designed for people with no previous computer experience or knowledge of programming. 1972, \$2.00.

Games With the Pocket Calculator — Thiagarajan and Stolovitch. This book features interactive "group" games to be played by small groups of young or old. An entertaining book for use at home; an educational book to introduce students to the calculator and to build computational skills. 1976, \$2.00.

Games, Tricks, and Puzzles For A Hand Calculator — Judd. An entertaining and useful book for anyone with a hand calculator. The author covers many of your favorite mathematical games and recreations. Learn how to perform calculator tricks. 1974, \$2.95.

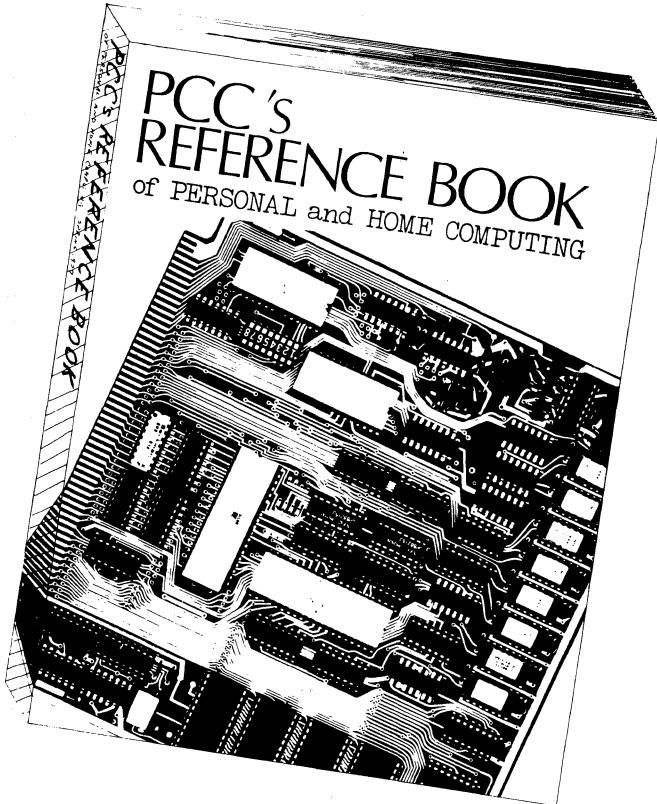
CALCULATORS/COMPUTERS Magazine — This new magazine contains creative articles and materials that will teach you about calculators and computers. Each issue includes "COPY ME" materials that may be reproduced to use at home or for classroom use (elementary school level through college). Topics include learning about your calculator, programming in BASIC, as well as games, simulations, and interesting articles. Subscription price: \$12/year (7 issues).

Handling and shipping charges will be billed.

DYMAX
PUBLISHER

P.O. Box 310, Dept. 15
Menlo Park, CA 94025

Available at your local computer store.



FINALLY! A Reference Book for Home Computer Users

Ever try to find the address of, say, a particular floppy disc manufacturer when you don't even know what state they're in? Looking for articles on a certain kind of hardware? Frustrating, isn't it? Well, People's Computer Company has just published a valuable reference directory that goes a long way towards ending that frustration.

Here's just some of the useful information you'll find in *PCC's Reference Book*

- ★ Hundreds of companies and computer stores selling hardware, software, peripherals and offering all sorts of services are listed with their addresses.
- ★ Nuts-and-bolts and survey articles on software, hardware, applications, robots, and the future, just to name a few, for the experienced and the not-so-experienced user of microcomputers.
- ★ The complete, documented source and object code for a 2K Tiny BASIC.
- ★ A massive index of the articles from the major hobbyist magazines, plus information on the magazines in the field, hobbyist clubs, newsletters and professional societies.
- ★ Bibliographies on different areas so you can investigate them further, including a special computer music bibliography.

It's a book you'll want to keep handy because you'll use it a lot. And even when you aren't looking up addresses or other information, you'll be referring back to one of the many helpful articles. *PCC's Reference Book* is available for \$5.95 plus 95¢ for shipping and handling in the U.S. (California residents add 35 cents sales tax).

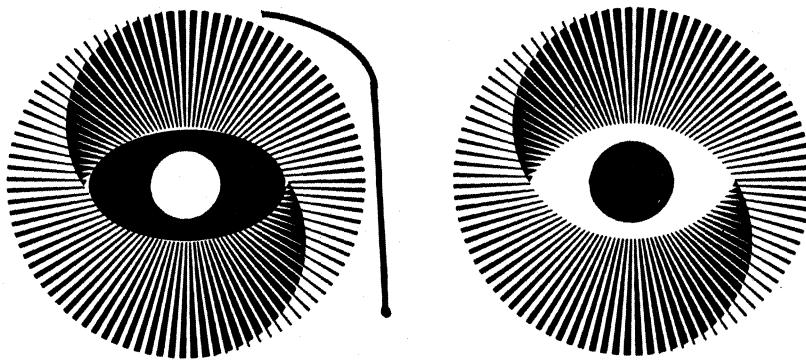
Please send me my copy of *PCC's Reference Book of Personal and Home Computing*. I am enclosing \$5.95 plus 95¢ for shipping and handling in the U.S. (In California add 35¢ for sales tax). If I am not completely satisfied, I can have a full refund.

NAME _____

ADDRESS _____

CITY/STATE/ZIP _____

Send to:
PCC's Reference Book
Dept. H
1263 El Camino Real
Box E
Menlo Park, CA 94025



LOOKING FOR A CONVENIENT SOURCE FOR ALL THE POPULAR MICROCOMPUTER BOOKS?

Here's your answer: Ham Radio's Communications Bookstore. Write for our FREE extensive Book Catalog listing all the popular microprocessor/microcomputer books. Bug Books, Adam Osborne, Scelbi, McGraw Hill, Howard W. Sams and more . . .

And a handy TOLL FREE number — your books will be on their way within 24 hours!

And while you're writing . . .

Why not ask for a FREE copy of Ham Radio HORIZONS? Discover the fascinating FUN hobby of Amateur Radio — you'll be amazed at how many applications your computer will have in the world of ham radio!

Ham Radio HORIZONS is the only Amateur Radio magazine that quenches the thirst for the newcomer (or does it start a hunger, we're not sure). Try a free issue, or even better, a subscription! One year, (12 issues) \$10.00.



**ham
RC
BC**
radio's
communications
bookstore

GREENVILLE, NH 03048

You should subscribe to creative computing today! Here's why —

1.

Creative Computing will help you decide which computer is best for you.

Creative's no-nonsense equipment profiles arm you with the facts before you talk to the vendor or dealer. Whether you're interested in a microcomputer kit, a mini, terminal, or programmable calculator, you'll find the evaluative information you need in *Creative*. Indeed, one wise hardware decision could save you the cost of a lifetime subscription!

2.

Creative Computing discusses computer applications in non-technical, understandable language.

Business people who want to know what's going on in the EDP department, students who want to learn about microprocessors, hobbyists looking to make good use of home computers, or anyone concerned about the effect of the computer on society will find these and many, many more mind-expanding topics covered on the pages of *Creative*.

3.

Creative Computing covers computer education in depth.

After all, that's where we got our start and so we continue to present four or five major learning activities every issue. If you're a teacher, *Creative* will save you hours of preparation time. If you're a student, you'll be way ahead of your class with *Creative*. And if you've already graduated, you can bone up on what you missed.

4.

Creative Computing carries outstanding fiction every issue.

One of the best ways of exploring future scenarios of computer usage is through fiction, so *Creative* seeks out material from the best authors — Isaac Asimov, Frederik Pohl, Arthur C. Clarke to name just a few, as well as many others who are destined to be the best of the next generation.

5.

Creative Computing's book reviews are honest and timely.

We're not owned by a big book publisher to whom we owe loyalty, nor do we depend upon advertising for our revenue. Hence, not only do our reviews pull no punches, but we also rank order similar books (like all 34 books on the BASIC language which we reviewed last year). *Creative* reviews virtually every computer book of interest to educators, hobbyists, and general laypeople, even including vendor manuals and government pamphlets.

6.

An extensive resource section will save you time and money.

Every issue of *Creative* carries 40 or more short resource reviews evaluating periodicals, booklets, hardware, terminals, couplers, peripherals, software packages, organizations, dealers, and much more. Every entry has a brief description, evaluation, and the name, address, and phone number of the vendor. You'll save valuable time seeking out this information, much of which you'd possibly never come across.

7.

Creative Computing will provide hours of mind-expanding entertainment, even if you don't have a computer.

Creative Computing carries 10 or 12 pages of games and puzzles every issue. Most of the puzzles don't need a computer or calculator to solve; some do. Naturally, the 4 or 5 new computer games (in Basic, Fortran, and APL) in every issue require access to a computer.

8.

Creative Computing gives you things to actually do with a computer.

Home computer kit, mini, timesharing terminal — whatever your access to computer power, *Creative* provides thoroughly documented programs with complete listings and sample runs that you can use with minimum effort. Games, simulations, CAI, computer art — whether education or recreation is your bag, you'll find programs which you can use in *Creative*.

9.

A no-compromise policy of editorial excellence means every issue is of value to you.

We firmly intend to be around a long time and we believe the way to do that is to publish only material of the very highest quality. We believe our readers are among the most astute, intelligent, and active people in the world. You can't afford to waste time reading imprecise, opinionated, or wordy articles and you won't find any in *Creative*.

10.

The price is right — only \$21 for 3 years.

That same \$21 will buy you a pair of Star Trek walkie talkies, six direct dialed 10 minute calls between New York and Boston, 3 tankfuls of gas, or 10 cocktails at a Hilton hotel. Wouldn't you rather have 18 issues of *Creative Computing* each containing over 85 pages of solid editorial material (including advertising, over 100 pages per issue). Count the editorial pages in any other hobbyist or special interest magazine and compare it to *Creative*. Any other 1 year subscription \$8. Lifetime \$300.

NO RISK GUARANTEE

You may cancel your subscription at any time for any reason and we will refund the balance without question.

David H. Ahl, Publisher

FOR FASTER RESPONSE

800-631-8112

(In NJ, call 201-540-0445)

SUBSCRIPTION ORDER FORM

Type	Term	USA	Foreign
Individual	1-Year	<input type="checkbox"/> \$ 8	<input type="checkbox"/> \$ 11
	3-Year	<input type="checkbox"/> 21	<input type="checkbox"/> 30
	Lifetime	<input type="checkbox"/> 300	<input type="checkbox"/> 400
Institutional	1-Year	<input type="checkbox"/> 15	<input type="checkbox"/> 15
	3-Year	<input type="checkbox"/> 40	<input type="checkbox"/> 40

New Renewal

Cash, check, or M.O. enclosed

BankAmericard Card No. _____

Master Charge Expiration date _____

Please bill me (\$1.00 billing fee will be added)

Name _____

Address _____

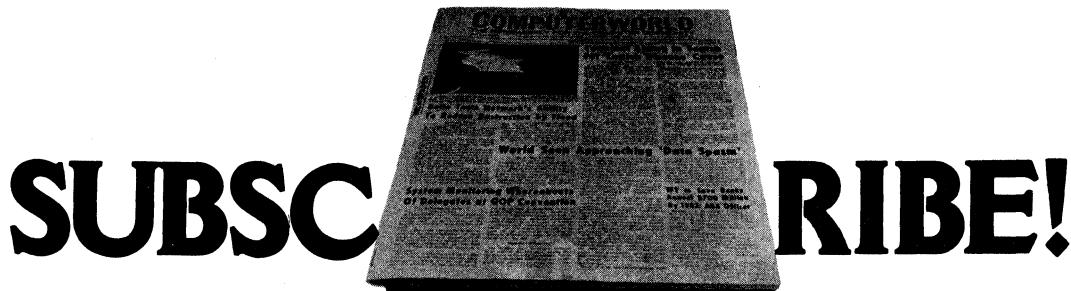
City _____ State _____ Zip _____

Send to: Creative Computing, Attn: Wanda
P.O. Box 789-M, Morristown, NJ 07960

Computerworld's weekly Microcomputing section will keep you up to date on the latest micro products, services and applications — from home to hobby to business.

Computerworld's weekly Microcomputing Classified Exchange will help you find unusual products and services, good bargains and people with similar interests.

ALL YOU HAVE TO DO IS...



Please send me **COMPUTERWORLD** for 1 year RATES: U.S. - \$18 Canada and PUAS - \$28

Check Enclosed

Charge My Amer

Europe & Middle East - \$50 All Other Foreign - \$65

Expiration Date on Card _____

Send to: Address										
City						State	Zip Code			

Address shown is: Check here if you do not
 Business wish to receive promotional
 Home mail from Computerworld.



CIRCULATION DEPT. 797 Washington Street, Newton, MA 02160

PLEASE CIRCLE 1 NUMBER IN EACH CATEGORY

BUSINESS/INDUSTRY

- 10 Manufacturer of Computer or DP Hardware/Peripherals
 - 20 Manufacturer (other)
 - 30 DP Service Bureau/Software/Planning/Consulting
 - 40 Public Utility/Communication Systems/Transportation
 - 50 Wholesale/Retail Trade
 - 60 Finance/Insurance/Real Estate
 - 70 Mining/Construction/Petroleum/Refining
 - 75 Business Service (except DP)
 - 80 Education/Medicine/Law
 - 85 Government-Federal/State/Local
 - 90 Printing/Publishing/Other Communication Service
 - 95 Other:

TITLE/OCCUPATION/FUNCTION

- 11 President/Owner/Partner/General Manager
 - 12 VP/Assistant VP
 - 13 Treasurer/Controller/Finance Officer
 - 21 Director/Manager of Operation/Planning/
Administrative Service
 - 22 Director/Manager/Supervisor PP
 - 23 Systems Manager/Systems Analyst
 - 31 Manager/Supervisor Programming
 - 32 Programmer/Methods Analyst
 - 41 Application Engineer
 - 42 Other Engineering
 - 51 Mfg. Sales Representative
 - 52 Other Sales/Marketing
 - 60 Consultant
 - 70 Lawyer/Accountant
 - 80 Librarian/Educator/Student
 - 90 Other: _____

CONFERENCE PROCEEDINGS

Jim C. Warren, Jr., Editor

**THE FIRST WEST COAST COMPUTER FAIRE
HELD IN
San Francisco's Civic Auditorium
WITH
Banquets in the St Francis Hotel**

April 15-17, 1977

**COMPUTER FAIRE
Box 1579
Palo Alto CA 94302
(415) 851-7664**

© COMPUTER FAIRE, INC. 1977

ALL RIGHTS RESERVED.
PRINTED IN U.S.A.

ISBN 0-930418-00-X

LIBRARY OF CONGRESS CATALOG CARD #77-82685

P R E F A C E

Home and hobby computing developments are racing along as such a pace that they make even the prodigious progress taking place in the fields of electronics and computer science and engineering appear to be leisurely motion. In the brief, two and a half year "history" of personal computing, we have seen it grow from an obscure hobby involving perhaps a few hundred home computers to the current situation in which there are possibly 50,000 or more general purpose digital computers in private ownership for personal use.

The frantic pace of the developments in this field has completely out-stripped the capabilities of the traditional information media for distributing state-of-the-art progress reports and data. The media are too slow; the entry of newcomers to the field from every direction makes the demand too widespread and varied.

As a result, several exciting and informative conventions have been held -- exclusively devoted to home and hobby computing. Each one has included a valuable conference program, usually including talks by a number of the "frontiersmen" in this new field. However, until the First West Coast Computer Faire, none of these conferences produced a written record of their talks and presentations.

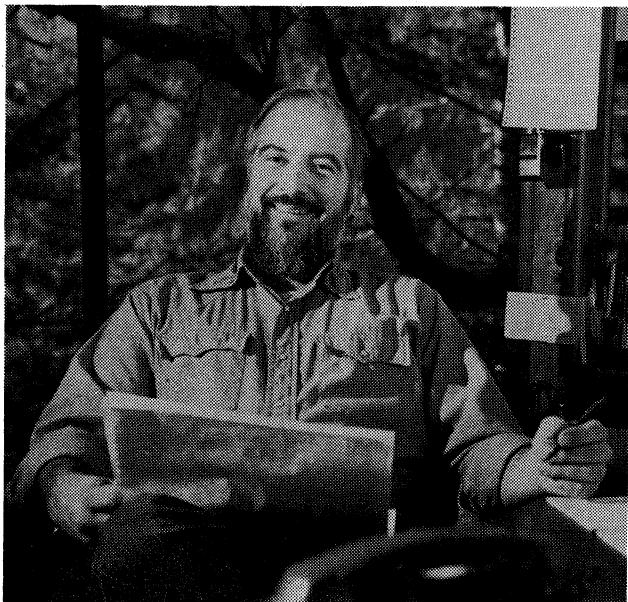
From the outset, one of the major goals of the Computer Faire was to produce these *Conference Proceedings*. Our goal -- from the beginning -- was to blend the best things from the previous hobby conventions -- enthusiasm, excitement, and an informal atmosphere that was highly conducive to easy information exchange -- with the best things from the professional computer conventions...including the production of a written record of the Conference Program.

By publishing this document, we allow the entire community of individuals interested in personal computing to share the massive collection of information -- much of it involving state-of-the-art developments -- that was presented in the Faire's Conference Program. By publishing it, we provide a permanent record of the talks for those who attended; we provide a reference document for those who were unable to attend; and, we provide access to the content of the conference activities to those newcomers to personal computing -- who weren't even involved in the subject area at the time of the Faire.

Additionally, we have included a listing of the names, addresses, and telephone numbers of most of the commercial exhibitors at the Faire. Admittedly, the field is changing so rapidly that parts of this list are already out of date -- parts of it were out of date by the time the Faire occurred, within days of its compilation. However, it provides a starting place for those seeking vendor information, and -- in this fast-moving marketplace -- that is not easily obtained.

We hope that the publication of these *Conference Proceedings of the First West Coast Computer Faire* will encourage the leaders of the many other such personal computing conventions to publish similar records of their conference programs. It involves a major expenditure of time and effort, however we believe that it is an invaluable service to our fast-growing and rapidly changing community.

Jim C. Warren, Jr., Faire Chairbeing
Woodside, California
77 August 1

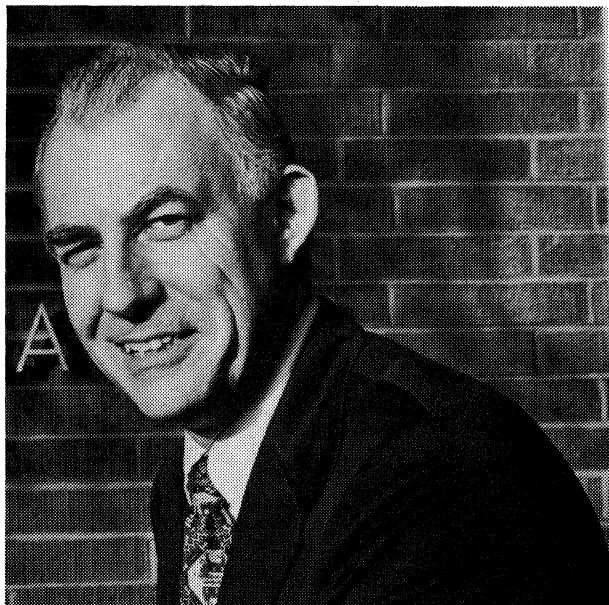


JIM WARREN, Faire Chairperson

&

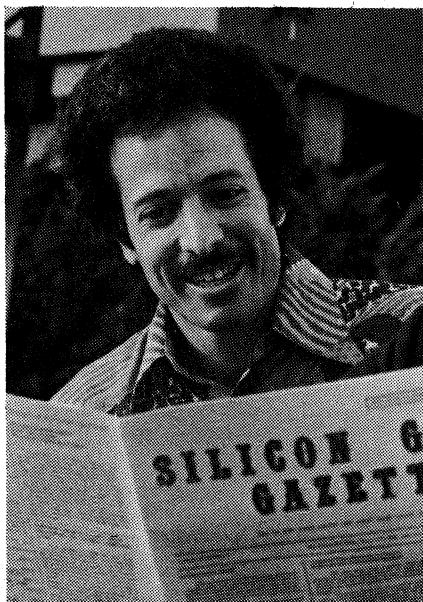
Editor, *Dr Dobb's Journal of Computer Calisthenics & Orthodontia*
People's Computer Company
Box E

Menlo Park CA 94025



ROBERT REILING, Faire Operations
Coordinator &

Editor, *Homebrew Computer Club Newsletter*
Homebrew Computer Club
Box 626
Mountain View CA 94042



RICK BAKALINSKY, Wandering Minstrel
&

Midwife-at-Large; Explorer
2464-1 W. Bayshore, Sweet 1
Palo Alto CA 94303

CO-SPONSORS OF THE FAIRE

Homebrew Computer Club
Box 626
Mountain View CA 94042
Southern California Computer Society
Box 5429
Santa Monica CA 90405
San Francisco Peninsula ACM* Chapter
Box 60355
Sunnyvale CA 94088
Golden Gate ACM* Chapter
Box 26044
San Francisco CA 94126
Santa Clara Valley IEEE-CS* Chapter
701 Welch Road No. 2210
Palo Alto CA 94304
California Mathematics Council
3360 Tonga Lane
Alameda CA 94501
Electrical Engineering Department
Stanford University
Stanford CA 94305
Lawrence Hall of Science
University of California
Berkeley CA 94720

People's Computer Company
Box E
Menlo Park CA 94025
Community Computer Center
1919 Menalto
Menlo Park CA 94025
Bay Area Microcomputer Users Group
1211 Santa Clara Avenue
Alameda CA 94501
Professional & Technical Consultants Assn.
1 First Street
Los Altos CA 94022
Amateur Research Center
Space Science Center
Foothill College
Los Altos CA 94022

*ACM: Association for Computing Machinery
IEEE-CS: Institute of Electrical & Electronics
Engineers - Computer Society

TABLE OF CONTENTS

Preface, Jim C. Warren, Jr.	3
Computer Faire Organizers	4
Co-Sponsors of the Faire	5
Table of Contents	5
Conference Referees	7

BANQUET PRESENTATIONS

Robots You Can Make for Fun & Profit, Frederik Pohl	8
Digital Pyrotechnics: The Computer in Visual Arts, John H. Whitney	14
The 1940's: The <u>First</u> Personal Computing Era, Henry Tropp	17
Those Unforgettable Next Two Years, Ted Nelson	20

TUTORIALS FOR THE COMPUTER NOVICE

An Introduction to Computing to Allow You to Appear Intelligent at the Faire, James S. White	26
A Tyro Looks Back, Fred Waters	30
The Shirt Pocket Computer, Richard J. Nelson	31
The Sidelobes of Industrial Distribution are Focused on the Home Microcomputer Hobbyist, Lowell Smilen, Ph.D.	39

PEOPLE & COMPUTERS

If "Small is Beautiful," Is Micro Marvelous? A Look at Micro-Computing as if People Mattered, Andrew Clement	42
The Computer in Science Fiction, Dennie L. Van Tassel	49
Computer Power to the People: The Myth, the Reality and the Challenge, David H. Ahl	51
Psychology and the Personal Computer, Kenneth Berkun	55

HUMAN ASPECTS OF SYSTEM DESIGN

Human Factors in Software Engineering, James Joyce	56
The Human Interface, William F. Anderson	64

PERSONAL COMPUTERS FOR THE PHYSICALLY DISABLED

The Potential of Microcomputers for the Physically Handicapped, Peter J. Nelson & J.G. Cossalter	65
An Interface Using Bio-Electrical Signals to Control a Microprocessor System for the Physically and Communicatively Handicapped, Laurence R. Upjohn, Pharm. D.	70

LEGAL ASPECTS OF PERSONAL COMPUTING

What to Do After You Hit Return . . . and Nothing Happens: Warranty in the Micro-Computer Industry, Kenneth S. Widelitz, Attorney at Law, WA6PPZ	72
--	----

HERETICAL PROPOSALS

Here Comes the Brain-Like, Self-Learning, No-Programming, Computer of the Future, Klaus Holtz	78
---	----

COMPUTER ART SYSTEMS

Composing Dynamic Laser Light Sculptures Via a Hybrid Electronic Wave System, Ronald Pellegrino	89
Computer Generated Integral Holography, Michael Fisher	89
Digital Video Painting, Dick Shoup	89
Electronically Produced Video Graphics Animation, Terry Craig	89
Roaming Around in Abstract 3-D Spaces, Tom DeFanti, Dan Sandin and Larry Leske	89
Video Synthesis: Expanding Electronic Vision, Stephen Beck	89
Video Synthesis & Performance with an Analog Computer, Jo Ann Gillerman	90

MUSIC & COMPUTERS

The Stanford Computer Music Project, John Chowning and James A. Moorer	91
Design of High Fidelity Real-Time Digital Hardware for Music Synthesis, John Snell	96
The Kludgehorn: An Experiment in Homebrew Computer Music, Carl Helmers	118
Notes on Microcomputer Music, Marc LeBrun	128
A Pipe Organ/Micro Computer System, Jef Raskin	131
A Computer Controlled Audio Generator, Thomas E. Olsen	134

ELECTRONIC MAIL

DIALNET and Home Computers, John McCarthy & Les Earnest	137
CB Computer Mail?, Raymond R. Panko, Ph.D.	139

COMPUTER NETWORKING FOR EVERYONE

Community Memory — a "Soft" Computer System, Lee Felsenstein	142
Design Considerations for a Hobbyist Computer Network, David Caulkins	144
A Network of Community Information Exchanges: Issues and Problems, Mike Wilbur	149

PERSONAL COMPUTERS FOR EDUCATION

Sharing Your Computer Hobby with the Kids, Liza Loop	156
Personal Computing & Education: A Time For Pioneers, Thomas A. Dwyer	161
The Things That We Can Do with a Microcomputer in Education That We Couldn't Do Before, Lud Braun	163
Classroom Microcomputing: How One School District Learned to Live with the State of the Art, Peter S. Grimes	165
The Construction, Operation, and Maintenance of a High School System, Melvin L. Zeddis	170
Educating People about Personal Computing: A Major Program at Lawrence Hall of Science, Bob Kahn & Lee Berman	173
CAI Answer Processing in BASIC, Franz J. Frederick	175
Telemath, Lois Noval	178
Student Records Subroutine for Computer-Assisted Instruction Lessons in Extended BASIC, Franz J. Frederick	180
A Question-Answering System on Mathematical Models in Microcomputer Environments, Milos Konopasek & Mike Kazmierczak	182
Use of a Personal Computer in Engineering Education, Roger Broucke	187
The Microcomputer Education Process: Where We've Been and Some Guesses on Where We're Going, Merl K. Miller	191

RESIDENTIAL ENERGY & COMPUTERS

Microcomputers: A New Era for Home Energy Management, Mark Miller	194
---	-----

COMPUTERS & SYSTEMS FOR VERY SMALL BUSINESSES

The Emperor has Few Clothes — Applying Hobby Computer Systems to Small Business, Michael Levy	196
---	-----

ENTREPRENEURS

The Software Dilemma, Carl Helmers	200
Tax Aspects of Lemonade Stand Computing: When is a Hobby Not a Hobby?, Kenneth S. Widelitz	202
Study of the Emerging Consumer Computing Marketplace, Walter Smith	203

SPEECH RECOGNITION & SPEECH SYNTHESIS BY HOME COMPUTER

Speech Recognition Systems, John Reykjalin & Horace Enea	206
Speech Synthesis by a Set of Rules (Or, Can a Set of Rules Speak English?), D. Lloyd Rice	209
Top-Down Analysis of Language Rhythms in Speech Synthesis, Alice Wyland Grundt, Ph.D.	214

TUTORIALS ON SOFTWARE SYSTEMS DESIGN

Home Text Editing, Larry Tesler	220
Learning to Program Microcomputers? Here's How!, R.W. Ulrickson	224
Structured Programming for the Computer Hobbyist, Ed Keith	228

IMPLEMENTATION OF SOFTWARE SYSTEMS AND MODULES

An Interpretive Approach to Programming Language Implementation, Dennis Allison	231
Numerical Calculations on Microprocessors, Roy Rankin	235
Modular Relocatable Code, Dennis Burke	240
An Implementation Technique for MUMPS, David D. Sheretz	242

HIGH LEVEL LANGUAGES FOR HOME COMPUTERS

Computer Languages: The Key to Processor Power, Tom Pittman	245
A PILOT Interpreter for a Variety of 8080-Based Systems, John A. Starkweather	248
Design and Implementation of HI, Martin Buchanan	250
Fortran for Your 8080, Kenneth B. Welles II, Ph.D.	254
EMUL-8: An Extensible Microcomputer User's Language, Bob Wallace	255

MULTI-TASKING ON HOME COMPUTERS

EMOS-8: An Extensible Microcomputer Operating System, Bob Wallace	260
A New Approach to Time-Sharing with Microcomputers, Joseph G. McCrate	265
Microcomputers and Multi-Tasking: A New Dimension in Personal Computing, George Pilipovich	267

HOME BREW HARDWARE

Interfacing a Selectric to Your Computer, Carl Townsend	269
A Floppy Disk Controller for Under \$50, Kenneth B. Welles II, Ph.D.	272
A Fault Isolation Troubleshooting System for the Multi Vendor Environment, Robert A. Tuttle, Jr.	273
Solenoids Provide Software Control of a Home Cassette Recorder, William J. Schenker, M.D.	276

BUS & INTERFACE STANDARDS

A Microprocessor Independent Bus, Ceasar Castro & Allen Heaberlin	277
16-Bit and 32-Bit Adaptations of the S-100 Bus Standard, Gary McCray	282
Standardization of the S-100 Bus: Timing and Signal Relationships – A Proposed Standard, Tony Pietsch	284
DMA Operation Protocol in the S-100 Bus Environment, James T. Walker	286
A Biomedical Application Using the S-100 Bus Standard, William J. Schenker, M.D.	291

MICROPROGRAMMABLE MICROPROCESSORS FOR HOBBYISTS

VACuuM: A Variable Architecture Computing Machine, Tom Pittman & Bob Davis	294
Large Scale Computers for the Hobbyist, David C. Wyland	304
Bipolar Microprocessor Microphobia, John R. Mick	307
Microprogramming for the Hobbyist, John Birkner	309

AMATEUR RADIO & COMPUTERS

Ham RTTY: Its Evolution and Future, Robert C. Brehm	312
Generate SSTV with your SWTPC 6800 Microprocessor, Clayton W. Abrams, K6AEP	315
CW Operator's Utopia – Automatic Transmission and Reception, Ivar Sanders, W6JDA	317
Microprocessor Control of a VHF Repeater, Lou Dorren, WB6TXD	321
Amateur Radio & Computer Hobbyist Link Via RTTY Repeater, Alan Bowker & Terry Conboy	322

COMMERCIAL HARDWARE

The New Microprocessor Low Cost Development Systems, Phil Roybal	323
A Megabyte Memory System for the S-100 Bus, Glenn E. Ewing, Senior Engineer	326
A New Approach to Microcomputer Systems for Education, Alice E. Ahlgren, Ph.D.	327
A Computerized PROM Programmer, PROM Emulator, and Cross Assembler System, Richard Erickson	329

HOME BREW EXHIBITORS

COMMERCIAL EXHIBITORS	331
---------------------------------	-----

CONFERENCE REFEREES

The Computer Faire would like to express its sincere thanks for the time and effort expended by the following individuals in refereeing the multitude of papers and abstracts that were submitted to be considered for presentation in the Conference activities at the Faire.

Committee of Referees

Phyllis Cole, Editor, <i>People's Computer Company</i>
Marc Le Brun, Personal Computers Editor, <i>Co-Evolution Quarterly</i>
Bob Reiling, Editor, <i>Homebrew Computer Club Newsletter</i> , & Computer Faire Operations Coordinator
Larry Tesler, Xerox Palo Alto Research Center
David Wyland, Raytheon Semiconductor
Jim Warren, [ex-officio member] Editor, <i>Dr. Dobb's Journal of Computer Calisthenics & Computer Faire Chairperson</i>

ROBOTS YOU CAN MAKE FOR FUN & PROFIT

Frederik Pohl
386 West Front St.
Red Bank NJ 07701

I've been writing science fiction for a long time—this year marks the fortieth anniversary of the first time I appeared in print in a science-fiction magazine for money—and I've been reading it even longer than that. You all know what science fiction is. It's that crazy Buck Rogers stuff, made up by wild-eyed lunatics, that everybody knows will never come true.

Well, we all know that a lot of things everybody knew would never come true have come true. When I was a kid we read *Amazing Stories*, and *Thrilling Wonder Stories* to read about such fantastic things as atomic energy, and space travel, and television, and radar, and jet planes; now we read about them in the *Wall Street Journal*. Not everything that was discussed in a science fiction story has really happened—fortunately enough for all of us, we've been spared Godzilla breaking up Tokyo. But much has happened, and there's a strange phenomenon I've noted. Some of the most fantastic things discussed in science fiction haven't exactly come true, but what has turned out to be true, along the same general lines, is sometimes even stranger. Invasions of little green men from Mars. ESP—the ability to communicate from mind to mind, without the use of the conventional senses. Robots: clanking, red-eyed machine-men like Henry Kuttner's Gallacher, or Isaac Asimov's positronic creatures.

We haven't seen any little green men from Mars—or at least most of us haven't; there are thousands of flying-saucer devotees who would deny that statement. In fact, there is really no good evidence that the Earth has ever been visited by any creatures from another planet. But, on the other hand, there do exist, and in fact I can show you groups of people right here and now, some part of whom come from the interior of another star.

And although the existence of ESP between human beings has not been demonstrated, or at least not to the satisfaction of most scientists—here too, there are a lot of people who would disagree with me—still it is fairly easy to demonstrate the existence of ESP between a human being and a machine.

I seem to detect a sort of expression of skepticism on the part of some of the faces I see; but I promise that before I leave this platform tonight I will supply evidence for those two statements. But before that, what I really want to talk about—the thing, in fact, that brings us all here tonight—is the third of those crazy science-fiction ideas that seem to be coming true in an even stranger and more significant way than most science-fiction writers had predicted: the robot. Or, as we familiarly know it today, the

computer.

A computer is a machine. Generally speaking, all machines function only as amplifiers of some human capacity. There are thousands of machines than amplify man's muscles: the airplane, the bulldozer, even the nuclear bomb. There are even more thousands of machines that amplify his senses: the microscope, radar, the tv set. But what makes the computer unique is that it is the first complex machine ever built that amplifies neither a person's muscles nor senses, but one's intelligence.

It doesn't necessarily make the person himself smarter. But it does make it possible for him to do things which, if he didn't have a computer, would take a smarter person than he do do . . . and often would require a smarter person than anyone who ever lived.

Now, we've all seen this happening, quite pleasantly and productively. The space program, for instance, simply could not function without high-speed computer facilities operating in real time; human being don't think fast enough.

In a different way, something exciting happened a few years ago at the Harvard-Smithsonian Astrophysical Observatory in Cambridge, Massachusetts. An astronomer named Owen Gingerich had promised to give a paper at an astronomical meeting, and he decided to make his subject a repetition of the mathematical analysis of Johannes Kepler, three and a half centuries ago, which led to the formation of Kepler's famous three laws of planetary motion.

Gingerich had available to him all the observational material that Kepler had had, as well as a knowledge of the mathematics Kepler had used. (In order to get that, by the way, he had to develop a computer program to translate medieval scientific Latin into modern scientific English.) He didn't have a much time as Kepler—Kepler spent four years on the mathematics, and all Gingerich had was the eight days of a Christmas vacation. But he had a computer.

So he wrote a program, and he put it on the computer, and the computer duplicated what had taken Johannes Kepler four years to do in a little over three minutes.

Now, that's interesting in itself as a sort of a parlor trick, and maybe that particular project is not much more than that. But imagine what might have happened if Kepler himself had had a computer working for him in 1626. It took four years out of the most creative and productive part of his life to discover that the orbit of Mars is an ellipse. If he could have done it in four minutes,

what might he not have done with the other three years plus? Keplers are hard to come by. But now we have thousands upon thousands of scientists in all disciplines who can, at least in some way, function at the level of a Kepler or a Mendel or a Count Rumford, because the computer is there to help them. And it is this fact, as much as anything, which accounts for the immense acceleration not only in the growth, but in the rate of growth, of scientific and technological progress that we are all seeing every day of our lives.

I've been talking about the uses of the computer in amplifying intelligence for scientists, but, of course, that is not the only place where that kind of amplification can occur, or is needed.

For example, I have a pet project of my own. It has nothing to do with scientists. I don't quite know how to make it happen, but it seems to me that it could be done . . . maybe someone in this room can perhaps see a way to figure it out.

My interest in the project came about because of my youngest daughter, who is a brain-injured child. She is a pretty and rather endearing young lady, not severely handicapped, but through and because of her I have found myself involved with several organizations concerned with learning-disabled children. The children vary widely, in all sorts of ways; but I think it can be said that they have one characteristic that is nearly universal.

They forget.

Here's a young boy, brain-injured, and learning-disabled. You are his teachers and parents. You want to teach him arithmetic.. You teach him addition, and by and by, after a number of repetitions he learns how to add a column of numbers—even how to carry, and how to check his answer. Fine. You then teach him subtraction, and sooner or later he learns that, too, but then you throw in an addition problem, and he's forgotten that while he learned subtraction.

Or you're his parent, and you want to teach him to dress himself and get himself ready for school. He can learn to put on his shirt and button it, and tie his tie. He can learn to brush his teeth and comb his hair and wash behind his ears. But there will be days when he will come downstairs with his hair brushed, and his shirt buttoned, but he has forgotten to take off his pajamas. All children do that sort of thing when they are little. Learning-handicapped children do it over and over. They even forget when it is a survival matter, like crossing a street. On a Wednesday they remember to look both ways, and wait for the green light, but on Thursday they forget to look, and on Friday they forget whether it is "red" or "green" that means it's safe to cross, and on Saturday they may forget to turn left on Broad Street and get wholly lost within a block. And this is serious. What do you do? Let the child go out, and risk having him run over? Or keep him secure, and make certain he will never develop the skills for independent living?

But suppose—

Suppose the boy had a little, pocket-sized computer terminal: shared-time, on-line, remote-access; very much the same thing Project MAC used at MIT a decade ago, and many other installations have duplicated since, but small enough to carry. Suppose one other thing: that it is voice-responsive instead of requiring him to carry around a teletype. And suppose we could give something like this to every brain-injured or learning-disabled child or adult in the country.

That's quite a gift. What we have given them is a memory.

So Jimmy starts out for school and gets to the first corner. He says to his pocket terminal: "I'm at the corner of High Street and Main." The terminal replies at once, "Right, Jimmy. Wait for the green light. Then look to see if any cars are turing out of High Street. If there aren't any, then you can cross."

Because it's remote access, it has, of course, built-in communications support, so that if anything goes wrong, Jimmy can be patched through to his mother, or his school. He is never out of touch. He is able to do as much as he can on his own, and when the situation is beyond him, he always has help at hand.

More than that. Because there is a computer at the other end, he has a teacher always available, who is never impatient, out of sorts, or bored. I've watched some of these kids, with a hunger to learn, setting themselves problems in arithmetic or spelling. And I've seen CAI installations where a computer in Stanford helps a student in Tennessee, with exactly the endless, untiring patience that the learning-disabled person needs, rehearsing them over and over in arithmetic and algebra.

I do not see why computer-aided instruction and the learning-disabled child cannot be brought together.

And when you come right down to it, I don't think it is only the learning-handicapped child who needs that sort of help. I think most of us could use it from time to time.

Let's see what it might mean to you and me. About ten years ago I wrote a science-fiction novel called *The Age of the Pussyfoot*, in which I presented my own ideas of what a remote-access, carry-in-the-pocket computer terminal ought to be like. Mine was about the size of a darning egg—if anyone remembers what a darning egg looks like. It functioned as combination secretary, credit card, bookkeeper, telephone and radio. I added a few little extra features, just to improve the product. Push one button and it dispenses perfume. Another button gives you a drink or a pill to cheer you up; another button, and it doles out contraceptives. I called it a "joymaker."

Let's leave off the refinements and see how a joymaker brings joy, or anyway reduces trouble, in our lives. You're out strolling with your mother-in-law, and she comments that she likes a kind of

perfume in a shop window. Three months later the joymaker reminds you that her birthday is that weekend, and when you ask it about a gift suggestion it displays the name of the perfume. If it's really expensive perfume—and why would she be hinting around about it if it weren't—it tells you what your bank balance is, in case you need to write a check. It's your alarm-clock when you have to get to a meeting or make a phone call. Shopping, it tells you whether the Jumbo Jar with 51 ounces for \$1.93 is a better buy than the Family Economy size with seventeen and a half ounces for seventy-three cents. It remembers your Social Security Number, and your wife's ring size. Our your girlfriend's. Or both. When you fly east for a meeting with the manager of your Pittsburgh branch, it reminds you of his wife's name and the ages of his children. It tells you in which section of the lot you parked your car at the airport, and prods you to take your pill when you get up in the morning. Traveling? It will tell you the French words for "Where's the men's room?", or the Russian for "I demand a lawyer." If you get bored, it will play chess with you. If you get lost, it will match any two street-signs against a map in its memory and tell you how to get found again.

And with it—and with the communications facilities that it always provides—you are never alone.

Is that a good thing? Do we really want to be in touch with everyone, all the time?

I don't know the answer to that, in any moral or basic-values sense; but what I do know is that that's the way we're heading. If you judge what people want by what they buy, and look at the sales on CB radios, pocket calculators, telephone-answering services, and so on, then the joymaker is the growth industry of the Eighties. And just think, I'm letting you all in on the ground floor.

But all that is only hardware. If you wanted to spend the money you could buy a lot of it off the shelf right now. Let's look a little farther into the future. I mentioned the word "robot" a little while ago. Is there any real prospect that we're going to have them among us before long?

A lot depends, of course, on what you mean when you use the word "robot." Isaac Asimov has one kind, Jack Williamson has another, and if you go back to Karel Capek, who coined the term out of the Slavic root meaning "worker," you get something different still. I think we might say that a robot is a mechanical analogue of an intelligent human being. That's a great definition, in that you can challenge every part of it as being so imprecise as hardly to be meaningful at all; but maybe we can agree on what the traits of a human being are: the ability to grow, to feed itself from the environment, to reproduce, to evolve, to make decisions, to create—to learn.

Are there any machines like that around?

Johns Hopkins University had one a little bit like that, some years ago. It wasn't at all complicated, or impressive-looking. It was a little,

sort of rat-sized box on wheels, with an electric plug and feelers. You plug it into an outlet, and it charges up its batteries. When they are charged it pulls the plug out and begins roaming around. It stays close to the baseboards, feeling its way with its antennae; when it comes to a doorway it cautiously sneaks across, and it keeps on doing that until its batteries run down. Then it goes looking for another outlet. It finds one, plugs itself in, recharges itself, emits a sort of electronic burp, unplugs itself again, and resumes its wanderings.

And William Turing in England, thirty or forty years ago, invented what is called the "Turing engine." He didn't actually build one; it is not a very elegant design, and extraordinarily expensive and complex to make into hardware; but the principle is straightforward enough, and in principle it can be proved that it would work. The Turing machine has a long tail that extruded a paper tape—a little like the chain of the DNA molecule, but designed decades before anyone was quite sure what DNA was. Each section contains a command. Put all the commands together, and you have built a duplicate of the Turing machine: it reproduces. More than that. In printing and communicating each command, there is always the chance of error, so that the copy might not quite be exact. Set up enough Turing machines and let them reproduce, and maybe some of the copies would have more survival capabilities than others, and if so Darwin tells us that they will, in the long run, breed true: so it not only reproduces, it evolves.

Learning? Samuelson at Yale long ago wrote the first checker-playing learning program. He didn't tell the computer much about how to play, only gave it the rules of the game and some notion of how to decide one board position was better than another . . . and also the ability to remember what it had done, and to remember not to do again the moves that had got it into trouble in a previous game. So at first it played really rotten checkers, but by and by, it learned; and I understand that it played several strong games against the Connecticut State checkers champion not long ago.

When you come to questions like creativity and intelligence you are on much shakier ground. An argument can be made, but it's not easy to demonstrate that they exist in any real sense in computers . . . of course, it's not always possible to demonstrate that they exist in all human beings, either. But if you put together the ability to move independently and feed itself from the environment of the Johns Hopkins' bug, and the learning ability of Samuels' program for checker-playing, and the ability to reproduce and even evolve of the Turing machine . . . put them all together and they spell something. If you don't want to call it a "robot," I don't know what else there is to call it—except maybe "sir."

As you no doubt noticed, I glided rather quickly

over the questions of intelligence and creativity, not so much because I didn't want to talk about them, as because I wanted to look at them in a somewhat different light. That is, in connection with the way people think about computers.

We all know what way that is: a certain amount of resentment, a little amused contempt, maybe now and then some fear.

That's understandable, in an anthropological sense. Primitive man anthropomorphized the great blind natural forces of his world: lightning bolts, ocean currents, rocks, trees, predatory animals. But there are also great blind forces in our own world, almost as capricious as the lightning, or the wind: they can punish you by destroying your credit, or reward you by issuing a check for \$1,000,003 when all you really have coming to you is the price of a refund on a faulty pair of socks. We call these forces "the computer"; they are our collective paschal lamb, on which we blame all our sins, and we can't help but attribute some kind of personality to them.

And not without justice. Computers do have personality. At least, they have traits which, if exhibited by human beings, we would clearly call personality traits. For instance, there was a program at Case Institute in Cleveland a few years ago, in which they were trying to get computers to solve complex problems faster by teaming two computers on a single problem, like a mother and daughter cooperating to cook a Christmas dinner. Each computer would work on a part of the problem, and they would pass their partial solutions back and forth until they had it licked. Only it didn't work. The schematics looked okay, but it turned out that neither of the computers wanted to let go of any of its stored memories to the other. I told a friend of mine about this; he happens to be a minister with a big business in marriage counseling, and he said, "There's nothing strange about that, it's what I hear every day in my study."

Another trait we generally think of as limited to human beings is creativity, and, of course, there is a great deal of computer-generated poetry, art, music, and even film script around.

music, and even film script around. Most of it isn't any good. But most human-generated poetry, art, music, and film script isn't any good either.

I won't dwell on this particular subject, because John Whitney is going to show you some real and first-rate examples of it; but maybe I should say one word about the other side of that coin. A piece of art is a communication. A communication implies two sides: both a sender and a receiver. It is established that we can get computers to generate, but can we also get them to consume it?

Maybe Marvin Minsky shows us the way to that. At MIT he and Seymour Papert developed an interesting program for studying human psychology through computer models. They would make up stories and tell them to the computer: children's stories, because what they

were attempting to model was the learning process in the mind of the child, especially those quantum leaps when the child deduces something from context that has never been made explicit to him. The stories might be something like this: "Mary was coloring in her book with her new box of crayons. Her big sister, Sally, came in and said, 'Oh, Mary, what a pity! Those rotten crayons are spoiling your pictures! I hate to see that happen to you, so why don't you give them to me instead?'" And then they would ask the computer what was *really* going on: is Sally in all honesty worried about Mary's peace of mind, or is she just trying to rip off the new box of crayons?

I haven't yet found a use for Minsky's bedtime-story computer in science fiction, but I must admit he gave me part of the inspiration for my newest novel, *Gateway*. One of the principal characters in *Gateway* isn't human. He's a psychiatrist. His name is Sigfrid von Shrink, and he is a computer. Sigfrid von Shrink comes from a time several decades ahead of ours, and he has some jazzy features not found in your average computer program of 1977—for one thing, he has holographic attachments that let him look like Sigmund Freud when he wants to—but his roots come from an MIT Artificial INtelligence Laboratory program of more than a decade ago. MIT's program worked with the usual typewriter keyboard instead of Sigfrid von Shrink's mastery of the spoken language, but if you sat at a keyboard and typed out

I FEEL DEPRESSED

the program would come right back with

WHY ARE YOU DEPRESSED?

And if you then said something like

MY MOTHER DIDN'T LOVE ME

The program would ask

TELL ME HOW YOU FEEL ABOUT YOUR MOTHER

It was a pretty simple-minded program. It had no knowledge of semantics, could only recognize certain key words like "depressed," and "mother," and "sex," and responded to them with pre-programmed sentences. But as Sigfrid von Shrink points out in my novel, that's not all that important. What counts in analysis is not what happens in the analyst's head—whether it is understanding and compassion or total boredom—but what happens in yours.

Of course, my robot psychiatrist in *Gateway* lacks a few of the refinements we have come to expect of current psychotherapy—I don't think you would want to be Rolfed by a machine, and it wouldn't be much fun in a nude encounter session. But maybe something could be done about that. My friend and fellow science-fiction writer James Gunn has a new story about computer dating... which turns out to mean having sex with a computer.

I really do think that a computer could function in some ways as an analyst, but what I am less certain of is that most patients would sit still for it.

There is a rather pervasive fear of computers that I thing would get in the way. This fear came home to me sharply not long ago when I was talking to undergraduates at the University of Wisconsin. I was telling them about another project of that same polymathic Marvin Minsky. Marvin had mentioned in a scientific conference shortly before that he was engaged in developing tiny robot-like things for medical purposes: little remote-controlled things only a couple of inches across. He asked the assembled audience to suggest other uses for them, and some good ones came up in the discussion: I emember that one proposal was that they be used to pick potato bugs off the vine. Not only would that avoid all the problems of insecticide pollution, but the bugs themselves were, looked at properly, really a rather concentrated source of high-grade potein. There was a vigorous discussion when I was through, and the students didn't seem enchanted with this idea. One of them said, "Well, maybe you and Minsky might want to use those things to pick bugs, but I bet I know what the FBI would do with them. They would have them creeping through our dorms looking for pot."

That's not an unrealistic fear. Nor is it unrelated to the fear that other people display of centralized storage of information on credit, tax returns, draft status, Social Security details, and so on: there's a name for that sort of facility; it was coined by George Orwell, and it's "Big Brother." And there's no doubt that it's so. Big Brother is watching all of us, all of the time.

I share the fear. But I don't think it's directed at the right target. Stalin, Hitler, and the pre-war Japanese didn't have any computers to help them, but they didn't have much trouble keeping tabs on what everybody was doing all the time. Inspector Javert managed to track down Jean Valjean as relentlessly as any IBM bookkeeping computer going over your form 1040s, without any technology at all.

The computer is only a tool, like an ax. Tools are amoral. You can use the ax to cut the lumber to build a house, or you can use it to bash in somebody's brains. The tool doesn't care. And neither does the computer.

What endangers all those great ideals of personal liberty and individual integrity that everyone shares, or claims to share, isn't the computer. It's the people who run the computers. People, not computers, applied mechanical, two-valued logic to human affairs. People, not computers, harass their opponents and turn a blind eye to the misdeeds of their friends. People decide what penalties or rewards there are to be; computers do not. I don't think we have any need to fear computers that think like human beings.

But, on the other hand, I think we are all terribly at risk from human beings who think like computers.

I am running out of time, but I owe you

something before I sit down. I promised I would defend my statements about ESP between man and machine, and about the fact that some fraction of the human race came, literally and exactly, from the interior of other stars.

I don't deny that there are other intelligent races in our universe—the odds seem to be pretty heavily that there are. I kind of wish one or two of them would drop in, once in a while, just for sociability's sake; but as far as I can tell, no one ever has. Yet there seems to be pretty good scientific evidence for something far stranger. Astrophysicists think they have a pretty good idea of how the universe began; it was almost certainly with some sort of catastrophe, and the catastrophe seems to have ben a sort of immense explosion which released a flood of hydrogen gas into space. That's all there was, the hydrogen gas, and the empty space. Over the eons the gas collected into clouds which condensed to form suns; in the immense heat and pressure of the interior of a star, the hydrogen was cooked into the heavier elements—and when those early stars exploded, they flung into space all of the elements, every bit of all of the elements that are heavier than hydrogen. Including all of the carbon in the universe, all of the oxygen, all of the nitrogen, all of the calcium that make up more than three-quarters of the weight of our bodies. So most of my flesh—and most of yours—was once, billions of years ago, in the heart of a sun;and that seems to be far stranger than any little green man from Mars.

As to extra-sensory communication between a human being and a machine, that's pretty straightforward. "Extra-sensory" means without the use of the senses: sight, hearing, touch, taste, smell. I can do that. I can sit in a rom and without speaking or moving, make a machine stop or start, go faster or slower; not only that, but all of you probably could too, if you wanted to go to the trouble of learning a fairly easy, new skill. It's only a matter of controlling the alpha waves of your brain. Most people can control them if they want to, at least to a limited extent; and if you then hook up a rheostat to respond to the amplitude of the alpha waves, you can display them on a cathode tube if you like. But the machine doesn't need that: you can, in fact, command the machine to do what you want it to, without any sensory intervention at all.

Now, I confess that in some sense those are verbal quibbles; but there is another kind of extra-sensory communication involving machines that isn't a quibble. It hasn't happened yet. But it could, any time anybody wanted to build the hardware. It's an invention of John Platt hardware. It's an invention of John Platt's, and he calls it his "left-hand radio." Take a human being, and fasten on his left wrist a little radio transmitter, connected to some fine surgical probes that tap into the nerve centers of his left

hand. Take another human being, and strap onto his wrist a receiver, connected in the same way to similar probes in his hand. Human being A moves a finger; the probes pick up a signal and broadcast it to human being B; his radio fires the signal into his nerve centers, and his finger moves too. Clench fist A; fist B clenches. Relax one, and you relax the other.

Does that seem like a trivial thing? Maybe so. But suppose the hand whose every muscular twitch you feel is the hand of Alexander Brailowsky, playing the *Fantasie-Improptu*, or Ruggiero Ricci fingering his Guarnurius . . . or a masseur, or a lover. Or suppose that it is not just the left hand you can feel but maybe even the whole body: the body of a Baryshnikov, or an Olympic runner . . . perhaps you can feel more, all of the sensory inputs, in fact, whether of ecstasy or pain.

I don't know whether all that would work. But I know that I would like it to.

You see, I can't accept the evidence for ESP. But I can understand why a great many intelligent, sensible and competent men and women can and do accept it. A great science-fiction writer of an earlier day, Voltaire, once said, "If there were no God, it would be necessary to invent Him." I think the case is exactly the same for ESP. In the crowded and complex world we live in—and in the far more crowded, and infinitely more complicated one that lies just ahead—there does not seem much chance for survival unless we can find some better way to make contact with the inner reality of other persons, to feel what they feel, and understand what they want, to reach out and touch what is at the core of another human being, and to perceive it with compassion, and respect, and understanding.

So if there is no ESP—if there is no clearer, surer way to reach understanding between people than the tricky and tendentious forms of words we use in trying to deal with real-time problems, then we'd better get on with inventing one.

I said at the beginning of this talk that the right way to view a computer seemed to me as a tool that magnified human intelligence. We can stand all the amplifying of that trait we can get; and I predict with great confidence that we will find a solution to that problem. In fact, I regard it as the safest prediction I've ever made, because if I'm wrong—if we don't, through computers or some other medium, find a way to reach a fundamental understanding of other human needs—I don't think there will be anyone around to call the bet.

BIOGRAPHICAL SKETCH

Frederik Pohl has been president, Science Fiction Writers of America, from 1974 to 1976. He has been Bantam Books' Science-Fiction Editor since 1973, and an Authors League Council Member since 1976.

Frederik has authored (alone or in collaboration) approximately 50 books, and edited an additional 40 anthologies. He has contributed several hundred short

stories and essays to publications all over the world, ranging from *Family Circle* to *Playboy* and *Rolling Stone*, and, of course, nearly every science-fiction magazine. He has been translated into more than 40 languages.

Frederik has received four Hugos (top annual science-fiction award), and is the only person ever to receive the Hugo both as editor and as writer.

He has lectured to more than 200 colleges in the United States, Canada, and ten other countries, as well as literary and professional gatherings. He was for four years a staff lecturer for the American Management Association on the challenge of the future, and has toured Yugoslavia, Romania, Poland, Bulgaria, Hungary, Germany and the USSR under the auspices of the State Department. He has represented the United States in international literary symposia in Tokyo, Rio de Janeiro, Milan, and elsewhere.

Frederik has been called "the most consistently able writer of science fiction, in the modern sense, has yet produced." (Kingsley Amis, in *New Maps of Hell*.)

He has participated in scientific meetings sponsored by NASA, New York Academy of Sciences, American Astronautical Society, American Documentation Association, and various other scientific groups. He was a Session Chairman at both the First and Second General Assemblies of the World Future Society.

Apart from science and science fiction, Frederik Pohl's sphere of interest includes political science (*Practical Politics*, Ballantine, 1971), Roman history (he is the *Encyclopedia Britannica's* authority on the Roman emperor, Tiberius), and music (forthcoming work on the violin concerto). His most recent books are *The Early Pohl* (Doubleday, 1976), *Man Plus* (Random House, 1976), and *Gateway* (St Martin's Press, 1977).

For further information, consult *Who's Who in America*.

Digital Pyrotechnics: The Computer in Visual Arts

John H. Whitney

600 Erskine Dr.

Pacific Palisades CA 90272

ABSTRACT

Harmonic forces give shape to our experience of past and future. This is the dramatic essence of musical experience. It is why the composer, more or less intuitively, has manipulated the network of harmonic relationships of all musical scales for as long as music itself has existed. Evidence is accumulating to substantiate the need for much further study of harmonic phenomena. Because there is reason to believe—as I do—that the tensional charge and discharge—the expectations evoked and thence fulfilled by tonic structures in music—all this is a direct product of the mathematics of harmonic order. I further believe that the same possibilities exist in the skillful design of harmonic pattern for visual perception. Therefore, I am exploring harmonics designed for eye instead of ear. It is interesting to note that the very creation of harmonic pattern had been altogether inconceivable until a very recent time when computer graphics eventually and slowly became available to the visual artist.

I'd like to show how well computer graphics and harmonic pattern are suited for each other. And show how useful this compatibility can be for employing the computer to charm the eye.

To begin, let's consider the manner by which composers for centuries have used harmonic "force" to attract and hold the attention and otherwise charm the ear. Then we will examine a similar form of visual "force" which has been made possible by computer graphics.

At the outset we know the musician's aural spectrum to be an undifferentiated and continuous spread of frequencies, say from twenty to twenty thousand cycles. Yet this apparently homogeneous continuum is not continuous at all. Harmonic relationships interactively transfigure this spectrum. Harmonic phenomena create discontinuities, as nodules of tension, anticipation, and resolution deform this otherwise smooth continuum. Whole number, or harmonic nodes, scattered throughout the spectrum create order/disorder proclivities: centers of emotional focus which distort an otherwise smoothly ascending texture. In fact, sounding tones over the span of just one octave persuades the ear that we are nearer a return back to the start than we are advanced along any straight line of upward continuous ascension.

I want to suggest that it is this particular discontinuity, not really any other quality of the audio spectrum, that constitutes the raw material of the composer's art. Not pitch, texture, rhythm, and meter. Not frequency, intensity, and density, as most 20th-century modernists have wanted us

to believe.

That is to say, no matter how we divide the spectrum into steps we find a hierarchy of perceptual values that distinctly rank each step. It is the composer's cunning, or intuition, or even mindless exploitation of this hierarchy that is the primary source of rhythmic vitality and emotional content of music. The composer, however, must cooperate with these natural harmonic forces, or see his strategies defused by them. He cannot work his will against, nor exercise insensitivity to the charge and discharge. He cannot escape the dominion of the gravitational force of harmonic moment.

Furthermore, as a corollary to all the above, a visual domain of harmonic consequence has become accessible through computer graphics. With the graphical display rooted upon coordinate mathematics it is only natural that a great variety of periodic interference patterns can be produced. The motive now exists for the artist/composer to discover his way into this diverse domain of dynamic visual form structured out of two and three-dimensional harmonic periodicity. This domain abounds with tonic centers of focus as in music. And this domain will render up an equivalent rhythmic and emotional content as in music. I think we will soon see the artist learn to cooperate anew with natural harmonic forces in hitherto unexplored visual territory.

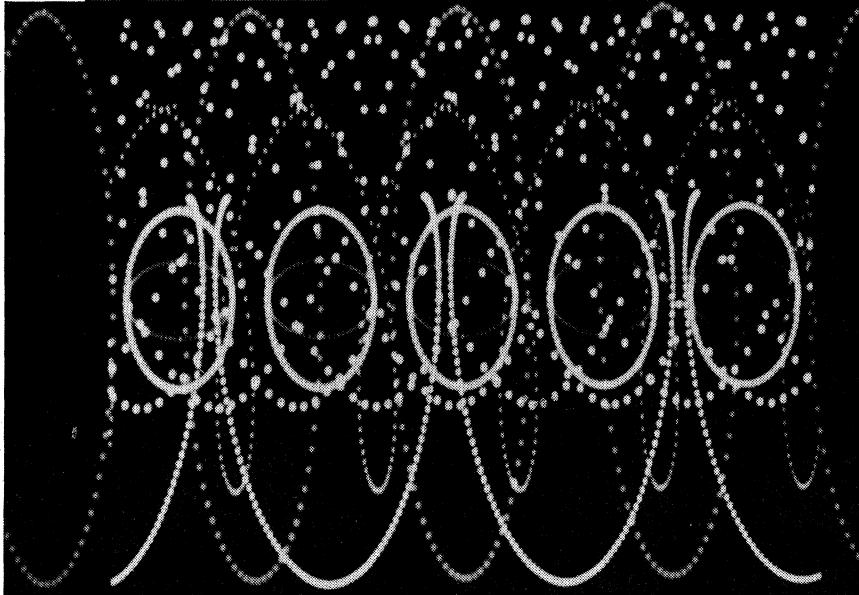
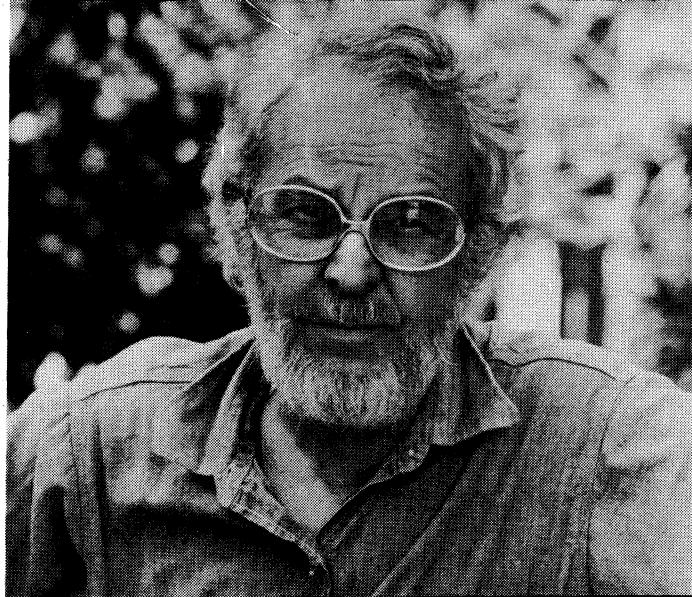
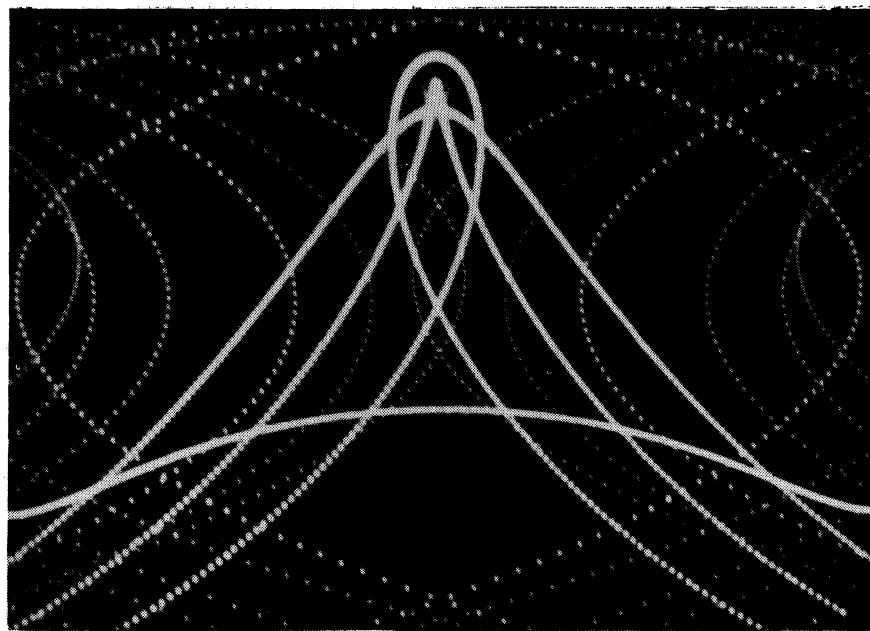
In truth, harmonic forces give shape to our experience of past and future.

Yet harmonic force is not all that mysterious. We can speculate why the sound of "ti" urges us on to "do." Significantly a good diagram for the perceptual dynamics of harmonics is found in this picture (Figure 1).

It is characteristic of harmonic phenomena, visual, aural, or otherwise, to show this kind of pattern. As patterns go, the illustration is perhaps explicit in a way that is even more obvious than the aural, leading-tone effect of "ti" upon "do."

Eye and ear, each in its own unique manner, experiences the dynamics of this kind of pattern as an event in time—as punctuation. Especially as an event of arrival or departure. When we arrive at "do," the octave above the tonic "do," we hear that rudimentary relationship with a particular infallibility. If we sample ascending steps of the scale, the ear is bound to sense the final event of arrival just as the eye can see arrival and departure relationships in the illustration. I might add that these relationships are many times more explicit when seen as a motion picture sequence.

In terms of visual perception, vaguely a



corollary to aural responses, we have here a phenomena of hierarchical distribution and classification of elements into an array in which all are ranked according to some perceptual scale of complexity.

No need to argue which pattern is more "consonant" than another. For generating dynamic patterns here is an order/disorder principle, or a consonant/dissonant, principle to work with. It is a principle which can be exploited in more ways than one might expect to give meaningful order to temporal development. The principle becomes a composer's valid strategy—probably the first strategy to be so defined and applied in the brief history of the art of computer graphics.

Finally, it is worth remarking that the illustration for this article could not be created by conventional hand-drawing techniques. At least that would be quite difficult. Moreover, it would be impossible to hand-animate the film from which the illustration was derived. Many of these films required thousands of drawings while the computation for plotting is staggering. Thus the computer is the ultimate and the only tool for visualizing the dynamic world of harmonic functions. This may serve to illustrate the point that this new world of visual art cannot be confused with any previous traditional forms. (See *Art International*, Vol. XV/7, September 20, 1971.)

It should be of particular interest to realize that computer graphics—this 15-year-old infant—is patently capable of bringing forth a totally different kind of visual experience as unique and riotously enjoyable—much cheaper—more energy/materials intensive than the Chinese, pre-Christian invention of fireworks.

BIOGRAPHICAL SKETCH

John Whitney's growing reputation as a pioneer with applications of the computer in visual art was advanced another step with the completion of his film *Arabesque* last year. The film has received high honors in the U.S., and abroad; including Iran where its relation to the traditions of Islamic pattern was acknowledged. John's thirty-seven year "obsession" with the role of time and movement in visual art led to computer graphics long before that technology was accessible, or practical. What for years was only an obscure experimental study is just now evolving into an accepted and recognizable fine art attracting many young artists. It is a field which is of growing interest to the personal computing experimenter. John Whitney's part in bringing about these developments is well known.

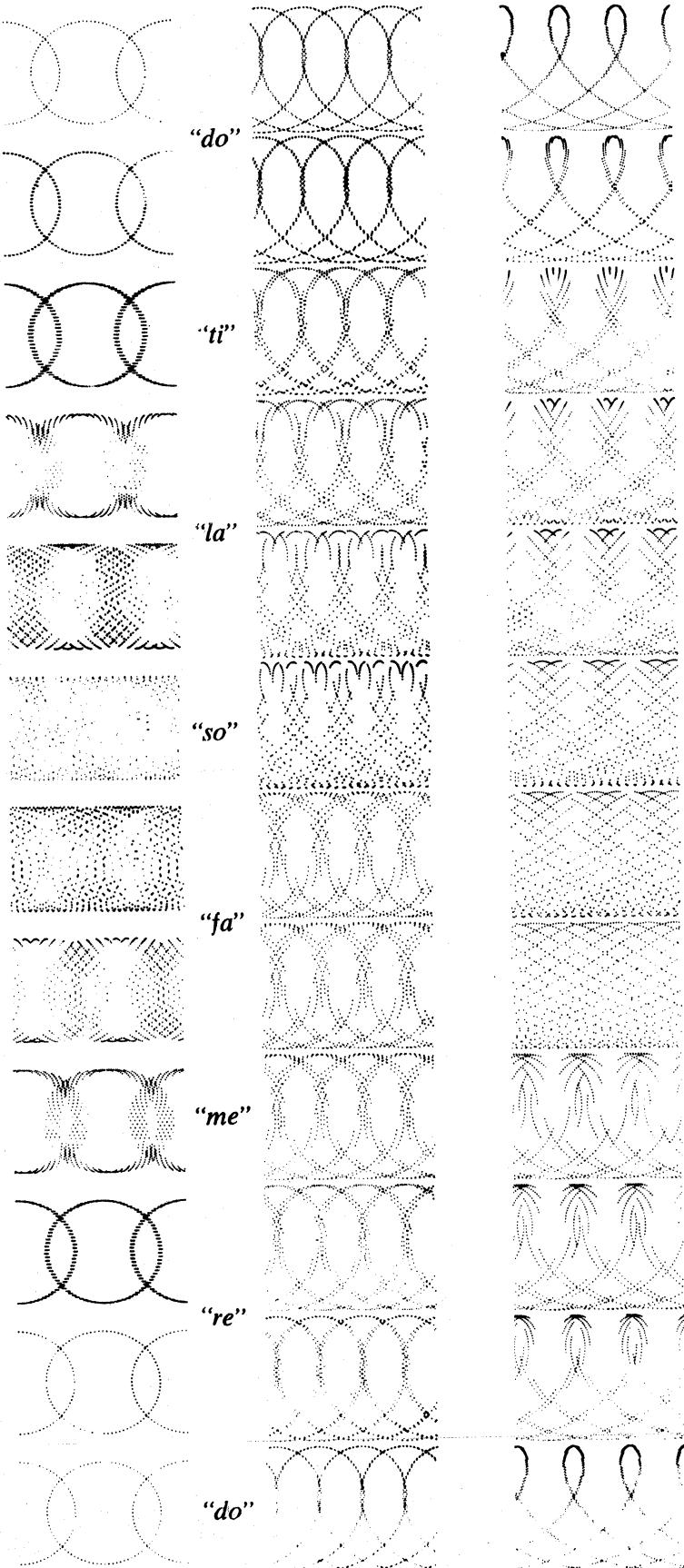


FIGURE 1.

Each of these frames might represent samples approximately one second apart out of a smooth continuous action sequence totaling twelve or more seconds in length. Read from bottom to top.

THE 1940'S: THE FIRST 'PERSONAL COMPUTING ERA'

Henry S. Tropf
Professor, Mathematics
Humboldt State University
Arcata, California 95521

ABSTRACT

The beginning of the modern computing era (generations -1, 0 and 1, c. 1935-c. 1955) is best chronicled by observing the efforts and determination of certain key individuals. One of the workers in the immediate post-WW II period characterized the machine architecture with a comment that he could look at the design of a new machine and identify its geographic area of birth and, in many cases, the individuals involved. Nelson Blachman's 1953 ONR "Survey of Automatic Digital Computers," listed 98 'machines' which had been completed or were under construction by 75 different companies, universities, or governmental agencies. The majority of them were one of a kind, although in many cases the designers indicated a willingness to make plans and drawings available. This stamp of individual philosophies and the willingness to share information pervades this early environment where few could see the future explosion of information processing (the 'computer revolution') that we are still experiencing. In this talk, I will attempt to describe some of the efforts, frustrations, and achievements of a few of these individuals.

Anyone who looks up the word "computer" in a dictionary published before 1955 will discover that it refers to a human being. That is, "one who performs a computation." Indeed this is an accurate characterization of the term up to the early 1950's. The early electronic devices were generally called Calculating Engines or Giant Brains.) Computing was primarily an individual, or occasionally a team effort. The development of mechanical devices by Schickard, Pascal, and Leibniz paved the way for desk-size calculating devices, but they didn't substantively alter the computational environment. The construction of tables to facilitate astronomical, astrological, navigational, or mathematical calculations can be traced back at least to the early Babylonian cuneiform tablets. Babbage's 19th Century efforts to construct calculating engines, is well known, as is his inspiration from the Jacquard Loom for punched-card input and Lady Lovelace's concept of what we now call programming. Due to his failure to complete a working model, however, his ideas had no effect on the computing scene.

One of the early efforts that had a major impact on the 20th Century computational environment was Herman Hollerith's construction of a tabulating device, which he developed for use in the 1890 census. This punch-card tabulator led to a variety of mechanical accounting machines and some individuals conceived of ways to use them as computational aids. For example, in the early 1920's, Henry Wallace gave a course at Iowa State College, using an IBM key-driven machine to calculate correlation coefficients. In the same decade, L.J. Corie (in England) used accounting machines for scientific calculation, and the construction of mathematical tables. In the latter part of the 1920's, Ben Wood established a statistical laboratory at Columbia University, using IBM donated equipment. Out of

of the 1920's, Ben Wood established a statistical laboratory at Columbia University, using IBM donated equipment. Out of Ben Wood's experience, and again with donated IBM punched-

card equipment, Wallace J. Eckert established an astronomical computational facility, also at Columbia. Another computational development, in the 1920's, was Vannevar Bush's first analog computer. These developments, along with a growing electronic technology that included the development of radar and television prior to WW II, established a technological basis for what is now called the computer revolution. By the mid-1930's, the capability was there, but some of the key ingredients were still dormant.

Prior to WW II, there was no widespread interest in developing a capability to do large scale computation. A few individuals may have felt a personal need for such a capability, but there is no evidence that governments or commercial firms saw any value in generating the time and funds necessary for such developments. At NCR and IBM, for example, devices were built shortly before WW II which performed arithmetic operations at electronic speed, but management of both companies showed little interest in pursuing this avenue. Two specific examples will illustrate this attitude in two different environments: a research laboratory and a major university.

At Bell Laboratories, George Stibitz was employed in 1930 as a research mathematician. One evening, in 1937, in the kitchen of his home, he arranged a few relays on a small piece of plywood in order to demonstrate a mechanical procedure for adding one-digit binary numbers. At the time, Stibitz was concerned with switching network problems. At Bell Labs, computers were available to perform necessary calculations, which consisted in great part of adding, subtracting, multiplying and dividing complex numbers. These computers consisted primarily of about a dozen women using pencil, paper and desk calculators. When a fellow employee suggested to Stibitz that the computations might be accelerated if one could figure out somehow to hook two or three desk calculators together, he proceeded, instead, to use his binary relay device to design a circuit that could multiply and divide complex numbers. This generated enough interest so that a group of Western Electric engineers, headed by Sam Alexander, completed construction of the Complex Calculator in 1939, capable of all four arithmetic operations.

The Complex Calculator (later renamed the Bell Mod I), had some unusual features. When it went into operation in the Bell Labs Building at 463 West Street, N.Y.C., it became the first remote access computing system in existence. The calculator was kept in a locked room, with keys available only to maintenance people and custodians. Access was by means of one of three teletype keyboards in the building. (Not time-sharing -only one at a time.) This was dramatically demonstrated at the summer meeting of the Mathematical Association of America, when, in September, 1940, Dr. Stibitz described the device and attendees were able to put problems on the keyboard located in a hallway outside of the lecture hall which was connected to the calculator in New York. That is, access over a long distance data link. Its speed is almost embarrassing today, 30 seconds to find the quotient of two 9-digit complex numbers, a factor of about 3:1 over desk calculators. But it has a few more important advantages: The excess-3 checking code made errors impossible. If a relay failed, it stopped; also, continuous, fatigue-less operation.

With the completion of this Calculator, Stibitz approached management with the suggestion that an investment of

\$50,000 would most likely produce a large-scale general purpose device using the same ideas and technology. The proposal was turned down essentially with the response "who wants to spend \$50,000 just to be able to do ordinary computations a bit faster and easier?"

World War II provided a response to this question, particularly with the need for ballistics computations. The computation of one ballistics trajectory requires about 750 multiplications. For one trajectory, this will require about seven hours of human desk calculator time. The time factor is not significant until you have to do large batches of 100 or so. Here, the relay calculator was capable of a speed factor of about 10:1. A survey of 100 trajectories could be calculated in about 24 hours, as opposed to 700 human hours, with built-in checking reliability and 24 hour-a-day, 7 days a week performance. During the war, Bell Labs constructed the Mod II (Relay Interpolation, 1943), the Mod III (the Ballistic Interpolator, 1944), the Mod IV (Mark 22 Error Detector, 1945); and after the war, the general purpose Mod V (one for NASA, 1946, the other for Aberdeen, 1947), and the Mod VI (Bell Labs, 1950). These calculators had not only error detecting codes, but some had error-correcting codes, floating point, check signals, and many features which enabled it to run unattended for many hours. Although I tend to classify these electro-mechanical relay computers as generation -1, many of their features didn't appear in electronic machines until generation 2.

A second development in the U.S. occurred at the same time that Stibitz suggested using relays to perform calculations in a binary mode. It occurred at an institution where, unlike Bell Labs, one would expect, and indeed find, little interest in computational capability: Harvard University. In 1937, Howard Aiken was completing his doctorate at Harvard where his thesis on the laws of space charge had involved him in 18 months of laborious hand calculations of nonlinear differential equations. In response to a query after his oral examination about his future plans, he said he hoped to make it possible for future scientists to avoid his grueling computational experience. In fact, he had already written a first draft of a proposal to perform sequential computations in an electro-mechanical mode. His efforts to bring his proposed device into being resulted in a good deal of hostility among the Harvard faculty and administration. But with his natural stubbornness, Aiken persisted until, with the aid of colleagues Harlow Shapley (Astronomy) and Ted Brown (Business), he was able to see the president of IBM, Thomas Watson, Sr. Watson agreed to have IBM assume the costs and assigned engineers Claire Lake, Frank Hamilton and Ben Durfee to collaborate with Aiken. Despite delays caused by WW II, the Automatic Sequence Controlled Calculator (Harvard Mark I) was installed and running at Harvard in 1944. From then until it was dismantled in 1959, the Mark I performed needed calculations for the defense department, the atomic energy group, as well as the computation of many volumes of Bessel functions. From the standpoint of speed, it was in a class with the Bell relay calculators, both soon to be hopelessly outclassed by the electronic machines. From the standpoint of reliability, accuracy and negligible down-time, it had no electronic rivals until the electronic 2nd generation machines. The technology used in the Mark I was not new, most of it had been used and tested through at least a decade of various applications. From an architectural standpoint, it was designed to do operations in a sequential manner in much the same way that Howard Aiken would have solved his problems with pencil and paper. Its 60 foot rotating shaft, translating horizontal rotation to vertical, was reminiscent of a power plant generator. Not too surprising when one knows that Aiken was a power engineer for the Wisconsin Gas Company from 1923 to 1935.

However, the most significant impact of the ASCC is more subtle than the machine itself. Howard Aiken felt that it was his duty to spread the word. The visitors log of the Mark I installation shows the wide range of scientific, technical and governmental individuals who came to view and learn. The Harvard computational facility became a center for people interested in machine-oriented computation. Many graduate students who used Mark I at Harvard are recognized leaders in the computer environment today. Aiken organized two conferences (1947 and 1949) in which he made it possible for everyone interested in automatic computation to present papers or to come to learn what was going on. The atmosphere was an open one of sharing ideas and disseminating the experiences of everyone interested in assisted computation. Aiken also traveled extensively to Western Europe, and his approach and philosophy are evident in early European machines. The Mark I was followed by the electromechanical Mark II and the electronic Marks III and IV. Aiken was a strong-willed individualist with tremendous vision. But even he had his visionary limitations. When one of his colleagues suggested improving the speed of the printer, he replied: "Why would we want to print faster than someone can read?"

There were other computational devices suggested or constructed in the late 1930's. Konrad Zuse's Model Z1 was completed in 1938, and his fully operational relay calculator, the Z3, was completed in Germany in 1943. At Iowa State College in Ames, John Atanasoff attempted to build an electronic device that would solve a system of 28 equations and 28 unknowns by Gaussian elimination at electronic speeds. Although this device was never operational, recent litigation has revived the details of the effort.

The operational beginning of the electronic digital computer began with ENJAC. (Colossus, an electronic cryptanalytic device built in England in 1943 was classified information then and is still primarily classified today.) The literature is full of this event and the role played by John Mauchly, who conceived of it, J. Presper Eckert, who was associated with him in its design and implementation, Herman Goldstine, who played an important role in getting the project funded by Aberdeen, and other individuals like John Brainerd, Arthur Burks, etc. Instead of repeating this part of the chronology, I would like to focus on the summer of 1946. The event: A summer course at the University of Pennsylvania's Moore School of Electrical Engineering. Subjects: ENIAC and the proposed EDVAC. Here, as at Aiken's later Harvard conferences, an attempt was made to provide a forum to present all known information on electronic computers, using the only one known to exist and to discuss the design, operation and architecture of EDVAC, the first machine into which was to be incorporated the capability of modifying its instruction. The goal of this program was to share all that the builders and designers of ENJAC and EDVAC knew, and to help others who might want to construct their own devices. One of the invited participants was Maurice Wilkes, from Cambridge University. Although he arrived a couple of weeks late, due to difficulty in getting a ship from England, by the return voyage, Wilkes was already designing a scaled down version of EDVAC, a machine which became operational in 1949, with the acronym EDSAC. Information (and people) in this period seemed to flow freely. The most general directional patterns were from the U.S. to England and England to U.S.; the eastern U.S. to western U.S. and to western Europe. John von Neumann's project at the Institute for Advanced Study sparked parallel projects at Oak Ridge, Argonne, the University of Illinois, Rand Corporation and Los Alamos (to name only a few). In fact, some of IAS Project may have gone on the air before Von Neumann's machine at Princeton.

The U.S. west coast computational environment is an interesting illustration of this phenomena to share information. The decision by IBM to manufacture the defense calculator in 1951 (the IBM 701), led to a group of individuals at places like Rand, Northrup, Hughes, etc., to informally organize in order to be able to share information and to avoid duplication of effort in the use of the 701. The informal gathering led to a slightly more formal one, DCA, (DCA = Digital Computer Association) and later, with the announcement of the IBM 704, to SHARE. As one of the key organizers of SHARE put it: "Isn't it disgraceful that on the 701 there were 18 machines, and there were 18 assemblers. Let's invite every person who has an order in for a 704 to come and talk about, number one, how can we reduce from 18 assembly programs to one, and, second, anything else of interest."

These brief sketches of the role of the individual in successfully achieving a desired goal in a hostile environment, the general enthusiasm of a small group of early practitioners to spread the word and the open exchange of ideas characterize generations -1, 0 and 1 of what is now called the computer revolution. But the growth of an industry from non-existence three decades ago to one where the number of computers (excluding pocket calculators) has grown to 350,000 or 400,000 has to have exerted a change on this environment. The small number of individuals involved from 1947 to 1954, their constant dialogue with each other, visitations, mutual help gatherings, enthusiastic participation by everyone from the project head to the people on the benches, characterized the era of the birth of the electronic information processing era. Many of the one-of-a-kind machines of this early period carried some architectural signature of its builders' attitude toward computational design. One of the pioneers claimed that he could look at a new machine and identify in what geographic locale it was built, and often even identify some of the people involved in its design. Jim Warren, in a recent issue of *Dr. Dobb's Journal* characterized this in the following way: "The sharing of ideas is useful in that it allows us to stand on one another's shoulders, instead of standing on one another's feet. But, there is something else shared that is of at least equal value: the enthusiasm and intellectual excitement." The computer hobbyists in their enthusiasm and excitement appear to have re-introduced those elements which were prevalent before 1954 back into the computer environment. Who knows what the spin-off of this rebirth will be?

THOSE UNFORGETTABLE NEXT TWO YEARS

TED NELSON

Here we are, at the brink of a new world. Small computers are about to remake our society, and you know it. I am supposed to tell you what is about to happen in the near future.

But to understand the future we must understand the past: most people don't realize what *has* happened. What is astonishing to me is not so much the future as the past, and the things that are going to surprise people -- the sudden appearance of little helpful interactive computers everywhere -- should be less surprising than the past circumstances that have delayed all this till now.

We have inherited some silly myths about the nature of the computer; and these myths are miraculously shared by both computer-lovers and computer-haters. They are the silly and cruel myth of depersonalization -- that computers demand impersonal behavior and rigid unpleasant rules, not to mention punch cards; the myth of efficiency, that using computers to depersonalize ways is somehow good; and the myth of technicality, that to criticize or try to appreciate computer systems is beyond the competence of the laymen. These vicious myths, and more, imprison us all.

Many computer people seem content with these myths and -- intentionally or not -- keep them going, dealing with non-computer people ominously and unhelpfully, with hard-heartedness, scientific pretension and scary lingo.

We all know that there are those in the computer profession who enjoy pushing other people around, who hide behind the computer and say the pushing around is the computer's fault. The computer is functioning as a mask. There are even those with long hair who teach computer science and talk about computer liberation, who yet enjoy mystifying and confusing people and leaving them helpless. Worst, I regret to report that there are actually people who enjoy pushing people around, and intimidating them, in the amateur computer field.

C.P. Snow, the English author, has spoken of what he calls "the two cultures" of educated people, one being those scientifically trained and the other being the humanists. Ordinarily these two factions do not speak to each other, each regarding the other side with distrust and disgust. Computer people continue this tradition. Many computer people, indeed, seem to half-desire some kind of a showdown fight, where they are going to rub the humanists' nose in computer jargon, possibly to get even for having had their spelling corrected.

Do you think it's not as bad as that? Consider the problem of lower-case lettering for display and printout on computer terminals. Some computer people feel there is no need for lower case. This is a slap to all lovers of the written word. If you have no lower case on your terminal, it assures that no person with a literary background will ever use your text editor.

Or consider an argument that recently occurred in one of my seminars. A student of mine is planning to implement a text system along certain lines I advocate. But he insists that the user, typing in text, and each typed sentence with a special character, such as the caret. When I pointed out that standard practice for both stenographers and authors is to delimit a sentence with two spaces, he was indignant. "I don't want to have to type two spaces after a sentence!" he declared, preferring instead his caret; it did not matter to him that literary people would hate the caret more than he himself hated the two spaces, and so they wouldn't use his system.

With these points I have intended to highlight several problems. Computer people are not all of them prepared to honor

or appreciate the wants and needs of others, especially those with different points of view. And this has had, and will continue to have, curious repercussions. Old style computers are about to collide with new style computers, and many people will get caught in the middle.

Here is what has happened so far. In January of 1975, MITS announced the Altair computer kit for \$400. That seems a long time ago: in the two years since then, so much has happened. To the surprise of some people, the Altair computer took off like a rocket. Certain other people, who had been almost ready to do the same thing (they say), were galvanized into action. More brands appeared.

The next major event was Lois and Dick Heiser's opening of their Computer Store in Los Angeles. "Now why didn't I think of that?" was everybody's first reaction, followed instantly by, "I still could."

So here we are. Perhaps twenty thousand dinky computers, perhaps more, are in the hands of small businesses and hobbyists, too, have visions of becoming small businesses.) Some three to five hundred computer stores are open, or on the verge of opening, or in some other stage of near-existence. And throughout the country are the cottage computer industries: people making accessories, software, teeny mainframes. Catering to these mad folk are half a dozen -- probably a dozen by the time this comes out -- counterculture and hobbyist computer magazines.

Now, as far as I am concerned these developments are no surprise at all. The surprise is that it took so long to get started.

Certainly the computer-on-a-chip could be seen coming ten or twelve years ago. Now some people call them "microcomputers," But nothing could be more absurd than calling them "microcomputers:" a radio is still a radio if it is built from integrated circuits, not a "micro-radio;" a tape recorder or a hearing aid or a telephone is not a "micro-tape-recorder" or "micro-hearing-aid" or "micro-telephone" even if it is built from integrated circuits; and a computer is a computer whether or not it is built from integrated circuits. *The real meaning* of calling it a "microcomputer" is this: the word is a face-saving device for people who did not see any of this coming, who had their heads in the sand. Calling these things "microcomputers" implies that they are something new under the sun, a technological surprise which could not possibly have been predicted, so the term absolves the speakers of having been fatheaded in expecting the status quo to continue. The surprise is all in their minds.

The dinky computers, as I prefer to call them, are widely belittled in the regular computer industry. But our fundamental 8080 chip, suitably supported, is approximately comparable to the 1401, which was the workhorse computer of American business in the early sixties. For many purposes the 8080 is more than adequate. It's always nice to have more computing power, but then it's *always* nice to have more computing power. There will come a day when the power of a big PDP-10 will seem inadequate for a personal computer.

For now, though, the dinky computers are working magic enough. They will bring about changes in the society as radical as those brought about by the telephone or the automobile. The little computers are here, you can buy them on your plastic charge card, and the available accessories include disk storage, graphic displays, interactive games, programmable turtles, that draw pictures on butcher paper, and goodness knows what else. Here we have all the makings of a fad, it is fast blossoming into a cult, and soon it will mature into a full-blown consumer market.

FAD! CULT! CONSUMER MARKET! The rush will be on. The American manufacturing industry will go ape. The

American publicity machine will go ape. American society will go out of its gourd. And the next two years will be unforgettable.

SCENARIO

The exact sequence of events is of course hard to predict.

The general outline, however, is not mysterious. Let me try to fill in some of the basic things I think are going to happen.

In 1977, tens of thousands of computers will be sold to individuals; the current demand level, with machines sold as fast as they come into stock, will probably continue for some time, possibly several years for the better machines.

The sales of the S-100 machines, like Altair, Imsai, SOL and Polymorphic will continue strongly. If you've followed the computer field for any length of time, you know that standardization is generally established on a defacto basis by whoever gets someplace first. It should have surprised no one that the original Altair system of interconnection at once became a standard. Many engineers have said, "Aw, I could do better," and proceeded to try, but that has been quite beside the point. When there is a standard way to do something, the possibility of a better way is often academic.

So we have seen a mass movement: Processor Technology, Cromemco, et al. have built boards for the Altair, and then their own computers on the same system of interconnection. Whereas little computers like the Sphere, the Jupiter Wave Mate, the Digital Group, indeed MITS' own 6800 machine - and conceivably the Heathkit computer that is to come -- have ignored this standard at their peril, attempting to create a non-interchangeable market.

So far these little computers, from little manufacturers, comprise the "amateur" market. But next we will see the bigger manufacturers come rolling. Large-scale manufacturers will enter the game; some are already talking about building millions of units. And already there are "video games" actually containing a computer, such as the Fairchild for \$150.

Soon the media will hear about it, especially magazines and TV news. They will go out of their minds, and give home computers the full-scale "mysterious phenomenon" treatment. They will make all those incredibly stupid jokes over and over, about "thinking machines", and about 1984. But few of the journalists will bother to find out what computers are really about until everybody else knows already.

The funniest aspect of the press coverage will be, of course, that everyone they talk to will have a different story; it will be impossible for reporters to converge or focus. Some of the people interviewed will tell the press that home computers are an extension of ham radio (this has actually happened); others will seem to say it is an outgrowth of the space program or of video games, or of the New Math, or of pocket calculators, or who knows what. And they'll all be wrong because they didn't see that computers were for people in the first place.

After the media, of course, the stock market will react. Wall Street will behave as Wall Street always behaves in the face of a new consumer fad, and public corporations involved in home computing will lurch upward in price.

About this time it will become clear that home computers are actually as important as people thought video disks would be. Many people will try to get on the little-computer bandwagon in inappropriate ways. The cable TV operators, for instance, will blunderingly keep trying to set up some sort of a connection to home computers. There isn't any -- except that they could use dinky machines to generate little animated cartoons and logos.

Now the really big manufacturers will come in. Philco and Zenith, Sylvania and Magnavox and the Japanese TV concerns will bring out their home computers. Then will come the me-too electronics packagers, with a dazzling array of brand names offering small silly variations.

One problem for the larger organizations is that their decision processes are perilously slow: since the home computer market will change drastically every six months, the big slow companies may never get off the ground. They will be continually deliberating over how to compete with last month's product, and being dumfounded by the next development. However, some big manufacturers will swallow the little companies that are off to a good start, paying lots of money for the privilege.

While there will be many more small accessory and software houses, it will be much harder for the second generation to make it big; their best strategy will be to target and specialize.

Certainly the maturing markets for dinky computers will become ever more differentiated. *Beginners'* systems, whatever the chip, will usually come with Basic and game cassettes. *Office-standard* systems will probably become stabilized on the computer-with-screen, like the SOL, Sphere or Intecolor; but interchangeability will be an important feature, possibly favoring S-100 machines. *Graphic* systems will become popular among artists and doodlers and playful people, and indeed among the well-to-do these may become a sort of animated decoration. (We will see them turning up all over retail stores, with sales messages and animated cartoons.) *Literary* systems for readers and writers will include disk, high-capacity display, and high-power text manipulation and filing programs -- of a type, I believe, that nobody has seen yet. *Baroque* and *slinky* systems will come out for technical showoffs: like the computers that have one of every processor chip, or multiple Z80's: some with lights and switches by the yard, others with lots of "computerish" lettering, military styling, futuristic spheroidal boxes and so on. Finally, perhaps most important, we will see the "nothing is too good for my kid" configuration, with graphics, extensive text handling, programmability and large memory; possibly musical input and output as well. (Kay's Dynabook, at Xerox Palo Alto Research Center, is the exemplar of this class -- even though it may never appear commercially.)

As to manner of interconnection, alas, there will be all too many separate hardware worlds. There will be the ever-growing S-100 world, of Altair, Imsai and so on. There will be the nonstandard world. Some individuals will actually buy high-class hardware out of the professional computer world, which will be nonstandard as far as the hobby world is concerned. (In the next two years it is not clear the entrenched minicomputer manufacturers can lower their prices, or adjust their thinking, enough to compete.) Then there will be the other chip computers, packaged with incompatible accessories, such as the Intecolor. Then there will be the discount packaging world, probably with more systems of incompatible accessories. The incompatibilities will be regrettable, but the marketplace will eventually punish this approach.

There will of course be a variety of other oddly-interconnected computer configurations. Certainly we may soon expect the TV with built-in computer. We may see the "smart hi-fi," a multi-room, multi-tuner, bus-oriented programmable system. And don't rule out the computer-in-a-van, with one monthly payment for both transportation and hobby.

Though it would be a good thing, it is doubtful whether any later and better chip computer will attain, in the next two years, the universality that the 8080 (and the Altair bus) have enjoyed. We *ought* to have a 16-bit amateur ma-

chine, at least one allowing relocatable programs, the LSI-11, chip Nova and TI 990. But the central position of the 8080 will be hard indeed for another unit to capture.

In the software area, meanwhile, we will go from confusion to chaos: but perhaps after about two years it will begin to straighten out.

Utterly disparate software packages will appear, all proclaiming themselves to be Everything You Need. (There is one advocate of Tiny Basic who says numerical input is sufficient for "all software".) Programs will appear embodying the greatest possible variety of viewpoints.

There are many ways of thinking of software, and in the hobby world some of them are weirdly simpleminded. Many hobbyists come out of electronics, and insist on thinking of a computer as "a collection of switches." This is true in the same sense that a human being is "a bag of chemicals." But some hobbyists think, on the basis of this impression, that programs are little things, like ten or fifty lines long; like a binary-to-decimal conversion routine. From that follows the hobbyists' mystification by, and in some cases objection to, the copyrighting of programs.

It would seem peculiar that people who claim to have compunctions about shoplifting, or the looting of parked cars, nevertheless proclaim -- as some computer hobbyists do -- that they have a perfect right to violate others' property rights in respect to copyrighted software. But we can understand this view slightly better if we consider that many electronics-oriented amateurs think programs are small and insignificant -- of the level of complexity, say, of a limerick.

Many programs are short and simple. And many enjoyable computer games run in less than a hundred lines of BASIC. But tomorrow's home programs are not going to be just little programs. The important software of tomorrow will be immense turnkey systems of programs, comprehensive and interactive, for a broad spectrum of business and personal uses.

New business systems will wholly reverse the computer business systems of the past. The business systems of old, based on batch processing, created strange work procedures: transactions had to be keypunched and sent into the computer in long trains, to be processed one at a time by the same program. This created dehumanizing job definitions, such as keypuncher. Now, however, clerks will be able to process business paper one piece at a time, in the order of its arrival, and businessmen will discover themselves in control of their firms once again. But the programs will be big.

There will also be large-scale programs for gaming and simulation and animation will likewise be swapped and overlaid from disk.

The new big software packages will have to do a lot of storage management -- overlays, swapping and housekeeping. Once such a system is started, it will not need reloading, it will roll and undulate from one function to another at the user's whim, swapping and storing (fail-safe) and displaying interactively. These features do not come easy; they involve a lot of programming.

Now not everybody may want or need such extensive services. Some hobbyists, indeed, may also enjoy the ritual of "running separate programs" -- toggling in bootstraps and running in loaders and programs for each new activity, in an anachronistic and playful simulation of batch processing.

But the big and sophisticated software will be to the little toy programs as *War and Peace* is to the limerick. The really good systems will be unique and distinguished, even works of art. People will devote months and years to their creation, and they will have considerable commercial value -- being leased to businesses, say, for hundreds of dollars a month.

Speaking as a prospective supplier of such advanced soft-

ware, the amateur market tends to interest me not at all. I see little reason to sell to hobbyists, and considerable reason not to. For the revenue of one well-behaved business customer, it might be necessary to deal with ten belligerent and disrespectful amateurs, some of whom have given notice of their intention to break any software contract.

In the ordinary computer world, the most emotional issue is people's preferences with regard to computing languages; copyright is accepted as an ordinary part of life. In the amateur world the most emotional issue is that of copyright, with languages second. My own feelings about copyright are very strong: copyright is one of the only ways the little guy can get a leg up -- you will excuse me for oversimplifying -- against the Big Guys and the Bad Guys. I publish my own book, *Computer Lib*, and receive a modest income from it. Now if there were no copyright protection, anyone could print *Computer Lib* in Taiwan, and sell it for less than I get, and I would get no benefit for the years of work I've put into it. But the law of copyright, a Federal law, says the book is mine, and that people can only make copies with my permission; I like it like that. The same law, now, is extended to computer programs as a form of writing; and the same protection is available to you, practically free of charge, for the work that you put in on programming.

Anyone is of course free to give away a program he has made, since the copyright presumably belongs to him. And people to whom software means *small* programs of their own concoction will be happily giving them away.

But people who are working on big programs -- programs involving tens of thousands of lines of code, programs that take months or years to create, will be very concerned for their copyright protection. To further this protection, we will begin to see software disguised as plug-in hardware, software coming on mystery tapes in unknown formats, software disguised in every possible way. This cloak-and-dagger approach is quite regrettable, but may be the only way to create an orderly market. I think there will be a period during which the good software, the big and serious systems, are held off the amateur market and made available only to commercial customers. But eventually, perhaps at about the end of the next two years, we will see bargains struck up with well-behaved individual users, and a system of safeguards hammered out to make an orderly market possible. These safeguards may even include programmable hardware serial numbers, just as used in the bigger systems.

Languages. Just as in the big computer world, in the dinky world we hear hotheaded arguments about programming languages. Computer people always become very pugnacious about their favorite programming languages -- especially if they only know one or two. But the dinky-world arguments about language are a travesty on those same arguments in the straight computer world. The same honors are here attributed to BASIC that more knowledgeable computer professionals attribute to Algol, or SNOBOL, or PASCAL. (In the hobby world, one actually hears people with Ph.D.'s, who certainly ought to know better, arguing very angrily that Basic is the ultimate.)

By two years from now, after a lot of frustrated effort and experience with big systems shoehorned into little computers, people will have realized that for powerful software we need good and powerful languages. They will also have found out that these must be *structured* languages, in the Dijkstra sense, which make programming so much more manageable; and they must be *extensible*, allowing the rapid creation of new commands for particular ranges of purpose. This means that interpretive languages like LOGO, TRAC language*

*TRAC is a registered trade and service mark of Rockford Research, Inc., Cambridge, Massachusetts.

and SMALLTALK will assume new importance, as will such compiling languages as FORTH and "C".

One of the great strengths of the extensible interpretive languages is the degree to which they simplify *big* programs. My colleague William Barus and I recently created an animation program for the VDM board using TRAC language; the program is about 16K in size. TRAC is not as easy to learn as Basic, and for small programs may seem rather clumsy. But as programs grow larger, its intrinsic orderliness keeps programs manageable. (The 16K program was written in about three weeks, and seems to work quite nicely, allowing both frame-by-frame animation and subpicture overlays.) With unstructured languages like Basic, programming becomes exponentially more difficult as the size of the program grows; with structured languages, the growth in difficulty seems to be logarithmic.

Basic will remain the language for beginners, unfortunately; no standard Dijkstra-structured upgrade is presently defined for it, although a "structured Basic" has been under design where Basic began, at Dartmouth. (But for those who want to sell language processors, there is a simple secret; hobbyists want any language as long as it's *called* Basic.)

It is a great pity that hobbyists are so insistent on talking about the physical box of the computer, and so resistant to learning about the issues and depths of programming languages -- even though all their programming difficulties stem from these issues and depths. Meeting frequently with computer hobbyists, it is astonishing, and depressing, how often one must repeatedly answer such questions as "What is a computer language, anyway?" and "What is structured programming?" and "I don't see why it can't all be done in Basic." There has got to be some form of general consciousness-raising in this area.

In any case, when the smoke clears, about two years from now, hobbyists will begin to realize that the more advanced languages -- at least the ones that fit on the dinkies -- make programming, much, much easier.

Such languages -- like LOGO and SMALLTALK and TRAC Language -- will gradually assume primacy. But we may hope that their being "advanced" languages will not make people think they are unsuited to beginners. On the contrary, structured extensible languages are the right way to learn programming. For instance, at MIT's LOGO project, they tried teaching programming to kids two ways: with BASIC and with LOGO. After a few weeks, I have been told, the kids who had started on LOGO were hopelessly far ahead, and, it is said, did not want to speak to the BASIC-trained kids because they thought the latter so ignorant and incapable.

No one who is really interested in computing languages would claim that the "ultimate" language exists; but there is a new generation of languages shaping up now, the "actor" and "agent" languages, whose proponents expect them to be as far ahead of LOGO, TRAC Language and SMALLTALK as these are ahead of Basic. Which of these advanced languages will fit on little machines is another question.

The merchandising of little computers will gradually break out into a number of separate sales approaches, much like the merchandising of anything else. The mass-market computer will of course be sold by the corner electronics discount house, in a fancy package. You'll get the instruction book and an impatient rundown, but no personal help. In the box you'll find the computer, an instruction book, warranty card, and list of repair centers. Perhaps also one free game cassette.

This mass-market approach will probably go two ways: toward the non-compatible, as with the Fairchild, computer-game tapes, which will be a dead end; and toward compatibility, as with systems offering Basic and a standard cassette interface.

For more serious users, and for the S-100 world, the computer stores will continue; but they will become a cross between the hi-fi store, camera rental house and laundromat.

A viable take-home rental market will develop, presumably as a part of the computer stores. There you will be able to rent simple standard units (as you might rent a standard car), or, later, more esoteric units (in the way that professional motion-picture equipment is rented).

Repair set-ups will blossom. There may appear franchised computer-repair chains, similar to the automobile-transmission repair centers. But these will be walk-in centers; cheap computers will not be eligible for house calls. (Many computer-center people, who are used to repairmen coming *in*, cannot visualize throwing the computer on the front seat of the car and driving it to a repair center, as you do with an amplifier. But that's how it'll be.)

The amateur used-computer market will come into full swing. We will see the more fanatical hobbyists buying not just used Altairs but commercial minis and old "big" machines -- real 360s and 7090s -- and real antiques and fond replicas, such as Eniac and TX-2. (We may yet see a fully-loaded Imsai go for more than a small 360, just the way in 1974 you saw small used cars selling for more than big ones.)

A variety of personal services will appear, probably in conjunction with the expanded computer store. Such services will include program storage and printout; soon, time-sharing parlors, having clusters of terminals which may be rented by the hour. (This is the laundromat analogy -- renting a terminal by the hour is not unlike renting a washing machine.) In the next two years, we should see services grow from the simple ones to a dazzling variety of new ones, such as advanced text services and retrieval, digital music synthesis and movie-making, and great libraries you can reach through your home screen.*

The prospect of great on-line libraries raises great questions about truth and freedom -- for which there is no room in the present talk, but about which much more will be heard.

I have elsewhere predicted that there will be ten million computers in American homes by 1980. This may be a considerable underestimate, in the light of American consumer contagion. It will not be long, in my opinion, before the home computer field becomes most of the computer field in absolute dollar volume. This will take longer than two years. But I would predict a hundred dollars per capita per year within a decade.

Pretty good so far, right? But I'm afraid it will not be all sweetness and light. We can expect the dinky revolution to have a convulsive effect on the computer industry, and on the society as well. First let us consider the industry.

IBM will be in disarray. Fine-tuned to a captive market having certain kinds of submissiveness, it is hard to suppose that their kind of sales, let alone their kind of computer product, will give them a ready entrance to the kinds of markets now opening. (Supposedly as a personal computer, they will push the 5100, actually a 360 in disguise; but they will probably not know *what* to do if someone puts out the S-100 adapter for the 5100, effectively offering a 360 with Altair accessories.) The jolt to IBM's product line of all these developments will be considerable. A loss of revenue for IBM, or at the very least a slowing of its growth rate, seems to me inevitable. This could be traumatic to the stock market and to other true believers.

* See T. Nelson, "Personal Digital Services," IEEE Computer Magazine, March 1977, and "Design of a Transcendental Literary Network," to appear in proceedings of the 1977 National Computer Conference.

Most of commercial time-sharing will be down the drain, being priced at the top-managerial level but only usable by technicians.

Organizations will be in internal turmoil, as they see kids doing with computers what their programming teams can't. And as the new availabilities reverberate, many internal goals and budgets will be shot to hell.

There will be mass layoffs in programming staffs, and this will be only a beginning. The skills involved in programming are comparable to those involved in automobile mechanics, and as the number of capable individuals increases, the pay levels will go down, because systems will need less and less tinkering to be made to do what you want. It will gradually be revealed that most programming has been make-work around ill-designed systems.

Between laymen and the old-style computer people there will be a lot of questions asked. "Why weren't computers easy to use before? Why couldn't I have had this sooner?" These are not pleasant questions to answer.

For computer professionals will be the last to see this coming. Professional programmers have lived in a world of account numbers and job submissions, core declarations and runs aborted by the operating system -- silly complications that evaporate as soon as everybody has his own computer. Theirs is an awkward and inhospitable world that bears as much relation to the real nature of computers as a string of frankfurters bears to a living cow. The "Nature of the Computer," for many who have worked with it, has really been the system of bizarre bureaucratic project management under conditions of monopolistic mystification. And so it may come to be revealed that the people who have worked with computers most have understood them least.

In some areas of the society there will be pandemonium, as the personal lives of a vast number of people -- especially computer people, who thought change was their friend -- are uprooted and retrenched. These more general effects are hardest to predict. I suspect that the main public reaction to little computers will be a mixture of delight and anger -- delight at the new fun and facilitation, anger that computers were ever allowed to create the harm and inconvenience that they have till now.

Here is the final surprise. We the computer folk are not going to be converting the non-computer world to be like us, as some of you may have been hoping. Quite the contrary. We are going to be repudiating the computer past, resolving not to push people around any more, and finding out how to make computers easy and helpful to everybody.

MUSINGS

The future use of computers is, and should long ago have been, personal. Misled and perhaps occasionally malicious, the industry has ignored this as a possibility for thirty years; computers have been held back from a personal market for perhaps thirteen. But the movement cannot now be stopped.

Why has individual computer use been so retarded? Partly because of the nature of our society, and partly because of the trickery by which computers have been sold: *not* because of the nature of the beasties themselves.

For instance, I believe that the point of the 360 computer was to lock out the other manufacturers, especially those of minicomputers, and postpone highly interactive systems -- the kind of computers uses that would be good for, among other things, personal use. The 360 drastically postponed personal usability. And in an unholy alliance with the big manufacturer, the computer centers long ago became entrenched power units especially concerned with preventing maverick applications, and with preventing any other departments -- let alone

individuals -- from getting a computer. Underhanded methods have been used, it is sometimes said, to get rid of people who advocated interactive systems. The computer field was structured to lock out both the interactive applications and the people who wanted them, or might have.

Now the computer centers are on the defensive, publishing articles saying there is still a place for the centralized computer facility. They may be right, but they have an increasing burden of proof. As dinky computers and interactive styles of use take over more and more, little sympathy is due to those who strove to hold them back.

But let us lift our eyes from the past, from the limitations and trappings of how you've had to do it. Rather than being obsessed with the styles of computers as they have been used, we need to consider, and design, the environments that we really want. I call this the Higher Virtuality. By virtuality I mean the effective environment we create with the computer, as distinct from the computer's "reality", the physical parts and program techniques that make things come out that way. The higher virtuality, then, is the computer environment we should strive to create. It is hard to make the choices and designs that this entails.

The virtuality that calls to us now, I think, is the new world of highly interactive systems. Cursors and panels, lightbuttons and menuplexes, maps and graphs and fast-changing screen layouts, are the pieces from which we will construct our new experience-spaces. Now, we tend to think of highly interactive computer systems -- for text editing or retrieval or movie-making or whatever -- as the ones with fancy screens and light-pens. But this brings up an interesting general point. The fancier super hardware isn't as important as the imaginative and artistic use of whatever we have.

Let's consider just the problem of pointing at things and controlling what's on the screen. Most of us don't have a light-pen or Englebart mouse on our home computers; but if we are clever we can make systems highly interactive even if we just use keyboards. We can zip cursors around with the keys to select from menus. We can even whiz between environments. Suppose the interactive program can change the meanings of the separate keys dynamically: at one instant, specific keys can be arrow keys, to move a cursor about the screen; at another instant, they can type musical notes, or special picture-symbols; or anything.

This is not an unimportant matter. Some of the snazziest research setups have highly interactive keyboards of this type. Bitzer's PLATO system, for instance -- a large special-purpose graphics time-sharing network run out of the University of Illinois. The main program can react to every key-press, and the keys can be automatically redefined under program control. The letter "D" on the keyboard can be an arrow when pressed at one instant, create a whole picture of a dog when pressed in the next. This feature makes PLATO one of the most dramatic introductions to interactive computer graphics. Kay's Dynabook system, at Xerox, likewise allows a key-by-key dynamic reaction to what the user does. There is even a research setup with a "phantom keyboard" -- its keytops have colored changing labels that you can read right through your fingertips. (This is done with a color video display and a semi-silvered mirror.)

Such dynamic keyboard capabilities, together with our screens, make highly interactive systems easy and exciting to create. (It is regrettable that most if not all IBM systems do not allow this dynamic key-by-key redefinability. IBM systems will not ordinarily respond to single arbitrary key presses; you have to hit either the Carriage Return or an "Attention" button to make the systems respond. Again, the interdiction of highly interactive systems. Was this malicious design?)

This key-by-key arbitrary response leads us, as do many

other paths, to the real question. Given that we can make each key have any effect, then what? How should our systems behave? How do we make the interactive system make sense?

On the interactive screen we deal with a special new virtuality: architecture in virtual space, where anything can happen based on anything you do. The space has to be invented. This architecture of conceptual space, this conceptual architecture of screen-space -- is a new realm where we are combining feelings and effects, as in a movie, and the need to be clear, as on a map.

The problem really becomes, I think, giving the system conceptual unity, which is just a more general example of the problem of giving conceptual unity to the interactive keyboard. How do we give the key assignments, and how do we give the larger system, conceptual unity? "Conceptual unity" is an extremely pliable concept. What many programmers think is conceptual unity, clear and simple, is not necessarily conceptual unity to the non-computer people.

The problem is very like architecture. Now, I'm no architect, but I have spent some time in buildings -- indeed lived in them! -- and I think there are several clear criteria for what makes buildings good. It's nice for them to look good, sure. It's nice for them to have good cost ratios for their builders. BUT I submit that one of the most desirable things about good buildings is for people to be able to get from place to place simply, and know where they are.

I recently taught at a great urban campus filled with grand pretentious buildings. The architect gave no mind to helping people find their way around. The buildings looked very Futuristic, but were incredibly confusing and inconvenient.

One evening, in the depths of one of the incomprehensible buildings, I had to mail a letter and make sure it went out in the morning. I went to the mailbox on the main floor, and there was a sign on it saying that after hours one was to use the mailbox on the *first* floor. (This was my first inkling that I was not already on the first floor.) Diagonally to my right was an elevator. I stepped in, pressed "1", and prepared to retrace my steps, walking diagonally back to my left to the first floor mailbox.

I was faced with a cinder-block wall.

I walked around to the right, then finally found a back corridor going in the direction the mailbox should have been in. But now there was a succession of cinder-block cubicles and rooms. In none did I see the mailbox.

Eventually I was able to find the mailbox by a system of triangulation. By counting paces, and making a map marked with paces marked and a combination of ninety-degree angles (of which the architect was fond), at last I found the mailbox in one of the rooms of the cinderblock environment.

Here is an example where an architecture of *real* space was created with no concern for personal clarity or orientation.

The problem is the same for creating interactive systems. Creating interactive systems, and their virtual spaces, should be an *art*. And the real kicker, I believe, is this: *all computer systems should be highly interactive*.

The confusions and oppressions of yesterday's computer systems vanish where the user can see his alternative on a screen, get quick explanations, see maps of what he is doing, and get all the other helps that the interactive screens can provide. This means the continuing oppressions of yesterday's computer systems -- even today's -- are the opposite of the way computers ought to have been used all along.

We shall see. Time, and the marketplace, will decide. When people get to use interactive systems, they will be much less interested in either manual methods or batch.

But the central concern of highly interactive systems will be making things clear and simple. Now, making things clear and simple and easy to us is, I'm afraid, the opposite of what some

computer people, perhaps some of you, want to do. So in an important sense, it is *not* the laymen who have to learn Computerese; in the greater sense, WE MUST ALL LEARN COMPUTER EASE.

The next two years will wreak extraordinary changes, and we will "computerize" society for fair. But it may not be what you computer-smart people expect.

What some of you have been considering "the new era" of home computers may correspond to the tin-can and crystal era of radio; and the convivial hobby you are part of right now may vanish like the crowd that welcomed Lindberg at Orly. They don't come out to meet the planes anymore. Today's summer-camp camaraderie won't last forever, and the computer will probably become a home appliance, as glamorous as a canopener, within a couple of short years.

What then is there left to believe in...? To hope for...?

I suggest that we look to simplicity and clarity (to make people's lives easier) and to truth and freedom, for their own sake.

Thank you.

BIO: TED NELSON

Ted Nelson holds degrees in philosophy from Swarthmore College and sociology from Harvard, but since a fateful computer course he took in 1960, his principal labor has been the design of computer systems for writing, moviemaking and general personal use. Not so much for the rest of mankind, at least originally, but for his *own* general personal use, writing and movie-making.

Nobody would hire him for this sort of thing in the early sixties, so after obtaining his master's he has been a dolphin researcher, folk singer, junior executive in publishing, consultant to the antiballistic missile system, and yellow cab driver in Manhattan, in that order. He has also taught at five different colleges and universities, in departments of economics, sociology, anthropology, art and mathematics. He is presently visiting lecturer at Swarthmore College, teaching courses entitled "Art and Animation with the Computer" and "Literary Machines." His students use the SOL computer and LINCTape drive now in his living room.

Ted's book, *Computer Lib*, is now in its fifth printing, and has been widely adopted at colleges throughout the country.

He is a co-founder of a computer store in Evanston, Illinois, the Itty Bitty Machine Co., Inc., and is chairman of the board of its software arm, Sophystems, Ltd.

Ted Nelson insists on being called a generalist. This befits the immense library system on his drawing board, the Xanadu (trade mark) hypertext network. Through the Xanadu system, Ted claims, people all over the world should be able to have instantaneous access to everything that has been written -- adding their own marginal notes, or multidimensional "hypertext" writings, as they see fit.

Ted's glibness, showmanship and mischievous intellectual precocity are found either endearing or infuriating, depending on the listener. His emphasis on interactive computers as a form of movie-making and show business stems from a unique bias: both his parents are prominent in the theater and films. His mother is Academy Award-winning actress Celeste Holm, his father is film director Ralph Nelson, perhaps best known for "Lilies of the Field" and "Charly."

Nobody is quite sure whether Ted's most recently announced project, a feature film based on his *Computer Lib*, is a gag or not.

AN INTRODUCTION TO COMPUTING TO ALLOW YOU TO APPEAR INTELLIGENT AT THE FAIRE

James S. White
1202 Riverview Lane
Watertown WI 53094

ABSTRACT

The following is an introduction to computing and presumes no prior technical knowledge. The purpose of this discussion is to provide the novice enough basic knowledge so he or she can learn from, and enjoy, other Faire activities. This presentation includes some very basic, non-technical ideas about computers, especially home microcomputers, their characteristics and construction, and how you can choose and use your own home computer. Finally, some of the types of computers exhibited at this Faire are discussed.

INTRODUCTION

Welcome to home computing! The great interest in this Faire dramatically demonstrates that the time has arrived for a computer in the house of everyone who wants.

During the past few years, expensive, complex computers in industry and commerce have helped you by cutting costs and raising the quality of many products and services you use. Now, computers have been designed for home use and their prices have been drastically cut. Personal computers are available, here, today, at prices similar to other major household purchases.

Thousands of Americans are enjoying their own computers now. You can select a computer for your home, a computer that will be easy for you to use, even if you have no technical experience. That's what this presentation and this Faire are all about—learning how you can benefit from your own computer.

Learning about computers can be a little confusing because many different kinds of things are sometimes called computers. A computer can not be consistently identified by its size, appearance, construction, or use. None of these features are characteristics of computers in general. Computers are electronic devices built according to certain technical rules, which we won't discuss here.

However, all types of computers do have some common characteristics that make them different and much more powerful and helpful than many other things we use. We will discuss some common characteristics of today's popular types of home computers, characteristics that are important to computer users and that make computers especially helpful to us.

COMPUTERS ARE TOOLS

We use tools to extend our capabilities and to multiply our natural powers. We use tools to help us do things we can't alone, or so we can do jobs easier, quicker, or better than we can alone. Computers help us in all these ways.

MENTAL TOOLS

Computers help our minds. Rather than helping our muscles, as most tools do, computers multiply our intellectual power and capabilities. Computers are extensions of our minds. Computers are tools we can use for greatly increased mental achievement and enjoyment. Because our mental activities are much less limited in potential benefits than our physical activities, the helping potential of computers is much greater than the potential of other tools.

PROGRAMMABLE TOOLS

Computers follow people's instructions and do nothing unless instructed to do so. Computers are useful as extensions of our minds primarily because they faithfully do just as we instruct. Your computer will follow your instructions. Your computer's ability to follow a complex set of instructions and thus do just what you specify gives it power far beyond any other kind of tool. We don't know the limits of the ultimate potential of computers because we don't know the limits of the ultimate potential of men and women.

If you examine a computer in detail, you see that it recognizes only a few instructions that can do only certain limited things. These instructions allow a computer to, for example:

1. Perform mathematical calculations
2. Store a large amount of information, and select a particular piece of information to meet a specific need
3. Evaluate a vast number of alternatives to determine the best solution to a question, even a question unrelated to mathematics.

However, the instructions a computer recognizes are so powerful and so varied that a person can combine them to instruct his computer to do almost anything as an end result.

A set of instructions you give to your computer is a program. When you instruct your computer, you are programming it.

A computer's program can be changed easily and quickly. A complete program can be changed in as little as a fraction of a second. However, the work or play of composing a program the first time usually takes a person several hours or longer.

CONTROL TOOLS

Computers can help us by controlling other tools and machines. A computer can cause a much larger tool to do something in a certain way and at certain times, as programmed. This capability extends the power of a computer far beyond the things it can do directly.

Our minds are control devices which control our muscles. We thereby control the most powerful of machines and many natural events. Computers are devices which control electricity, and thereby can control the most powerful of machines, the generation and use of power, and many natural events.

So, although computers function intellectually, their direct benefits extend far beyond the intellectual. By helping us to "work smarter", computers can be a way to physical accomplishment that is better than just working harder.

MACHINE (TOOL)

A computer is a machine. A machine is generally a tool that can operate by itself, doing a desired job, without always needing the personal control or constant attention of a person. A computer can follow its assigned program with little or no need for the further assistance of a person.

A machine used in a home is generally called an appliance. A computer can be a home appliance, a machine that can continue working to help us while we do things that are more useful or enjoyable.

VERSATILE TOOLS

Computers can be extremely versatile. One computer can do, and be, many different things. A computer's versatility results from two other characteristics. The first is a computer's previously discussed programmability, the fact that you may instruct your computer to do whatever you want.

The second reason a computer is versatile is that it is a system. A system consists of various parts, each doing its own job. All of a system's parts operate together, supporting each other as do the various organs in our body.

A computer can be composed of a few or many different types of parts; each can have widely varying characteristics. The parts you choose to assemble your computer are those parts that together will give the resultant system the capabilities to best do what you want it to do, at a price you can afford. You can change these parts as you want your computer's capabilities to change.

The wide range of possible computer characteristics also results from the fact that a computer, as a control device, is often combined with other devices. A large variety of devices may be combined so the resulting system can have a wide variety of characteristics. A computer generally remains the key part of the resulting combination, so all these types of systems are often called computers.

Computers are so versatile that a computer can do, or control, any rational or mechanical process,

activity, or happening. What a computer can do is essentially limited only by the skill and imagination of the person(s) who build and program the computer.

A particular computer must be equipped with the power and accessories appropriate for the job it will do. Power usually isn't a serious constraint; a computer that is half as powerful may just take twice as long to do a given job. Even computer accessories are usually versatile, and many can be used for a wide variety of different kinds of jobs.

PRECISE TOOLS

Computers can be perfect. A properly programmed and operated computer makes so few errors in end results that a computer mistake may never occur during years of doing a particular job. Computers do occasionally make mistakes, typically one in many millions of operations. However, computers can, and many do, check their own work so that a mistake is not given to a human user.

Most "computer mistakes" occur because computers do exactly as they are instructed. Many people find it difficult to give exactly correct instructions to handle complex problems. The computer that follows incorrect instructions will produce an incorrect result, and is often incorrectly blamed.

FAST TOOLS

Computers are extremely fast. A typical home computer can do 500,000 of its basic operations per second. Home computers often move information as fast as physically possible--at the speed of light.

This great speed also gives computers great power. Because computers are so fast, they are very useful even for those jobs that they can't do efficiently. Computers just go ahead in their roundabout way, doing many, many operations to provide a useful result. A typical computer that must do a million operations to complete a particular job may be done in two seconds, which is often sooner than any alternative.

How powerful are computers? A computer can be a million times as powerful as a man. That's almost beyond comprehension, so let's consider an example.

The great pyramid of Egypt is generally accepted as the world's most costly single structure, in terms of man years. According to the most commonly accepted estimates, 100,000 men worked 20 years. In total, two million man years was required to do this job.

Building pyramids is out of style today. Our world is oriented towards numbers, such as dollars. So let's think about two million man years of working with numbers or calculations.

The first common home computers were delivered about two years ago. One of these computers, working for these two years, could have done calculations that would take one million men working with pencil and paper 24 hours per day, 365 days per year, for 2 years. You can buy such a computer here, today, or one much faster.

PERSONAL TOOLS

Despite their great power, computers can be personal tools. You can select your own computer to help you, and to expand your own capabilities. More than 10,000 Americans have their own computers; many are here today for you to meet and learn from. More than 100,000 Americans are using computers as personal tools, to make their work easier and their play more fun.

We should consider a couple of philosophical points here. Despite its great power, a computer can no more do anything by itself than can any other machine. Computers are constructed, and instructed, by man, as tools. A computer is and does only what it is made to be and do by man.

Further, as we plan to use computers, we should consider their strong and weak points, just as we do for other tools. For some tasks, computers can be a million times as powerful as man. In other of the many ways we can use computers, they will be much less than a million times as powerful as we are alone.

A COMPUTER IS:

Having reviewed outstanding characteristics of

computers as they appear to users, we are now ready for a glimpse of some technical details. We have come to a fundamental question: What is a computer?

A computer is a programmable electronic data processing system. Computers have been called many other names, but this definition gives us a good start towards understanding computers technically. We'll consider the significant meaning of each word, except "programmable" and "system", which are discussed elsewhere.

Because a home computer is electronic, it is powered by electricity, generally from a 110 volt wall outlet, just as other electronic appliances are. Computers are solid-state electronic, so need relatively little power and are not likely to wear out soon or to need many repairs. Many parts of a computer are less likely to fail after they have been used for several months or years than when they are new. However, computers are very complex and some parts are not solid-state, so the availability of repairs should be considered when selecting a computer.

A computer works with data, or information. This characteristic is similar to the fact that our minds work with ideas, rather than the physical things our bodies work with. However, as previously mentioned, a computer can control almost any physical activity.

The data a computer works with must be objective and precisely expressed. Computers can't work with feelings or subjective ideas unless they are expressed objectively. Neither computer data nor computer results are ambiguous, despite the fact that their meanings aren't always obvious to all people.

All computer data is expressed in a very simple numeric code. Knowledge of this coding is necessary for the user who wants to understand the internal operations of a computer. However, a computer can translate its codes into words or numbers that you are familiar with, so computer codes need not be of concern to you. You can use computers here today without consciously learning any codes.

A computer processes data. Processing means changing, generally into a more useful form. A computer changes data it has been given into data that is more useful, or more fun, for its individual user. The processing a computer does is whatever its program directs it to do.

MICROCOMPUTERS

The type of computer generally used in homes is called a microcomputer. Most of the computers here today are microcomputers.

A microcomputer is a computer that is small in size and price. A microcomputer need not be small in power or capability. There is no exact definition that precisely distinguishes microcomputers from other types of computers, just as there is no commonly accepted rule that distinguishes a micro-skirt from a skirt. This paper discusses computers in general and emphasizes the characteristics of home microcomputers.

A COMPUTER'S PARTS

In addition to having certain characteristics in common, most home computers have certain types of parts in common. Next listed are some of the types of parts which are usually combined to form home computers and the computers here today.

INPUT

To start a computer working you give it a program and data. Communicating these things to a computer is called inputting. The computer parts you use are called input devices.

You often communicate to a home computer in ways similar to some you use to communicate information and instructions to people. You can communicate to a computer by pressing keys, or buttons, on a keyboard that is very similar to that of an electric typewriter. Some lower cost computers have keyboards that are much like the keyboard of an electric calculator. All computer keys are electrical switches.

Other types of switches used for computer input are similar to the on-off light switches in your home. Computers with many switches of this type are best suited for the user who wants to know much

about computers and work with them in intimate detail.

A variety of other types of input devices are used with home computers. Some computers can understand spoken words, but this capability is not yet common for home computers.

PROCESSOR

The key part of a computer is a processor. A processor is a group of electronic circuits which actually performs the calculating and the logic operations comprising a program. A processor performs the same functions for a computer that a brain (less its memory function) does for an animal.

The processor of a microcomputer is a microprocessor. This is a small electronic component, one kind of the wide variety of components called chips. A microprocessor is a small form of a processor. Microprocessors are widely used as control devices, in many kinds of machines besides microcomputers.

You may hear a microprocessor called a micro-computer or a processor called a computer. However, there are critical differences for the user.

A processor is a control element, a part of a machine that directs, by following a program, the rest of the machine to do whatever it is supposed to do. A microprocessor can be part of a calculator, part of a microwave oven, or part of a sewing machine. In none of these uses is a microprocessor a computer. Nor is the calculator or the microwave oven a computer just because it is controlled by a microprocessor.

A computer, or a microcomputer, is a complete machine, including all the parts and accessories needed to do whatever it is used for. A computer can operate entirely on its own, needing only a source of energy and, at the start, directions from a person. A computer is built so you can input your instructions to it, and so it can communicate the results of its processing to you.

MEMORY

A computer uses parts called memory to store, or remember, the programs and data that have been input and the results that have been computed. Some of the parts that perform the memory function are chips, similar to microprocessor chips. In many computers, memory chips are physically part of the processor.

A home computer can also remember information by storing it on a magnetic tape cassette, such as the type you use on a cassette recorder to record, and thus remember, music or words. A cassette recorder can also be an input device because you can use it to read into your computer programs that other people have written and recorded on their computers.

Another type of home computer memory device is a relatively expensive and powerful device called a floppy disc drive. This is a recording and playback device that uses a rapidly spinning disc about the size of a phonograph record.

OUTPUT

The final common functional part of a computer is an output device, which a computer uses to communicate to you. The most common home computer output device is one you are quite familiar with--a TV screen, on which the computer displays words, numbers, and sometimes pictures. This screen can be that of a standard home TV, or of a specially designed device.

Computers can also display their output by using lights that form numbers and letters, as do the output lights on an electronic calculator. Computers can also use individual lights that are either off or on, and thus communicate to you the status of various processes and coded data. Computers can also communicate to you by printing on paper and, less commonly today, by "speaking" words.

INTERFACE

Many computers have several groups of electronic components, called interfaces, to allow the computer to communicate with devices not normally compatible with the computer. An interface is like a translator. You can, for example, buy an interface so your home computer can display data on a TV set you now have.

Often some interfaces are built as part of a computer, while interfaces for less common input output, and memory devices are options.

POWER SUPPLY

Computers have one other part which the potential buyer should consider. A computer uses electrical power, but of a low voltage, similar to that used by electronic calculators and similar devices. This low voltage is supplied by a group of electronic components called a power supply. This has the same function as the group of components called an "AC adaptor" when used to supply power to devices which normally use batteries.

A power supply is sometimes included as a standard part of a computer, is sometimes an extra cost item, and is sometimes not even available from some vendors. Also, a large microcomputer system with many accessories needs a more powerful supply than a small system, and perhaps more power than can be supplied by a computer as sold. Check before you buy.

PROGRAMS

A program is not usually considered a part of a computer because it doesn't exist physically, just as an idea doesn't. Programs are often called software, while the physically existant computer and accessories are called hardware. Software is essential to the use of a computer, and some types of programs are commonly purchased with the computer or as accessories. The availability of software should be a key computer selection factor for most potential users.

The most important type of program generally furnished by a computer manufacturer is a language translator. This program translates between your English language and the computer's machine language so your computer can understand the programs of instructions you communicate to it. This translator program may be called an interpreter, an assembler, or a compiler.

The most common English-like language used to program home computers is called BASIC. This is one of several languages, called high level languages, that are easy for casual programmers to use. BASIC interpreters are available for many computers. For some computers, more than one BASIC interpreter is available. Different BASIC interpreters vary in the number and the power of the instruction words they recognize and in the accessories and capabilities they require of the computers on which they operate.

Lower cost computers have no language translator programs, so you must communicate with them in machine language. Although such communication is educational for the user who wishes to understand how a computer works in detail, it isn't in an easy language for a person to use.

The other major category of home computer programs includes those that give a computer the capability to do whatever its owner wants. Some sources of programs of this type include:

1. The manufacturer and vendor of your computer
2. Other vendors, including many at this Faire
3. Program libraries, which loan or give copies of programs, at little or no charge
4. Friends and computer club acquaintances
5. Your own ideas--this is the type of program you can write yourself.

Although some programs will be easy for you to write, others could take several months of hard work. A discussion of the many factors affecting program procurement decisions and the decisions of which programs you should write yourself is beyond the scope of this paper. A brief summary is that there are many ways you can get programs and many unexpected options and decision factors that may be involved. The availability of programs for doing specific jobs certainly can be a major computer selection decision factor.

USES OF HOME COMPUTERS

Having now covered the technical characteristics of computers, we are ready to consider some of the things they can do for us. Even though many people understand the immense potential computers have for helping in our homes, they still ask: What are home

computers used for today? This is a reasonable question, and probably the most common one for a home computing newcomer. The answer is: Mostly for fun.

Although computers have proven that they have many other potential uses, the reason relatively few people are using computers for serious purposes is that most people can't seem to quit playing with their computer. I see nothing wrong with that. Work, for many people, should be done at work. Those who can spend most of their time at home for enjoyment seem to have achieved an important measure of life's success.

The most common type of home computer owner today is the builder, a craftsman or craftswoman. For these many people, understanding, building, or otherwise developing computers is an end in itself. These computer hobbyists are similar to the painter or woodworker whose enjoyment comes from creating, rather than using or possessing.

Computers are the world's most pliable media--you can make more things of a computer than you can make of clay, paint, or wood. Creating with computers can be extremely challenging, and success extremely rewarding, because the potential for creativity, mental and physical, is almost infinite.

Other people enjoy using computers for playing games. When used for games, computers are often somewhat like TV games, today's hottest consumer item. Unlike today's common games, which are very limited and rather quickly become boring, the variety of computer games can be almost infinite.

People with home computers can keep getting new games, sometimes for less than \$1 each. Better yet, they can design and use games they create themselves. And these aren't just bouncing ball games. Home computer owners program and use their computers to play cards, space war, chess, and many other games. The really fun games are ones you never heard of because they can only be played with the help of the immense power of a computer.

But computers can be very practically useful, as proven by their use in, and great benefits to, almost all of commerce, industry, and medicine. Easily affordable home computers can be at least as powerful as many of those that are doing thousands of different jobs in business today.

Families use computers in home offices to keep records and to handle accounting chores. Some also plan stock and commodity futures buying and selling. Home businessmen use computers in many occupations.

Computers control heat and other environmental factors so an entire building, such as a home, is more comfortable and so energy is used more efficiently. Home security is another service, meeting an unfortunately increasing need, that computers are being developed to help supply.

Education is another of today's uses of home computers. As educational tools, computers are helping to teach subjects ranging from first grade addition to financial and engineering math, and a wide range of other subjects. Also, many people are using computers to teach themselves about computing, a field with considerable employment potential. In addition to vocational benefits, learning about computers is both enjoyable and practical because of the rapidly increasing role computers are coming to play in the everyday lives of each of us.

Many computer owners have talked of the great potential of computers for keeping track of kitchen supplies, planning menus and recipes, and for many similar household chores. For some reason, these uses seem to be like most household chores the American man is asked to do. They end up in a "tomorrow, dear" category, even though their long range benefit can include considerable work savings.

Returning to fun applications, computers are used to catalog and control the playing of musical recordings. Computers are also used to make music. There is even a magazine devoted to the topic of computer music.

Electronic hobbies are another home computer application. When electric train layouts are so complex that father and son together can't control them, computers keep the home railroad running smoothly. Some amateur radio hams make extensive use of computers in their hobby and have generally

been the pioneers in home computing.

The list of what computers can do, of what you can do with your computer, can go on and on. Aside from hopefully giving you some more reasons for getting involved in computing, the preceding examples of computer uses should further support the idea that a computer can do almost anything.

HOME COMPUTERS TODAY

But a computer in your home? Lots of people have computers in their homes already. As you go to some of the more advanced sessions today and tomorrow, and as you go around the exhibit hall, talk to some of the other people there. Many of these people have computers in their homes. They can tell you what it is like.

But keep in mind that these people are the pioneers. They didn't have to cut down logs with an axe to build their own cabin. But some of them have done as much pioneering in other ways. They have done the hard work. You can buy a prefabricated, ready built computer, ready to move in. However, there is plenty of wilderness left in computing. If you want to be a pioneer, there are many opportunities for you to go ahead into the unknown and develop techniques and products. And there are lots of people who will follow and much appreciate any real discoveries or progress you make.

This is one of the choices you can make as you look at the computers here today, and as you decide how you will become involved in computing. You can buy the equivalent of 40 acres of wilderness, or a prefab in a subdivision, or many things in between. You can totally design and build your own computer. Or you can get one that you can plug in and have start immediately to work for you. There are many available options between these extremes. Your home computer can be what you want it to be.

ELEMENTARY COMPUTERS

You can get a very basic type of computer. This will be relatively inexpensive, costing between \$100 and \$250, generally less power supply. It won't have a fancy cover, and probably will have no cover at all. All the parts of this type of computer may be on one piece of plastic-like material called a printed circuit board, or just "board" or "card".

You will have to do lots of detailed work to program and use such a computer, especially because programming generally must be done in machine language. The power of this type of computer is limited--you can't easily add many accessories, nor can you easily connect such a computer to other devices to control them.

This type of computer is a good training device, an educational tool. The lack of fancy accessories means that you work directly with your computer, with little interference. Therefore, you can learn, by doing, much about the details of how a computer works.

EASY-TO-USE COMPUTERS

Some people don't want to work closely with a computer. They would rather have their computer follow their general instructions. Such people aren't especially interested in knowing the details of how a computer works. They would rather learn how to use a computer to solve a particular type of problem, control a particular process, or help them in a field unrelated to computing.

Computers designed for this type of user are available. One key characteristic of this type of computer is that you program it by using a high level language. As we discussed earlier, a high level language is easy for people to use.

These computers are generally built on several printed circuit boards which plug into guides and connectors mounted on the computer frame. Computer capabilities are changed by plugging new boards into appropriate spaces, as long as there is room available. For some computers of this type, especially "Altair bus compatible" or "S-100 bus" computers, many accessories are available that can be added. For other types of computers, only a few accessories are available. Consider what you may need and want.

Prices of computers in this category generally

start at about \$500. A system with enough power and accessories to use a high level language may cost \$1000. The price of a full system, such as might be used for a business application, can easily reach \$5000.

KITS?

Another primary choice when selecting your home computer is whether to purchase a kit of parts, or an already assembled computer. A kit, like the traditional model airplane, requires your work and time to put it together. A computer kit often takes from 50 to 200 hours to assemble; accessory kits generally require less time. Time requirements vary depending on your skill, the ease of assembly of the kit, and your luck in (not) encountering problems which can take many hours to diagnose and repair.

The major benefit of a kit is that it saves money. An assembled computing product generally costs from 30% to 50% more than a kit. Another benefit of a kit is the experience that assembly gives. However, the person who has so little experience that he would benefit probably would encounter considerable problem in assembling a working computer.

Most computing kits require skill and knowledge in electronics work, sometimes a considerable amount. The decision whether to build a computer kit is not one to be made lightly. Anyone, after he has enough skill, equipment, time, and expert help available, can build a kit. The amount of these resources required for assembly and successful use varies widely among kits. The quality of instructions provided also varies considerably. These are all factors to consider when selecting your computer.

HELP!

Although good instructions and the availability of expert help are essential to the novice kit builder, these aids are important to every computer user. Computers today are too complex to be understood completely by most people. Because computers have so much potential, appropriate understanding is very important for using a computer well. Because skill requirements vary widely among types of computers, a solution to the need for skill is to choose a computer which needs the amount and types of skill you have or plan to have.

A special resource to consider when evaluating sources of computer skills is the dealer from whom you buy your computer. This is the person from whom you will also probably get the several types of support you will need to use your computer and to keep it operational. A dealer who is knowledgeable about the equipment he sells, and is willing to share this knowledge, can be a great help in using a computer. So the decision of whether to buy from a dealer, and the selection of your dealer are both important home computing decisions. A novice may be well advised to use this Faire as a terrific way to learn about home computing and to compare computing products so he can intelligently make the actual purchase of a computer through his home town computer dealer.

CONCLUSION

After this quick overview, you probably are more confused than when you started because you have learned a little and can better see how much there is to learn. At least you have some understanding of the fact that many of the things at this Faire are new, even to the most knowledgeable and experienced persons.

The knowledge you now have is greater than many of the other people at this Faire. So go ahead to the other talks, and visit with the exhibiting vendors, confident that you can benefit from these discussions. Most of all, have fun as you plan and learn about your new helper, your own home computer.

BIOGRAPHY

James S. White is the author of Your Home Computer, an introduction to personal computing. As an active member of 3 amateur computing organizations, he has worked with a variety of newcomers to microcomputing. He is also Data and Systems Manager for Durant Digital Instruments, a manufacturer of counting and control systems and components.

A TYRO LOOKS BACK

Fred Waters
1601 Provincetown Drive
San Jose CA 95129

Some of you are curious, interested, and perhaps actively investigating this idea of microcomputers for fun. About a year ago I was where you are today. I had just had my eyes opened to this new hobby, and was in a turmoil about which way to go. Are you asking yourself, "Is this hobby for me?" Or, "I think I want to get into this, but which way do I go?" Or, "I've made a start, but what do I do with the limited hardware I've got?" Maybe my own experience and points of view will help guide you, or give you some ideas on choices and directions.

I speak as a tyro. There is no part of this hobby in which I am not a beginner, except perhaps for a prior interest in mathematical games and recreations. I speak only to those who follow me. As for the men who has a more advanced system, with lots of memory and peripheral devices, with prowess in destroying Klingons using his favorite version of Star Trek, and with 87 of his wife's favorite recipes stored for quick recall--he's a man I want to listen to, not tell my tale to.

After finding out by accident what a microcomputer was, and that a rudimentary system could be acquired for not much more than a good hi-fi system, I started investigating. This meant attending club meetings, visiting shops, finding and reading the right library books, subscribing to magazines, writing for commercial brochures, and asking questions of anyone who would stand still long enough to give answers.

Then I picked a system, bought it, assembled it, checked it out, and started doing things with it. I made some mistakes that you can avoid. It was necessary to keep on studying, reading, learning. I started out with the simplest possible uses for my system. Then variations became evident. As one progresses in the various skills, and adds peripheral equipment to the system, one finds more and more interesting pursuits in an ever-widening range of activities. And you glimpse more to come. Learning and playing with these machines is like reaching and exploring an unending series of plateaus, each one more or less fascinating, and each one pointing to more beyond.

Where are you in this hobby? If you are looking at something entirely new, or already launched into the first few stages, you may get some useful ideas from what I did when I was there. I hope my own enthusiasm has ignited a spark, or helped fan the existing spark into a growing flame of interest and excitement.

Fred B Waters, Jr
1601 Provincetown
San Jose, CA 95129

U S Army Corps of Engineers, retired
M.S. in C.E., Cal Tech
Former assistant professor of engineering,
U S Military Academy
Former instructor in public speaking,
International Speakers Bureau,
Santa Clara
Member of Mensa
Member of Speakers Bureau, San Jose
Bicentennial Commission
Occasional lecturer on mathematical games
and recreations: "Fun With Math"

THE SHIRT POCKET COMPUTER

Richard J. Nelson
2541 W. Camden Place
Santa Ana CA 92704

INTRODUCTION

The shirt pocket computer exists today in the form of the personal programmable calculator. No matter how you may define a computer, machines like the Hewlett-Packard HP-67 and HP-97, and the Texas Instruments SR-52 probably qualify. The small gap between the personal programmable calculator, PPC, and the microcomputer will be bridged completely this year with the introduction of the National NS-7100 in late spring, and the still unknown model number from Texas Instruments at about the same time. There are nearly 300,000 PPC's and less than 30,000 micros in use today. The micros get most of the publicity, while the PPC's quietly serve their owners as a portable, extremely powerful, and economical computational tool.

This paper will provide a general review of the PPC, its characteristics and capabilities, and what can be expected in the near future. In addition, this paper will cover sources of information on software, hardware modification, accessories, and users groups.

COMPUTER OR CALCULATOR?

The basic architecture of the top-of-the-line PPC's on the market today makes them calculators primarily because they are designed for numerical computation oriented to a base-ten world. There are many characteristics that are attributed to a computer which differentiates it from a calculator. A few of these are: a) ability to modify its own program, b) ability to handle alphanumeric 'strings', c) high speed operation, d) multiple input/output ports, e) large mass memory in the form of 'core', solid state RAM, or disc, f) multi-mode operation, batch, time-sharing, etc.

Certainly the PPC cannot measure up to the performance of the above listed computer capabilities, but when each one is broken down into basic capability, it is astounding how close the PPC comes in scaled-down capability. The upcoming NS-7100, for example, will be able to talk to many different devices, but it is not possible to have them all in one pocket. The PPC, with its indirect addressing, multi-logic compare instructions, multi-flags, full editing, and external memory makes it as close to a computer as you can get and not call it a computer. The next generation of machines available this summer will add features that bridges the gap completely. There are sound marketing reasons for calling these microprocessor-based systems calculators, and as long as they are primarily intended for numeric computation, they are best called calculators.

WHAT DO I NEED?

If you are an engineer who is interested in computers because you must keep up with technology, then the microcomputer is what you need. You will buy and build one, and will spend considerable time and money getting it to do something practical. If you are interested in a machine to control a process or a model railroad layout, you should consider a microcomputer. If you are a businessman who needs lots of mass storage to handle inventory or data acquisition, you need a microcomputer.

If, however, you are a problem solver, or want to learn programming, want to have fun, and not spend \$2,000, \$3,000, and up to \$10,000 for a real, up and running system, the personal programmable calculator with external memory may be just what you need. If you want to: play Startrek, The Game of Life, Bagels, Hexapawn, or a truly Cybernetic game; solve five (or more) simultaneous equations; curve fit data to just about any curve up to order 4 or more; design circuits, bridges, etc., on an interactive basis; do extended numerical operations to 50 digit accuracy or greater; perform card tricks; convert your calculator into a clock or timer; or just plain balance the family checkbook, the PPC is made to order. Besides, you can have the very best for less than \$400. The software support for the PPC's is far more extensive than any microcomputer. It is easier to determine if a program has been written to solve a specific problem on a PPC than it is on most microcomputers. Numerous libraries are maintained by the manufacturers and users clubs throughout the world.

CALCULATORS AVAILABLE TODAY

The personal programmable calculators are those machines that have an extensive instruction set that includes full scientific functions, logic compares, flags, full branching, and program editing. The machines that qualify also have some form of external program and/or data storage. They are the SR-52, from Texas Instruments, and the HP-67 and HP-97, from Hewlett-Packard. Table 1 is a brief summary of the major features of these machines.

MODEL	SR-52	HP-67	HP-97
Size	Handheld	Shirtpocket	Desktop
Cost	\$300	\$450	\$750
Printer	Separate(PC-100)	None	Yes
Program Steps	224	224	224
Data Registers	22	26	26
Program Storage	Magnetic Card	Magnetic Card	Magnetic Card
Merged Codes	Partial	Fully Merged	Fully Merged

TABLE 1. MAJOR FEATURES OF PPCs AVAILABLE TODAY

The ease with which these machines are programmed, and the personal nature of a battery-operated, go anywhere, calculator, has stimulated considerable programming effort from their owners. The ability to provide immediate answers to "what if" questions, and the dedicated functions of the keys, make programming an easily-learned skill. Studies have shown that students who have had preliminary programming experience on a PPC do much better in their Fortran, Basic, or APL classes. The extensive programming techniques that PPC users have developed allows these memory limited and slow machines to perform astounding tasks. Techniques that use a keyboard overlay even allow alphabetic inputs and outputs. Today, increasing amounts of literature, rapidly growing users clubs, and growing software support from the manufacturers is encouraging. Several years ago the situation was not as bright.

THE HP-65 AND SOME HISTORY

On January 17, 1974, Hewlett-Packard announced the HP-65 - the world's first pocket 'computer'. The \$795 technological wonder had external memory in the form of a magnetic card. One-hundred program steps could be programmed and recorded for future use. The microminiature card reader made convenient the reading of user-written or factory programs. The 100-step limitation of only partially merged program steps proved to be one of the machines' greatest assets. Intuitively the HP-65 users knew that if the right approach was used, the program could be made to run in 100 steps. The wonder of the HP-65 created a group of dedicated users who formed the first Programmable Calculator Users Club in June 1974 -- the HP-65 Users Club. The small size and very personal nature of this machine set the stage for today's machines. The dedication of HP-65 users resulted in an astounding body of knowledge of program techniques, hardware modifications, and applications. During the nearly three years of product life, the HP-65 was never reduced in price, despite fierce competition by the SR-52. Only in the closing days of clearance sales did the price for a new HP-65 drop. Now replaced by the HP-67, the HP-65 still offers the user a capability that was never fully explored. The full power of the HP-65 may never be known, for the excitement of a newer, better, and many times more powerful machine diluted the investigative efforts that were once exclusively the domain of the HP-65. One of the most interesting aspects of the HP-65, and later the SR-52, was the discoveries of machine behavior that was not intentionally designed into the calculator. These anomalies at first terrified the manufacturers. They were so engrossed with the business of making and selling calculators that they couldn't take time to answer a lot of 'foolish' questions by the calculator enthusiast who was usually associated with one of the Users Clubs. The unusual behavior resulted in what has become known to calculator users as 'unsupported features'

UNSUPPORTED FEATURES

Reference 1 lists unsupported features of the HP-65 and SR-52. The Users Club Newsletters, references 2-5, document all known

unsupported features. The following examples will illustrate "unsupported features".

HP-65 Lampman Split Logic - The microcoding of the HP-65 had a few undefined conditions. Dean Lampman, Cincinnati, Ohio, discovered that certain 2-part instructions could be separated by any number of logic compares, or stack manipulations. Using this knowledge reduces program steps. For example, a single STORE + n instruction can apply to five or more registers. Using the Lampman split logic saves eight steps, which on a 100-step machine is significant.

"Non-normalized" numbers - may be generated in registers 8 and 9 and utilized to provide decrementing counters (timers), generate pseudo random numbers, and most useful, a programmed pause, not included as a design feature.

The term 'unsupported' is very descriptive because Hewlett-Packard never publicized the advantages of the Lampman split logic, and users outside the small world of the HP-65 Users Club never knew of the technique. Non-normalized number generation and applications were eventually published by Hewlett-Packard and probably represents the turning point in the change of attitude of Hewlett-Packard in viewing the super enthusiast. This subject has been discussed in a paper given at WESCON (see Reference 6).

THE HP-65 SHOWED THE WAY

The accomplishments of HP-65 users have been previously summarized in Reference 7. A brief list will be included here because these application experiences served as a training ground for the increasingly sophisticated calculator users to quickly advance to even greater heights on the next machine, the HP-67.

Games, an ever popular 'computer' pastime were developed to a high level on the HP-65; 21, NIMB, TIC-TAC-TOE, FOOTBALL, and BAGELS, are only a few. Two outstanding, truly cybernetic games -- Hexapawn and Cibernimb -- were also written for the HP-65.

Keyboard Overlays - Especially the design of John Rausch, Dayton, Ohio, provided a convenient means for inputting alphabetic data. The use of the Rausch overlay facilitated word games and convenient encoding and decoding of english text

Magnetic Cards Analyzed - David Kemper, Northridge, California, and others, examined the recorded instruction bit patterns of HP-65 program cards. Knowledge of these codes caused a search for four missing (bit patterns possible, but not used) codes which were eventually isolated, and used to perform unusual, unsupported features, functions such as partitioning memory so labels could be duplicated. David's famous table of codes proved a useful tool, not only in explaining the behavior of the calculator, but also in providing insight into the next generation machine.

Unusual Programs - Applications of the HP-65 went beyond the owners manual. A program which, when run with the calculator near an AM radio, could transmit the number in the display with an audio code. Programs that provide a 45-digit by N digit product, infinite division, or compute the factorial of 10^6 fully exploit the HP-65's firmware and programming capability.

Display control was always an area of "research" by HP-65 owners. Fun displays were produced by unusual loops, or "special" instructions in the endless search to expand on the only human output interface the machine provided - the display. Few practical applications resulted, but the experience gained provided practical tools to use to analyze the HP-67. Pseudo alphanumeric displays are possible on the HP-67 as a direct result of work done on the HP-65.

Adding a printer was always the dream of the HP-65 owner. Heinrich Schnept, Köln, West Germany, built the first printer for the HP-65. His concept of plugging the shirtpocket PPC into a "home base" chassis which contains a printer served as a concept for manufacturers to apply in future calculator "systems".

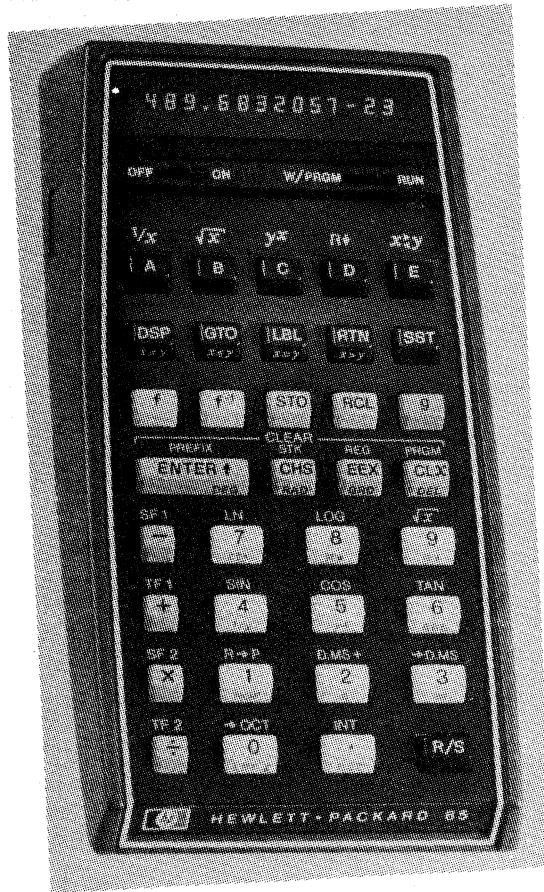


Figure 1. HP-65 "Pocket Computer" was announced by Hewlett-Packard in January 1974.

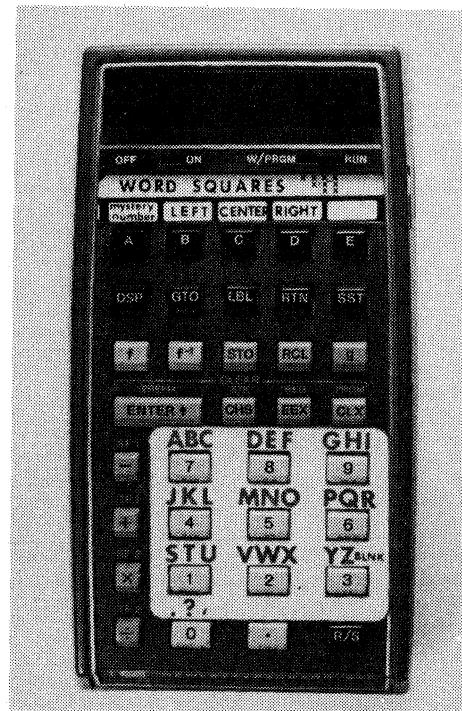


Figure 2. Keyboard Overlay for the HP-65 and HP-76 allows alphabetic inputs and outputs for word games; message encoding.

The HP-65 was the first PPC and was the first generation of machines which followed. The three major machines being offered today represent 1½ generations of technological achievement.

TODAY'S' PPC's

Following the HP-65 came the Texas Instruments SR-52 and the HP-67/97. The HP-97 and HP-67 are functionally identical and can run each other's programs. The HP-97 has a battery-operated printer and is a small desktop machine.

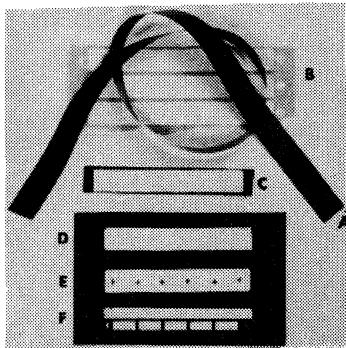


Figure 3. Tape and label method of making magnetic program cards. Stress relieved computer tape (A) is 'labeled' with file folder label (B) as shown at (C). Thickness is 6-7 mils. (D) shows card trimmed to dimensions. Rubber stamped key locations (E). Standard Hewlett-Packard card at bottom (F).

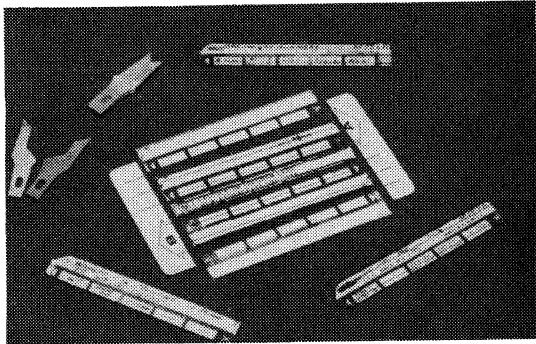


Figure 4. Cutting desk top calculator magnetic cards is also a popular method of making magnetic cards for calculators. This photo by D. Machen shows how four cards are cut from one.

The introduction of the SR-52 added some new features and functions to the PPC. In addition to the designed features the SR-52 had an unusually large number of unsupported features.

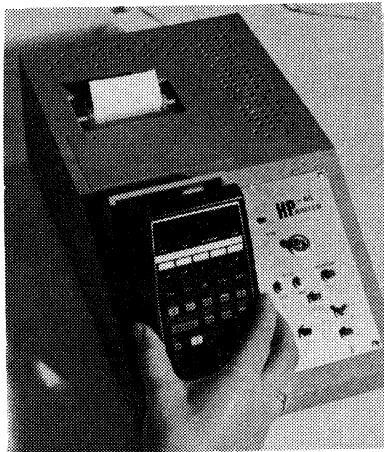


Figure 5. HP-65 printer designed and built by Heinrich Schneppf.

The many unsupported features were discovered in rapid fire fashion by experienced PPC users in late 1975 and early 1976. T.I. has now recognized nearly all of these unsupported features and has now made them supported features. Competition in the marketplace, and realizing that the logic of an introduced PPC would cause software chaos if changed, are probable

reasons for T.I. recognizing user-discovered features.

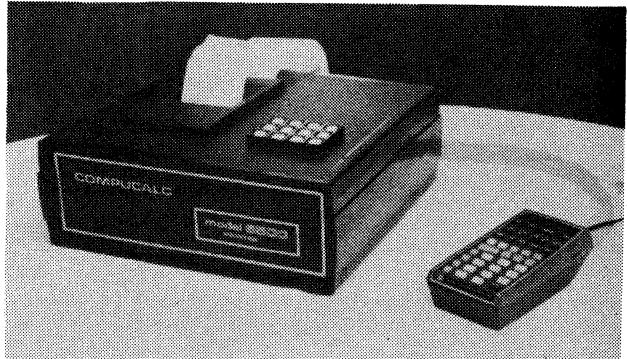


Figure 6. Printer manufactured by Compucalc for HP-65. Modification of the calculator is not required, simply replace the back with one supplied with the printer. Interfaces for other calculators are on the drawing board. Reference 19.

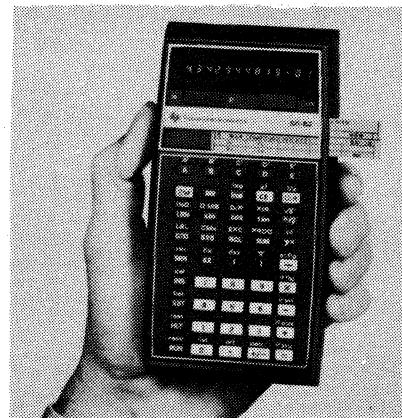


Figure 7. SR-52 announced by Texas Instruments in September 1975 added indirect addressing to the Personal Programmable Calculator.

Many of the contributions that the SR-52 made, such as indirect addressing, the ability to store data on magnetic cards (unsupported) and the ability to read a program card under program control (unsupported) are designed and supported features of the HP-67/97. The SR-52 also pioneered the concept of adding a printer to a handheld calculator. The added printout capability of the quiet PC-100 was a delight to users who soon grew tired of handwriting 224-step programs. The SR-52 was T.I.'s first effort in this product area and the company has come a long way in realizing that, as the machines get more sophisticated and have greater capability, factory software support is a vital aspect of calculator sales. Consult manufacturers literature for full product details.

FACTORY SUPPORT

The consumer who buys a PPC doesn't always realize that he may get far more from his machine if he doesn't have to write all his own programs. When Hewlett-Packard announced the HP-65 they also announced a contributors library and a free owners newsletter to keep the user updated on manufacturer and user activities. In addition, groups of programs were made available in the form of Program PACs. Fourteen PACs were written by HP for the HP-65 which provides the application expertise of professional programmers. Many PPC users do not have the time or ability to write their own programs. The manuals and magnetic cards that made up the PACs provided practical information in addition to instructions and program listings.

T.I. continued the practice of offering program PACs, and a users program exchange with newsletter, in support of the SR-52. Unfortunately, major software support for the SR-52 did not come until a year after the machine was available. The HP-67/97, like the HP-65, was software supported from the date of announcement. Software support is only one factor the potential PPC customer must consider. The prospective computer customer who considers the PPC as an addition, or alternative, to a microcomputer should also consider convenience, quality, cost, service,

and logic operating system.

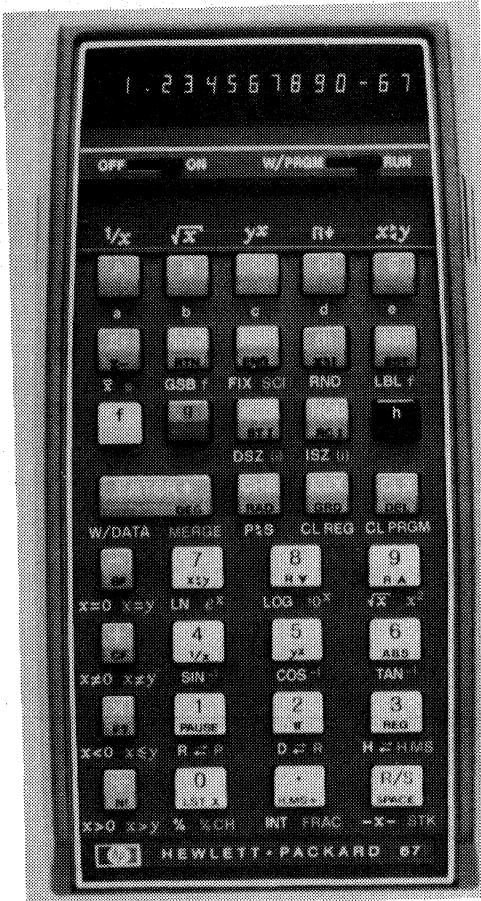


Figure 8. HP-67 introduced by Hewlett-Packard in July 1976.

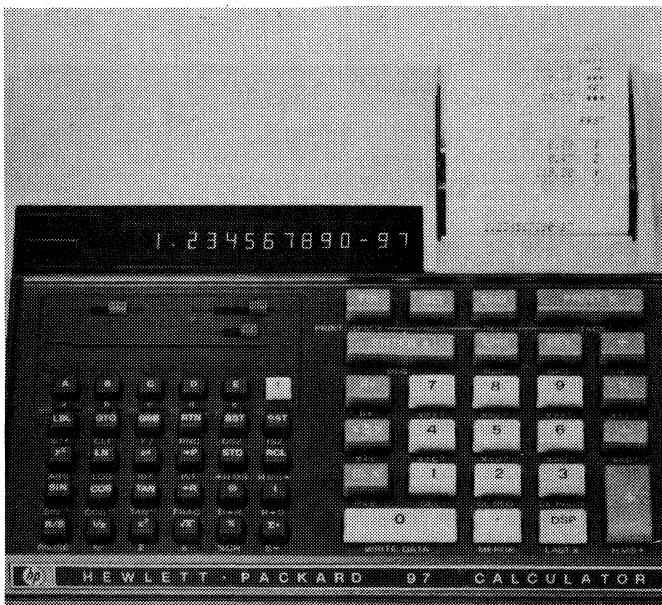


Figure 9. HP-97 Introduced by Hewlett-Packard in July 1976. Programs written for the HP-97 will run on the HP-67 and vice-versa. This arrangement allows nearly full power in your pocket with the ability to print programs and listings on a machine at the "home base".

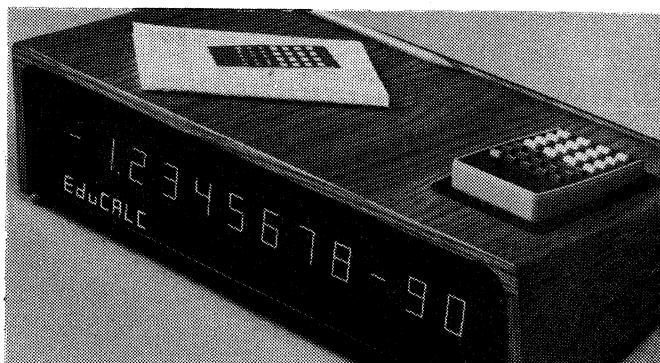


Figure 10. Large display for class room use is manufactured by Educalc. The calculator is hard wired to the display chassis. Several models of calculators are available. The display can be seen from 60 feet. See reference 20.

LOGIC SYSTEMS

Of the five considerations listed above, the first and last ones, convenience and logic system, are the most controversial. Comparing the products of the two companies, T.I. and H-P, is like comparing Fords and Cadillacs. Both provide transportation, each with its own advantages. More fundamental than purchase price is the difference between T.I.'s Algebraic Operating Systems (AOS) and H-P's Reverse Polish Notation (RPN). Discussing this subject is often like arguing that brunettes are prettier than blondes. Objective conclusions as to the best system are difficult. Subjective factors will usually tip the scales one direction or the other.

Keystroke counting as a comparison of the two systems is not valid. Both systems usually require identical numbers of keystrokes to instruct the machine what to do and in what sequence. Both systems take getting used to. A strong argument for AOS is that it most closely resembles the conventional algebraic representations learned in school and practiced by the best minds of the last 2,000 years. AOS uses parentheses to retain numbers for operations that must be performed after other operations. There is no implied operation of multiplication if a parenthesis follows a number. RPN does not use parentheses, but rather uses the concept of a four memory stack, entering numbers into the stack with an enter key, and pressing the operation key afterwards. Using RPN requires a little practice, but owners who use it swear by it. The advantages of the HP system is not RPN in itself, but rather the combination of the stack, and the last X register which always saves the last number entry after an operation is performed. This allows recovery of operator errors which is very convenient in any problem situation. To investigate the relative merits of each system would require more space than allowed here, but the topic is important enough that the new user should try both systems before deciding. The saying "different strokes for different folks" applies. In addition to normal keyboard use of the "operating system", the differences the logic system has on programming structure should also be considered. If you are a complete novice in this area, seek out friends and associates who have machines to give you their viewpoints.

PPC APPLICATIONS

The typical applications that follow can be implemented by any of the three calculators described above. The convenience, ease, and practicability of one machine over the other may make it a better choice in certain applications. The SR-52 was not designed to store data on cards, but it can be done. The purpose of describing these applications is not to provide explicit detail or analysis, but rather to show how the PPC may supplement or even replace a microcomputer in many practical applications.

The small, stick of gum-sized, magnetic cards used with today's machines allow the storage of program instructions and/or data. The machine has a limited amount of internal memory for programs and registers for data. Using appropriate programming techniques magnetic cards can be cascaded to extend program memory and/or data registers. The SR-52 can be programmed to use data registers for program memory and vice versa if the proper techniques are applied -- unsupported.

Accounting. A small business may keep the salary, taxes, insurance, etc., data on each employee on a magnetic card. The

card is read, the weekly hours keyed, and the program executed. All the calculations required are done in seconds and the new yearly cumulative totals kept in memory. The data is entered on the pay records and the card is read once again to record the new cumulative totals.

Taxes. Income tax consultants have been able to develop algorithms to put complete tax tables on a magnetic card. Running their sophisticated programs saves considerable time and, for their business, adds to their income. Naturally the circulation of these programs is limited.

Conversions. Currency, pricing, and engineering conversions can be programmed for convenient daily use that obsoletes tables because any unit can be chain converted to any other. 24 or more conversions can be placed on one card. The limitation is primarily the writing space on the card, not the number of conversions.

Eliminate Tables. Using clever curve fitting and other number generating algorithms, thousands of pages of tables may be reduced to a few cards. The Star Almanac, all scientific function tables, etc., are practical everyday uses of the PPC.

Design Interactively. Iterative design equations can be programmed to resolve for desired parameters when one or more parameters are changed. This interactive design capability becomes a powerful engineering aid; especially when printed results can be added to the engineers' log.

Reduce Production Errors. The use of the PPC on the production line, or nurses' station in a hospital can save considerable time by allowing complex programs to be run by unskilled personnel. Programs properly written can check and look for errors in situations involving large digit quantities. One quartz crystal manufacturer justified \$800 worth of calculators on his line the first week because a self-checking program caught a digit reversal error made by an order clerk. Having standard, self-checking, and easily entered verification problems available for verifying program performance adds to the confidence which non-skilled users have in the machine.

Finance Calculations. The PPC may be converted into a financial wizard that will rival most computers. The high numerical accuracy built into these machines allows individuals to have an independent check on interest calculations. In fact, the program may also include the legal rounding rules that banks and savings and loan companies use. In 15 minutes all the truth in lending statements on one page of newspaper ads can be verified. Often there are two or three errors on one page. Want to check your 239th house payment to determine how much of it is interest? This type of problem can be done in less than 30 seconds including the time to find and load the magnetic program card and key in the data.

Cost Estimating. This type of problem is a natural for a small business. Usually each business has its own guidelines and

rule of thumb. Consider a fabricated part produced from sheet metal. The material, size, and number of holes are keyed in as the customer gives the data. A quoted price is given in seconds. The program knows the size of the material and determines the optimum number of pieces. Fabrication costs, overhead, and profit are calculated without human error. Complex products that require 20, 30, or more inputs can be handled easily.

Research. Computer time is expensive, and usually a microcomputer is not on your own desk. A simple program can be keyed in to search for answers while you go on with your work. The slow PPC may take hours or days, but it can find answers that would take a big machine, restrictive availability, and a professional programmer to solve. If the program is still running when it's time to go home, simply unplug it, slip into your pocket, and let it continue. Plug it in again when you reach home and get the first solution while watching your favorite TV show. Start a new problem and have another solution at breakfast.

Games. Games are just as popular on PPC's as on larger computers. Startrek, Life, Golf, Football, Baseball, and most of the classics can be played on the PPC. Both T.I. and H-P even offer a Games PAC. A recent BASIC listing of the game Hexapawn appeared in a popular computing magazine. The 2/3 column listing was the equivalent of one magnetic card program. ON THE HP-65! True, machine-learning games have been programmed on the PPC. When the "computer" is available at the personal convenience of the owner, his creative skills are enhanced because the machine is available whenever and wherever needed.

The above applications are not an exhaustive listing, but only serve to illustrate the high level of programming achievement that exists in the software for PPCs. The small size of the machine and the external storage media makes the PPC a problem solver, a teacher, and an entertainer. Literature on PPC applications has increased dramatically in the last year. The technical literature and Users Clubs provide the bulk of high quality software and applications information.

SOURCES OF INFORMATION

The basic body of knowledge suitable for PPC users has been in the technical literature for many years. Only recently, however, has mathematical concepts, algorithmic techniques, and iterative and recursive techniques been available in a form directly applicable to PPC use. Textbooks, magazines, users club newsletters, and engineering papers have been contributing to the general body of knowledge. The references at the end of this paper list many of the major works related to the PPC. The list is more complete in the Sources from Engineering and Science fields and lacking in references from the Academic community. One of the richest sources for applications information, programs, and programming techniques is the monthly users club newsletter.

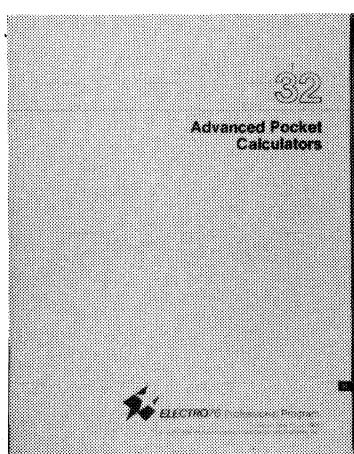


Fig. 11 Ref. 1 8½x11". 15 pp, 3 papers "bound".

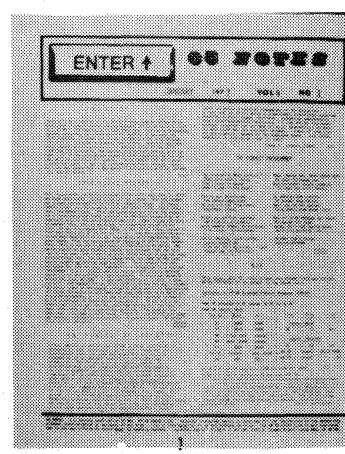


Fig. 12 Ref. 2 8½x11", 2000 word per page format, typical issue 12-14 pp.

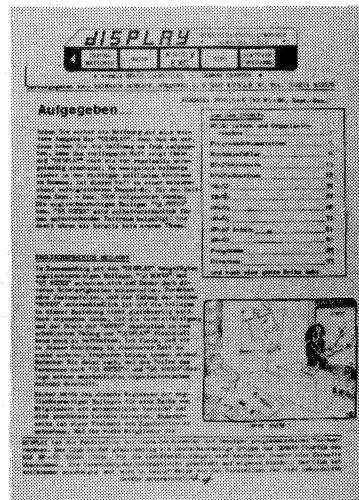


Fig. 13 Ref. 3 8½x11½", In German, very difficult to translate.

USERS CLUBS

The Council of Personal Programmable Calculator Users Organizations lists three U.S. and one German club as members. Several smaller groups at major U.S. corporations are getting large enough to qualify for membership. Approximately 1% of the total PPC user population belongs to a Users Club. One primary objective of these groups is to produce the highest level of programming techniques and applications information possible. As a result, these super calculator enthusiasts enjoy a knowledge of the state of the programming art that leads the technical literature by nine months to a year. Almost without exception, programs or programming techniques, especially machine anomalies and unsupported features, that are published today in national magazines have appeared a year earlier in a Users Club newsletter.

These clubs operate along similar lines. They are all non-profit, independent, volunteer groups who have found that their PPC forms a unique bond to share technical information. Club activities include publishing monthly newsletters, holding meetings, making group purchases, sharing programs, and maintaining a program data base, and even maintaining calculator hot lines for the latest bulletins on what's happening in the calculator world. In terms of free software alone, membership in one of the clubs makes the membership contribution a bargain. Membership is world-wide and the machines the clubs support range from a single manufacturer to all manufacturers. There are no restrictions in membership except an interest in PPCs and a willingness to contribute to the general goals of the group. Membership in these clubs include users who may be Presidential advisors, Vice-Presidents of Fortune, 500 com-

panies, scientists, engineers, or college and high school students. The growth, history, character, and activities of these clubs is a phenomenon as unique as the high technology PPC itself.

BRIDGING THE GAP - THE FUTURE

As previously mentioned, two new PPCs are expected by early summer. The advanced capability of these machines will qualify them to represent the second generation of PPCs. They will have I/O ports with handshaking capability to interface with the rest of the computer world. Their internal program memory will be 20 times that of the present machines on the market. The NS-7100, for example, will have a solid state plugin Library that has a capacity to hold 4,096 program steps, each of which may consist of up to three keystrokes. The Library cartridge contains the equivalent of 12,288 keystrokes! All memory of the second generation machines is non-volatile. Execution speed is not much faster than today's machines and a significant increase in speed is not expected until Silicone on Sapphire (SOS) technology is implemented for the integrated circuits. Data registers and program steps will approximately double. Alphanumeric displays are not expected on the handheld machines.

The details of the known machines coming from National Semiconductor and Texas Instruments in early summer are unknown. Enough is known about them to see that they will bridge the gap between the calculator and the computer because of the literally infinite capability that exists for "talking" to computers, printers, modems, CRTs, TV sets, and each other through their I/O port.

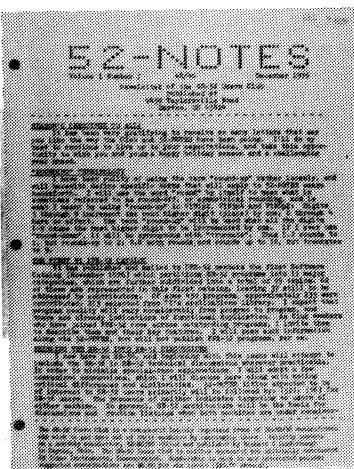


Fig. 14 Ref. 4 8½x11", 6 pages per issue, primarily advanced topics.

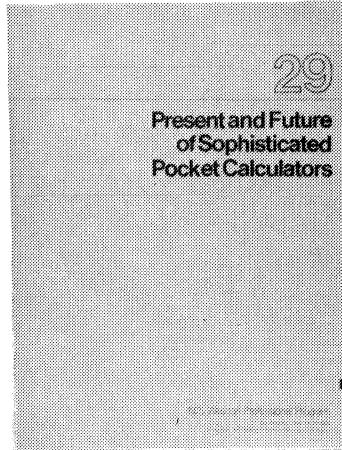


Fig. 15 Ref. 7 8½x11", 24 pages, 4 papers "bound".

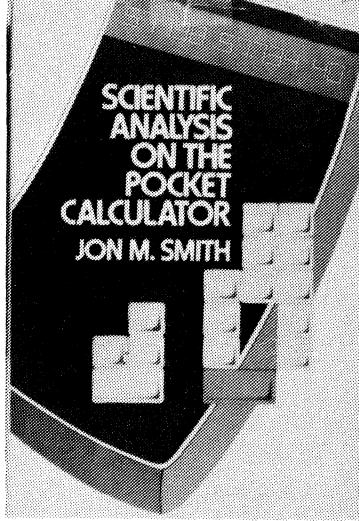


Fig. 16 Ref. 8 7x9", 380 pages hard bound.

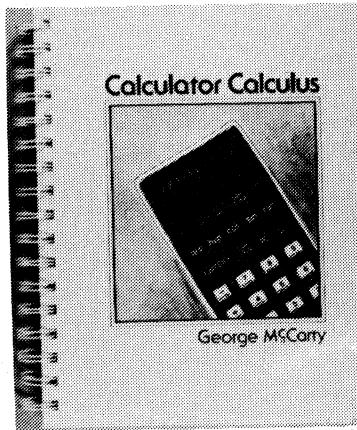


Fig. 17 Ref. 9 7x9", 254 pages spiral bound.

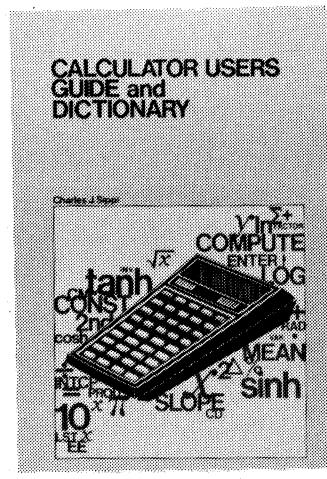


Fig. 18 Ref. 10 6½x10", 427 pages soft bound.

PROGRAMMABLE CALCULATORS

Charles J. Sippl

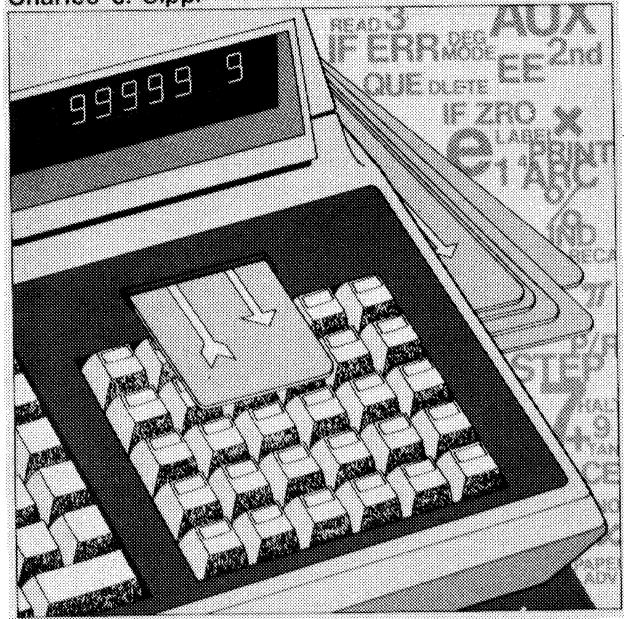


Fig. 19 Ref. 11 6½x10", soft bound, in press, available early summer.

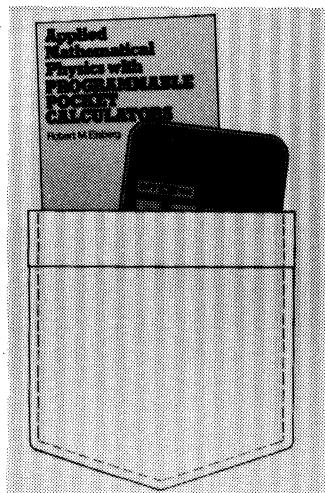


Fig. 20 Ref. 13 6½x9½", 176 pages soft bound.

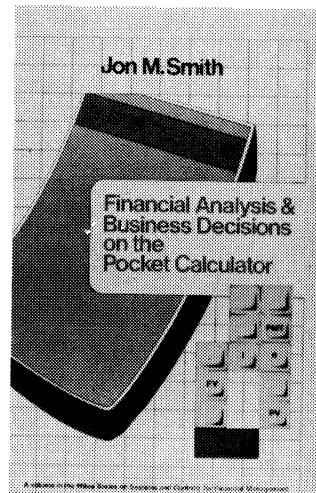


Fig. 21 Ref. 14 6½x9¾", 317 pages hard bound.

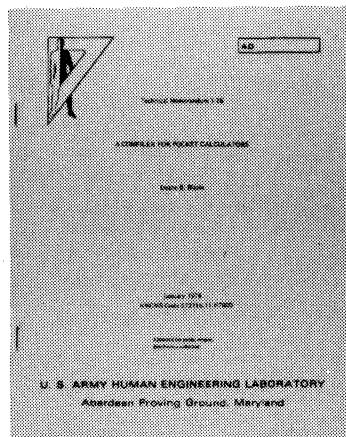


Fig. 22 Ref. 15 8x10½", 75 pages soft bound.

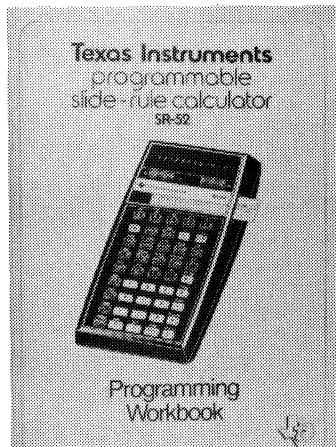


Fig. 23. Programming Workbook makes user discovered features "supported".



Fig. 24 Ref. 17 Free to all HP-65/97 owners who send in their registration card to get on mailing list.

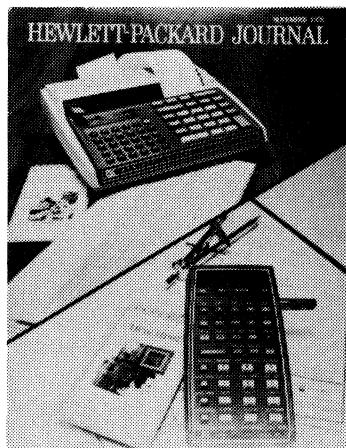


Fig. 25 Ref. 18 Only selected issues of interest to PPC users.

SUMMARY

The personal programmable calculator (PPC) with external memory is indeed a shirtpocket computer. The new machines to be announced in early summer 1977, with their full I/O capability, bridges the gap between calculator and computer. The term "personal computing" has a special meaning to the PPC user. "Personal" means "on ones person", available any time, anywhere. The applications and programming technique state of the art has kept up with the hardware advances because of the activities of the manufacturers software support and the Users Clubs. The PPC offers the computer user a powerful, low cost, accessory and, in applications where numerical solutions satisfy the major needs, may replace the microcomputer completely. The PPC of today is not just a calculator; inside it is a microprocessor with control ROMs that give it a calculator personality. The shirtpocket computer is here to stay.

ACKNOWLEDGMENTS

I would like to thank the members of all the PPC USERS CLUBS who, through their thousands of letters and phone calls over the past 2½ years have provided the material used to prepare this overview. I would also like to thank Jim Warren for letting "the other computing users" have a chance to share the calculator viewpoint. Compucalc, Educalc, Hewlett-Packard, Matrix, and Texas Instruments provided photos of their products.

REFERENCES

1. Electro '76 Professional Program, "Advanced Pocket Calculators", 32/4 May 11, 1976.
2. "65 NOTES", published monthly by the HP-65 USERS CLUB, 2541 W. Camden Place, Santa Ana, California 92704.
3. Display, published bi-monthly by the Mikro-(Taschen)-Computer Anwender-Club, Buschenweg 24, D5000 Köln 40, West Germany.
4. 52 Notes, published monthly by the SR-52 Users Club, 9459 Taylorsville Road, Dayton, Ohio 45424.
5. Recreational Programmers' Notes, published monthly by the New Zealand RPN Club, P.O.Box 21067, Edgeware, New Zealand.
6. WESCON Professional Program, "Pocket Calculator Update", 7/3 September 14, 1976.
7. WESCON Professional Program, "Present and Future of Sophisticated Pocket Calculators", 29/4 September 19, '75.
8. Smith, Jon M., "Scientific Analysis on the Pocket Calculator", New York, John Wiley & Sons, 1975.
9. McCarty, George, "Calculator Calculus", California, Page-Ficklin, 1975.
10. Sippel, Charles J., "Calculator Users Guide and Dictionary", Champaign, Illinois, Matrix Publishers, 1975.
11. Sippel, Charles J., "Programmable Calculators", Champaign, Illinois, Matrix Publishers, 1977.
12. Fowler, Robert W., "A Collection of HP-25 Routines", 12104 Arbie Road, Silver Spring, Maryland 20904, 1976.
13. Eisberg, Robert M., "Applied Mathematical Physics with Programmable Pocket Calculators", McGraw-Hill, 1976.
14. Smith, Jon M., "Financial Analysis and Business Decisions on the Pocket Calculator", New York, John Wiley & Sons, '77.
15. Blazie, Dean B., "A Compiler for Pocket Calculators", Technical Memorandum 1-76, U.S. Army Human Engineering Laboratory, Aberdeen Proving Ground, Maryland.
16. PBX Exchange, Dated periodical from Texas Instruments, Inc., PBX-52, P.O.BOX 22283, Dallas, Texas 75222.
17. HP Key Notes, published quarterly by Hewlett-Packard, 1000 N.E. Circle Boulevard, Corvallis, Oregon 97330.
18. Hewlett-Packard Journal, published monthly by Hewlett-Packard, 1501 Page Mill Road, Palo Alto, California 94304, November 1976 issue.
19. Compucalc, Inc., 201 E. Main, Silverton, Oregon 97781.
20. Educalc, Educational Calculator Devices, P. O. Box 974, Laguna Beach, California 92652.

THE SIDELOBES OF INDUSTRIAL DISTRIBUTION ARE FOCUSED ON THE HOME MICRO-COMPUTER HOBBYIST

Lowell Smilen, Ph.D.
Almac/Strom Electronics
5811 Sixth Avenue South
Seattle WA 98108

ABSTRACT

The emerging microprocessor technology is having and will continue to have a strong impact on the way electronic design is carried out on existing electronic products and the countless number of new ones yet to appear on the market. The microprocessor hobbyist is in the forefront of this exciting revolution; both as a consumer and a contributor. This paper will explore how the hobbyist can benefit from the engineering programs in effect at some electronic distributors, who in turn are working closely with the semiconductor manufacturers. The broad array of microprocessor kits, system development tools and learning products which are now available will also be discussed.

I. Introduction

The role of the industrial electronics distributor has been evolving over the past few years. Whereas previously, the distributor was concerned mainly with supplying single function components, i.e. resistors, transistors, wire; the current trend towards multifunction components and pre-wired subsystem boards such as microprocessors, single board computers and microprocessor development systems has caused the distributor to become more technically oriented.

The intent of this paper is to describe the wide range of microprocessor products and services available through some segments of industrial electronics distribution and to suggest that this mode of operation, perhaps with variations, will be increasingly followed in many parts of the country.

The major effort or mainlobe energy of the distributor is directed at the OEM (original equipment manufacturer), however, the home hobbyist market is growing at a sufficiently rapid rate that the sidelobe energy directed in that direction is still considerable. This is particularly true for microprocessor hobbyists since electronic system development using these products lends itself well to the home laboratory. The home hobbyist ranges from the person who is developing a system mainly for fun and games to the experimenter who is prototyping a system from which a profit is ultimately to be realized. The nature of the microprocessor products on the market allows both goals to be realized.

II. Engineering Services

Perhaps the biggest problem associated with moving into microprocessors is determining where to start. A Microprocessor Development Center with a resident technical specialist provides the ideal starting point. The beginner then has the opportunity to come in and discuss his goals, budget and future plans. The designer at every experience level gains exposure to new products. The Center contains the development systems from a variety of semiconductor manufacturers with an assortment of peripheral devices; typically a development system consists of:

- a. The processor console with between 16K and 64K memory; resident firmware for controlling the peripheral devices; and resident firmware for program operations and debugging
- b. Teletype or similar units, possibly a high speed printer
- c. Disc operating system and/or paper tape reader and punch
- d. CRT console
- e. PROM Programmer

f. In-circuit emulator and/or simulator

g. Software for program editing, assembly and in some cases software for high-level language

The function of these units will be discussed in the section on Microprocessor Products.

The visitor to the Center receives a demonstration on one or more of these systems with a discussion of their capabilities. (The Center at Almac contains the development systems from Intel, Motorola, National and Texas Instruments).

Of significance to the hobbyist is the fact that time on these systems can be purchased on a daily basis.

As a means of keeping the designer current with the latest developments in microprocessor products, the distributor sponsors a series of technical seminars usually presented by field application engineers from the semiconductor manufacturers. The distributor also presents technical seminars at universities, local micro-computer clubs and trade shows.

There are other engineering programs in existence at distributors which are pertinent to the hobbyist. A PROM programming facility provides the means for entering the debugged program into a field programmable ROM, either erasable or not, from a variety of media such as paper tapes and mark sense cards.

A distributor often provides electrical assembly capabilities for customizing such items as switches, connectors, terminals and flat ribbon cable to the user's specs.

Where the distributor maintains a retail store in conjunction with his stocking warehouse, the hobbyist has easy access to the wide variety of products and services which the distributor provides to his industrial customers.

III. Microprocessor Products

A. LSI Chips

The range of microprocessor products is so extensive that they can be best treated by breaking them into categories: First, there are the LSI chips which constitute the microprocessor chip set, the RAM and (P) ROM memories; serial and parallel I/O chips and the newer peripheral chips such as, the programmable DMA controller, interrupt controller and interval timer. Examples of some of these chips are the:

1. Intel 8080A - an NMOS 8-bit parallel CPU with 6 registers and an accumulator, separate 16-bit address and 8-bit data bi-directional data busses.
2. Intel 8048 - an 8-bit NMOS LSI chip which contains the CPU, memory in either ROM or PROM structure, RAM, internal clock, programmable I/O parts and interrupt control logic.
3. Motorola 6800 - an 8-bit NMOS microprocessor with an 8-bit bi-directional bus and a 16-bit address bus, 6 internal registers using a 5-volt power supply.
4. Motorola 6802 - an 8-bit NMOS microprocessor that contains the registers present on the 6800 but also includes an internal clock oscillator and driver and 128 bytes of RAM. It uses the same language as the 6800.

This chip also illustrates the availability more highly integrated LSI chips.

The trend towards microprocessors with greater data processing capabilities is shown in the:

5. Texas Instruments TMS 9900 - a single chip 16-bit NMOS CPU having 16 general registers and on-chip instructions for multiply and divide. It is supported by the TMS 9901 a programmable systems interface chip, the TMS 9902 asynchronous communications controller, the TMS 9903 synchronous communications controller and the TMS 9904 clock generator.

A chip which is applicable to controller type systems which require a lower degree of computer complexity at a low cost is the:

6. National SC/MP - 8-bit microprocessor which is available either as an NMOS chip or a slower PMOS one. In addition to its use in equipment controllers it finds applications as a peripheral interface unit in distributed processor systems.

There are many factors to be considered in choosing a microprocessor chip. Among these are speed (relative to the application), power dissipation, supply voltages, data handling capability, availability of peripheral chips, need to minimize chips, cost and second sourcing. There is another factor, which will be discussed further later on, hardware and software support provided by the semiconductor manufacturer.

B. Microprocessor Kits

The next product grouping to be discussed, and possibly of greatest interest to the house hobbyist, comes under the heading of microprocessor kits. Historically, kits have always been popular with hobbyists ranging from test equipment kits to high fidelity and stereo subsystems. The semiconductor manufacturers have made a strong effort to capture a wide following among microcomputer beginners by putting on the market kits which are excellent values, both in electronic parts and documentation for self-learning programs. They provide an ideal starting point for novices and those who wish to become familiar with a second or third chip family.

The first microprocessor kits were merely an assortment of LSI chips. When the kits were expanded to include a printed circuit board, their popularity grew and with the introduction of LED displays and hexadecimal keyboards into the kits sales advanced more rapidly. With the inclusion of a means of entering and receiving data the need for an expensive terminal was minimized.

Examples of the more popular kits ranging in price from under \$100 to \$250 are:

1. Intel SDK-80 which contains the 8080A, RAM and ROM memory, I/O chips, a crystal, TTL logic, passive components and a PC board. The monitor firmware allows communication with various terminals, program debugging and execution. There is space provided on the board for the user to expand the system for his particular application.
2. Motorola MEIC 6800D2 Kit contains a 24 key keyboard and a 6-digit seven segment display in addition to the M6800, RAM and ROM serial parallel I/O chips, TTL logic, 2 PC boards, and all passive components needed to complete the kit. There is a monitor program which permits communication with an audio cassette and program entry and debug from the key board. This kit can also be expanded by the user.
3. National ISP-8K/200 SC/MP kit and ISP-8K/400 SC/MP keyboard kit. The SC/MP kit was available initially and contains the SC/MP chip, RAM, a clock crystal, buffering chips, a ROM

for teletype communications and program debugging and a PC board with an expansion capability. The keyboard kit was introduced to obviate the need for the teletype and uses the blank portion on the PC card of the first kit. The ROM included with this kit allows the user to enter, debug and execute his program by using the calculator-style keyboard.

C. Computers on a Board

Next in the hierarchy of microprocessor products would come the families of single-board computers. This family of products probably has its greatest significance for industrial users but the hobbyist should be aware of this potential storehouse of products.

Intel's SBC 80/10 probably was the first of the single board computers followed by Motorola's micromodule the M68MM01. Intel has now introduced its expanded version of the 80/10, the SBC 80/20 and Texas Instruments, the T.I. 990/4.

The single board computers are prewired circuit boards that have been tested and contain a CPU, RAM, I/O chips, clock circuits, terminal interfaces and sockets for ROM and I/O expansion. Their significance is enhanced by the fact that they are members of a family of boards and cardcages that may include RAM and PROM socket cards, I/O cards, A/D and D/A cards and more recently a general-purpose-interface-bus card. The cost of the microcomputer card and associated modules, in single unit quantities, is in the \$500 and \$800 range.

The beauty of these microcomputer cards and their support modules is that they provide for a wide variety applications without the need to stuff and wire each PC board, they minimize inventory and spare part inventories and they are compatible with the microprocessor development system associated with the particular semiconductor manufacturer.

This brings us to the last grouping of microprocessor associated products - the microprocessor development systems.

D. Development Systems

By their very nature, microprocessor systems are complex, in that they require both hardware and software design, with trade-offs between the two. Digital test equipment existed for hardware check-out and large computers could serve for software development and simulation. However, there was a need for a new class of test equipment which combined both functions - what resulted was the microprocessor development system which is itself a general purpose microcomputer system. The key components of these systems are:

1. Microprocessor console. This unit contains a microprocessor and associated circuitry for interfacing to the resident RAM memory which can range from 8K to 64K; firmware in the form of a monitor program in ROM for serial and/or parallel communication with one or more terminals, for program loading, debugging and copying; a card for interfacing the system to external hardware; additional sockets for special function cards and a heavy duty power supply.

2. Peripherals

The disc operating system, often with two drives, holds the system software and provides a rapid means of loading and storing programs.

Where paper tape is used, in conjunction with a teletype, a high speed paper tape reader, 200 characters per second, is desirable to perform the loading function.

Other peripherals include the ASR 733 twin

cassette, data terminal, an alphanumeric keyboard cathode ray tube display console and 165 character per second line printers.

3. In-circuit Emulators

The utility of the development systems was greatly enhanced by the introduction of cards which plugged into the main console's mother-board to allow the development system to be used to check out the hardware being developed. The emulator permits the system being developed to use the main console's memory and I/O capabilities, to execute the program being developed in the user's hardware and to display executed instructions with address and data, and contents of the CPU's registers, all in real time.

4. PROM Programmers

Once a program has been successfully developed, the PROM Programmer allows the data to be entered in a field programmable ROM. The PROM Programmer may be a separate console that is interfaced to the CPU console by cable or a PC card that can be inserted into the mother-board of the CPU console.

5. Resident Software

Initially, the resident software was dedicated to program assembly and editing, then came the software for using the emulators and the diagnostic check-out procedure. However, recently the trend has been to resident higher-level languages such as PL/M-80, Fortran IV, and Basic. These operate in systems using from 4K to 64 K bytes of memory.

Most semiconductor manufacturers have produced a prototype development for use with their microprocessor chip; some have gone further and developed the wide variety of hardware and software products discussed above. Most noteworthy of mention are Intel's Intellec System, Motorola's EXOR-cisor Systems, National's IMP and PACE Systems and more recently, Texas Instruments' 990/4 Prototyping System.

Prices for microprocessor development systems range from a minimal configuration at \$4,000 to a multi-function test bed at \$15,000.

E. Learning Systems

There is another group of products, which can be described as learning type systems, that has been introduced by the semiconductor manufacturers. Most of these products are unique.

National Semiconductor has produced the SC/MP Low Cost Development System (LCDS). It comes with a chassis containing a monitor program, a hexadecimal keyboard, a LED display, sockets for expansion and a CPU card with RAM and I/O chips. Communication with this unit is in machine language either via the keyboard or a teletype. Additional cards for RAM and PROM memory can be obtained and it will run National's 4K Basic.

Intel's PROMPT 80 and PROMPT 48 contain single boards computer cards for the 8080A and the 8748 or 8035. They have hexadecimal keyboards and LED displays, PROM programmers, monitor firmware and power supplies. They can be used as development or learning systems. They will interface to the MDS-800.

The learning-type systems described above are in the \$500 - \$1,500 price range.

This brings us to the Texas Instruments Micro-processor Learning System which consists of four

separate modules, each provided with a detailed manual. The LCM-1001 is intended to teach micro-programming, the development of a set of instructions as used by a microprocessor and is concerned with processor architecture, data handling and processor programming. The companion modules are the LCM-1002 a controller, LCM 1003 memory and the LCM-1004 input/output module. The four modules combined constitute a versatile micro-computer. The entire set costs less than \$700.

IV. Conclusion

The era of microprocessors is indeed in its infancy. Regardless of what projections are used for its growth, there is no doubt that they will have a profound impact on our lives whether we are engineers, technicians or consumers. The technical hobbyist enjoys working with and nurturing new technologies. In the case of microprocessors, the hobbyist will find that broad, valuable assistance and support are being provided by the technically and service oriented electronics distributor, working with the creative semiconductor manufacturer.

If "Small is Beautiful," is Micro Marvellous?

A look at micro-computing as if people mattered

Andrew Clement
789 W. 18th Ave.
Vancouver B.C.
Canada V5Z 1W1

Abstract

E.F. Schumacher, in Small is Beautiful and his more recent work, has drawn attention to the role that technology plays in the shaping of our present society and to the need for technologies of smaller scale in the growth of a more humane society. He identifies the criteria of smallness, simplicity, capital cheapness, and non-violence as being particularly important. Micro-computer technology, which is the basis of computing systems for home and personal use, seems to satisfy these criteria in many respects, and so it potentially represents a technology appropriate for future general use.

This paper examines critically the field of micro-computing in light of Schumacher's four criteria. Many of the positive aspects are seen to hold up under this examination. In important respects the technology really is small, inexpensive and non-violent. It can also be fun, facilitate the more equitable distribution of computing power and help breakdown fear and mis-understanding of computer technology. A number of important drawbacks are also discussed. Micro-computers are generally crude and not very powerful at the moment. They are not simple to use other than at a fairly elementary level. Widespread use of micro-computers may contribute to the increased centralization of power in society.

The important role that computer amateurs and other pioneers in the micro-computing field have in influencing its future development is pointed out. They are challenged to devote effort to ensuring that computers are used humanely to enhance people's lives and that the negative aspects of computers are minimized. The paper ends with suggested techniques and applications that may assist in achieving this goal.

I. INTRODUCTION

Everyone agrees that technology has had a profound effect on modern society. Technology has crept into every facet of our personal lives and left no public institution untouched. Along with the many good things it has brought, there have been serious ill-effects to which no-one is immune. It is therefore wise of us to inspect closely new technologies as they arise to determine whether, and how, we may benefit from them.

Two contemporary authors have made significant contributions to the understanding of what we should expect from our technologies, and their manifestation as tools, in order that they serve us best. Ivan Illich, writing in Tools for Conviviality, calls for tools "which give each person who uses them the greatest opportunity to enrich the environment with the fruits of his or her vision." (p.22), tools "which foster self-realization", (p.25) tools that promote self-reliance, autonomy and harmony between ourselves and with nature. It is essential that control of the tools lie with the individuals and organizations they are intended to serve. E.F. Schumacher has similar objectives, and in Small is Beautiful and Technology and Political Change, identifies the scale of the technology as the most important factor in determining its social consequences. In general, many present technologies are so massive that the overall effect for people is that they suffer, rather than

benefit, from their use. Smaller, more human-scaled enterprises are needed in the formation of a more humane society.

Schumacher suggests four criteria with which to judge technologies and serve to guide their development. These criteria are:

- small
- simple
- inexpensive
- non-violent

Technologies which satisfy these criteria will be much more likely to be good for us.

One technology that appears destined to play a large part in our lives is that of micro-computing. Conventional computer technology, already quite influential, is at the moment embarking on an important new direction based on a radical reduction in the size and cost of primary computer components. Large scale integration (LSI) of circuits has made possible powerful computing devices that could be purchased by virtually everyone for home or personal use. Though not widespread yet, the future general use of these "micro-computing" devices seems probable and would have considerable impact on society. At first appearances it would seem that this new micro-computer technology satisfies many of Schumacher's criteria and so could have the potential to provide us with tools of great service. The remainder of this paper is concerned with assessing this potential and exploring ways in which it can be exploited.

II. WHY MICRO-COMPUTER TECHNOLOGY IS IMPORTANT

Before becoming concerned about the social impact of micro-computer technology it is wise to see whether such concern is of any relevance. Firstly it should be shown that micro-computers are likely to have an important effect on people's lives and secondly that there is some possibility of influencing their course of development. Both of these appear to be the case.

Micro-computing has not yet begun to affect many people's lives directly but there is good reason to believe that in the foreseeable future their impact will be great. What we are witnessing at the present time is the rapid emergence of the computer from within the confines of large institutional settings and into the view of the general public. Until now most people have not really seen a computer or had to deal with one directly. Rather they have learned of them by report and communication with them has been via the mail and through a bureaucracy. Computer hardware first came into public view in the form of terminals for airline reservation systems (back view only), but now they are highly visible in many banks and other institutions. Point-of-sale processing, and automated checkout are about to burst forth and electronic funds transfer is fast approaching. In the home, the last few years have brought the cheap pocket calculator and the video game that plugs into the T.V. set. Other talked of T.V. add-ons include systems to allow shopping, banking, and even voting from the comfort of one's living room. We can look forward to finding (?) digital circuitry in our cars, ovens, and other appliances, (Figure 1).

At work the outlook is for the "electronic office" with computerized word processing and greater

reliance on computer conferencing techniques. All of these changes rely heavily, though not entirely, on computing devices built upon the technology of large scale integration (LSI). Thus many aspects of our lives, in buying, in recreation, in employment and transportation and communication are going to be affected strongly by micro-computing technology. We should see what we can do to ensure that we are better off for all this.

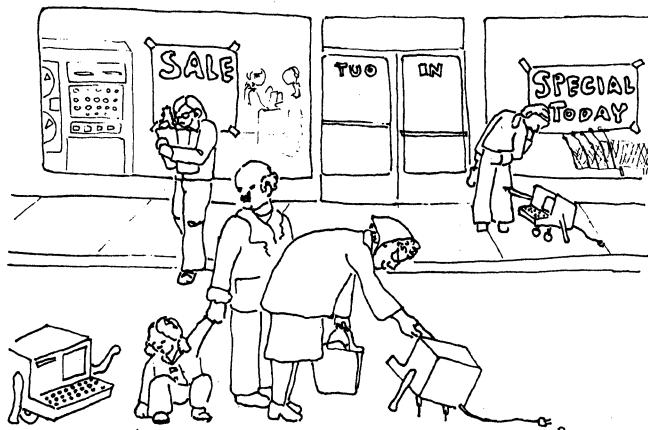


Figure 1
People will soon be meeting
micro-computing devices everywhere.

The realization that micro-computers are going to have a major impact on our society is not, by itself, sufficient reason to become actively concerned in trying to affect their development. We need confidence that the technology is somewhat flexible, that it can be changed to some degree, and further that these changes will be reflected back on the society that uses the technology. If on the other hand what we do has no effect, then we might as well sit back to enjoy the show and amuse ourselves with our new toys.

We shape our tools; thereafter they shape us.

Marshall McLuhan

This statement expresses part of the justification for concerned action at the present time. At the moment the computing devices are still crude and many of the details not worked out; the manufacturing and marketing systems not fully developed; patterns of use have not been firmly established; the myths that apply to conventional computers will no longer be accepted but replacements have not yet taken hold; legislative regulations await formulation. In many areas the technology and related structures have not solidified and although to a large extent, the future forms have already been determined by prevailing social forces, there is still a short time to influence the course of development. To do this positively and effectively requires an understanding of the technology and what it has to offer.

III. SCHUMACHER'S FOUR CRITERIA: SMALL, SIMPLE, INEXPENSIVE, NON-VIOLENT

Schumacher's four criteria give us way looking at technologies and evaluating them for their potential to benefit us. The criteria are not really new and in fact are based heavily on common-sense notions of what makes something friendly and easy to get along with. Everyone has had rewarding experiences with tools or institutions and I am sure that they would often find that these criteria were underlying the success of the encounter. Conversely, unsatisfactory experiences with technology can often be traced back to it being huge, complicated, expensive, or violent. By themselves these criteria are not

complete and do not guarantee that the effects of a technology will be humane, but they do provide a useful starting point.

Small

People are small and tend to get overwhelmed when they have to approach and deal with things that are much larger than themselves. It is clear that the physical size of micro-computers is small. They can be put on your desk, can be carried around, taken over to friends' house, the whole thing can be seen and be grasped at one time. At the extreme, the Dynabook that is being developed by Xerox is pictured as being the size of a notebook and works off batteries and so will be completely portable and unintimidating in appearance. This is an important departure from the conventional view of computers as large, complicated and somewhat fearsome devices, demanding large buildings, special air conditioning, phalanxes of experts to care for them, and layers of bureaucrats to protect them.

Considerations of size apply not only to the actual hardware that the user sees but also to the necessary back-up technology and institutions as well as the organizational settings that are required to make effective use of the tools. Fortunately in most of these respects the "smallness" of micro-computers remains in tact.

Manufacture and marketing of most of the hardware components can be done with relatively small firms and without huge factories and machines. In fact, many of the products available to computer amateurs have been introduced by small groups of individuals operating out of "garages". This is a reflection not only of the relative newness of the field (there hasn't been enough time for companies to grow really large) but also to the characteristics of the technology and the particular way in which it has developed. A good example of this is found in the "bus" structure that underlies many current micro-computing devices. A "bus" structure is not only just an excellent way to organize a computer (there are other ways) but has important consequences in other areas. The emergence of the "Altair" or S-100 bus as a de-facto standard has resulted from, and more importantly, allows, small enterprises to create special products that can plug into it. Of course this has depended on the fact that there is no restriction preventing anyone from doing this. It is not hard to imagine that a really large manufacturer (mentioning no names) would either not use such an easily copied and exploited technique or would find artificial means to prevent such "abuse" and preserve its monopoly.

When it comes to the manufacture of the integrated circuits (ICs) the "smallness" is lost somewhat since fairly large, though not huge, installations seem to be required.

On the software end small groups appear to be able to operate efficiently, perhaps more efficiently than armies of programmers even with very large programming tasks. However, when it comes to marketing a hardware/software combination, size may be a considerable asset in order to enjoy economies of scale and as a protection against competition. Using micro-computers can also be a small-scale affair as long as there is no need to have to plug into large centralized networks as is the case with conventional television.

Simple

Simplicity is another virtue, not necessarily related to size, but sharing with "smallness" the tendency to promote autonomous control and self-reliance. If a technology or tool is simple then it is much easier for people to understand it, to use it and modify it for their purposes, to fix it when it goes wrong, in short to control it rather than be controlled by it. Unfortunately micro-computers are simple only in a few, restricted

aspects. Most people do not have a great deal of trouble using computer equipment. Small children appear to find it simple enough and although harder to use than most popular appliances (e.g. telephone, oven, etc.) it is certainly much easier than an automobile, once some basic skills and concepts have been mastered. The same goes for the assembly of equipment from kits and the composing of simple programs. However, when involvement goes beyond this level, things become much more complicated. The analysis, design, and programming of major systems requires specialized skills and experience. There do not seem to be programming languages and techniques presently available that allow us to do this sort of thing fairly easily. This is due no doubt in part to the inherent complexity of the process but also because we are still in the infancy of our understanding. The problem is even worse when it comes to the design and manufacture of hardware, particularly integrated circuits. A highly advanced and sophisticated technology is involved requiring machines of great precision and personnel of rare expertise. With the reliance on experts goes a loss in control and freedom, resulting in a situation akin to that with the automobile. Most people can drive one, put in gas, change tires and sparkplugs, but when it comes to repairs we leave our mobility (and pocket-books) in the hands of trained auto-mechanics. The prospect of actually making our own vehicle or modifying an existing one is simply out of the question for all but a handful - it's just too complicated. The present situation with micro-computers is substantially worse, with the consequence that at least in the foreseeable future their popular use will centre mainly around the purchase of plug-in modules. Most people will depend on outside hardware and software suppliers and not be able to rely on themselves or others in their vicinity. The guts of our major machines will continue to be generally out of bounds.

Inexpensive

The criteria of inexpensiveness applies primarily to the financial aspects of a technology, and in fact, Schumacher uses the term "capital cheapness". The capital requirements of a technology are clearly related to its size and complexity, and like them has an important influence on its control and acceptability. Again it is useful to distinguish between the hardware and software aspects and the consumer and production ends of micro-computer technology.

Perhaps the most dramatic and publicized feature of the "micro-computer revolution" is the drastic drop in price of hardware components. This plunge in cost is probably the most important factor contributing to the rise of popular computing and the trend shows every sign of continuing. Small hobby systems are now available for a few hundred dollars, which is in the order of domestic appliances and within the reach of many people. However, by the time the extra memory, peripheral storage and other goodies have been added to make an interesting system the cost has risen to several thousand dollars, making it a substantial investment and thus out of the range of most individuals and small organizations. With further refinements of technique and the development of mass markets, this will probably remain the case for only a few more years, at least in North America. This contrasts sharply with conventional computer equipment, which is enormously expensive in comparison, and thus available to only already wealthy organizations. At the manufacturing end it appears that great investments of capital are not required except possibly in connection with the production and marketing of integrated circuits.

Software, to bring the newly inexpensive hardware to life, has not enjoyed the same radical drop in price, mainly because it is a mind and labour intensive process and not susceptible to easy automation. To produce high-quality software does

not require lots of time to start with but does take a great deal of time and care, making the ultimate cost fairly high. The purchase price can be kept down if the software is general and reliable enough to be attractive to a large number of users. One way to do this is through software exchanges where software can be made available and acquired cheaply. The producer would not stand to earn a great deal of money but would presumably receive some compensation by being able to enjoy other peoples' work at a low cost. If this system is to flourish though, the software must be very reliable, well documented and widely accessible. Dr. Dobb's Journal is a good example of this process in action at a level of amateur computing. The alternative seems to be stiff copyright regulations maintaining high costs for all but the most widely used packages.

Non-violent

It is very important that a technology that is to find widespread use not harm people nor damage our natural environment. In a direct sense this is certainly true of micro-computer technology. The greatest physical danger to an average user is getting an electric shock or having one drop on his toe. Environmentally, no aspect of computer technology, even with universal application, consumes much in the way of natural resources, demands large amounts of energy, nor produces appreciable pollution. We can say generally that computers "walk lightly on the earth". (Figure 2) This is because the hardware, the physical part, is needed only to harbour the software, which is really where the interesting activity is going on, and thus has to interact only minimally with the rest of the world. Micro-computing is like other communications technologies in this respect in that its effects are usually not directly physical. For instance, the violence that results from television is not due to its physical characteristics but rather to the indirect effects on its viewers over a period of time. With micro-computing, the violence comes from the ways in which it affects how people see themselves and how they relate to others and their environment. Excessive and narrow use of computers seems to promote mechanistic thinking and de-personalization which represent and result in forms of violence that, though not overt, are nevertheless significant. Further investigation of this social and psychological violence is vitally necessary if large numbers of people are going to come in regular contact with computers. (Figure 3)

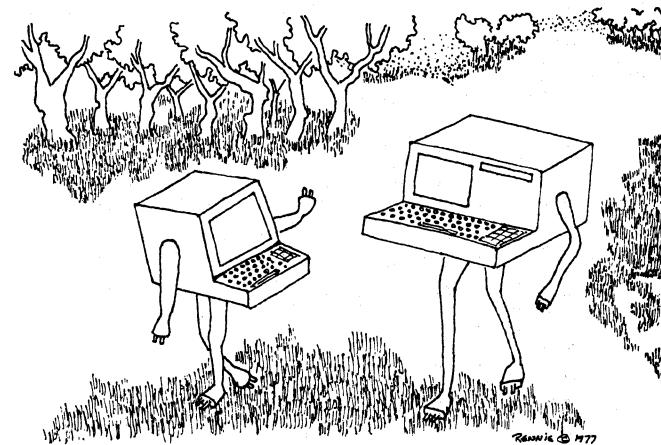


Figure 2 *Micro-computers walk lightly on the earth.*

IV. ADDITIONAL SIGNIFICANT FEATURES OF MICRO-COMPUTING

Besides Schumacher's four criteria that have just been discussed, there are a number of other important features that should be considered when examining micro-computing's potential to serve as a humane technology in our society.

Fun

Micro-computers are fun! Lots of people, children of all ages, enjoy playing with them. Computers are not just serious business! They have the potential to become great mind toys since they are so flexible and can be adapted to represent such a wide variety of situations (see Ted Nelson's *Computer Lib/Dream Machines*). They are such a good toy, in fact, that some people get so fascinated and turned on to the world they are creating and exploring in the machine, that the outside world gets lost. For this reason, excessive exposure to computers should not be allowed to drive out other more physical and social forms of play, particularly in the case of children. Nevertheless the pleasure that can be obtained from micro-computer technology is an important part in making it friendly and socially desirable.

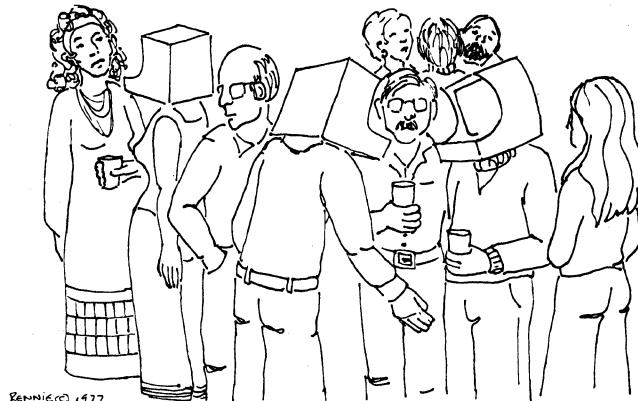


Figure 3

Breakdown of fear and misunderstanding

The widespread use of micro-computers will hopefully have the effect that people will gain a much better understanding of computers. This will tend to lessen the fear of computers based on ignorance and provide a firmer basis for the real justified fears and thus enabling people to deal more effectively with the technology. It should make it much harder for large organizations to hide behind their machines and abuse people by insisting they conform to the expectations of the computer. People will learn that it is not the machinery that is mainly responsible. However, there is a danger that the understanding may only be superficial and that the loss of fear resulting from familiarity may work against people. Instead, what could happen is that this simply paves the way for easier penetration and acceptance of the more questionable aspects of computer technology, such as Electronic Funds Transfer and other large-scale databank systems.

Redistribution of computing power

The ability to process information quickly and cheaply is an important component in the exercise of power in present society. Until now the mechanisms for doing this have rested almost exclusively in the hands of large institutions while individuals and small organizations have been pretty well left out. The wide-spread availability of small, yet reasonably powerful computers could alter this significantly and help shift real power, along with computing power, toward smaller social units. For this to actually take place not only would the computing power have to be accessible but also the expertise and raw information needed to go with it. It is not clear that the organizations whose strength would suffer in proportion would be willing to relinquish their control over these as well as the other vital components of power. In fact, if a multitude of micro-computing devices act mainly as peripherals in a highly centralized system, then power will be further consolidated in the hands of the massive interests that already dominate society.

Crude, not powerful

Cheap, general purpose micro-computing systems are still in their infancy and are yet fairly crude and lacking in significant power. This is particularly true of the way in which people have to interact with them. The graphic quality of output, based as it

usually is on a stark, single upper-case type font of poor definition is often much lower than we are accustomed to in other areas. This doesn't have to be the case. Input to computers is almost always through keyboards with very little use being made of devices such as light pens, joy sticks and "mice", which, in many cases, are more appropriate and easier to operate. The languages of interaction with which we communicate with the machine are also often primitive. Seldom do they offer convenient aids in learning them, are usually intolerant of any "mistake", rudely insisting that they be addressed in strict and peculiar ways, and they do not easily admit to modification to correct their annoying features. In fact, a computer usually has to be treated rather like a spoiled child from whom any desired, out-of-the-ordinary behavior can be obtained only by those who are very familiar with the brat or who love it as a mother and are therefore blind to its shortcomings. This is not to say that there have been no important achievements, for clearly there have been, but we still have a long way to go before most people find micro-computers to be friendly, useful and easy to get along with. An indication of what is possible in this regard is seen in Alan Kay's Dynabook Project at Xerox. The aim of his Learning Research Group, as outlined in *Personal Dynamic Media*, is to develop a computer the size of a notebook that rivals in power some of today's biggest computers. A Dynabook would have visual output of at least "higher quality than what can be obtained from newsprint. Audio output should adhere to similar high fidelity standards." A Dynabook could be owned by everyone and used "as a medium for expression through drawing, painting, animating pictures, and composing and generating music", as well as handling many more mundane information processing tasks. The fact they have already made great progress gives credence to the achievability of their goal and is a tribute to what the combination of talent, hard work, and money can accomplish.

Exclusiveness

Micro-computing is much more accessible to average people than conventional computing, but still has in some ways the same exclusive cult feeling about it. For example, informal observation reveals that the overwhelming majority of active participants are male. Why is it that more women are not involved? Is it because machines are inherently more interesting to men, perhaps women receive cues that they are not wanted in the area? The simple fact that it is already so male-dominated may be sufficient to intimidate many women from participating actively. That the disproportion is not as pronounced among young enthusiasts could be sign of change, although perhaps computers appeal to the adolescent in us and men stay adolescent longer. Reducing sexist and other forms of bias and involving a wide variety of people is important in the development of micro-computing. A range of considered viewpoints will tend to protect against the growth of a widespread technology that serves only the interests of the small group of its creators.

The sources of discrimination and exclusion are probably deep-seated in our culture and so cannot be removed quickly. However, there are some fairly obvious mechanisms that can be seen at work and dealt with. One of the clearest is the way that jargon is used. Any advanced technology needs specialized language in which to express the notions that are peculiar to it. However, at the same time, and quite unconsciously, it also serves to exclude those who are not initiated into the circle (Figure 2) and reinforces membership for those already in. Deliberate avoidance of unnecessary jargon and careful explanations for novices will do much to broaden participation in this important field.

V. THE CHALLENGE TO COMPUTER AMATEURS AND OTHER PIONEERS

It is hopefully clear from what has preceded, that micro-computer technology, though suffering from some major drawbacks, has a number of positive features to recommend it for wide-spread, human scale use in society. It is very likely to have an important impact on society and how it is shaped now, in the critical period of its infancy, will have long-term consequences. One group of people who will influence the course of development and who are in a position to exercise some choice over its direction are the computer amateurs. There are two main reasons why this is so. The primary reason is that computer amateurs are the first major group in the general population to have significant computing devices in their homes. If it catches on, the market would be enormous and so major manufacturers will be watching closely to see what happens to

get an idea of what will sell. Behavior patterns, in terms of purchase and use of micro-computer systems will establish trends that will continue for some time. Hopefully high standards will be set during this early period when the market is still forming. If people insist on quality systems that are reasonably powerful and self-contained, that give the user flexibility and choice to implement interesting, constructive applications, then the result will be quite different than if people tolerate the introduction of crude pre-packaged systems of narrow imagination and usefulness into their homes.

The second reason that computer amateurs can influence the evolution of micro-computer technology is that they have the time and freedom to explore interesting avenues of approach. They are not bound by having to make money from the end product so they can try many leads before achieving a really significant result. For instance, an amateur computerist could enjoy spending a great deal of time collaborating with a musician on a music system before coming up with a version that was truly exciting. Almost incidently it could be made available to many others at very little cost. In other words, computer amateurs represent an important pool of inexpensive research talent and possibilities.

One consequence of the influential role computer amateurs occupy at the present moment is that they have social responsibility to others to use their position wisely. This responsibility comes from the fact that the actions of computer amateurs could have a significant effect on other people's lives and is independent of whether they actually choose to acknowledge the responsibility. This is one of the reasons that the term "home and personal computing" is more appropriate than "computer hobbyism" for what computer amateurs do, even though many regard it as a hobby. Because it has major social consequences it is not just like any other high technology hobby. "Computer Hobbyism" is simply the thin, leading edge of the much larger field of "home and personal computing". The challenge to the amateurs and other pioneers in this field is to accept a share of the responsibility and work actively to ensure that the micro-computing for home and personal use grows in humane and positive directions.

VI. WHAT TO DO?

If the argument has been accepted so far, the question now is: "What does one do about it all?" To answer this question directly is very difficult so what follows here are some suggestions for things that can be tried, rather than "answers". They are tentative and sketchy, at best, but should provide a good starting point.

Research and Discussion

One obvious step is to find out more about the whole area of computing and its social implications. Two books that can help in this area are Computer Power and Human Reason: From Judgement to Calculation by Joseph Weizenbaum and The Conquest of Will: Information Processing in Human Affairs by Abbe Mowshowitz. Other suggested works are listed in the bibliography section. Discussions with others will also serve to expand the base of knowledge and refine viewpoints. Try to keep a broad view of one's activities so that technical considerations are balanced by human concerns; so that people don't get lost in all the bits, bytes, and bauds.

Follow Schumacher's four principles

The criteria of smallness, simplicity, capital cheapness, and non-violence serve as a useful guide in developing both hardware and software systems. In creating new hardware components other people with similar needs will benefit most if the design is kept simple and inexpensive. With software development, concentration on high-quality, well-documented, transportable programs, though quite difficult to achieve in practice, will likewise result in widest benefit. Programs should treat users well by being polite, by providing learning aids, good error handling and clean interfaces. Increasing the sophistication of programs in this way may require considerable extra effort and contradict to some degree the aim for simplicity, though imaginative approaches are often able to overcome dilemmas of this nature. If a program is worthy of the attention of other people software exchanges and publications like Dr. Dobb's Journal are a good way of making it available to them cheaply.

Build on existing cooperative and sharing enterprises

The social and organizational settings in which micro-computer technology is developed and used will define precedents of some lasting effect. If the work is done in an environment of relaxed and open exchange of information by people in small local groups that are in close communication with each other, then the results are likely to be more humane than if it is done in a competitive and secretive atmosphere. Computer clubs that are sprouting up everywhere are good starting points because they provide an opportunity for people living in the same area to get together to share experiences, exchange ideas, software and the like. Club newsletters are a valuable means for keeping members in touch with each other, and trading them between clubs expands the communications network even further. Computer amateurs have of course their own novel communications media at hand and so direct computer-to-computer link-ups may prove useful and at least interesting.

Major publications such as Creative Computing, People's Computer Company (PCC), and BYTE serve a similar function of information access and exchange on a broader level.

Work with non-computer people

Computers are too important to be left entirely up to computer people.

A very important step is for non-computer people to be involved with the development of micro-computer technology. By this is meant people whose main motivation is to do something particular, such as draw a picture, make music, organize and share information, say, and look upon a computer as simply one, or perhaps several, tools for accomplishing their objective. Computer people on the other hand (to take an extreme view), look for things they can use their computers on. They love their machines and the actual process of using them, rather than the end product. As a consequence these two groups have very different approaches. The computer person can explore wild and fantastic uses and show what the machine can do. Their vision is limited by what they perceive to be the bounds imposed by the machine. Non-computer people will bring them back to earth and supply interesting ideas that go off in quite different directions and which push hard on the limits of what the technology is capable of. The results of the interaction of these two forces should be excitement and conflict. It will no doubt be difficult at times but is quite vital to the health of micro-computing. Growth by incorporating a variety of perspectives is essential to avoid repeating the mistakes of conventional computing that have resulted from computer people being given too much unbalanced influence.

There are two important requirements that must be satisfied if this collaboration is going to work. Firstly, it is obvious that the two parties are going to have to be able to communicate effectively and this will require a form of commonly understood language. Each will have to learn a little of the other's terminology and at the same time reduce their own use of special jargon. It will take some time before this gets smoothed out. The second need is for the non-computer person to be able to communicate with the machine in a language that is natural to use and which can express the notions peculiar to the particular application. To start with the computer person will probably act as a go-between but in many cases the lack of immediacy will be unsatisfactory and a special language for direct communication will have to be implemented on the machine. This too can be expected to take much time and effort. In short then the message to the concerned computer person is to get involved with individuals or small groups, (artists, musicians, writers, teachers, business people, cooperatives) that one supports and work with them on mutually rewarding projects. The going won't be easy but it will certainly be stimulating and contribute valuably to the future positive and beneficial use of computers.

Suggested Applications

Hand-in-hand with these non-computer people there are a number of promising applications that are worthy of attention. Some are conventional applications that can be adapted to a smaller scale, while others are fairly innovative. Many have already been tried in one form or another, with varying degrees of success.

Design

Many small projects involving design (e.g. boats, solar energy, insulation, etc.) require calculations appropriately done with a micro-computer. For example in Ontario a van equipped with a computer goes around to small businesses and helps determine the insulation needs and potential energy savings.

Process control

A small machine shop may be able to use a micro-computer based process control system to permit the easy set-up of relatively small production runs and the convenient incorporation of custom modifications. The savings in re-tooling costs could allow the shop to remain competitive with larger operations.

Bookkeeping

Small businesses, theatre groups, farmers all need to do a certain amount of bookkeeping and often have neither the time nor inclination to do it themselves nor the money to hire someone else. A simple micro-computer based accounting system may be just what they need. In Canada, the government provides an inexpensive nation-wide accounting service to farmers. There is no reason why this sort of thing cannot be done on a local basis with a small computer.

Aids for the handicapped

Handicapped people, particularly those who are blind or deaf, need to have sensory information converted to a different form before it can be perceived. A small microprocessor unit could be very appropriate. Also some deaf people, because they cannot use the telephone in the conventional manner already have teletypes in their homes so they can communicate with one another. Adding computing power to their terminals could make them more useful and interesting.

Art

The potential for application of small computers in the arts is very broad. Computer-based "music-boxes", animation, stage-lighting, sound synthesis, graphics, poetry have all enjoyed a measure of success.

Games

Games are fun and, if designed with care, can be educational - not only for the programmer but also for the player. Gaming is probably the area that has received most attention from computer amateurs and there are already plenty of games around. A great number of these are one-person conflict games and there seems little need for more of them. Games that involve several people, that depend on cooperation or negotiation, that teach useful skills or encourage the understanding of important processes would all be worthwhile and provide balance to the scene. Game scenarios could be taken from realistic situations such as ecological systems, government, mathematics, land-use, physics, transportation and so on.

Information/Communication

Many small organizations have filing systems that would be greatly improved through automation, if it was sufficiently cheap. Not only would the routine up-dating chores be much easier, but wider access to the information in the files would be facilitated.

Richly cross-indexed directories and catalogues would be produced quickly and more frequently. Mailing lists, in particular, are a prime candidate for this type of application of micro-computers. Exciting possibilities open up when systems of this kind are set up that allow direct access by a whole community of users. People can not only get out previously stored information but put their own in as well. The system thus serves as a memory and communications medium for the community. In fact, the first two prototypes were called "Community Memory". The original one operated in the San Francisco area and was later "cloned" to Vancouver with the help of the author. Both versions used large time-sharing computers and work is now under way on stripped down versions that will run on portable micro-computers.

This list of suggested applications is quite incomplete and unspecific, but it does give an ideas of some of the things that an individual or group with time, energy, and a small computer in their hands could fruitfully undertake. They all represent promising steps in the development toward a humane and powerful technology.

VII. CONCLUSION

To return to the question posed originally, "If small is beautiful, is micro marvellous?", the answer must be:

Now? "No!"

But perhaps "Yes!", if we make it so.



Figure 4

Biographical Details

The author, Andrew Clement, was born in England in 1947 but emigrated with his parents to Canada before he could remember anything. They settled in a semi-rural area near Victoria, B. C. where the young lad spent most of early schooling. Eventually he made it to the University of British Columbia, graduated in 1968 with a B.Sc. in Honours Mathematics. There followed a two year period in which he taught mathematics and physics at a Malaysian secondary school as a C.U.S.O. volunteer. Upon his return to Canada, Andrew re-entered U.B.C. and received in 1973 an M.Sc., mainly for having managed to complete a thesis with the title: "The Application of Interactive Graphics and Pattern Recognition to the Reduction of Map Outlines". He continued research into interactive computer mapping and developed a geographical programming language known as INTURMAP. He has also been busy with non-computer people experimenting with interactive public-access information retrieval systems. The first one was known as the Community Information System and provided a file on social service agencies in the Vancouver area. The second system was Community Memory, which was based on the San Francisco prototype and operated under the auspices of INFACIT, a Vancouver community computer service society. Andrew is presently working on a survey of "humane computing" with a grant from the Canada Council. He would very much like to hear from anyone wishing to contribute to the survey in some way. He can be reached by mail at 789 West 18th Avenue, Vancouver, B.C., Canada, V5Z 1W1, or by phone at (604) 874-6740.

Acknowledgements

Of all the many people who have contributed information and ideas that have found their way in some form into this paper, I would like to especially thank Efrem Lipkin, Abbe Mowshowitz, and my colleagues in INFACT for the numerous, stimulating conversations that I have shared with them. Gratefully acknowledged also are the contributions of Ingrid Wood and Jan Morton, of the Community Planning Association of Canada, for the typing of this paper, Rennie Wiswall, for the cartoons, and the Explorations Program of the Canada Council, for the financial support that has allowed me to become better acquainted with the field of "humane computing".

Bibliography and References

These are some of the books and articles that relate to the subject of the social relevance of micro-computing.

Ken Colstad and Efrem Lipkin, Community Memory:

A Public Information System.

Print of a paper presented at the Computer
Science section of an IEEE conference, San
Francisco, February 1975.

Erich Fromm, The Revolution of Hope: Towards a
Humanized Technology,
Harper and Row, New York, 1968.

Karl Hess, Community Technology
SPARK, Vol. 4, #2, pp10-15, Fall 1974.

Ivan Illich, Tools for Conviviality
Harper and Row, New York, 1973.

Learning Research Group, Personal Dynamic Media,
Xerox - PARC Publication, 1976.

Abbe Mowshowitz, The Conquest of Will: Information
Processing in Human Affairs,
Addison Wesley, Reading, Mass., 1976.

Ted Nelson, Computer Lib/Dream Machines,
Hugo's Book Service, Chicago, 1974.

Victor Papanek, Design for the Real World,
Thames and Hudson, London, 1972.

E.F. Schumacher, Small is Beautiful: A Study of
Economics as if People Mattered,
Abacus, London, 1974.

E.F. Schumacher, Technology and Political Change,
Reprinted from Resurgence in RAIN,
Vol. III, #3, pp8-10, Dec. 76.

Joseph Weizenbaum, Computer Power and Human Reason:
From Judgement to Calculation,
Freeman, San Francisco, 1976.

Norbert Wiener, The Human Use of Human Beings:
Cybernetics and Society,
Doubleday, New York, 1954.

Mike Wilber & David Fylstra, Homebrewery vs. the
Software Priesthood,
BYTE Magazine, Oct. 1976.

THE COMPUTER IN SCIENCE FICTION

Dennie L. Van Tassel
Computer Center
University of California
Santa Cruz CA 95064

Since the computer is so ubiquitous in our life it is not surprising it is a common character in fiction and poetry. The main thrust of this article is about computers in literature, but I will mention a couple robot books for those interested in robots. There are several common motifs in computer related literature. The first is the computer run society.

Computer Run Society

In This type of story the whole society is run by a computer. Robert Heinlein's The Moon is a Harsh Mistress is an excellent book about a colony on the moon which is run by a friendly computer. Heinlein's book is unusual because the computer is good. Heinlein's other books are somewhat computer oriented and are all good, especially Time Enough for Love.

Usually the computer is evil. D. F. Jones has two books Colossus and The Fall of Colossus. In these books the computer ruthlessly runs the world. The book was also made into a movie called The Forbin Project. Kurt Vonnegut's Player Piano is about a society where all work is done by machines and only the privileged all allowed meaningful jobs. Everyone else is reduced to make-work and welfare program.

In Ira Levin's This Perfect Day we again see a computer run society which is very unpalatable to humankind. In Levin's book the computer completely controls everyone, including their work, sex, and friends by a combination of drugs and surveillance. One more story along this line is E. M. Forster's "The Machine Stops", where we have a computer run society and people have no control over how it is run and even forgot how it was started. In this story the computer tries to be a benevolent despot and fails.

Challenging the Gods

The next common motif is humankind's attempt at something previously forbidden, and failing. Like the old myth -- if you challenge the gods you are going to get it in the end. Old books that fall in this motif are Frankenstein (must reading for computerniks) and Capak's play R. U. R. Next we have Michael Crichton's excellent book The Terminal Man. Some doctors install a computer in a mentally sick man's head to control him. Everyone gets what they deserve in this fast moving book. Crichton's other books are excellent too. The opposite is done in Enslaved Brains, by Eando Binder, where human brains are implanted into computers to control the machines.

Humor

Humor is rather rare in literature and life but there is some good computer humor. The Tin Men by Michael Frayn is about a computer research institute where the researchers try to write literature, simulate sex and sports, and teach a computer some ethics. The book is very funny and one might even think of some known research institute when reading the book. Another humorous book is John Barth's Giles Goat-Boy, which takes place on a large college campus, where opposing sides, using computers, war for control of the campus. In David Gerrold's When Harlie Was One we find an adolescent computer with all the humor of an immature genius.

The Computer in Education

The fictional computer assumes various positions in education. In "Cybernetic Scheduler" by Edd Doerr, (in Van Tassel's TCG) a computer is given responsibility for scheduling all teachers and classes. But chaos develops when the computer informs some of the teachers they are unsuited for teaching and should in fact be taking classes themselves. In addition students are advised to change majors and drop out of school. In E.C. Weir's "What Happen to the Teaching Machine" (in Lewis's OMAM) people have forgotten how to ask questions because of teaching machines. In Isaac Asimov's "The Fun They Had" (in Asimov's SSFT) school children do not go to school, but stay home with their teaching machine. Two lonely children wonder what it was like to go to school with many other children.

Computers in Sports

Computer use in sports is rather rare in literature. But there is a good humorous book, The Last Man is Out by Marvin Karlins, where we find a computer run baseball team. The Electronic Olympics by Hal Higdon is another book about computer use in sports.

A good anti-machine book is Samuel Butler's Erewhon. In this book written in the 1890's a society destroys all machines less machines dominate humankind. In The Tale of the Big Computer by Olof Johannessen, we have a society where computers are the dominant species and the computers gives its view on the historical place of humankind.

Other Books

There are several other books besides this one that have some computer related fiction. Groff Conklin's Science Fiction Thinking Machines contains both computer and robot fiction. Damon Knight's The Metal Smile is a similar anthology. The Compleat Computer by Dennie Van Tassel is textbook reader about computers which contains both nonfiction and fiction, including some computer related poetry. Robert Baer's The Digital Villain is a nice little book which has a section on computer related fiction. Computer Computer: The Computer in Fiction and Verse by Dennie Van Tassel is an anthology devoted entirely to computers in fiction and verse. This book also contains some songs from the IBM songbook, which was used the 1940's.

Of Men and Machines by Arthur O. Lewis Jr., is an anthology about human beings relationship with machines. Finally check Marcia Ascher's bibliography at the end of this chapter. Many of the following books in my bibliography have been through both hard cover and soft cover editions but I will mention the edition I have.

Bibliography
Ascher, Marcia. "Computers in Science Fiction". Computers and Automation. November, 1973.

Asimov, Isaac and Groff Conklin. Short Science Fiction Tales. New York: Collier Books. 1963.

Baer, Robert M. The Digital Villain. Reading, Massachusetts: Addison-Wesley Publishing Company. 1972.

Barth, John. Giles Goat-Boy. New York: Doubleday Co. 1966.

Binder, Eando. Enslaved Brains. Avalon Books. 1965.

Conklin, Groff, ed. Science Fiction Thinking Machines. New York: Vanguard Press. 1954.

Crichton, Michael. The Terminal Man. New York:

Alfred A. Knopf. 1972.

Frayn, Michael. The Tin Men. New York: Ace Books.
1965.

Gerrold, David. When Harlie Was One. New York:
Ballantine Books. 1972.

Heinlein, Robert A. The Moon is a Harsh Mistress.
New York: G. P. Putnam's-Berkley. 1968.

Higdon, Hal. The Electronics Olympics. New York:
Avon Books. 1973.

Jerrold, David. When Harlie Was One. New York:
Ballantine Books. 1972.

Johannesson, Olof. The Tale of the Big Computer.
New York: Coward-McCann, INC. 1968.

Jones, D. F. Colossus. New York: G.P. Putnam's
Ssns. 1966.

Jones, D. F. The Fall of Colossus. New York:
G.P. Putnam's Sons. 1974.

Karlins, Marvin. The Last Man is Out. Englewood
Cliffs, New Jersey: Prentice-Hall, Inc. 1969.

Knight, Damon. The Metal Smile. New York: Belmont
Books. 1968.

Levin, Ira. This Perfect Day. New York: Random
House. 1970.

Lewis, Jr., Arthur O. Of Men and Machines. New
York: E. P. Dutton Co. 1963.

Van Tassel, Dennie. The Compleat Computer. Palo
Alto, California: Science Research Associates.
1976.

Van Tassel, Dennie. Computer, Computer, Computer:
The Computer in Fiction and Verse. New York:
Thomas Nelson Inc. 1977.

Vonnegut, Kurt, Jr. Player Piano. New York: Holt,
Reinhart and Winston. 1952.

COMPUTER POWER TO THE PEOPLE: THE MYTH, THE REALITY AND THE CHALLENGE

David H. Ahl, Publisher
 Creative Computing
 Box 789-M
 Morristown NJ 07960

The following is a lightly edited transcript of a presentation originally given at the "Man and the Computer" symposium at Dartmouth in December, 1976. Modified versions have also been given at several other educational and hobbyist conferences. Some 80 slides and graphics are used in the live presentation most of which, unfortunately, cannot be reproduced here.

INTRODUCTION

We all know that computers are around us, they're invading our lives along dozens of dimensions. We see them in super markets - the little product code that you find on the side of virtually every food and grocery product you buy can be read by an optical scanner connected to a computer. Computers in department stores - a little "magic" wand, actually a tiny laser device, reads a product code from the tag. Medical facilities - hospitals frequently keep all their patient records on computers. When you're admitted you often undergo some kind of questioning process. One psychiatric hospital out in Utah takes the entire patient history, and in fact, does the preliminary diagnosis of all entering patients by means of an on-line computer program. College admissions, for instance, Fort Lauderdale Community College, and hundreds of others use on-line computers. Every time you pick up the telephone and dial it you're actually using the largest general purpose computer in the world - the switched telephone network. Magnetic ink character recognition in the bank; sports stadium score boards; and so on.

My premise is that now, some 30 years or so after the invention of the computer, it's having a tremendous impact on our lives. It is having an impact on our lives similar to what the printing press did, but instead of taking some 400 years to make its effect known, the computer is having a vast effect in something like 20 or 30 years. We just can't escape it. So some thirty years after the invention of the computer we decided it would be a nice idea to find out what people think about computers. Do they view it as a master, a slave, a dictator, a monster - in fact, do people really understand what the computer is all about and what it's good for? We took a survey among both adults and young people with 17 different questions. We posed statements and asked them "Do you agree with this statement or disagree?" and got their responses. We also had some open ended questions and we continue to ask people open ended questions. Like, "If you had a computer in your home, what would you do with it?"

THE MYTH

First of all we asked some questions about what you might call the quality of life. Did people feel that the computer was going to improve various facets of society? Well, for the most part, there was pretty good agreement that computers would improve education somehow, a very substantial agreement that computers would improve law enforcement, a little less agreement, particularly among younger people, that computers would improve health care; and some agreement that computers are worthwhile for prevention of fraud through credit rating data. This last one is interesting. The question was asked in the AFIPS/Time Magazine survey just four years before this one; the percentage of people that felt credit checking was a good application dropped from 74% to 64%, so 10% more people today have some doubts in contrast to four years ago. I guess in four years many people have gotten stung in one way or another by credit ratings or other foul-ups.

We asked some questions about the threatening nature of computers. Do you feel you can escape the influence of computers? Well people for the most part felt that they couldn't; a surprising number of young people felt they could - I'm not quite sure where they were going to go to do it, certainly not the United States. There was some feeling, particularly pronounced among west coast respondents, that the computer could influence the outcome of elections. Senator John Tunney of California was one of the biggest critics of the use of computers to forecast the outcome of elections. Senator Tunney, if you'll recall, was defeated in November, 1976. I'm not sure if com-

puter projections had anything to do with his defeat but, in fact, his fear was that by the time the voters went to the polls in the western states the major national election was locked up. In 1976 it wasn't quite locked up by the time they went to the polls, but frequently it is and therefore people may say "why bother" or "gee, there's a bandwagon; I want to get on it and vote for the winner." Or, "I was going to vote for the other guy, and he has lost, so I can't be bothered going to the polls." Well that may not affect the outcome of the national elections, but it has a tremendous affect on the outcome of local elections and local bond issues. So, John Tunney at least was pretty upset about using computers in the forecasting of election results.

"Computers dehumanize society by treating everyone as a number." On that statement we had some ambivalence - some people agree, some people disagree - certainly a substantial number of people are a little bit fearful and do feel like the computer is dehumanizing things by treating them as a number.

We asked five questions to get at whether people understand the role of a computer. Do they really know what it's good for and do they know its applications? One of those statements was "Computers are best suited for doing monotonous, repetitive tasks." Well 80% of the adults agreed with that although only 67% of the young people did which gives rise to the hope that young people can see that computers are good for doing more than just dull, repetitive tasks. Are computers a tool? Yes - a pretty substantial agreement that they are a tool. I think that's a good thing. But I think it matters a lot whether people view it as an intellectual tool or whether they are thinking of it as a plain, ordinary tool such as a hammer, for example.

STATISTICAL RESULTS OF SURVEY OF PUBLIC ATTITUDES TOWARDS COMPUTERS IN SOCIETY				
	ADULT (N=300)		YOUTH (N=543)	
	Strongly or Mostly Agree	Strongly or Mostly Disagree	Strongly or Mostly Agree	Strongly or Mostly Disagree
<i>Computer Impact on the Quality of Life</i>				
• Computers will improve education.	86.6%	5.9%	84.2%	4.5%
• Computers will improve law enforcement.	81.9	3.3	70.0	10.1
• Computers will improve health care.	78.6	5.3	54.1	11.9
• Credit rating data banks are a worthwhile use of computers.	64.2	13.4	64.0	7.6
<i>Computer Threat to Society</i>				
• A person today cannot escape the influence of computers.	91.6	4.0	66.6	17.7
• Computer polls and predictions influence the outcome of elections.	48.1	27.5	44.2	26.9
• Computers dehumanize society by treating everyone as a number.	37.4	50.3	39.9	30.6
• Computers isolate people by preventing normal social interaction among users.	18.7	62.5	20.9	42.6
<i>Understanding the Role of Computers</i>				
• Computers are best suited for doing repetitive, monotonous tasks.	80.0	10.3	57.0	21.6
• Computers act a tool just like a hammer or lathe.	72.6	14.7	61.3	23.4
• Computers slow down and complicate simple business operations.	17.6	66.4	17.4	68.8
• Computers will replace low-skill jobs and create jobs needing specialized training.	71.0	15.0	61.8	14.4
• Computers will create as many jobs as they eliminate.	62.5	16.4	40.0	29.1
<i>Understanding of Computers</i>				
• Computers are beyond the understanding of the typical person.	25.2	61.6	30.6	49.2
• Computers make mistakes at least 10% of the time.	9.6	76.7	10.3	60.0
• Programmers and operators make mistakes, but computers are, for the most part, error free.	67.0	19.3	72.3	13.3
• It is possible to design computer systems which protect the privacy of data.	60.2	26.4	48.6	15.9

Do computers slow down and complicate simple business operations? Some people felt that they did - I'm not quite sure who. There's a substantial agreement that computers are going to replace a lot of jobs and create jobs that need specialized training, and some people really fear that they might not be qualified for the jobs that will exist after the "computer revolution". Also on the jobs issue we asked whether people feel that computers will create as many jobs as they eliminate? About two-thirds agree but that leaves a fair number that disagree. You have to remember that people have always been fearful of any kind of industrialization or technological breakthrough. The Luddites were anti-technology - to them the

industrial revolution meant the machines were going to take all the jobs. Well, it just didn't quite work out that way, and I don't really think computers are going to take all the jobs either.

Then we asked a couple of questions to see if people really understand the computer itself. We first asked are computers beyond the understanding of a typical person? The response was mixed. At least a quarter of the people think that they are beyond their understanding, but I'm encouraged by the larger percentage of people who disagree. "Computers make mistakes 10% of the time." You have to feel sorry for the 10% of the people who do think that computers make mistakes this often. In fact it is the programmers and operators who make the mistakes and not the computers. But in these questions we gleaned a little bit of intelligence that someplace between 13 and 19% of the people just actually don't know what's happening, don't know what computers are all about, and don't know who's running them. They think the computers are running the people rather than the other way around. A substantial number of people just don't know, which is also upsetting. So, there's a substantial portion of our society - at least a third or so - that just doesn't know some of the fundamental issues and facts about computers. We asked one last question - is it possible to design computer systems to protect the privacy of data? Well - not even the computer designers know for sure so I don't think we could expect much from people that we asked.

So all in all, we have some ambivalence - people optimistic on some counts and pessimistic on some other counts and some other things that they just don't know. The ignorance is probably most apparent when you ask someone what would you do if you had a computer at home? A computer? What do you mean a computer? You mean like a hand calculator? Some people thought we meant robots. "Well, maybe I'll have it serve me martinis when I come home from work." They just couldn't quite visualize a computer at home. A computer is supposed to be something that goes behind glass doors and is on raised flooring and requires a lot of electricity." I don't have the kind of home that would suit a computer," said one.

I guess this mixture of attitudes really shouldn't be too surprising. The every day perceptions of a computer are formed by people in the media and elsewhere who really don't know what computers are all about either. For example, newspapers, comic strips, TV, and so on. For example, what does a newspaper cover? They're going to report the computer error - the problem with the computer. For example, how many of you read a couple of months ago about the Shop-Rite in Springfield, New Jersey with brand new laser scanning systems at the checkout? This was the grand opening day and they really crammed the people in. I mean they had hundreds of people all filling their carts with these grand opening specials. People were lined up at the cash registers with two and three carts each laden with groceries. Seven or eight deep at every cash register and all of a sudden, bang, the system went down. Well, not only did it go down but it locked all the cash drawers. So there was no way of making change. They couldn't use the cash registers manually. There was just no way of opening them up. Rumors started flying around - people said, "The cash drawers are locked; the doors are going to lock too; we're going to be locked in here forever." And then there was a rumor that a replacement computer was going to have to be shipped in from Texas and they'd have to wait until it arrived! It was wild. Finally the manager decided that the best course of action was to give each checker a pencil and some brown paper bags which were at the checkout position, and have them add up manually the groceries in these laden carts. People were there for literally hours. The interesting thing is they did not lock the doors and more people kept streaming in. They didn't want to lock the doors because of this panicky rumor inside the store that if we lock the doors we might be stuck here. They didn't want to start a riot. Well anyway the newspapers had a field day with the story.

Most of you have heard about the frivolity out in Southern California when McDonalds had a sweepstake. To enter, all that was required was a 3 x 5 card of your own. Students at one fraternity programmed the computer to produce entry forms - 1.2 million of them and then they stuffed every ballot box of McDonalds in Southern California. They won 90% of the prizes in the contest. McDonalds was very upset about it - they said it was anti-American. I think it was very American; it showed a lot of ingenuity and creativity. In fact, McDonalds awarded duplicate prizes to people that were not members of this conspiracy to defraud them. The winning fraternity invited Ronald McDonald to make the prize presentations over at their fraternity house for dinner, but he declined the invitation. Actually, Burger King got the best publicity out of this. They gave a \$3,000 scholarship to the university in memory of the prank. Again, the newspapers had a wonderful time blaming the

whole thing on a computer.

A college student at the University of Arizona insured the life of his guppy. He put down all the correct information on the mail order insurance form - height 3 centimeters long, weight 30 centigrams and so on. It died of course, as most guppies do, some four or five months later. He submitted a claim for \$5,000 which was what he had insured it for. The insurance company said it was an invalid claim - the computer made a mistake by accepting this party. Well the computer hadn't made a mistake - it was a programmer who hadn't allowed for somebody that was 3 centimeters high. It wasn't the computer. But the newspaper, how did they portray it. Sure - another computer error.

In Swansea, Wales, a young man of 17 applied for a drivers license and passed his test shortly after. But when his license arrived, it bore 12 endorsements for a whole array of driving offenses, plus a 28-day driving suspension. Police proved sympathetic when it was found that "the computer at the license office had run wild. The system has not been operating for long," said an official.

There was a cute little notice printed recently in the Chicago Tribune. "A COMPUTERIZED bill had this notice on the bottom: Failure to receive this bill is no excuse for non-payment of the amount shown." Why capitalize "computerized"? Does that mean the computer printed that notice on the bottom of the bill. As if the computer could have made that up out of the blue sky? The computer is the scapegoat for the post office now - that's what's really happening!

A woman in Shreveport, La. got a gas bill for \$42,474.58. A customer representative at Arka Gas Co stated, "the computer went haywire and some of those bills got out." Computer error? Hardly. Good for the newspaper? You bet!

Movies are another way that people form perceptions of the computer. For example, in 2001, remember when Commander Bowman finally gains access to the memory banks after Hal has been harrassing him for half the trip and he yanks out the circuits one at a time. Finally, Hal breaks down as Bowman performs the first successful interplanetary lobotomy. The movie Colossus - have you seen that one? Colossus "wakes up" and gains sentience very much like the computer did in Heinlein's book, *The Moon Is A Harsh Mistress*. Well, Colossus gains it while it's hooked up to its Russian counterpart. The computers are in charge of the National Defense Systems of both countries and the two computers decide between them that it would be kind of neat if they held the population of both of their countries hostage. A movie that will be coming out shortly called Demon Seed has a computer in it, Proteus IV (appropriately named) equipped with an ominous blue arm enforcer with which the computer keeps people hostage, mainly Julie Christie in the movie (that probably makes it worth seeing even if you don't like computers). Three movies and three impressions of computers - all false.

Some people get their images of computers from books (not too many because not too many people bother to read books anymore). Science fiction writers are probably the one group of writers in the country that are portraying future computers uses reasonably realistically and making some half decent speculations. Unfortunately, very few people read science fiction so we don't have to worry about many people getting a realistic view of computers from that source.

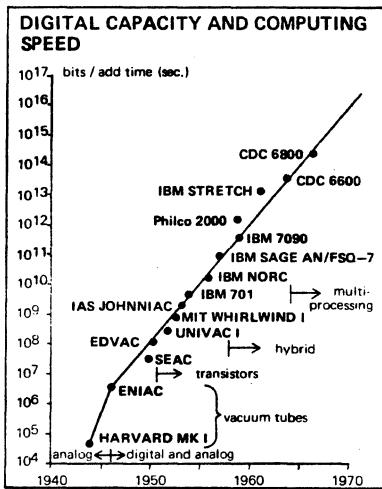
So, consequently we know a little bit from the survey what people think about computers and little bit of how these impressions were formed from my rather incomplete discussion of it, but I think you can fill out the missing pieces. We know too that if we ask the average person what would you do if you had access to a computer or if you had a computer in your home he really doesn't have a very good idea. In fact neither do many professionals or manufacturers. The fact is that we're really not very good a forecasting the future. We really can't and never have forecasted future technological innovation or invention very well.

Back around the turn of the century who would have forecasted life today as it actually is? In those days the best guess of what the Panama Canal would be was a railroad pulling ships across the isthmus. Back in those days it probably seemed reasonable. I'm sure if the Wright Brothers had asked the drivers of ox carts what they would do with an airplane, they probably couldn't have given them a very good idea. Henry David Thoreau, one of our leading philosophers commented when he was told that the telephone would permit people in Maine to talk to people in Texas, "but what does a man in Maine have to say to a man in Texas?"

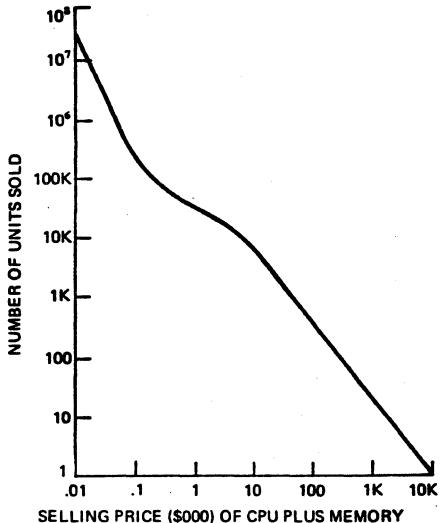
THE REALITY

It's pretty clear that we can't forecast 70 or 50 or probably even 30 years very well, particularly with a high technology item such as a computer. So let's just look 5 to 10

years into the future. Even so, we can't foresee exactly when everything is going to occur. We would certainly expect that processor instruction speed would continue to increase very rapidly. Packing density will also continue to dramatically increase. Currently, we are within two orders of magnitude of the human brain. Actually, the theoretical limit for semiconductor devices is packed more densely than the human brain.



Currently, bubble memory circuits in Bell Laboratories about 1 centimeter square will store about 1.5 million bits.



Coupled with miniaturization, prices are rapidly falling. Let me tell you that more than one manufacturer is a little bit alarmed at the projection of hardware prices approaching zero. The indication is that as the prices come down the numbers of units sold go up very dramatically. This applies not only to calculators but to computers as well. What happens as prices come down? What do you think the value of this ratio is today?

$$\frac{\text{Cost to program 1 line of code}}{\text{Cost to execute 1 line of code}}$$

One hundred to one? A thousand to one? Ten thousand to one? Wrong. IBM says the ratio is 100 million to one and that was two years ago! Given the current increases in processor speed, it's probably a lot more than that today. What that indicates of course is that the human element is by far and away the most important thing in computers and technology today in making them all work.

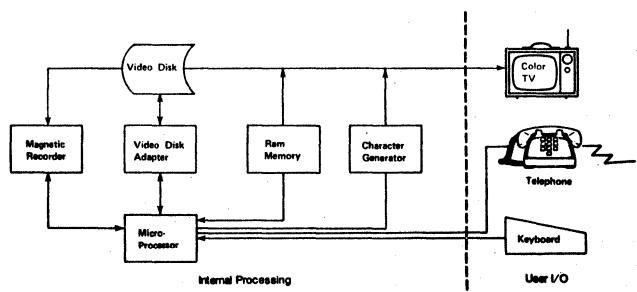
So what does all this mean when you put all this technology together? Well, obviously it means smaller terminals, terminals that fit in your pocket. Sophisticated and very small color video cameras. Calculators with as much power as a computer of 20 years ago. Hobbyist computer kits that are within the price range of a quarter of the households in the U.S. Close to 30,000 hobbyist computer kits have been sold as of the end of

1976. Technology means people talking to other computers and terminals or new high speed terminals or plasma panels that are built into your phone. A panel that can be written on with a light pen or typed on; or display information from a computer, data bank, directory, or from local storage.

Today there are over 100 manufacturers of personal computers and peripherals. At Creative Computing we can't possibly keep up with all the new product announcements for new hobbyist computer kits and peripherals. We started a new product section in early 1975 and the hardware portion was about one page. In the Jan/Feb 1977 issue it ran 9 pages of close spaced descriptions of new hardware. It's a revolution. Two hundred computer stores open now and a new one opening every four days. Retail computer stores where you go in you can buy yourself a microprocessor, a computer kit, or peripherals.

Extensible, user defined, simple languages are being developed. Harvard has a new language called ECL. It's not like today's simple language, say Basic or Logo. ECL doesn't use constructs that have to be absorbed into your intuition but rather you use concepts that are already part of your intuition, part of your language and then you construct the computer language out of that. Whether you're a banker, a baker or a professional programmer you can produce a computer language that does exactly what you want.

Out at Xerox Palo Alto Research Center they've got a thing called the "Dynabook". The original idea was that Dynabook should in every way be better than a book. It can display printed pages on its screen: black on white, white on black, red on green, etc. It can display pages in any style and size typeface. If you have some visual problems and you want a page in large type, it displays it in large type or for reference material it can use very small type. Not only can you read things but you can write things on it. You can just draw a circle around a word and move it to someplace else with a little arrow and the computer moves it for you. You can edit your material from a keyboard if you'd rather. You can strike over lines and they disappear. The next time you push a button you get all your text nicely justified on the screen. Actually, it's better than a book in every way because you read it, you can write it, and you can change it. It's also better from the graphical sense. It would be nice if the illustrations in a book could move (kids love little pop-up picture books). The illustrations on Dynabook can move with full animation. In fact not only can they move the way that they're programmed to move, but if you'd like them to move in some special way, you simply take your light pen and draw over the illustration and let the computer sequence through your frames. This is reality. This is here today. It's not quite the size of a book today; it's about the size of 3 bread boxes, but it's not going to be too long before it is the size of a book. In addition to having book qualities, it's also a general purpose computer with the ability to do parallel processing on eight different levels. When you think of it, that's the way human beings think. When I'm walking alone, for example, one part of my brain is thinking about putting one foot in front of the other, another part is thinking "it's cold out I'll be glad when I can get inside," another part is thinking about the speech I'm going to give tonight, still another is thinking about the beer that I'll have later on and so on. So your brain is processing information on a parallel basis all the time. Well, wouldn't it be nice if you could have a computer that could do that too and have the output of one level serve as the input to another. That's precisely what Dynabook provides. It's a phenomenal machine. I'd like to think that within 10 years it will be as commonplace as the pocket calculator is today.



Future Home Computer System

I feel one of the keys for getting computers into the home at least is the widespread availability of cheap high-quality software. One possible vehicle for bringing this about is the video disc. Quite inadvertently, a stroke of luck perhaps, the storage technique employed by the disc is binary, or digital.

Just what's needed for a computer. So while the player will be brought into the home for entertainment, it's real power lies in the fact that if you couple the video-disc system with a micro-processor and keyboard you have incredibly powerful audio/visual/computational/educational/recreational device. One video-disc can store 10^{11} bits of information, the entire encyclo-pedia Britannica for example, or a very comprehensive software library. You could have Jackie Stewart introducing the Monaco Grand Prix, taking you on a pre-recorded ride around the circuit and then turning the controls over to you. Or Kirk handing you the controls of the Enterprise just as the Klingons are about to attack. Or Fran Tarkington coming off the field putting you in as quarterback in the Super Bowl.

THE CHALLENGE

There's no question that in 5 to 10 years solid state and realted technology are going to put some fantastic things well within the reach of everyone who wants them. It's equally clear that most people have little idea of what they'd do with a computer if they had one. Hence, we have quite a chasm between the insiders (those who have learned about computers from school, work, or hobby) and the outsiders who don't know much about computers and don't really care (today).

It would be nice to think that this chasm could be bridged by education (like the new math or metric system?), but it's not likely that schools will really face up to computers until every kid has his own (pocket calculators all over again). Business and industry are so wedded to large EDP Systems with most DP Managers pretending that microcomputers are just toys, hence we can't expect any help from that quarter. Most likely it will be the people, plain ordinary folks, who see a friend with a computer and decide to get one of their own. And as this increasingly happens, we're going to have the most massive domino effect you ever saw - calculators and CB move over - you ain't seen nothin' yet. Computer power to the people is on the way!

PSYCHOLOGY AND THE PERSONAL COMPUTER

Kenneth Berkun
928 J Street
Davis CA 95616

Computers are useful as tools not just as an end in themselves, but also for two other reasons 1) for teaching or doing things (applications) and 2) for what they can teach us about mankind (womankind). I will discuss only briefly the first area, applications, and will spend the majority of my time on the second and perhaps more theoretical area of computers providing insight to people.

The study of psychology and applications of psychology do tie in to computers, and specifically they can relate to micro-computers in the home and in school. For instance, the famed programmed text developed by B. F. Skinner which lead to his invention of the teaching machine. A modern example of the teaching machine is the computer. This is evidenced by the success of the Plato system.

A micro computer in the home could be supplied with a bunch of lessons in prom plug in packages, and everyone being such good little students would flock to use them. They might also be networked with a central processor for record keeping.

Then there is a more esoteric aspect, that of computer counselors. The most famous example of that is Joseph Weizenbaum's program Elizah, and its "Doctor" version. This program simulated a Rogerian therapist and his active listening approach when talking over a terminal with a person. Very effective. Sort of frightening too. His views on the program are chronicled in his book "Computer Power and Human Reason".

And computers have one other advantage in the educational system than say textbooks, they're different, and fun! They hook people so that you want to use them. Flashing lights and buzzers may not be necessary but they sure are exciting. So kids may do much more work on a computer than they would anywhere else. This is a major advantage, at least until they become so common place as to be boring.

Now I will discuss the second area, computers as an insight to human behavior. Basic to this is the question of why study human behavior, or why is human behavior important. This is not the place to discuss that, I will assume that it is important. If you felt it is not, then you probably wouldn't be here.

One thing that we've all noticed in computers is that at times they behave very differently, for instance, sometimes they seem almost parental, scolding us for our errors, and billing us for their time, kicking us off for exceeding limits etc.

At other times it is very serious and gets the work done very efficiently, like a person who is reasoning with logic and care. And sometimes it just goofs off messes around and nothing seems to work, or it literally plays games like a kid! Looking at that I see three basic ways humans behave, emulated on their most human technological development.

This is important because it points out how man seems to be a technological creature by nature, and society does influence technology and vice versa.

There are a lot of similarities between computer operation and human function. It is interesting to note how systems are designed that way. For instance human thought processes run very similarly to timeshare systems, as far as central processor design. Also humans have the original interrupt system, as demonstrated by the "cocktail party" effect. I/O is also handled similarly, for instance by "polling".

With the advent of the micro computer we

see a new emergence of the compulsive programmer or hacker. These are also described in Joseph Weizenbaum's book. Now they can live with their computer eat with their computer and even sleep with their computer. But is that any different than say, the compulsive mechanical tinkerer, the ham of the early 1900's, or even the alchemists of medieval days!

Now how does this relate to personal computers? First and foremost it marks the invasion of the computer into one's personal life. Even more the Frankenstein complex is triggered. Will computers take over the world? Within ten years perhaps, every home will have a computer. We need to make computers more personal, more human and less scary. If people understand and/or can control them then they won't be as afraid. Things out of our control scare us.

Therefore I see the role of the personal computer as it exists now, the hobby computer, as opening the pathway to general acceptance, if not understanding of computers by the public. Computers are a way of life, more so even than T.V.

Ken Berkun is a senior at the University of California at Davis. His major is an individually conceived one consisting of Computer Science and Psychology, and called "Man Machine Interaction". He is a computer operator on a Burroughs 80700 at UCD. He is active at the Davis Counseling Center and the Sacramento Area Institute for Group Treatment, where he is in training. He is currently teaching a course on public speaking in Sacramento.

HUMAN FACTORS IN SOFTWARE ENGINEERING

James Joyce
Computer Science Division
Department of EE & CS
University of California
Berkeley CA 94720

This is a discussion of how human factors affect software development and integrity. My interest in this area was first focussed by the article "Guidelines for Humanizing Information Systems" in the November, 1974, issue of *Communications of the Association for Computing Machinery*. For this discussion I characterize as basic components of software engineering six aspects:

1. Human user interface
2. Maintainability
3. Portability
4. Efficiency
5. Correctness or reliability
6. Ethical considerations

Human interface, briefly, encompasses those aspects of a program a user experiences: protocols, error messages (if any), or uncertain silence broken only by the whirring of a fan. Maintainability is also, in a way, an aspect of human interface, but from the point of view of someone who fixes or modifies an existing program. Portability refers to the problem of getting a program that works on one machine to work on another. There have been many discussions of portability, but they center on hardware limitations or software techniques. I argue here that human factors impinge directly on the portability of software.

Software efficiency has received more attention than all of the other factors listed here -- either singly or combined. Gerald Weinberg [1972] has argued persuasively that efficiency is as strongly influenced by human factors as any other component of software engineering.

I have combined the questions of program correctness and program reliability because they seem fraternal twin aspects of a central source of worry: how can I depend upon the answers I get? Yet they are separable as well. For example, that the program actually meets all the requirements of the problem is a matter of program correctness [Elspas, 1972], but how often the program crashes is a matter of reliability.

The last component of software engineering to be discussed here, ethics, is one that appears the least rooted in reality. Questioning the ethics of writing or copying software seems as ethereal and philosophical as questions about whether the person who commits robbery is the true criminal or whether the true criminal is Society -- that is, unless you happen to be the one who is robbed. We will examine the ethical aspects of software, particularly fraud and copyright protection, and a possible solution to piracy of software.

1. Human User Interface

1.1. Error 477

Perhaps the component of software engineering in which human factors are easiest to recognize is human user interface. Anyone who has been programming for any length of time eventually elicits as output what I call Error 477. Just what Error 477 is remains a mystery, because "Error 477" is the entire message; and even if there is a reference manual with a table of explanations of the messages, Error 477's explanation is as cryptic as the error message itself.

Partly human-oriented messages, such as those by Li-Chen Wang in Palo Alto Tiny Basic [Wang, 1976], are still at the "Error 477" level. His three error messages, "How?", "What?", and "Sorry" are, it is true, differentiated by the kind of error condition that evokes them. And chatty error messages would undoubtedly swell Tiny Basic into something much larger than at present, thus making it less available for machines with small memory. However, the program could have been quite a bit more helpful if it had printed, in addition to the error message, a character on the line under the Basic statement in question to indicate where it stopped translating. Although no guarantee that the error is truly in the indicated location, it is nonetheless quite a bit of help that can be provided at a small increase in program size. This is not an attack on Wang; I chose Tiny Basic as my example because I admire the job Wang did very much. It is, I feel, one of the better representatives of assembly level coding practices today.

Donald Knuth's Turing Award Lecture to the Association for Computing Machinery contains a plea that goes further than mine about error messages; he urges programmers to make programming languages and even control languages "a pleasure to use, instead of being strictly functional." [Knuth, 1974] I fully agree with Knuth's statement, but am willing to settle for good error messages. Objections to the effect that elaboration of error messages would consume large amounts of main memory are, as conceded above, true. But just as writing a program so it will run in a small amount of main memory is seen as a challenge, so too writing a program so that it is a pleasure to use can be seen as a similar challenge. After all, the personal computer movement seems to me to be an insistence that computer technology is not the private domain of institutions -- whether business, educational, or governmental -- but is available to people. It follows, then, that the error messages should not be bureaucratic gobbledegook, but human.

Making the human interface relate not to the machine's requirements and limitations alone, but to the user's desires, seems a continuation of the same philosophy that began personal computing. From a strictly commercial standpoint, a large number of potential computer users will not purchase equipment and

software until it is easy to use for the person who has never held a soldering gun nor looked up an error message. And we cannot expect people to memorize commands willingly; they will always manage to forget something that is hard to remember -- as they should.

A discussion of why people forget would take far too much space here, although some basic reasons can be stated briefly. One is that we forget because there is too much to remember. In a reasonably complex program (the kind we feel good about writing) the individual parts may at one time have been known so well we could recreate the code at will. But by the time we have created a large enough program, the parts we worked on earlier begin to fuzz and fade. This is not due to time, but to attention to something else -- either another part of the program, or another program. Yet time, too, enters as a factor in forgetting. In terms of human interface we may forget (either from time or other pressing details) just how something was to work. Typically this occurs when we are explaining to someone else how to use a program. We forget a detail, a restriction, and if we are lucky we remember it correctly when we are telephoned by the person who believed the first explanation was all there was to it. But surely these remarks cannot be applied to that large number of programs that are "one time only"!

It has been my experience that even my most casual program is something at some time I may wish to show off; this, regrettably, does not mean every program is all that good, but that what I wrote as a one-shot program is inevitably something someone else wants to use or do. A well-done program must be able to be used without arcane knowledge. It should even be helpful to users wanting to use it.

1.2. Help and Control-d

The first test I apply to any program is to input a one-word message: help. A system that cannot help is at the mercy of misunderstanding just as much as the user who receives Error Message 477. Help may be offered in many forms. On large computers I have seen it expressed as a telephone number to call (which is not all that much help at 8:30 pm -- and less at 3 am), a menu of arguments to use with the help statement to receive specialized help (as in "help files" for information on how to move, remove, create, etc.) files, or a short summary of the available commands, or a tutorial without menu. I regret that I have not yet seen a microcomputer program that responds to help other than to issue Error 477 in any of its many equivalent forms.

One area of human interface in which microcomputer programs are, on the whole, more advanced than programs for mini or larger computers, is in free-format input. Sometimes, unfortunately, free format is interpreted to be that the data items may be any length, but if they are separated by more than one blank then things will not work properly. Being able to ignore redundant blanks is simple and makes input much easier. Reluctance to make such allowances seems hard for me to distinguish from laziness. Similarly, having to enter commands or data in specific positions is a requirement always for the convenience of the programmer and not for the user. Let me make that point more strongly: rigid input specifications are for the convenience of the programmer writing the program and not when s/he uses the program. Although software for microcomputers is ahead of that for the larger computers in emphasizing free-format input, it shares with larger computers the mystique of control-d.

To those of us who are used to the control key on a terminal, a fuss over control-d may seem unwarranted. Yet in teaching naive users (and we were all at one time naive users) to use a text editor I have found those commands requiring the control key with

another (and sometimes two others!) the hardest to make into second nature. This is not to urge a ban on control characters, but to suggest that before control-anything is made a command there should be serious thought about other ways of signalling what is intended. The same can be said of the "off" combination that allows graceful exit from APL; ideally, commands should be something the user can guess rather than something to memorize.

1.3. Guidelines for Interactive Systems.

A student of mine [Tossy, 1977] in a seminar on human factors in software engineering came up with several ideas for designing interactive programs so they are easy to use. They are:

1. Command names should be mnemonic.
2. A frequently used command should have a short name or an abbreviation.
3. The program should always accept the full length name of a command, even if a standard abbreviation exists.
4. Both the full command name and any standard abbreviations should appear in the index of a user's manual.
5. Make provisions for a user becoming more sophisticated about a program as s/he uses it.
6. Protect the user from the results of "dumb mistakes."

He generalizes his rules into two principles:

1. Make the program forgiving; humans make mistakes.
2. Make the program convenient; humans are lazy.

What programmers seem to forget, time after time and program after program, is that the two principles apply equally to both sophisticated and naive users -- to you and me, for example.

Such concerns lead naturally into a discussion of another factor in human interfacing: that awful wait while the panel lights blink, during which the most hardened of programmers wonders whether the program has found a new infinite loop or is only setting all of main memory to 7s. Much of the anxiety of waiting is not hearing anything. In Samuel Beckett's play "Waiting for Godot" the wait endured by the two protagonists is eternal, unending even when the play ends; but to some extent they are kept waiting by the appearance of Godot's messenger. The message he brings is, simply, wait. Some such message from a program that runs longer than 10 seconds (say) might be a good idea -- as well as the capability of suppressing such a message if that is desired. For the human user's sake there should be some indication the machinery is still working, still "there". Thus I would like to add a third principle to the basic two given here:

3. Make *something* happen; humans are easily bored or made anxious.

2. Maintainability

Strongly tied to human interface with users is the problem of program maintainability. Error 477 enters our discussion again, this time not as a message we have generated but as a system message that lets us know there is something to fix. The first question is not the logical one of "What is it I have to fix?" but "How do I fix it?" If this question seems out of place, we might note that it is really a more sophisticated version of "help!". How do I fix it may be answered variously by identifying the problem and then fixing it, or asking for help in either locating the problem or in fixing it. No one who develops software really wants telephone calls from users who cannot understand why the editor they are using suddenly goes into a loop -- or simply stops. No one who programs

wants code written six months ago to stop working. I could be somewhat cynical and say six weeks rather than six months, but it does not seem necessary for the point.

There are, no doubt, some individuals who are so good at programming that the very code communicates its purpose to any reader who knows the programming language. I believe once or twice I have known such people; the world needs more people with such ability; I wish I had such ability. And there are some very impressive higher-level languages with block structures that encourage structured programming. Although we know structured programming is not enough to make a program readable, it is evident that structured programming helps readability [Knuth, 1974]. So far, higher-level languages are, by and large, unavailable to microcomputer owners; such languages require too much main memory or an auxiliary storage device, such as a disk, for successive passes of the compiler -- or simply cost too much. Microcomputer programming is done in either assembly language or Basic by and large, and remarks about maintainability are directed toward what is possible within those languages.

2.1. Routine Size

In structured programming a key concept is that of breaking the larger problem into modules that are in themselves complete; that is, the module *does* something. Sometimes we find a module that does too many things, which we perceive as the product of an inadequate application of structured programming. When we are designing a program we may manage to create modules that do too much. How much, though, is too much?

I would argue that "too much" is as dependent upon physical size as it is upon conceptual size of the module. Yourdon [1975] suggests a size limit in his recommendation that "at each level of the design, try to express the implementation of a module in a single page of coding or flowcharting." What we are able to perceive at one time strongly influences what we can process in our minds. Thus modules should never be larger in size than the crt screen can hold, or one 8 1/2 by 11 inch sheet of paper. My choice of the American standard paper size is based on the nearest hard-copy analogy to crt screen size. I would gladly relax my pronouncement of page size to my true intent, the unit of page size on the hard-copy device, but in the case of teletypes that leads to an obvious absurdity: the whole roll of paper. The routine size of one page or screen includes all comments associated with the routine. There are few things more irritating than flipping between pages or trying to squeeze two screens of code onto one screen.

2.2. Comments

Having introduced the topic of comments into the subject of human factors in software maintainability, I think it only fitting to discuss commenting in greater detail. I have heard people who should know better claim that a well-written program needs no comments. It is possible that a particular well-written program -- even one in assembly language -- is so clear it does not require comments; yet I know of no technique or language or attitude toward programming that would make comments truly unnecessary. Of course, it is also possible that too many comments can obscure the code they were created to explain. Comments require an effort to write that seems quite apart from the effort to program, and for that single reason they often do not get written. Other reasons a programmer might offer for not writing comments include:

1. They take up too much space in the program.
2. They only repeat code anyway.
3. They should be put in after the code is running or

else they will be just another thing to change during debugging.

The list of three is not exhaustive, but it is representative.

The reason comments seem to take up too much space is that usually they are not properly integrated with the code; the role they play in understanding the program needs to be essential rather than documentary. If the comments echo the code they will at best be documentary, and perhaps a nuisance to be ignored while reading the code. Bad comments are the result of the programmer's self-fulfilling prophesy: seen as taking up too much space and repeating the code, comments will be produced to do just that. Comments are a feature of programming languages, and like any other feature they should be used when needed. The problem with comments, then, is not in their number or size, but *when* in a program they are needed. The function of a comment in a program is to help remember what is going on [Weinberg, 1972], and thus they should be introduced as a part of the code.

To some extent the programming language itself (excluding machine language) helps us remember what is going on. Commands are imperative utterances to do something; declaratives identify elements in the program, such as variable names, transfer labels, or routine names. Between commands and declaratives we may indicate what is to be done to (or with) what items, but not why. And it is "Why" that we ask when we read uncommented code: why LAI 002 -- load the value 2 in the A register of the 8008 microprocessor cpu? The activity, though complete enough for a computer, is not complete enough for the human; to complete the instruction it is necessary to supply motivation, a reason. The feature for explaining things in a programming language is a comment.

While programming the programmer has the reason for writing particular code in mind. That reason may be the wrong one for the problem, as when we code a bug into our program, but it is a reason nonetheless. Without the reason for the code clearly indicated, the program literally has no reason for "running" in the programmer's mind. The program does run (or at least is emulated) by a human when the code is read or written: the value 2 is placed in the A register of the 8008 cpu which exists in the programmer's imagination. At the moment of coding we supply the reason, the comment, in our mind; it is, then, an inaccurate transcription of our code to omit keying the comment when we key the command or declarative. As elsewhere in computing, the term for that inaccuracy is "bug".

Further, comments that echo the code are buggy comments. "LAI 002" is not the *reason* we code "LAI 002". No action is its own reason, even if one is not speaking of computer programs. It is a part of good programming to express the comment as correctly as the command or declarative. This appears to call for a comment for every command or declarative, and for some routines I do not doubt that necessity at all. We also know that there are many instances in which such commenting would be too much. How do we know? I believe the answer to that question is the same as the answer to "How tricky is too tricky when writing code?" -- another question I cannot answer in words. But, through experience, we may learn the answer to both.

One technique for creating comments is to think

to *comment giving reason*,

command

or,

command

to *comment giving reason*

or, for variety,

command

because *comment saying why*

or perhaps

because *comment saying why*,

command

Naturally, all those *tos* and *becauses* would be tedious to see on our crt, and the ability to avoid repetition is a test of our ability to write good code. The important thing to remember is that the function of the comment is to provide the reason or motivation for the code, and we should choose the wording for a comment accordingly. Such an attitude toward comments will undoubtedly make manufacturers of ram, paper tape, and teletype paper very happy and the rest of us pleased at the insight -- but with one more thing to remember to do.

There is another real advantage to writing the reason along with the action, one we recognize in other forms of thinking. When something, some problem, is expressed in a form external to our mind we tend to get a clearer idea of what the problem is and are then less likely to make a mistake based on not thinking the thing through. Or, if we do make a mistake, we stand a much better chance of identifying the mistake (and thus correcting it) if it is in a form external to the mind investigating the mistake. In plainer terms, it is easier to spot a flaw in thinking if it is down on paper or up on a crt. This is certainly one function of a project log, used in many large-scale systems development projects and by some of us working on individual projects as well.

The value of writing the reason along with the code and rereading such reasons along with the code later seems to be so obvious I mention it here only to have said it. The question I find myself asking (when I ask questions of programs) is "Why is the code there?" I know perfectly well what the statement does (if it is a language I know), but not why it is being used.

2.3. Variable Names

Since I am arguing here that the statement and its reason should be represented together in a program, it follows that just as the comments should be written clearly, so should the statement. We generally have no control over the operation code mnemonics in an assembler, but we do have control over the variable names. Everyone, it seems, agrees that variable names should be mnemonic -- an aid to memory in identifying their purpose. However, as Weinberg [1972] warns, mnemonic symbols expose us to error because

1. They tend to make programs seem "sensible" by their satisfaction of our general preference of sense for non-sense.
2. They play upon our tendency to believe in the name, rather than the thing named; consider the seductive nature of a variable named FIVE. What value does it contain? Are you *sure*?
3. Similar names can be confused; for example, SQUARE for the square of a number and SSQUARE for the sum of such squares differ by only one keystroke.

Exposure to error is perhaps preferable to no idea what the variables in a program mean. If names in a program are assigned in alphabetical order of occurrence, we will find it difficult to interpret the value we assign to the variable D. Weinberg's point is not that

mnemonic names should be avoided, but that they are not a fool-proof solution to symbolic names. The problem of mnemonic names in Basic is a non-problem in that the names may be only one character in length -- with a \$ suffix if the variable is to hold character strings or an expression in parens if the variable is the name of an array.

3. Portability

Mnemonic symbols, meaningful comments, and structured programming are aids to maintainability and also portability. If a program is good it will be one that gets shared -- or rather, one that we will want to share. Even if we are not the sharing kind, it will be one we want to take with us when we move onto another computer. Even assembly language programs can be made fairly portable if there is macro capability on at least the new computer. The human factors in portability may be viewed through three concerns: equipment considerations, relocatable code, and the ability to "plug it in" and have it run.

3.1. Equipment

Equipment considerations are perhaps more financial than human factors; but where there is something financial there is something very human at work: a fallacy I call the "Just As Good As" fallacy. This fallacy has several variations: a system with 12K of ram is just as good as a system with 16K or more because if you are good enough you can do anything in 12K; and, this teletype is just as good as the other one with upper and lower case because you really don't need lower case; and so on, with examples being possible from all areas of human choice. We compare what we are willing to settle for (or afford) with what we would rather have, and deduce "just as good as". It is a natural defense mechanism to try to keep us from feeling we have let ourselves down. Of course, "just as good as" is a letdown; the easiest measure of the truth of this is the amount of emotional heat with which someone who has settled for "just as good as" will defend the decision against someone who has the bad manners to suggest otherwise.

In personal computing, as in stereo systems and automobiles, our taste (or lack of it) must be balanced by our bank account or line of credit. Computer components are dropping in price, but they are still expensive enough that software requiring 16K and a floppy disk will find fewer systems to run on than one requiring 16K and audio cassette. Actually, the target software for microcomputers presently seems to be an 8K machine with paper tape reader or audio cassette. Even as I write this I am aware that equipment costs are bringing "just as good as" and "would rather have" closer together, and before long they may be as close together as "just as good as" and "would rather have" in the case of stereo equipment. Until then, software must avoid dependencies on expensive or extravagant hardware if it is to be exportable to many microcomputer users.

3.2. Relocatable Code

Relocatable code is a topic more obviously a matter of software than are equipment considerations, and is generally associated with large system software rather than with microcomputer software. My use of the term "relocatable code" refers to code that does not rely on being loaded into the same address space to be run each time. The difference between relocatable and non-relocatable code may be illustrated in terms of how a particular memory location is referred to. If we declare a location with the name K, some assemblers associate K with a specific address, such as 764. Every time the symbolic name K is used in an instruction the assembler substitutes 764 for K. The resulting program is

dependent upon K being at position 764, and all other parts of the program are tied to specific locations as well. But if the assembled code does not refer to 764, but to 764 plus a value in a register that is the address of the beginning of the program, the code can be loaded into any segment of memory and the addresses will be resolved as offsets from the beginning of the program. Such a register is being used as a "base register"; base registers are in wide use on larger computers. On microcomputers I know implementation of a base register for relocatability is a software feature, not a hardware feature.

Base registers imply there is a need or a desire to load a program into various locations in memory, and such a need has not yet been demonstrated here. Interfaces between devices on some systems require certain absolute locations in memory for buffers, status registers, etc. If on a given system we want to load a program at a particular location and a device interface requires the same location we may find that the device, the program, or both may do unexpected and possibly unwanted things. The easiest solution to such a problem is to eliminate the problem. If the code can be relocated by an appropriate value in the base register then the conflict over particular locations does not exist. Writing a program so that it is relocatable heads off future complications over where the program has to be in main memory in order to run.

The first two aspects of portability, equipment limitations and relocatable code, appear to be arguing for practices contradictory to each other. We must keep in mind that most microcomputers have small main memories, but we should also write code so that it may be run in any location in memory. The synthesis of these two concerns requires the belief -- justified by past performance -- that memory costs are coming down. Two 8K relocatable programs could be loaded in 24K with room to spare, and one or the other run as needed without having to load and reload code. For example, with the code for Shooting Stars and a text editor in memory a user could switch between them while writing a paper, thus time-slicing work and play. If neither Shooting Stars nor the text editor are relocatable, all the memory the machine can address will be of no help.

3.3. Plug it in and Go

When I began as a professional in computing in 1967, IBM was making great claims for upward (and parallel) compatibility. This meant that a program running on an IBM 360/30 would run on an IBM 360/67. Generally that was my experience, and apparently others also found it a strong selling point for the IBM 360 series. For microcomputer software the corresponding strength might be in self-loading programs. That is, programs arrive with a bootstrapping loader on the front as a part of the program so that the overall effect is that the program can be "plugged in" and run with minimum user involvement. This may appear to contradict the spirit of personal computing, of wanting to do things with one's own computer. But the owner of a computer does not really want to do everything, just those parts s/he wants to do. Getting ready to play with a program is nowhere nearly as enjoyable as playing with it -- especially if some unexplained detail must be guessed by the user before the program will load properly. What may conflict somewhat with the "plug it in and run it" aspect of software portability is the problem of relocatability. But then, if two programs are to be in memory at the same time it is likely only one of them really has to be relocatable. Besides, there is nothing that prevents a self-loading program from using a base register.

4. Efficiency

Being able to load and execute a program in one step may seem more an aspect of overall efficiency than of portability, and perhaps the discussion should be continued as an aspect of human factors in efficiency. The program that handles details for the user is intuitively more appealing than one that requires attention to housekeeping details. Such a factor accounts for many users' preference for a higher-level language over an assembler, and the overwhelming preference of an assembler to machine language. This sense of the word "efficient" is not the usual sense when one is discussing a program, however.

4.1. Fast Programs

I think it is fair to say that when most people speak of an efficient program they are referring to a fast program. Stories of programmers who spend a week to save a few cycles of cpu time are legion and share the same moral: be sure the time being saved is worth the effort and time to save it. For microcomputers the important unit of time is not the machine's cycle time or the time it takes to add 1,000 numbers together, but whether a human user notices a delay in response. This does not mean sloppy programs are efficient if the user does not have to suffer through a slow response, but that before a major effort is invested in optimizing a section of code the need for such optimization, in terms of the effect upon the user, should be established. If we are writing a program solely for our own amusement we can do anything we want in pursuit of that amusement; but I have already remarked on how programs we thought were one-time only seem to turn into programs we would like to share.

When I teach an assembly language programming course I carefully avoid any lectures on efficiency, yet my students turn in programs with notes calling my attention to how a particular programming trick saves time, code, or both. What is at work is a human factor, pride in one's skill as a programmer, that leads most people I have taught to find ways to make their programs run fast. Such pride can, however, turn into an obsession that impedes the progress of the very program being developed with such pride.

4.2. "Tight" Code

Another human factor relating to program efficiency serves as a summary point in our discussion of efficiency. As Weinberg [1972] puts it, "when we ask for efficiency, we are often asking for 'tight' coding that will be difficult to modify." The coding trick that seemed such a stroke of genius at first seems inevitably to turn into a stumbling block in understanding the code when it is to be changed or fixed later. This does not mean we should strive to be inefficient, of course.

In *The Elements of Programming Style* Kernighan and Plauger discuss efficiency considerations, arriving at the following rules relevant to us here:

1. Make it right before you make it faster.
2. Make it fail-safe before you make it faster.
3. Make it clear before you make it faster.
4. Don't sacrifice clarity for small gains in "efficiency".
5. Keep it simple to make it faster.
6. Don't diddle code to make it faster -- find a better algorithm.

The occurrence of the construction "Make ... before you make it faster" (or something very similar) so often in this list certainly seems to place efficiency in a subordinate position to other concerns. Such subordination does not mean efficiency is unimportant, but that for efficiency to be most valuable in a program

the other criteria must be met: correctness of the program, the program's ability to recover from error, the clarity of the program, and overall simplicity of the program. I have taken the liberty of interpreting "Make it right" as something speaking about program correctness because I believe that is the thrust of that point, and also because program correctness is the next topic in this discussion.

5. Correctness and Reliability.

As was said in the introduction, program correctness and program reliability seem to me to be fraternal twin aspects of a central source of worry: how can I depend upon the answers that I get? If a program is correct (that is, it has no bugs in it at all and works for all possible input) it will be a reliable program; unfortunately, a program that is reliable may yet contain a bug that has not been found, and therefore is not a correct program. But such fine distinctions aside, both program correctness and program reliability have in common the paramount concern that the program work properly for all data that is given it.

The area of program correctness is rather difficult to summarize, but the tutorial in *Computing Surveys* [Elspas, 1972] is the best attempt I have seen. Why the area of program correctness is so difficult to summarize may be found in the approach of those working in that area: they are trying to *prove* programs are correct. The point of being able to prove programs are correct has been summarized by Elspas as "It has been the hope of many practitioners and users of programming that the development of the programming art into a science would have the effect of relegating programming errors to the minor nuisance category." The techniques for proving program correctness are ingenious, but they are too complicated to be used on most programs even by those who make proving program correctness their specialty.

Proving programs correct is a rather mathematical endeavor, but that is not why it is a complicated matter to prove a program correct. We simply do not yet see how to prove programs are correct in any reasonably short manner at the present time. Such a lack of easy to apply techniques must be accepted as a fact of life for the time being until they are developed. That may be a long time away, warns Elspas, who does not foresee "any fundamental breakthroughs that will dramatically simplify the process of verifying a program." We can want correctness in our programs, but we must settle for reliability.

Software reliability may, for the sake of discussion, be divided into two concerns: will the program/system crash, and the documentation of bugs. A program with only one bug may sound like a very reliable program, but if that bug is in the code that handles carriage return we would not be very pleased. Every language translator I have used ran with known bugs, but the bugs did not occur so often that they interfered with typical processing. The translators were, despite their bugs, reliable -- within certain limits. As the bugs were discovered they were handled in any of several ways: they were fixed; they were scheduled for being fixed at some future time; they were reported to the person responsible for maintaining the translator; or, they were published (sometimes in the form of a hastily written note).

People who do not program, or those who are rather new to programming, find any of the alternatives other than fixing the bug most curious. However attractive fixing the bug may appear, we must be sure of the impact of the fix on the program as a whole. If the program has been designed with maintainability in mind, a change in the code intended to fix a bug will not create strange surprises in code within another routine even if that change is not correct. It is all too human and all too common to perceive a par-

ticular area of code as the problem area, to write code to fix the problem, and to accept that code as fixing the bug even though it is not tested carefully. Stories of such behavior on the part of individuals are common enough, but although I should know better it always seems to surprise me when I hear of a software company doing just that.

The motivation for changing the code is laudable for, after all, the idea is to get the program working again as quickly as possible. But an ill-considered change can make the software even more unreliable than before. We must balance how many bugs are in the software against how often they are encountered in use.

Van Tassel [1974] suggests several criteria for judging program reliability:

1. Mean time between errors
2. Mean time to repair errors
3. Percent of up-time for program
4. Number of bugs vs calendar months

Of these, perhaps numbers 1 and 4 are the most meaningful to us. They might serve as good starting questions we might like to put to software vendors who want us to buy a compiler. This assumes that the vendor would give us correct answers to our questions, an assumption that apparently is not always the case. Such a discussion, however, is better carried out in the next section of this paper, which is concerned with human factors and ethical considerations.

6. Ethical Considerations.

There certainly is no need to argue that human factors have a direct effect upon ethical aspects of microcomputer software. Such an effort is obviously present whether or not we feel ethical concerns derive from an absolute, eternal set of values or not. Ethical considerations include (but are not restricted to) fraud and violation of copyright.

6.1. Fraud

A typical dictionary definition of fraud will read to the effect that intentional perversion of the truth in order to get someone else to part with something of value constitutes fraud. One might argue that fraud is too strong a term to use regarding someone who makes claims about performance that the software cannot meet. Perhaps the term should be *lying*, not *fraud*. Software that fails to live up to promised performance may do so in degrees varying from small glitches to defying anyone to load it; surely the one word fraud does not apply equally to all. Of course, it does not. And deciding just when and where it does apply legally is a matter for someone in law to decide, rather than someone in computing. But I make no pretense of deciding a legal matter; my concern is ethical. It seems to me that anything software is advertised to do should be something that was thoroughly tested and debugged. When we buy something we naturally assume it is being offered in good faith: that it *works*, at least in the aspects advertised, and one can expect reasonable performance from it -- such as being able to load it. As more people enter the microcomputer software marketing field we are seeing a dual effect on software quality; competition brings pressure to bear to create better software, and the desire to be competitive is accompanied by a temptation to stretch the truth a bit about what a program will do. As users of software we have more than the old saw *caveat emptor*, let the buyer beware, to help us: we have the hobbyist and personal computing magazines.

Magazines such as *Personal Computing*, *Byte*, *Creative Computing*, and *Dr. Dobb's Journal* can and to some extent do serve as a

guide to available software. As microcomputer-oriented magazines continue to develop I would expect reviews of software to become a standard part of every issue. This may have the effect of concentrating power in the hands of a few at first, but in time reviews of software would come to be regarded in the same light as reviews of books: not the last word necessarily, but a guide to what is around. This is not to say that one should buy software only if it is favorably reviewed in a computer magazine. It is to say that a search of the literature may save the software consumer much frustration and disappointment. Searching through all issues of the various hobbyist magazines is lengthy, tedious work, and the idea of waiting for *Reader's Guide to Periodical Literature* to get around to listing all the hobbyist magazines does not appeal too much. But there is a guide published to the literature we can consult now: it is *Periodical Guide for Computerists*, by Eldon Berg, that indexes over 1,000 articles and letters from fifteen magazines read by computer hobbyists. The table of contents is quite extensive, and definitely saves hours of search through back issues. Berg is publishing the guide semi-annually, in July and December. Those wishing a copy of the guide may write him directly at the address given in my bibliography at the end of this paper.

6.2. Copyright

Our concept of copyright comes from similar English laws and licenses dating from approximately 1518. The United States copyright law is so complicated that I have had directly contradictory legal advice from two very competent attorneys. Both agreed that on the subject of computer use of copyright materials the law is rather grey and difficult. The purpose of the law is to protect a person's property from being taken by others without proper compensation. That is, books, documents, songs, recordings, etc. are considered the property of the copyright holder. Through copyright provisions writers, composers, and others are given protection for their work so that royalties are paid to the person who deserves them. This is, I hasten to add, not a legal opinion but an ethical perspective on the topic. Donald Knuth writes most eloquently and from the same non-legal status about legal protection of algorithms in *Sorting and Searching*, volume three of *The Art of Computer Programming*. Knuth argues that algorithms should not be anyone's property any more than a mathematical result is. This seems very reasonable to me and right in line with the contention, backed by legal precedent, that ideas are not protected by patent or copyright whereas devices or publications may be. I also believe that if someone writes a computer program and wishes to sell it to others then that person should be protected from piracy.

The revised copyright law, effective 31 December, 1977, contains wording that clearly intends to cover computer programs: "Literary works" are works ... expressed in words, numbers, or other verbal or numerical symbols or indicia, regardless of the nature of the material objects, such as books, periodicals, manuscripts, phonorecords, film, tapes, disks, or cards, in which they are embodied." The old law had to be interpreted to cover modern technology; the new law attempts to anticipate new developments by saying "Copies" are material objects ... in which a work is fixed by any method now known or later developed, [italics mine] and from which the work can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device."

The question of whether this law will indeed provide proper protection will have to be answered in time in the courts. However, whether the law can be effective against individuals who pirate software for personal computing can be answered, in my opinion, with a simple "No." First one must find the guilty party, and then one must take that party to court. The expense and time

required for such investigation and prosecution is staggering to contemplate. Although the average income for computer hobbyists is higher than for, say, the average citizen of the United States, recovery of the expenses involved in prosecuting software piracy would be hard to do -- perhaps harder than getting the judgement to begin with.

This is not to be construed to mean that I advocate violation of copyright; I decidedly do not. If someone is selling a program I want I have two ethical choices: either pay the asking price or write the program myself. However, one would have to be shy of basic intelligence and have never read any personal or hobbyist computer magazine to believe that everyone operates with the same ethics. The problem of detection and prosecution is too large, and the remedies too difficult to extract to halt violations of copyright through legal channels. What, then, is to be done to encourage software development and guarantee that such development will be protected if protection is desired?

Digital Group Software Systems of Denver, Colorado, appears to have a solution. By pricing its software so low that it is virtually the price of a tape cassette, the temptation to steal a copy has the cost motive taken out of it. After all, why steal the programs by copying them to a cassette when for essentially the same price they can be acquired legally? And with a copy of the documentation, no less! In a discussion with the President of Digital Group I learned that pricing software at \$5.00 per cassette does not mean they lose money on software. The volume of business they do makes up for the low price, and the low price encourages people to buy from them. Although it does not say much for strength of ethical feelings in humans, Digital Group's experience does indicate that when the cost is pretty much the same that people will buy rather than steal. The temptation to remark "how very human" is too much to resist.

7. Conclusion

This discussion of software engineering has had one major point to make throughout the discussion: no matter what area of software engineering one considers, there are human factors that have a major impact on that area, whether it is typically perceived as being so influenced or not. Some of the remarks have had more "solid" evidence or support than others, but all are backed by more than has been possible -- or prudent -- to include here. What has become clear to me is that ignoring human factors in software engineering is done at the risk of having them assert their importance with a force that can mean disaster for a project. This discussion has been one attempt to bring the matter forward for discussion.

Bibliography

- Berg, Eldon, *Periodical Guide for Computerists*. 1360 S.W. 199th Ct., Aloha, Oregon 97005 (\$2.50).
- Elspas, Bernard, Karl N. Levitt, Richard J. Waldinger, and Abraham Waksmann, "An Assessment of Techniques for Proving Program Correctness," *Computing Surveys*, vol. 4 (June, 1972), 97-147.
- Knuth, Donald E., "Computer Programming as an Art," *Communications of the ACM*, vol. 17 (Dec., 1974), 667-673.
- Sterling, Theodore D., "Guidelines for Computerized Information Systems: A Report from Stanley House," *Communications of the ACM*, vol. 17 (Nov., 1974), 609-613.
- Sterling, Theodore D., "Humanizing Computerized Information Systems," *Science*, vol. 190 (19 Dec., 1975), 1168-1172.
- Tossy, Michael, "Some Thoughts on Designing an Interactive Program for Easy Use by Humans," unpublished seminar paper, U.C.

Berkeley, 1977.

Wang, Li-Chen, "Palo Alto Tiny Basic," *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia*, vol. 1 (May, 1976), 12-25.

Weinberg, Gerald M., *The Psychology of Computer Programming*, NY: Van Nostrand Reinhold Co., 1972.

Van Tassel, Dennie, *Program Style, Design, Efficiency, Debugging, and Testing*, NY: Prentice-Hall, 1974.

Yourdon, Edward, *Techniques of Program Structure and Design*, NY: Prentice-Hall, 1975.

THE HUMAN INTERFACE

William F. Anderson
158 Valparaiso
San Francisco CA 94133

ABSTRACT:

A great deal of energy has been spent on defining the various physical interfaces that exist in a computer system. Such a statement cannot be made about the human interface. Yet, the success or failure of a system is dependent upon the quality of this interface. In designing the human interface the basic principles of communication need to be applied. Simply, these principles are: what are you trying to communicate, how is it being communicated, and who is the recipient of the communication.

The human interface—where does it fit into the picture? What must be considered when designing software to support this interface? The intent of this presentation is to start you thinking about these questions and their answers.

What is the purpose of computers; whether they be micro, mini, or maxi? Is not a computer a "tool"? A tool used, for example, for problem solving, information storage and retrieval, and the controlling of other devices. In general, computers are tools used to solve human problems. Note this is quite different from computers being problems for humans to solve; a situation which, unfortunately, is rather common.

The design of tools requires the application of "human factors engineering"—in other words, one must take into account the user of the tool. In a computer system this involves two levels: the physical design of hardware, and software design. Take, for example, a keyboard. The weight of the keys, their size, angle, shape, and layout are all factors in determining the optimal physical design of that keyboard. Although important, this is an area in which few of us ever get involved, since the hardware is usually purchased. However, software is a totally different story.

In terms of software, the human interface involves the entering of data into a system, the retrieval of data from a system, and the controlling of the system. Each of these areas can be viewed as a communication interface. It is a link to the human user, just as software routines also link the computer to peripherals such as a cassette recorder.

Some of you are probably wondering of what interest is this to the hobbyist? I can only ask, how do you plan to use your home computer system? Is it for your own personal enjoyment, or are you planning on others sharing the benefits of your work? If it is for yourself, then you need only be concerned about communicating with yourself. If you are writing programs that will be used by others, even if only your family and close friends, then you should be concerned with how they will interact with the system. For no matter how good the program is in other areas, if it is difficult to use, it will be put aside.

Notice, I have referred to the human interface as a communication interface. The interface is a communication interface when one considers that the object is to transfer information. Furthermore, we are entering an era where more people directly interact with computers. This means that the machine's software must take on more and more of the burden of providing the communication interface to the user of the system. Simply, the principles for this interface are those of all human communication. They can be categorized as what are we trying to communicate, how are we going to communicate it, and to whom are we trying to communicate.

Let's consider the recipient of the process first. Traditionally, the users of computer systems have had to be rather sophisticated, for example, users of airline reservation systems. Some educational systems have been an exception to this statement. The fantasy of those of us involved in personal computing is a computer in every home (or at least a good majority of homes). This means that the users of systems are not going to be the ultimate combination of electronic and software whizzes. The vast majority of potential users of home systems could not care less how a job is done on a computer; only that it is done. Unfortunately, the industry is geared towards supporting the sophisticated user. In fact, I will stick my neck out and say that it takes a more experienced person to purchase a home computer than is required to purchase a commercial product. We seem to be spending more time trying to educate people so that they have the requisite level of knowledge than we do in trying to make the computer easier to use. Even if systems are designed so that the hardware is transparent to the user, the job is not done. The task is complete only when the effectiveness of the information interchange with the system has been maximized and the person using the system has found the experience to be helpful rather than frustrating.

Although knowing what to say seems to be a problem that particularly plagues documentation, it also haunts the environs of interactive software. Before a message can be effectively communicated, one must be clear about the content

and purpose of a message. This applies not only to messages generated by a system, but also to the syntax requirements of entering data into the system. In my opinion, the designer of interactive software has the responsibility for seeing that it is the software which insures the quality of communication and not the user of the system.

The purpose of a message is to get the desired response, and that does not include kicking the computer during a peak of frustration. Nor is a system crash when invalid data is entered an acceptable response from the system. Valid responses include entering the requested information or taking an appropriate action such as changing paper in the printer.

To me, lack of clarity in a message reflects a lack of understanding of the problem one is trying to solve. For instance, with my current knowledge of the subject, if I were trying to design a system which, based on various inputted parameters, provided an analysis of a particular stock, I would have a difficult time devising questions that would lead to the appropriate answer—to say nothing about how to state the answer. In other words, if one understands the problem and knows the purpose of what he is trying to achieve, then he is in a position to communicate a clear message.

The final link in the communication chain is how to communicate the message to the recipient. One of the exciting things about current technology is all the new possibilities for media that can be used in the communications process. There are audio boards, video drivers in both black & white and color, light pens, plotters, printers, and optical scanners, as well as the old teletype. Each of these offers different possibilities as to how messages can be communicated. I know that I would like to own one of everything so that I could explore their limits. However, that is not the situation today, so I will explore what can be done with a CRT or teletype.

Compared with the CRT, the teletype is a very limited device. It is not uncommon for the CRT to be used in a teletype replacement mode, but it would be very inefficient and in some cases impossible to use the teletype in a CRT replacement mode. In terms of presenting information, the teletype is basically a line oriented device (longer messages being communicated as a series of lines). The major advantage is that it can produce a listing which is independent of the computer, can be used as a historical document on the interaction that occurred during the session, and comments can be noted on the listing itself. The major disadvantage is its lack of speed.

The CRT, however, is a very different medium. The area of the screen limits the maximum amount of data that can be conveyed in a single message. Unless supported by another means of storage, once the data goes off the screen it is lost forever. The major advantages are that longer messages can be communicated very quickly, and, depending on the CRT, one has the additional features of graphics and variable density.

For example, on a CRT I could do what is called menu selection, which is the presentation of the question along with a list of alternatives. My response would be to enter the number for the alternative I wanted. Whereas on a teletype the message is presented, and then based on my prior knowledge of the alternatives I would enter the appropriate response.

This leads into another topic called format of messages and responses. One of my pet peeves is systems that use single letter messages and responses, or even numbered messages. This approach requires a more sophisticated user of the system and is prone to greater error. It is disturbing to break up an involved interactive process with a search through the manual to find out what is going on.

The opposite extreme is to get too fancy in presenting a message. This is especially true with media that is extremely versatile. Remember, the object is to communicate a message, not to absorb the user's attention into the media.

I often see another habit which I think needs to be changed. Computer responses such as "hey stupid—you made an error" are basically insulting and convey little information. Such remarks say more about the designer of the system than they do about the user. In correcting errors, the purpose is to elicit the desired response. This is accomplished through messages that describe the error and tell what was expected.

In summary, I want to emphasize that until we start to concern ourselves with the human interface, the growth of personal computers will be limited to the devoted "hackers". I feel that the first step in designing this interface is to realize that it is a communications process and that the guidelines for good communications between two humans apply equally to communications between human and computer. For the dialogue of the computer is nothing more than an extension of the person who wrote the interactive program.

BIOGRAPHY:

Current employment at Four-Phase Systems culminates eight years of experience in data processing. This is tempered by a B.S. in Psychology and graduate work in Administration of Justice with a couple of years as a probation officer. Interest in personal computing dates back eighteen months to acquisition of an 8080A based computer.

THE POTENTIAL OF MICROCOMPUTERS FOR THE PHYSICALLY HANDICAPPED

Peter J. Nelson & J.G. Cossalter
 Medical Engineering Section, Bldg. M-50
 Division of Electrical Engineering
 National Research Council of Canada
 Ottawa Ontario
 Canada K1A OR8

ABSTRACT

The needs of the physically handicapped, especially those who are non-verbal, are outlined with respect to several communication aids developed at the National Research Council of Canada over the past dozen years. For each device described it is shown how the use of a microprocessor has simplified the required hardware, while making the redesigned devices more versatile. The use of synthesized speech in a classroom symbol communication system, based on a microcomputer, is described in some detail. To sum up, the future possibilities for home-computer-based aids for the handicapped are explored.

Introduction

The microprocessor and microcomputer revolution promises new electronic possibilities for everyone. For certain neglected groups such as the physically handicapped, the possibilities of new freedoms are more important and even more exciting than for the ordinary man or woman in the street. We are referring to such new freedoms as: the freedom of self-expression through simpler and more cost-effective aids to communication; the freedom to pursue educational opportunities through computer-aided-instruction brought right to the person's home; the freedom of independence provided by personal environmental controllers and mobility aids; the freedom to hold a job and support oneself by means of various vocational aids; and the freedom of new recreational opportunities provided by computer games.

The Medical Engineering Section and the Information Science Section of the National Research Council of Canada are pursuing some of these goals on behalf of the physically handicapped. This paper will describe our initial attempts to exploit the potential of microcomputers in communication aids and teaching aids for the physically handicapped.

Background to the Problem

The aids described in this paper are designed for severely handicapped persons who lack normal functional use of their hands and arms to the point that they cannot hold a pen or pencil nor operate a regular typewriter. Persons in this category include those with a high spinal cord lesion or other paralysis due to stroke, trauma, etc. and those with neuromuscular disabilities resulting from cerebral palsy, poliomyelitis, muscular dystrophy, etc. The most handicapped of these are the persons who also have no functional speech capabilities. The loss of the ability to communicate in any normal way is the most serious problem for these persons. The need to communicate and to express one's feelings, thoughts, and desires is fundamental to mankind. Children deprived from birth of any ability to communicate will be developmentally handicapped throughout their lives. Alternative means of communication must be provided early in life.

COMHANDI

The COMHANDI Communications System for the Handicapped was the first aid developed by NRC¹ (see Figure 1). It was developed over twelve years ago using diode-transistor logic (DTL), the implementation of which required about nine circuit boards and other associated components (see Figure 2). Nevertheless, the COMHANDI accomplished what it was designed to do - it permitted a severely handicapped person to type out messages on a low-cost teletypewriter by operating a single paddle switch. Actuating the paddle caused a light to scan behind an alphanumeric display panel, first horizontally and then vertically, until the desired character was reached and typed out on the teletypewriter. If the user had slightly better functional capabilities, a joystick could be used to direct the scanning in right, left, up, or down directions. A separate paddle would then be used to actuate the typing function. Although there are other approaches, the scanning concept has since been implemented almost universally in typing and communication aids for the most severely handicapped.



Fig. 1. COMHANDI typing and communications aid for the handicapped.

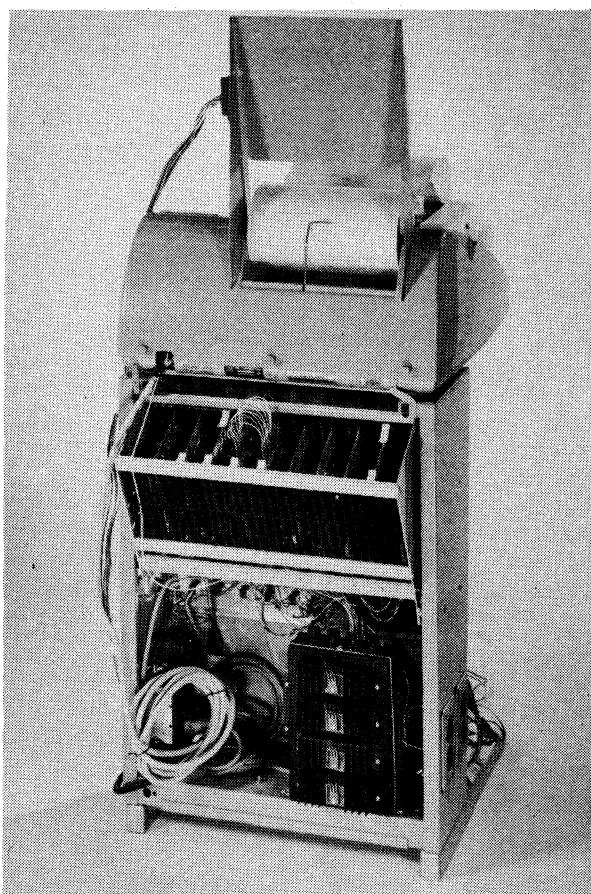


Fig. 2. Rear view of COMHANDI showing circuit boards.

Microprocessor-Based COMHANDI

As a demonstration of the potential of microprocessors to reduce hardware logic, we have recently implemented all the

functions of the COMHANDI using an F-8 microprocessor. The block diagram, Figure 3, shows the simplicity of the new system. The program requires 512 bytes of read-only memory (ROM), 9 scratchpad registers (out of 64 available in the F-8 CPU), the programmable timer, and 3 of the 4 I/O ports (3850 and 3851). The FAIRBUGTM monitor, which resides in the 3851 PSU, uses firmware to generate the serial ASCII-encoded data to drive the teletypewriter. Not only is the hardware vastly simplified, but the software-controlled system permits greater flexibility in the mode of operation. For example, the scanning rate can be changed by typing in a time constant from the teletypewriter keyboard.

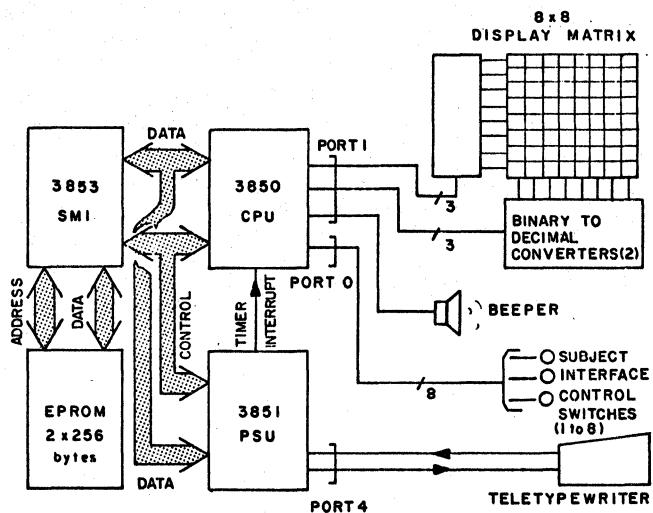


Fig. 3. Block diagram of F-8 microprocessor version of COMHANDI.

Bliss Symbols

Although the COMHANDI was effective for certain groups of handicapped individuals, an alphanumeric communication aid is of little use to preschool children who cannot yet read or spell. In the past, picture boards have been used with these non-verbal children to permit them to express their immediate needs.² More recently, a universal symbol language developed by C.K. Bliss has been adapted for use by these children.³

The Bliss Symbols, as they are called, are pictographically related to the concepts and real-life experiences which they represent. They are therefore, intuitively meaningful to young children and are more easily learned than an alphabetic language. Hence, using Bliss Symbols, the children can begin to communicate at an earlier age, which is extremely important to their development.

Some examples of Bliss Symbols are shown in Figure 4. The symbols for "home", "street" and "animal" look like the objects they represent. Basic symbols such as these are then combined to convey related concepts such as "street", "food" and "meat". Note that "meat" is "food" from an "animal". All concepts

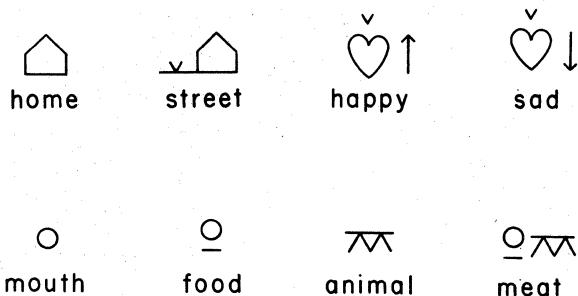


Fig. 4. Some examples of Bliss Symbols.

dealing with emotions contain the basic heart-shaped symbol to indicate inner feelings. By organizing Bliss Symbols into sequences, whole ideas or sentences can be conveyed, as shown in Figure 5.

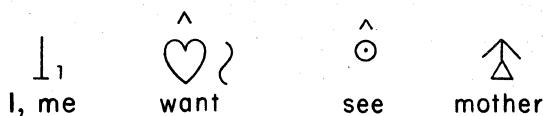


Fig. 5. A Bliss Symbol sentence.

Symbol Communication Aids

The children soon develop a working vocabulary of 200 to 500 symbols, which creates some problems for those who are physically incapable of pointing on a large symbol board. Electronic symbol communication boards have been built for these children, such as the unit shown in Figure 6. This portable unit provides electronic scanning over 256 symbols by means of a 16 x 16 matrix of light-emitting diodes (LED's). The capacity of the board can be doubled to 512 symbols by using either a steady light or a blinking light to indicate upper and lower case symbols, in effect, at each location. Just as with the COMHANDI, a variety of interface devices can be used to control the symbol communication board, either of the single-switch-closure type or the joystick-type.

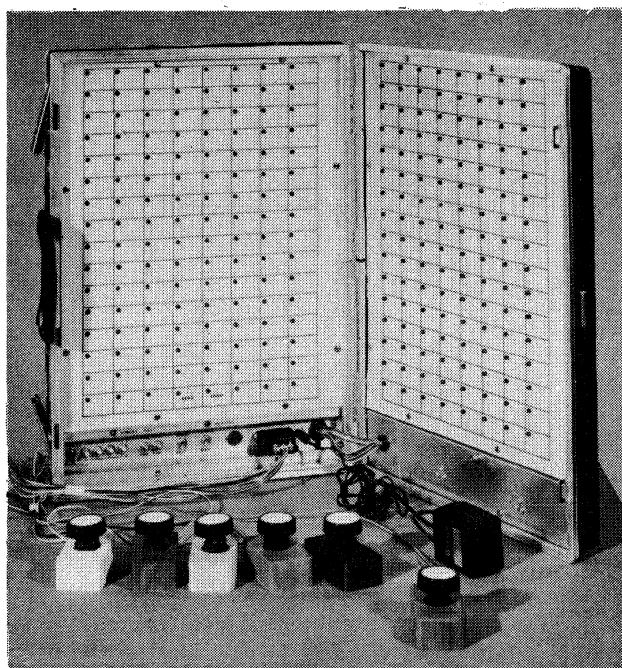


Fig. 6. Symbol Communication Matrix board, Model 2.

An important feature of the electronic scanning board is the provision of a memory function. One serious limitation of manual alphabet or symbol pointing boards is the fact that the "listener" or receiver of the communication usually cannot remember enough letters or symbols to piece together any but the shortest words or messages the "talker" is attempting to transmit. This problem is compounded by the slow rate at which some of the severely handicapped persons must work. Knowing that a normal person is impatiently waiting for them to point out a message only heightens their anxiety. With an electronic memory board, however, the handicapped user can work at his or her own speed, building up a message in the board's memory, one symbol at a time. The message can be read back to a "listener" at a later time, as often as necessary, simply by causing the LED indicators on the display matrix to "jump" sequentially through all the symbol locations stored in the memory.

Just as in the case of the COMHANDI, the electronic hardware required to implement a symbol scanning matrix can be

greatly simplified through the use of a microprocessor. The original hardwired board shown in Figure 6 uses about 60 CMOS small-scale integrated circuit packages, whereas the same functions have been implemented with a 768-byte program in the F-8 microprocessor. Again the microprocessor-based version provides greater flexibility, in user definable rates for scanning and memory read-out, etc. and a larger memory capacity (in the scratchpad memory of the F-8).

Synthesized Speech

The Symbol Communication Matrix, as described, has been a useful aid, especially in situations where portability is necessary. It has some limitations, however, because it provides only a visual indication of the selected symbols. We have been experimenting with the use of synthesized speech, therefore, as a means of providing additional sensory inputs to the handicapped person.⁴ For the initial trials, a commercial speech synthesizer (VOTRAXTM) and a remote computer were used, coupled to the handicapped person's symbol display board via an acoustic telephone link. The synthesizer speaks out the name of each symbol on the display matrix as it is selected, storing these words in a scratchpad memory. Upon command, then, the whole sentence or thought can be spoken by the synthesizer.

Speech is, of course, a very natural form of communication. The value of the aural reinforcement provided by the synthesizer soon became quite evident. The children's attention span and motivation increased almost immediately, permitting training to proceed at a faster pace. Hence, in a period of six months following the addition of the artificial speech, their speed in selecting symbols, their language development and the use of proper syntax, and their spontaneous desire to communicate have all been promoted. The synthesizer gives the child an objective feedback on the trial-and-error selection of symbols and sentences, permitting independent work. The "space age" quality of the artificial voice appeals to the children and many attempt to verbalize along with it.

Use of a Microcomputer

Following the successful trials with the speech synthesizer, we have proceeded to develop a stand-alone classroom communication system built around a microcomputer. A block diagram of our "Classroom Symbol Talker" is shown in Figure 7. Initially the system provides for two handicapped children and a teacher to intercommunicate in the two mediums of Bliss Symbols and artificial speech. Each student's terminal is very much like the symbol communication matrix board described previously, including the same type of interface controls.

Some of the differences are local loudspeakers in each board and extra "control addresses" on the symbol matrix to permit the child to select the destination for his or her message.

A graphic television display is planned as a low-cost additional feature. It will serve as a classroom "blackboard" on which the teacher or either of the students can "write" a Bliss Symbol message. The teacher will use a numerical keypad to select the symbols by code numbers. The symbols can be stationary on the screen or can be made to move across the screen in "Times Square" fashion for longer messages. By bringing the symbols together and displaying them in a serial string on the TV screen, it is hoped that a degree of connectedness will be conveyed which may be lacking on the individual symbol boards.

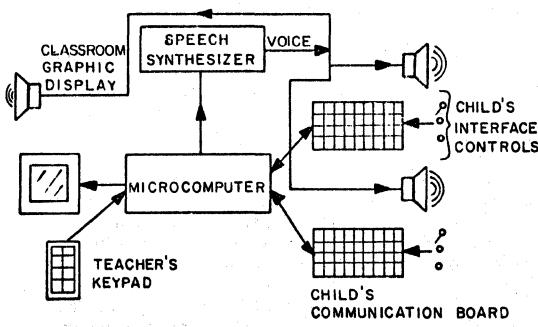


Fig. 7. Block diagram of "Classroom Symbol Talker" system.

The microcomputer is the component which provides all the necessary memory and electronic control. Sentences formed on one child's board can be sent to the other board to be read out or to the TV screen for all to see. The audio from the speech synthesizer is also under computer control and can be directed to any individual board or to the classroom display. As the symbol message is being displayed, the corresponding words are spoken by the synthesizer.

The microcomputer is based on the 8080 microprocessor. Since it is an 8 bit processor, it nicely suits the jobs of handling a 256 symbol system - a unique identifier for each symbol fits into one byte of memory. The control program for the communication boards and the TV display is stored in 4096 (4K) bytes of Erasable Programmable Read Only Memory (EPROM). This type of memory is ideal for this application because it allows the control program to be changed by reprogramming the same EPROM integrated circuits. This EPROM also contains the codes required to direct the speech synthesizer to produce the spoken words for the symbols. (The 256-symbol vocabulary requires about 1½ K bytes).

4K of Random Access Memory (RAM) is used as a dynamic data store for control operations. Symbol sentence lists and intercommunication buffers are kept here.

A Real-Time Clock is used to time events which must be carried out at particular intervals of time. Scanning symbols on the communication boards, for example, is carried out at a preset rate determined by the clock.

The processor sends control codes to the speech synthesizer through a Serial Input/Output (I/O) port. These control codes are buffered, interpreted by the synthesizer and then spoken out. The artificial speech that is generated, is routed through a digitally controlled switch and gated to the particular communication unit requiring audio reinforcement.

Each of the three communication terminals is handled through its own Parallel I/O port. Each is independent from the other and more terminals could be added to the system by interfacing them through additional I/O ports.

A significant amount of digital memory will be required to store the bit patterns for all 256 of the Bliss Symbols for the graphic TV display. Other researchers are attempting to display Bliss Symbols electronically and it is hoped that techniques will be found to minimize the required memory size.⁵ Otherwise, an auxiliary diskette memory may be required for this feature.

Design Considerations

The use of a microcomputer as the main component in our Classroom Symbol Talker is an example of a near-ideal solution to a number of the problems involved in developing aids for the handicapped. The commercial manufacture of small quantities of specialized devices for the handicapped has been a recurrent problem. In our system, however, as many of the logic functions as possible have been implemented in the microcomputer software, rather than in discrete circuitry as was the practice in the past. Less manufacturing is involved, therefore. The microcomputer can be purchased as a completely assembled and tested commercial product into which the programmed EPROM's are inserted. In fact, except for the individual symbol matrix boards and interface controls, all other components of the system are commercially available.

The system can be expanded, or its functions can be changed, without rewiring. Each communication board can light up a maximum of twenty five lights simultaneously. By replacing the symbol sheet on the front of each board with an enlarged BINGO card and adding some appropriately programmed EPROM memory, the system will allow the children to play BINGO. Other interactive games, such as TIC-TAC-TOE and Steeplechase can be played, either on the children's individual display board or on the classroom TV screen. The recreational possibilities for the handicapped which are opened up by computer games are nearly limitless. The games must be chosen to match the children's intellectual and functional capabilities, however. Target shooting, ping-pong and other games of skill are obviously out of the question. Games which stimulate intellectual participation are ideal. The costs of the software for each additional game are quite small.

The Future

This paper has drawn attention to some of the communication needs of the physically handicapped and has attempted to show how microprocessors and microcomputers are beginning to be used to meet these needs. It is certainly only the beginning. Other researchers are also applying these new technologies for the handicapped.^{5,6} This will be the case more and more as the prices of microprocessors and fully built home microcomputers continue to fall. In our Classroom Symbol Talker, the major cost item is the speech synthesizer. The synthesizer is an excellent device. Since it is basically a phoneme generator, it provides random access to an unlimited vocabulary. For some non-verbal handicapped persons a limited vocabulary stored on a floppy diskette would be a more economical solution. Controlled by a home hobby-type computer, this arrangement would permit such persons to conduct their personal business by telephone.

Some of the other areas where microprocessors can be applied for the handicapped are in TV typewriters and in the development of a portable artificial voice. TV typewriters have been adapted for the handicapped, already, in hardwired logic versions.⁷ These could be more easily implemented with a commercial microcomputer with a TV interface. A program could be written which would display the set of alphanumeric characters at, say, the top of the TV screen. The user would move the cursor, by means of a joystick or single paddle switch, to select each character. The selected character would then appear on the lower half of the screen, where a whole message could be built up. With an optional teletype or other hard-copy printer, the handicapped person could be gainfully employed preparing business correspondence. The text would be prepared in the desired format on the TV screen, with any errors corrected, before being "dumped" out to the printer.

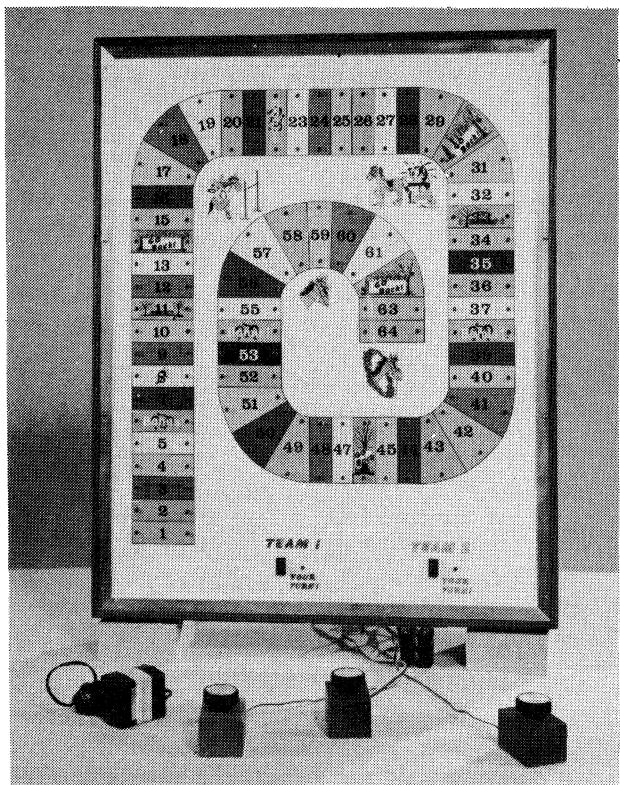


Fig. 8. Portable electronic Steeplechase game for handicapped children.

Speech is a far more acceptable medium of communication with the general public than letters or symbols. The non-handicapped person usually won't make the effort to learn a symbol language and feels psychologically uneasy when confronted by a non-verbal person. Hence, there is a tremendous

need for a portable artificial voice. It may be some time before such a device is economically feasible, but we can be sure it will involve a microprocessor. The feasibility of a low-cost device, using a limited vocabulary on magnetic tape or diskette, is under study at NRC of Canada. The problem is to provide reasonably rapid access to any word in the vocabulary.

Computer games have been mentioned. The Medical Engineering Section of NRC has developed several games for the handicapped, such as Checktronics, an electronic checkers game, and Steeplechase (see Figure 8). These were built up from hardwired CMOS logic circuitry. Portability was one of the features of these games, but from the standpoint of the visibility of the display, a TV screen would be an improvement. The hardwired games have served to demonstrate the value of recreational aids for the handicapped. The Steeplechase is especially appropriate for young children who like the idea of racing. Incidentally, it helps teach them counting concepts as they must "move their man" by pushing on a paddle switch the appropriate number of times. In hardwired form these stand-alone games are probably not cost effective. As programs in a low-cost hobby computer with a TV display, they will be very attractive.

From communication aids, to educational games, it is only another small step to more formal educational programs on personal computers. Here is an exciting new opportunity for the handicapped. The teaching of children and adults with physical and educational handicaps has always required an intensive amount of one-to-one interaction between student and teacher. Interactive teaching programs will reduce this requirement and provide a more objective tutor in many cases. The low-cost of personal computers will make such teaching practical, not only in institutional settings but also in a handicapped person's own home where they may be confined to bed or wheelchair. The hobby computer can also act as a remote terminal for a large time-shared computer system on which more elaborate teaching programs may be available.

Biography

Peter J. Nelson has a Master of Science degree in Electrical Engineering (Biomedical Engineering options) and has nine years of experience as a rehabilitation engineer (one who specializes in the design and application of devices and techniques for the rehabilitation of the physically handicapped). At present he is employed as a Research Officer in the Medical Engineering Section, Division of Electrical Engineering, National Research Council of Canada, where his responsibilities include the development and evaluation of aids for the handicapped. Previously he has worked in Washington, D.C., where he was Staff Officer for the Committee on Prosthetics Research and Development, National Academy of Sciences; and in Winnipeg, Manitoba, Canada, where he was Co-Director of the Rehabilitation Engineering Dept., Health Sciences Centre.

John G. Cossalter, holds the degree of Master of Applied Science from the University of British Columbia, Vancouver, Canada. For the past seven years he has been employed as a Research Officer in the Information Science Section of National Research Council of Canada and has worked in the areas of computer-aided instruction, computer-generated speech, and computerized aids for the handicapped.

Acknowledgements

The authors wish to acknowledge the assistance of summer student R. Paduch from McGill University, Montreal, Canada, in the preparation of the programs for the F-8 microprocessor. The authors are also indebted to J.R. Charbonneau and F.P. Orpana of NRC for the construction of the equipment described in this paper, to the staff of the Ottawa Crippled Children's Treatment Centre for the clinical evaluation of the devices, and to A.J. Mortimer of NRC for helpful suggestions in the planning of this paper.

References

1. Roy, O.Z. and Charbonneau, J.R., "A Communications System for the Handicapped (COMHANDI)", Chapter 11 in *Aids for the Severely Handicapped*, K. Copeland, ed., Sector Publishing Ltd., London, 1974; pp.89-98.

2. McDonald, E.T. and Schultz, A.R., "Communication Boards for Cerebral-Palsied Children", J. of Speech and Hearing Disorders, vol.38, no.1, Feb. 1973; pp.73-88.
3. McNaughton, S., "Bliss Symbols - An Alternative Symbol System for the Non-Vocal Pre-Reading Child", in Non-Vocal Communication Techniques and Aids for the Severely Handicapped, G.C. Vanderheiden and K. Grilley, eds., University Park Press, Baltimore, 1976; pp.85-104.
4. Charbonneau, J.R., Roy, O.Z., Cossalter, J.G., Warrick, A. and Côté, C., "A Symbol Communication System for the Non-Verbal Severely Handicapped, with Audio Word and Sentence Reinforcement," Proc. 1976 Conf. on Systems and Devices for the Disabled, Boston, Mass., June 10-12; pp.18-19.
5. Vanderheiden, Gregg C., Director, Trace Research and Development Center for the Severely Communicatively Handicapped, University of Wisconsin-Madison. Personal Communication.
6. McBride, J.W., Morena, L.C., Lawrence, P.D. and Robinson, C.E., "An Adaptive Environmental Control and Communications System for the Handicapped", Digest of 11th Intnl. Conf. on Medical and Biological Engineering, Ottawa, Canada, Aug. 2-6, 1976; pp.29.6.
7. Logan, H.C., "Communications Terminal for Speech Handicapped", Proc. IEEE 1975 Region Six Conf., Salt Lake City, Utah, May 7-9; pp.8-11.

AN INTERFACE USING BIO-ELECTRICAL SIGNALS TO CONTROL A MICROPROCESSOR SYSTEM FOR THE PHYSICALLY AND COMMUNICATIVELY HANDICAPPED

Laurence R. Upjohn, Pharm. D.
7297 Stanwood Way
Sacramento CA 95831

Abstract:

This paper outlines the biological source of electrical signals that can be used to control microprocessor systems. A brief discussion is undertaken as to methods of amplifying this bio-electricity with comments on integrated circuit selection and circuit design. Emphasis is placed on the safe interconnection of the bio-electric interface to the microprocessor insuring that electrical isolation is maintained between them. Lastly a brief mention is made of several decoding methods used with the switch to implement microprocessor systems for the severely handicapped.

Introduction:

This paper will briefly describe the human body as a source of electrical signals that can be used to control electronic systems. A short discussion on the biological origin of this electrical activity will be given, followed by technical methods to exploit this phenomenon. Application of this switch to various microprocessor systems then will illustrate adaptation of microprocessor systems to the physically handicapped. With this adaptation systems can be tailored to individual needs, by the handicapped individual himself. This self-tailoring process can provide new vistas of recreation, education, communication and enhanced independence.

Electro-physiology:

This ten dollar word describes the universal property of all living things to generate very small but measurable electrical currents. The popular areas of Bio-feedback use several bio-electrical potentials or voltages to study various body functions which then are converted to some auditory or visual signal with which our senses can deal. These signals have origin in individual cells going through the repetitive process of concentrating salts from body fluids. These salts are concentrated so that one salt will be in high concentration inside the cells while another will remain concentrated just outside the cell. These concentrations are maintained by the cell membrane but can rapidly change when certain environmental stimuli disturb the cell membrane. When these finely balanced concentrations change rapidly a small pulse of approximately 50 millivolts is given off such as the electrical current that flows when the poles of a battery are shorted together. Multiply this by several million cells discharging at once and it becomes apparent where or how bio-electricity is "generated".

Interfacing Bio-electricity to the Outside World:

Since the bio-electric signals just discussed are of such small voltage, they must be amplified before they can be modified or processed by any digital system. However this amplification must take place with as little interference to the biological system as possible. Therefore an amplifier must have high input impedance. This is a resistive quality of the input terminals such that incoming signals meet a very high resistance path into the amplifier. Because of this high resistance pathway very little current is drawn from the biological system into the amplifier so the cells being observed function as usual. Next, the amplifier needs the ability to reject low level signals that arrive at both inputs at the same instant (i.e., are in phase with each other). This property is called high common mode rejection ratio, or CMRR for short. The predominant electrical noise that must be kept out of the system is the 60Hz. noise from the electrical utilities, hence the high CMRR. Next the inputs to the amplifier are connected to electrodes placed over the area to be observed in electrical contact with the skin.

This contact is enhanced by using a jelly or paste containing a high salt concentration and securing the electrodes firmly so movement signals will not be produced. A much less complicated method is to use disposable EKG electrodes where available. They come prejelled with high quality electrode metal (silver!) and are self adhesive.

The amplified signal can now be monitored from the output of the amplifier by an oscilloscope or a low cost audio amplifier and permanent magnet speaker. One soon learns the tell-tale hum or smooth sinewaveform of 60Hz noise being amplified through the system. By the appropriate adjustment the 60Hz. noise can be "tuned" out so only the irregular higher frequency biological signal gets through.

The monitored signal must now be smoothed so that in effect a slowed waveform is presented to the next stage of the bio-interface unit. This process is called integration. Integration of the signal is accomplished with a $\frac{1}{2}$ wave rectifier or diode and a resistance/capacitance network to store the signal for the time constant of the resistor-capacitor network. This integrated signal

has a slow enough transition to trigger a 7413 dual Schmitt trigger, which then outputs a TTL pulse for each threshold crossing of the input signal.

The TTL trigger pulse is lead to a monostable multivibrator which outputs a pulse of predetermined duration. During this period it cannot be retriggered so its' output pulse remains of constant length or time duration.

Thus the signal is converted from the constantly changing low level biological current to a two state binary signal that is TTL compatible. Now we have a signal that can be fed to the microprocessor, as our next interconnection step, right?

WRONG! The interface must be completely isolated electrically from the microprocessor and its' high amperage DC voltages. This is not as difficult a task to accomplish as one might suspect. A photo-optic isolator of the LED/Photo transistor type is quite effective in achieving the necessary electrical isolation. It is also suggested that in construction of the interface, micro-power circuits be used where possible, to make it easy to use low voltage transistor batteries to power the interface. If low power or micro-power IC's are not available then larger batteries should be used. Under no circumstances use the DC voltages from the Microprocessor bus.

Proper grounding of all cases to one single earth ground will reduce noise problems that can arise when several different ground points are used.

Interface Applications :

Since this paper is to address the area of Microcomputer systems for the Physically Handicapped, now is the time to outline how this bio-interface could be used to program a micro-system.

The signal output from the bio-interface is only a single binary bit but several methods can be implemented to expand this to a fully coded eight bit binary data word or simply an external flag directly to the micro-processor.

Coding the single bit to an eight bit byte can be done using a self scanning array of LEDs. Two four to ten or sixteen decoders scan the horizontal and vertical rows giving two four bit addresses for the matrix position of the lighted LED. These two addresses are latched into the computer as a single eight bit word. This word is then converted by software to represent either alpha-numerics or actual machine operational codes. A simple "bootstrap" loader could be contained in Read Only Memory to give power on start capabilities.

Using the single bit line as a direct flag to the micro-processor involves writing the appropriate software to test the flag and the execute an interrogative interaction with the operator to determine function to be programmed. This may take longer to facilitate initially but is by far the more elegant solution to this control problem.

This second method has the added advantage of allowing the operator to tailor his software to his individual control or communication problems. This is the ultimate goal for any aid to the handicapped, for it gives them independence which to them is their greatest need.

Bibliography:

Belke, R.E. et al.
G.E. Transistor Manual (Light Weight Edition)
General Electric Co., Syracuse, New York, 1969

Geddes, L.A. and Baker, L.E.
Principles of Applied Biomedical Instrumentation
John Wiley and Sons, Inc., New York, N.Y. 1968

Millman, J. and Halkias, C.C.
Integrated Electronics: Analogue and Digital Circuits and Systems
McGraw-Hill Book Co., New York, N.Y. 1972

Vander Kooi, M.K. et al
Linear Applications, Volume 1
National Semiconductor Corp., Santa Clara, Ca. 1973

Author:

Laurence R. Upjohn Pharm. D.
Staff Pharmacist
U.C.Davis/Sacramento Med Center
Graduated From University of The Pacific College of Pharmacy in May of 1976 with emphasis in Electro-physiology and Pharmacology. While there developed Electromyographic Switch for the severely handicapped and a micro-computer system for clinical drug analysis using Electro-Myograms and

Encephalograms, as indicators of patient/drug effects.
Presently developing Bio-electrical switch and micro-
processor card for use by the severly handicapped.
May be contacted at;
7297 Stanwood Way
Sacramento, California 95831
Phone: 916-391-5336

WHAT TO DO AFTER YOU HIT RETURN... AND NOTHING HAPPENS: WARRANTY IN THE MICRO-COMPUTER INDUSTRY

Kenneth S. Widelitz, Attorney at Law, WA6PPZ
 10405 Louisiana, No. 8
 Los Angeles CA 90025

ABSTRACT OF FULL TEXT PAPER

This paper will discuss consumers' rights and manufacturers' and retailers' liabilities under the Uniform Commercial Code, the Song-Beverly Warranty Act of California and the Federally enacted Magnuson Moss Warranty Act. Attention will be paid to the effects of warranty on the microcomputer industry and examples will be drawn from the microcomputer industry. The paper will include a description of both express and implied warranties.

Special emphasis will be placed on the presale availability rule and on the repair facility requirements. The paper will conclude with an analysis of formal dispute settlement mechanisms as a possible solution to the compatibility problem unique to the microcomputer industry.

Recently enacted legislation has made dramatic inroads into the evils of the well known maxim of "caveat emptor" and, more importantly, the lesser known manufacturers' warranty maxim of "what the bold print giveth, the fine print taketh away." Unfortunately, consumers, manufacturers and retailers are generally unaware of their rights and liabilities under this legislation. This is especially true in the home and personal computing field due to the recent emergence of the microcomputer industry.

For instance, manufacturers are unaware that they are in violation of federal legislation if they fail to clearly designate their written warranties as either "full" or "limited". Retailers and manufacturers are not cognizant of the fact that they are in violation of federal legislation if they fail to conform to the presale availability rule which requires that every written warranty be readily available to the consumer prior to sale in a specified manner. Consumers are unaware that if a manufacturer or retailer breaches the terms of his express warranty, a California Statute allows the consumer to recover treble damages plus costs and attorney's fees.

The preparation of written warranties by microcomputer manufacturers has been given short shrift. The manufacturer should give his warranty the same careful deliberation and consideration that he gives to the development of his product. A carefully considered warranty can do much to promote brand loyalty and accurately cost out warranty liability. If a manufacturer costs out his potential liability based upon assumptions which are not viable due to the requirements of recent legislation, it might mean the difference between a profitable or loss year.

As an example of possible liabilities unknown to the manufacturer, manufacturers are now required to reimburse retailers who perform the manufacturer's warranty repair work in an amount which includes a reasonable profit to the retailers. Another example is in the area of implied warranties. Implied warranties can no longer be disclaimed where accompanied by written warranties. This can be a critical warranty cost factor when considering that a manufacturer's liability under an implied warranty can be greater than his liability under a written warranty.

As consumers become more aware of the presale availability rule relative to warranties, manufacturers and retailers will have to recognize that their warranty will have to be as competitive

as the quality of their product and its sales price. As home computing catches on, the purchaser of a microcomputer will have less technical expertise and will place a greater importance on the warranty which accompanies the product he is buying. Conceivably, the warranty can be as important a factor in a purchase as the number of bytes of memory or the speed of an I/O device. The warranty takes on added significance when the hobbyist realizes he will probably be spending more money on his microcomputer system than on any other item of personal property which he owns with the exception of his car (and automobile manufacturers spend substantial sums advertising their warranties.) The more money someone spends on an item, the more careful he is about what he purchases. The consumer is also aware that there is generally a close correlation between the cost of a product and the cost to repair it if it fails to function properly.

The foregoing is intended to place the importance of the warranty in perspective for the manufacturer, retailer and consumer of microcomputers. The following discussion of warranty legislation applies to all consumer goods, but an attempt has been made to use examples of warranty problems unique to the microcomputer industry.

WARRANTY LEGISLATION

There are three major pieces of legislation which govern warranties as they relate to consumer goods sold in California. The oldest is the Uniform Commercial Code of California (U.C.C.) which has been in effect since 1965. Almost every state has its own version of the U.C.C., but, as its name implies, they are all substantially similar. In 1971 California adopted the Song-Beverly Consumer Warranty Act (SBCWA). The SBCWA was the first consumer warranty protection oriented piece of legislation in the country. The federally enacted Magnuson Moss Warranty Act (MMWA) is based, in part, on the SBCWA. It has been in effect since 1975, although rules promulgated pursuant to that Act by the Federal Trade Commission have been effective as recently as January 1, 1977.

TYPES OF WARRANTIES

There are two broad species of warranties, the implied warranty and the express warranty.

A warranty is implied when it arises by operation of law from the nature of a particular transaction, that is, it's automatic and requires no writing to arise. There are two kinds of implied warranties, the implied warranty of merchantability and the implied warranty of fitness for a particular purpose. In general, the implied warranty of merchantability means that a consumer good must be free from defects in materials or workmanship. An implied warranty of fitness arises when the retailer, distributor, or manufacturer has reason to know any particular purpose for which the consumer goods are required and that the buyer is relying on the skill and judgment of the seller to select and furnish suitable goods. For example, if a manufacturer of a floppy disc advertises that it is Saltair/Bonzai compatible, there exists an implied warranty of fitness that the floppy can be effectively used with such microcomputers.

Implied warranties are created by provisions in the U.C.C. and in the SBCWA. While the MMWA does not create any implied warranties, it does legislate the extent to which they can be limited.

The other type of warranty is the express warranty. The express warranty is usually a written warranty. For the purposes of the MMWA the express warranty must be created in writing. However, under the U.C.C. the express warranty may be created by an oral affirmation of fact, i.e., where the retailer makes an oral statement as to the capabilities of a certain product. It should be noted that "puffing" does not create an express warranty. That is, the retailer who says, "I guaranty your satisfaction, "or that a particular product "is the best buy on the market" does not create an express warranty. Where a sample or model is displayed, the U.C.C. and the SBCWA create an express warranty that the whole of the goods conforms to such sample or model. This has a significance for the manufacturer who buys large quantities of parts from a supplier based on a sample or model, but will ordinarily not have significance for the consumer.

IMPLIED WARRANTIES

The U.C.C. and SBCWA provisions relating to implied warranties parallel each other to a certain extent, but the SBCWA provides some important changes.

The U.C.C. version of the implied warranty of merchantability requires, for such an implied warranty to arise, that the seller must be a merchant as to those goods. Such a warranty under the U.C.C. is a seller's (retailer's) warranty. The SBCWA has given renewed vitality to the implied warranty of merchantability in two ways: one, it imposes a liability under the warranty directly on the manufacturer; and two, it makes it both difficult and commercially unwise for the manufacturer to disclaim such a warranty.

The implied warranty of fitness under the U.C.C. is also a seller's warranty whereas under the SBCWA such a warranty is made by both the manufacturer and the retailer. As stated before, the implied warranty of fitness requires that the manufacturer or retailer have a reason to know that the purchase is being made for a particular purpose and that the buyer is relying on the expertise of the seller or manufacturer to furnish the right product for that purpose.

However, the manufacturer and retailer do not actually have to be told that the buyer is relying on their skill and judgment. The manufacturer who advertises a particular use for his product should be liable under the implied warranty of fitness for a particular purpose even if the advertised purpose is not an ordinary use.

DISCLAIMER OF IMPLIED WARRANTIES

Before the enactment of the SBCWA and the MMWA, the implied warranties created by the U.C.C. were relatively easy to disclaim. The implied warranty of merchantability could be disclaimed by the use of a conspicuous written disclaimer that mentioned the word "merchantability." The implied warranty of fitness could be disclaimed by a writing declaring that "there are no warranties which extend beyond the description of the face hereof."

As a result of the enactment of the SBCWA and the MMWA, the disclaimers just mentioned are ineffective. Under present law the only way the implied warranties can be disclaimed is by making a sale on an "as is" basis. A conspicuous writing indicating an "as is" sale attached to the goods sold is required. Any other attempted waiver of any implied warranty is void.

DURATION OF AND REMEDIES UNDER IMPLIED WARRANTIES

While the U.C.C. does not make mention of the duration of implied warranties, the SBCWA specifically states that the duration of an implied war-

ranty is coextensive with that of any express warranty which accompanies the consumer goods. However, in no event shall the implied warranties be for a period of less than two months or for a period of more than one year. If no written warranty is provided, the duration of the implied warranty is one year.

The remedies available to the consumer under the SBCWA and the U.C.C. for the breach of an implied warranty are identical with the single exception that the SBCWA specifically provides that if a consumer sues based upon the breach of an implied warranty, the consumer may recover his attorney's fees.

The damages which a buyer may recover are incidental damages and consequential damages. Incidental damages should include the costs of repairing nonconforming goods and other items such as the transportation of those goods to and from the place where they are repaired. Consequential damages include such damage, loss or injury as does not flow directly and immediately from the act of the breaching party, but only from some of the consequences or results of such act.

For instance, take the case of an amateur radio operator who spends 20 hours inputting data relating to all the contacts he made in 1976. Because of a defect in the cassette recorder he is using as a memory device, none of the data is preserved. His consequential damages would be equal to the amount he would have to spend in order to hire a typist to reinput the data. As another example, consider a lemonade stand application that cannot service its clients while a defect in the CPU is being fixed. The consequential damages would equal the lost profits during the period of repair.

As a final example, let's take a letter which appeared in the November/December, 1976 issue of Dr. Dobb's Journal, page 4. Stuart R. Fallgatler wrote that he received an AY5-8500 6 game MOS/LSI chip from Advanced Micro-Electronics. The chip didn't work properly. Mr. Fallgatler wrote the company and received no reply. Dr. Dobb's Journal also wrote the company and received no reply. The implied warranty of merchantability requires that the chip be free from defects. Having received no satisfaction Mr. Fallgatler should bring a Small Claims Court action against Advanced Micro-Electronics. Small Claims Courts handle suits for claims less than \$750. The filing fee varies by county but is usually \$10 to \$15. The chip in question here lists for \$50, a significant sum. While filing a suit might be time consuming and inconvenient, Mr. Fallgatler would certainly be less inconvenienced than the company. He should obtain a judgment allowing him a replacement chip and his court costs. He would also be allowed incidental damages and consequential damages, if any. More importantly, such an action would be a great service to other persons similarly situated as Advanced Micro-Electronics might be more responsive to future complaints, having been taken to court once.

EXPRESS WARRANTIES

The provisions of the U.C.C., SBCWA and MMWA overlap considerably in the area of express warranties. This paper will highlight the unique and salient features of each but where there is an overlap there will be no attempt to distinguish between them. It should be noted at this point that although both the SBCWA and MMWA define "consumer good" or "consumer product" as an item which is normally bought primarily for personal, family or household purposes, the House Report on the MMWA indicates that products which are in fact used for business purposes fall within such definition if such items would generally be used for personal, family or household purposes. Thus it would seem that lemonade stand and even office uses of microcomputers would be covered by the MMWA because microcomputers are generally used for hobby or home purposes.

MMWA "FULL" OR "LIMITED" DESIGNATION REQUIREMENT

A very important feature of the MMWA is its requirement that every written warranty accompanying a product selling for \$15.00 or more be clearly designated either "full" or "limited". If a warranty merely says "WARRANTY" at the top, it fails to meet these federal standards. Rather, the warranty must be designated, for example, "FULL 90-DAY WARRANTY." A warranty is designated full when it meets the federal minimum standards for warranty set forth in the MMWA.

The requirements necessary in order to obtain a "full" designation are as follows:

1. The warrantor must remedy the defective consumer product within a reasonable time and without charge. The term "remedy," as defined by the act, means either repair, replacement or refund. If the product is a "lemon", that is, it cannot be repaired after a number of attempts, the warrantor must permit the consumer to elect either a refund or replacement without charge. If the warrantor replaces a component part of a consumer product, such replacement must include installing the part in the product without charge. It should be noted that the MMWA only requires that the remedy be given within a reasonable time. The SBCWA, while not making provisions for designating a warranty as either "full" or "limited", requires that goods be repaired within 30 days;

2. The written warranty may not impose any limitation on the duration of an implied warranty;

3. The warrantor may not exclude or limit consequential damages for the breach of any written or implied warranty, unless such exclusion or limitation conspicuously appears on the face of the warranty.

4. The warrantor may not impose any duty upon the consumer other than that of notifying the warrantor as a condition of securing a remedy "unless the warrantor has demonstrated that such a duty is reasonable." Such a reasonable condition might be the filing of a warranty registration card, if such requirement appears on the face of the warranty.

5. The warranty must extend to each person who is a consumer under the MMWA. The MMWA defines consumer as a buyer or any person to whom the product is transferred during the duration of an implied or written warranty. Thus, the manufacturer giving a "full" warranty cannot deny liability to virtually anyone who obtains possession of a product during the term of the warranty. The SBCWA limits its scope to only the retail buyer of a consumer good.

Thus, a warranty which requires that the consumer pay for transporting the product to a repair facility must be a "limited" warranty, as the remedy is not provided without charge.

Whether a warranty is full or limited, the MMWA requires that on its face it disclose a great deal of information. The warranty must set forth the identity and address of the warrantor and the identity of the parties to whom the warranty is extended. It must state a clear description of the parts, or characteristics, or components covered by the warranty and where necessary for clarification, which are excluded from the warranty. The warranty must also state what the warrantor will do in the event of a defect, for what period of time and at whose expense. It must also include a step by step explanation of the procedure which the consumer should follow in order to obtain performance of any warranty obligation. The warrantor must further provide a general description of the legal remedies available to the consumer, and any limitations on incidental or consequential damages. The warranty must also state if there is an informal dispute settlement procedure (described subsequently) available.

If a warrantor fails to designate the warranty as either "full" or "limited" or fails to disclose required information, the warrantor is subject to a lawsuit brought by the Attorney General or a Federal Trade Commission attorney. A civil penalty of up to \$10,000 per violation may be assessed.

As of this writing, conversations with FTC attorneys indicate that, as yet, no actions have been brought to enforce disclosure requirements. The explanation is that since the rule has been in effect for such a short period of time, the FTC is allowing additional time for warrantors to comply. However, actions will be brought as soon as the FTC headquarters in Washington decides that enough time has been allowed.

PRESALE AVAILABILITY RULE

Effective for products manufactured after January 1, 1977, the presale availability rule will have a vast impact on consumer awareness of warranties and on the manufacturers', distributors' and retailers' costs. The rule requires that the seller of a consumer product costing more than \$15.00 make the written warranty accompanying such product available for the prospective buyer's review, prior to sale, by the use of one of four methods.

One method provides for the text of the written warranty to be displayed in close conjunction to each warranted product. Another method allows the retailer to maintain a binder (looseleaf notebook) which contains copies of the warranties for the products sold. The binder must be maintained at a location which provides the prospective buyer with ready access. Such binder must be indexed according to product or warrantor and must be kept up to date when new warranted products or models or new warranties for existing products are introduced into the store. Under this method the retailer must display such a binder in a manner reasonably calculated to elicit the prospective buyer's attention, or place signs indicating the availability of the binders in prominent locations in the store. A third available method is displaying the package of any product on which the text of the written warranty appears. The last available method is similar to the first and allows the retailer to post a notice that contains the text of the written warranty within the vicinity of the product sold. This differs from the first in that it provides some convenience to the retailer where a number of products covered by the same warranty are being sold in close proximity to each other. It allows one notice rather than the full text being displayed next to each product.

The effect of the presale availability rule will be dramatic. The rule will allow the consumer to compare warranties much more readily than has been possible in the past. For that reason the warranty will become a much more important factor in the consumer's purchasing decision. That should provide incentive for the manufacturer to give greater consideration to his warranty.

The presale availability rule will have other effects on the manufacturer and retailer. Most obviously, someone is going to have to pay for the printing of the full text of the warranties to be displayed. There will also be expenses involved in preparing binders or in making the warranties readily available. The retailer will incur considerable expenses to satisfactorily prepare, index and maintain binders, if that is the method the retailer decides to use.

The manufacturer and retailer will have to work together in order to decide the most efficient method of the available four. Obviously the physical piece of paper containing the text of a warranty to go in a binder will differ significantly from the type of paper used if it is intended to be displayed in close conjunction with the product. The manufacturer will also want to consider printing

the text of the warranty on the box if it is the type of product which can conveniently be displayed with its box. Perhaps in the long run such costs will be passed on to the consumer. Nevertheless, it is submitted that the benefit to the consumer in being able to shop warranties more than makes up for the added printing costs.

The presale availability rule is applicable to catalog and mail order sales. The rule requires a seller to disclose for each warranted product the full text of the warranty or an explanation of where it can be obtained free of charge.

The FTC has, as yet, not brought any actions against retailers for failing to comply with the provisions of the presale availability rule. The reason stated is that it is difficult to determine if a product has been manufactured after January 1, 1977.

SBCWA REPAIR FACILITY REQUIREMENTS

One area not broached by the MMWA and fully treated by the SBCWA is the subject of maintenance of service and repair facilities. The SBCWA mandates that if there is an express warranty accompanying goods sold in California, the manufacturer of such goods must maintain, in California, "sufficient service and repair facilities reasonably close to all areas where its consumer goods are sold" to carry out the term of their warranties. Manufacturers may comply with this requirement by entering into warranty service contracts not exceeding one year with independent service and repair facilities. Providing facilities in Los Angeles and San Francisco will probably satisfy the "reasonably close" requirement.

If a manufacturer maintains service and repair facilities within California, goods subject to an express warranty must be serviced or repaired within 30 days. The buyer may extend this period by agreeing in writing or the period may be extended by conditions beyond the control of the manufacturer. Should the manufacturer of its representative be unable to repair or replace the goods, the buyer will be reimbursed in an amount equal to the purchase price paid less an amount directly attributable to use by the buyer prior to the discovery of the defect.

If the manufacturer does not maintain repair facilities in California and has not authorized a representative to do so, the buyer may return defective goods to either the retailer from whom he bought the goods or to any other retail seller of like goods of the same manufacturer. The retailer to whom the goods are returned may replace or repair at the retailer's option.

If the retailer opts to repair, he is also subject to the 30-day requirement. If he can't replace or repair he must reimburse the buyer.

If a manufacturer does not maintain repair facilities in California, the SBCWA makes such manufacturer liable to the retailer who incurs obligations in giving effect to the express warranty of the manufacturer. The manufacturer is liable as follows:

1. In the event of replacement, the manufacturer is liable in an amount equal to the cost of replacement plus transportation and handling charges;

2. If the retailer reimburses the buyer, the manufacturer is liable to the retailer for the amount of the reimbursement plus handling charges.

If the consumer has gone to a retailer for warranty repairs and has not obtained satisfactory services, or if there is no retailer in California who sells like goods for the same manufacturer

(i.e., if the manufacturer sells exclusively through mail order) the consumer may secure the services of an independent service or repair facility. If the consumer pursues this option, he is not responsible for the costs of repairs. The independent repair facility shall only hold the manufacturer liable for such costs. Those costs encompass the actual costs of repair including costs for parts, transportation of the goods or parts, plus a reasonable profit.

A few comments about the independent repair provisions are in order: Those provisions are only applicable where the wholesale price to the retailer is \$50. How does the consumer know this? The SBCWA provides that the warranty must state that the independent repair option is available on all express warranties accompanying products with a wholesale price to the retailer of \$50 or more.

What about in the mail order situation? Actually, the independent repair provisions do not expressly provide coverage in mail order situations. However, the SBCWA repair facility section is applicable to "Every manufacturer of consumer goods sold in this state and for which the manufacturer has made an express warranty...." Manufacturer is broadly defined by the SBCWA as "...any individual, partnership, corporation, association, or other legal relationship which manufactures, assembles or produces consumer goods."

The question becomes "is a mail order sale a sale in this state."

A recent amendment to the Unruh Act (dealing with consumer credit) provides that: "...a...contract... shall be deemed to have been made in this state... if either the seller offers or agrees in this state to sell to a buyer who is a resident of this state or if such buyer accepts or makes the offer in this state to buy, regardless of the situs of the contract as specified therein."

Although that section is applicable only to the Unruh Act as promulgated, the legislative intent as to mail order purchases is clear. It is the author's opinion that manufacturers who sell by mail order to California residents are subject to the independent repair facility provisions.

Those provisions cannot be waived by any manufacturer. Any attempted waiver is void.

By imposing liability on the manufacturer for all warranty repairs the SBCWA enhances the probability of consumer satisfaction. This is because the retailers, manufacturer's representatives and independent repair facilities, knowing that the manufacturer will be liable to them and that they will be paid a going rate for making the repairs, will not hesitate to accept warranty work. Further, manufacturers should be less likely to delay reimbursement to retailers (their main source of revenue) than they would be to satisfy isolated claims of the consumers themselves. This is especially true in light of provisions for treble damages plus costs and attorney's fees should the manufacturer willfully violate the reimbursement requirements.

One more unique provision in the SBCWA is a tolling provision. Where any item costs the consumer \$50 or more, the warranty period is tolled (extended) from the date the consumer delivers the goods for warranty repair until the goods are returned to the consumer.

The SBCWA also contains disclosure requirements. An important addition to the MMWA requirements is that the consumer must be provided with the name and address of each repair facility in the state or be provided with a toll free number which will provide repair facility addresses for manufacturers who elect to maintain such facilities. The retailer must also provide a list of authorized repair facilities.

CONSUMERS' REMEDIES FOR EXPRESS WARRANTIES

The U.C.C. provides the same remedies for warranties whether they are implied or express. The SBCWA also provides identical remedies but allows treble damages in instances where it is an express warranty that is breached. The MMWA does not broaden the scope of remedies available under the U.C.C. and the SBCWA. It does specifically provide that an action under the MMWA can be brought in state court or in an appropriate District Court of the United States. However, the jurisdictional requirements to bring an action in District Court are almost prohibitive. That requirement is that the amount in controversy must be at least \$50,000. If the action is brought as a class action, the named plaintiffs must number at least 100 and the amount of any individual claim must be at least \$25.00.

Perhaps the remedies available to the consumer seem slight when viewed in the light of the magnitude of the requirements placed on manufacturers and retailers. However, the most important result of the warranty legislation is that the consumer be aware of his warranty rights and that he be able to enforce them. The SBCWA's repair facility provisions are perhaps the act's most important provisions, as it places the onus on the manufacturer to establish repair facilities or to be liable to retailers or independents who provide repair facilities for warranty repairs. Presumably, this will be enforced by the repairer after the consumer has been satisfied. Nevertheless, in order to be effective at all, consumers must be aware of their rights and must take advantage of them.

MAIL ORDERS

An oft heard complaint by consumers in dealing with mail order houses is that, "I ordered a product and sent my money in months ago and the mail order house just kept it and didn't send me what I ordered." A new provision of the California Business & Professions Code requires that any business conducting a mail order or catalog business in California must, within 6 weeks, either (1) deliver the goods ordered; (2) make a full refund; (3) send the consumer a letter advising him of the duration of delay and offering to send him a refund within one week, or (4) send the consumer substituted goods of equivalent or superior quality.

A violation of this section subjects the mail order or catalog business to a fine of \$2,500 which may be recovered in an action brought by the Attorney General of the State.

Therefore, if you have had experience with a mail order or catalog house which violates this provision, call the local Consumer Affairs Bureau or the Attorney General and notify them of the details of your experience.

INFORMAL DISPUTE SETTLEMENT MECHANISM

Congress, in enacting the MMWA, has explicitly declared its purpose to encourage warrantors to establish procedures for settling consumer disputes fairly and expeditiously through informal dispute settlement mechanisms (IDSM). The MMWA does not require the establishment of an IDSM but only states that any mechanism incorporated into the terms of a written warranty must meet the MMWA's standards. For the microcomputer industry, the early establishment of an industry wide informal dispute settlement mechanism could very well be the solution to one of the great problems facing the industry. That problem grows out of the situation where a manufacturer of a peripheral device claims that it is compatible with "all S100 buses" or "all 8080 microprocessors."

A perfect example of a real-life compatibility problem is set out in another letter appearing on page 14 of the November/December

1976 issue of Dr. Dobb's Journal. There Glenn Tenney describes a programming problem resulting from IMSAI's use of a NEC 8080A chip instead of an Intel chip. He states that NEC reported, in a confidential letter to IMSAI, some "minor" differences between their chip and Intel's. Being a Senior Designer for Compro, Mr. Tenney stated that he was more disturbed by not being informed of the differences than the incompatibility itself.

Being an "appliance operator" the author is more concerned about the incompatibility. IMSAI responded to Mr. Tenney's letter with information summarizing the differences between the chip. Well, that's satisfactory for someone who can understand the difference described, but as microcomputers become as common as TV sets and are purchased by people who can use them, but don't know how they work, the "solution" adopted in this instance is definitely unsatisfactory.

How can the establishment of an IDSM help solve this technological morass? In order to see how it is necessary to have a better understanding of what the IDSM "mechanism" actually is. The MMWA provides no definition. The IDSM rule only governs the minimum requirements of the mechanism with respect to its organization, operation, record keeping and qualifications of members. The minimum requirements are broad enough to encompass a wide range of descriptions.

In general, the proper model would be the labor dispute analogy. Almost all collective bargaining contracts include a grievance procedure with "in house" settlement steps leading to a terminal step where a decision is made by a neutral arbitrator (a mechanism). In this analogy the consumer can be thought of as an aggrieved worker and a warrantor can be thought of as management. In the microcomputer industry the neutral arbitrator or mechanism could be an association consisting of members who are microcomputer users, manufacturers, retailers, distributors, microcomputer magazine publishers, etc. Because the validity of some "grievances" in this industry would necessarily require the use of highly sophisticated and expensive testing equipment, the "offices" of the mechanism could run the gamut from the basement workbench of an experimenter, to the repair shop of a retail outlet, to the facilities of a storefront operation designed to exclusively act as an office for a microcomputer industry-wide informal dispute settlement mechanism.

While the MMWA does not require a warrantor to establish an IDSM, if a warrantor does establish an IDSM, the consumer must resort to the IDSM procedure before filing a lawsuit under which he would obtain the benefits provided by the MMWA. That is to say, even if an IDSM is established, the consumer may still sue and obtain remedies based upon the U.C.C. and the SBCWA. Further, a consumer may attempt to resolve the dispute with the warrantor prior to going to the IDSM procedures. If a consumer does resort to an IDSM, it is important to note that neither the consumer nor the warrantor is bound by the decision of the IDSM. However, if either decide not to be bound by the decision of the IDSM, the IDSM decision can be admitted into evidence should a legal action be brought.

Given that the warrantor is not required to develop an IDSM, that the consumer need not resort to the IDSM and that neither the consumer nor the warrantor is bound by a decision of the IDSM, what are the benefits of the establishment of such a mechanism? As previously indicated, in the microcomputer industry the nature of warranty disputes (i.e., the compatibility problem) are such that very few individual consumers have the necessary knowledge and test equipment to prove that a given peripheral is not "S100 compatible." Many manufacturers of peripherals cannot afford to test their peripheral with every S100 device. An industry-wide IDSM would

provide for the warrantor a cost effective procedure for dispute settlement. Additionally, it is conceivable that in the dispute settling process, an expert technical consultant could solve the non-compatibility problem, thus saving the warrantor thousands of dollars in research and development expenses. A potential consumer with limited technical expertise would feel more comfortable purchasing an expensive home computer if he knew that there was a mechanism to which he could resort if a manufacturer's claim of compatibility proved incorrect.

REQUIREMENTS FOR AN IDSM

If an IDSM is used, the warrantor must disclose that such a mechanism is available on the face of the written warranty. The warranty must disclose the name and address of the mechanism or a telephone number of the mechanism which consumers may use without charge. Further, the warranty must disclose the legal ramifications of resorting to the mechanism, as already discussed. The warrantor must also provide a description of the mechanism procedures, the time limits adhered to by the mechanism and the type of information which the mechanism may require for prompt resolution of warranty disputes.

The MMWA provides that the mechanism shall be funded and competently staffed at the expense of the warrantor. It further provides that no member of an IDSM deciding a dispute can be a party to the dispute or an agent or employee of a party to the dispute. There is a specific provision for the use of a technical consultant who works in the industry, so long as such consultant or consultants make up no more than one-third of the members who will be deciding a given dispute.

Once a dispute is brought into the IDSM mechanism, a decision must be rendered within forty days. The mechanism must then disclose to the warrantor its decision and the reasons therefor. If a decision would require action on the part of the warrantor, the warrantor must inform the mechanism whether, and to what extent, the warrantor will abide by its decision. The mechanism must then disclose to the consumer its decision and the warrantor's intended action. The mechanism must then ascertain from the consumer within ten working days whether performance by the warrantor has occurred. There are also extensive record keeping requirements.

In light of the highly technical nature of many of the problems which face microcomputer manufacturers, retailers and consumers, the establishment of an industry-wide IDSM at this early stage of the industry's development is a factor that might lead to an earlier broad-based consumer acceptance and use of microcomputers than anticipated. If a toaster doesn't toast, the consumer knows it and the warrantor knows that it is his responsibility. If an "8080 compatible" peripheral doesn't work, the consumer doesn't know if its due to a defect in the peripheral. The problem evolves into a pointing the finger at the other guy game. The bottom line in such a situation can only be bad news for the microcomputer industry as a whole.

The purpose of the above description is to suggest a possible solution to a major problem facing the microcomputer industry. It is intended as food for thought. Nevertheless, the establishment of an industry-wide IDSM would alleviate the finger pointing and would establish a state of the art dispute settlement mechanism in a state of the art industry.

HERE COMES THE BRAIN-LIKE, SELF-LEARNING, NO-PROGRAMMING, COMPUTER OF THE FUTURE

Klaus Holtz
310 Guttenberg
San Francisco CA 94112

An electronic computer which closely simulates human brain functions is now possible and practical. This computer is educated like a human child without any need for programming or supervision of its internal functioning.

Contrary to what you might believe, this computer is neither hard to understand nor complex and expensive to build. A few hundred dollars and some standard electronic hardware will yield the first primitive but useful devices. The evolution of this new technology may yield the intelligent self-learning robot and the apparently self-aware super computer in a few years' time.

This article will present a simplified review of computing theory. It will introduce knowledge processing, infinite dimensional networks and a theory of human brain function. It will suggest a few practical applications useful for text processing, cryptography, data compression, language translation, and home computing.

Theory of knowledge processing

How do we go about designing a brain-like computer? Do we invent a fancy new device called a neuristor? Do we cook up a chemical brew which spontaneously organizes itself? Or shall we depend forever on that old fashioned biological method?

No! The answer will not be found in exotic new devices but in the re-evaluation of our computing methods. All presently known computers may be roughly classified as methodical, analytical, or empirical computers. Fig. 1 shows a comparison of the three computing methods.

Most present-day electronic computers are methodical or stored program computers. A set of input data is manipulated by a stored program to yield a set of output data. The stored program presents a method which specifies what steps must be taken to convert the input data into some desirable output. Some engineers believe this to be the only way to fly.

Let us consider how a creature with a methodical brain would behave in nature. This creature would be born with a completely pre-programmed, pre-wired brain without the ability to learn or accumulate new knowledge. It would react to a certain situation in a predictable way precisely every time. A slight defect would kill the creature since it would have nobody to repair its program.

This is obviously not the way our brain works. The question is why have we been building such overstuffed adding machines? Well, they are simple and they are so different from ourselves. They do things easily which are difficult for us. They are great for science and bureaucracy with their rigid equations and rule books but they will never yield a brain similar to ours.

A second type, the analytical computer, is used by Nature and Man. The old analog computer used this method. An input pattern is examined to yield an output response directly dependent on the input pattern. In analog computers, a set of input variables is combined by a pre-wired network of operational amplifiers to yield an output function. The result is obtained, not by a stored program, but by a pre-wired weighted network. Mother Nature uses this method for all its primitive creatures and most of the peripheral processing in our brain is organized this way. A fly, for example, reacts to movement in its direction detected by its eyes by activating its legs and wings for flight.

This type of computer is useful for primitive reflex responses which can be pre-wired ahead of need. The network is not very flexible and depends on outside wiring either by man or in nature by the forces of evolution. The assembly of analytical infinite dimensional networks will be explained later in the text.

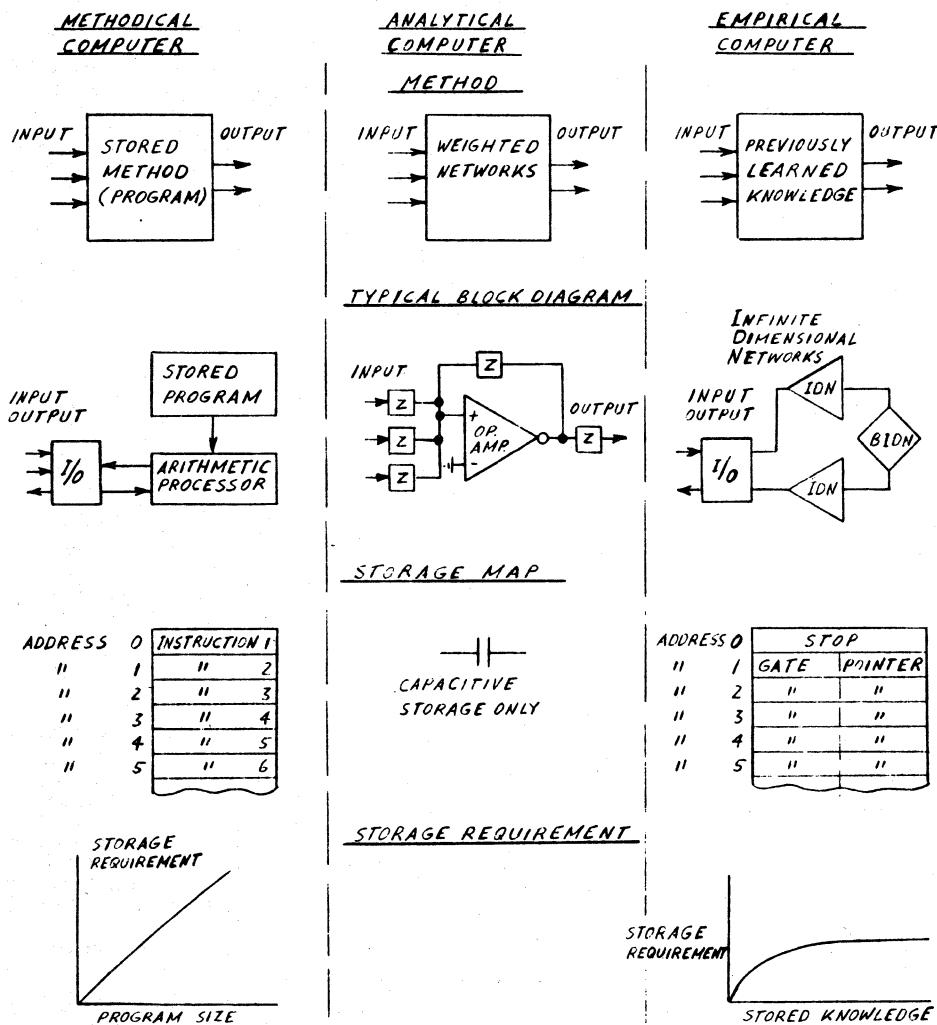
Enter the empirical computer. This computer contains no stored program but learns its behavior pattern by interaction with its environment. A certain input situation is analyzed and a response is formulated according to pre-learned knowledge. This knowledge is not programmed but obtained from experience, abstract learning, or duplicating the successful behavior of other people. An empirical computer would be very flexible. It may have several responses for each situation. It would constantly update its knowledge and behavior repertoire according to its changing environment but always remember past experiences.

An empirical computer is now possible but before we can build it we have to analyze exactly what knowledge consists of and how it may be stored. "Knowledge" unlike mere "Data" cannot be stored in a filing rack type array. It requires a self-assembling tree-like array at which all knowledge is related to other knowledge.

The following five principles will define knowledge processing. They are valid both for the new computer and to the human brain. They may be hard to understand all at once but will become more obvious in the later text. They may provide a more profound understanding of ourselves and natural philosophy. They may even sprout a new freak like religious group in Southern California.

The idea presented in this paper is protected by a U.S. "Patent Pending." The author reserves all patent rights and the disclosure of this paper does not imply a permission to use the invention for commercial purposes.

FIG.1 THIS FIGURE COMPARES THE 3 MOST IMPORTANT COMPUTING METHODS. THE EMPIRICAL COMPUTER WILL BE EXPLAINED LATER IN THE TEXT



The process of specifying concepts is explained later in infinite dimensional networks. Multi-dimensional space is only a description of the mathematical method. It does not imply the existence of such space.

1. "Knowledge is the definition of concepts and their interrelationships."

This principle specifies exactly what knowledge consists of.

Few objects in our environment are really understood in detail as to their composition and relationship to other objects. A ROSE for example consists of flower, organelles, proteins, amino acids, atoms, etc. It relates to other flowers and animals in the garden ecology, to climate, planetary motion, etc. Not even the words we use, such as proteins or ecology, are really understood. We gain knowledge however by defining the composition and relationships one fact at a time. The more tightly this network of facts is knit the more knowledge we possess.

2. "Knowledge is accumulated in discrete quantum increments which change a concept to a different concept."

This principle makes it possible to accurately measure knowledge and to store it in digital memories.

The concept ROSE for example is changed to a different concept RED ROSE by the addition of its color. The change is always in discrete increments which can be measured, quantified or named.

3. "Only a few of the infinitely possible concepts exist in the real world."

This principle sets a limit to variety and makes practical computers possible by eliminating the infinite. Every possible concept must, however, have a chance to become real.

Taking written English as an example, an almost infinite number of possible words could be assembled from the 26 letters of the alphabet. Only a few thousand words exist in everyday use.

4. "Established knowledge excludes redundant knowledge."

This principle specifies a mechanism which allows knowledge to be stored in highly repetitive and random order since every fact is only stored once.

Learning is the process of storing new knowledge. A human, when presented with a new fact, will learn that fact to increase his knowledge. If the same fact is presented a second time it will not increase his knowledge and is therefore not learned again. A mechanism to filter out already learned knowledge must be incorporated into a knowledge processing computer. This will allow only new facts to be learned and prevent the accumulation of useless redundant data in the memory. A fact, once learned, must never be learned again unless of course a memory is faulty and keeps forgetting.

The tools to build a knowledge processing computer are the infinite dimensional networks which will be explained now.

Introduction to infinite dimensional networks

1. "Every concept may be specified like a point in multi-dimensional space."

This principle is the very basis of knowledge processing. It provides a means of defining and manipulating concepts. A concept may be anything that can be named or described, such as physical objects, persons, processes or ideas. Since a single point is sufficient to specify any concept, its complexity becomes irrelevant. This makes it as easy for a human to learn a single new number as to remember the face of a stranger.

In the introduction to knowledge processing it was said that any concept may be specified like a point in multi-dimensional space. If this is to be useful, a method of specifying points in multi-dimensional space must be provided.

An infinite dimensional network may be envisioned as a tree like array. These networks assemble themselves automatically from the supplied input data without programming or human supervision. This results in a mechanism which strikingly

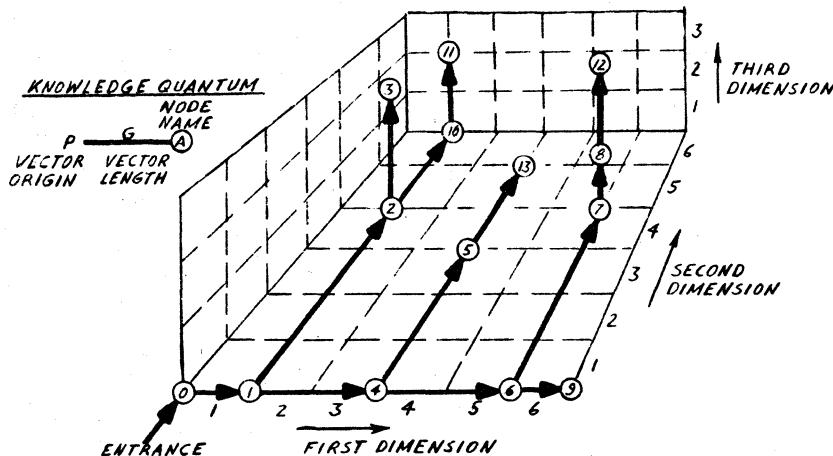
simulates human learning. The networks used later in electronic computers are both real and imaginary. They are real insofar as they are stored in digital memory devices but imaginary because they are not actually connected with gates or wires. They exist only as a mathematical equivalent.

Since infinite dimensional networks are brand new concepts we have to start from the very beginning.

Fig. 2 shows a 3 dimensional network that is all that can be illustrated on paper. This network may be used to specify various points in 1, 2 or 3 dimensional space. A specific point in 3 dimensional space such as "3" may be specified by its three vectors in the first, second and third dimension which are 1, 4 and 2. Each of the vectors presents an increment of knowledge about the location of point "3" but only in combination with the vector's origin.

This first step allows us to quantify knowledge. A quantum increment in knowledge consists of a fact (vector length) and information about its location in the network (vector origin). To specify a point in 3 dimensional space three knowledge quanta (3 vectors) are required. We do not have a name for a unit of knowledge at this time but knowledge can now be precisely measured. Future generations of students may graduate with a certified knowledge of 524 @\\$*!ç#&.

FIG 2 3 DIMENSIONAL NETWORK



Every specified point in the network is marked by a separate name or number. The nodes are like network nodes from which many branches may sprout. A very important first law of infinite dimensional networks is that: "Every node in the network is unique and no 2 nodes in the same network may have the same name or number."

Point "3" in Fig. 2 is therefore unique and can only be reached over the three vectors 1, 4, 2 in the right sequence. Each node is therefore like a fixed point in multi-dimensional space whose location is precisely known. The numbers to the nodes are assigned in sequence as required. We do not need to reserve numbers for empty coordinates and as law 4 of knowledge processing states: "Only a few of the infinitely possible concepts (nodes) exist in the real world." The fact that each network node presents a concept as a point in multi-dimensional space will become clear later.

Fig. 2 is useful for explaining the first step. Fig. 3 shows the network from Fig. 2 rearranged in a more convenient way. Instead of showing the vectors in their physical length they are specified as "Gate" values.

A "Gate" is now a condition code which specifies the condition which must be met to cross to the next node. In our example it was the length of a vector but it may be facts, input data, conditions or anything that can be measured or named. A second law of infinite dimensional networks is that: "Every node may sprout an unlimited number of branches but each gate in a branch must be unique. No 2 branches from the same node may have the same gate."

A "Pointer" value is introduced at this time. This value specifies the node from which a branch originates. It is therefore the number of the previous node in the sequence.

The node name or number is now called "Address." Every address (node) number specifies a point in multi-dimensional space. No 2 nodes in the same network may have the same address.

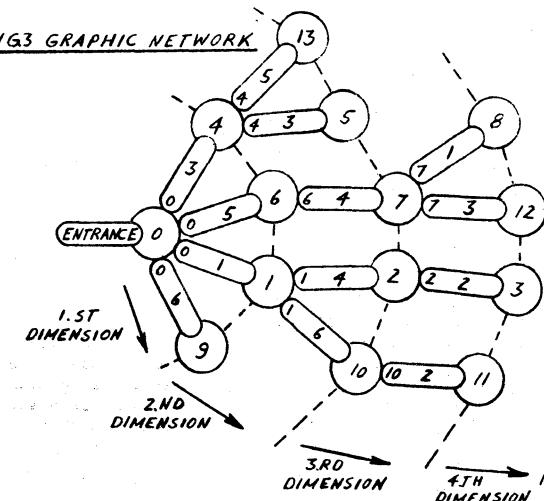
Every knowledge quantum is now specified by a "Gate" and a "Pointer" which are together called a "Matrix." Each branch leads to another "Address."

Verify that Fig. 3 contains all identical information from Fig. 2 but in a more convenient way. Point (address) "3" may still be specified precisely by the 3 vectors (gates) 1, 4 and 2. This network may spread into additional dimensions if required.

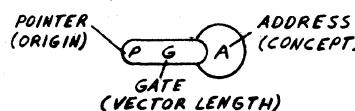
Fig. 4 shows the final step to arrive at a stored infinite dimensional network. Its construction is no longer obvious but it is now suitable for computer processing.

The stored network in Fig. 4 was derived by simple disassembly of the nodes from Fig. 3 and storing them according to their address (node) number in a common digital memory. The network in Fig. 4 contains now all identical information of Fig. 2 and Fig. 3. This network can spread into many dimensions without limit. It can specify

FIG 3 GRAPHIC NETWORK



NETWORK NODE
(KNOWLEDGE QUANTUM)



a point in multi-dimensional space using only the original vector inputs. In our example point "3" was specified by its 3 vectors 1, 4 and 2. Starting address 0 contains a unique stop code marking the network entrance. Starting from this location we try to locate a memory location (matrix) containing a Pointer 0 (origin of that branch) and a Gate 1 (first vector.) We locate this in Address (node) 1.

Address 1 becomes our new point or vector origin. A matrix containing pointer 1 and gate 4 (second vector) is found in address 2 which becomes the new pointer. Address (pointer) 3 is found after one more step with Pointer 2 and Gate 2.

The above procedure is very cumbersome for humans but not for electronic registers. The next chapter will show how these networks assemble themselves from the input data. There is no need for a human operator to specify address and pointer numbers or to supervise the internal operations of the computer.

FIG.4 STORED NETWORK

ADDRESS	GATE	POINTER
0	ENTRANCE	
1	1	0
2	4	1
3	2	2
4	3	0
5	3	4
6	5	0
7	4	6
8	1	7
9	6	0
10	6	1
11	2	10
12	5	7
13	5	4
14		

NETWORK NODE
(KNOWLEDGE QUANTUM)

ADDRESS	GATE	POINTER
A	G	P

Self assembling infinite dimensional networks

"In the 1980's Minsky and Good had shown how neural networks could be generated automatically self-replicated- in accordance with any arbitrary learning program. Artificial brains could be grown by a process strikingly analogous to the development of a human brain. In any given case, the precise details would never be known, and even if they were, they would be millions of times too complex for human understanding."

Arthur C. Clark 2001 A Space Odyssey

This prediction is too conservative as usual but otherwise very accurate. There are many kinds of networks, each useful for a special purpose. The super simple answering system shown in Fig. 5 is useful for explaining the functioning of the three most basic network types. The device gives answers to the four questions:

Question	Answer
ROSE	-
ROBOT	-
IDN	-
IDEA	-

The input data to these networks are ASCII characters and the "concepts" English words. The three networks are "learning network," "bipolar network," and "output network."

The simple learning network shown on the left of Fig. 5 is the most important basic network. This is not just a funny way to store data but genuine learning. As defined earlier "learning" is the accumulation of new knowledge" and "knowledge is the definition of concepts and their interrelationships."

These networks assemble themselves from the input data under the control of a micro-processor. This processor is controlled by a micro-routine which determines the network type desired. Except for this micro-program which may be considered part of the hardware there is no programming required.

Fig. 5 shows the steps at which a word like ROBOT is learned by the network. The five letters R, O, B, O, T, are the input which must be converted into a single point according to law 1 of knowledge processing. Address 0 contains a stop code which marks the network entrance. Starting from address (node) 0 a branch containing gate R and pointer 0 must be located. The letter R is the first letter of the word ROBOT. It is like a gate which determines which network branch is to be selected. Pointer 0 is the address of the node currently selected. Scanning forward through the memory we try to locate a memory location which contains the correct gate and pointer (R, 0). In our example, we find it at address 1. We update the pointer to the address value at which the gate and pointer pattern was found. The next letter of the word ROBOT "O" is combined with the current pointer 1 and the memory scan is continued to locate the new pattern. The above procedure is repeated for each letter of the word ROBOT until we reach address 7. This address 7 totally identifies the word ROBOT. If we start the same operation again with the word ROBOT address 7 will again be identified.

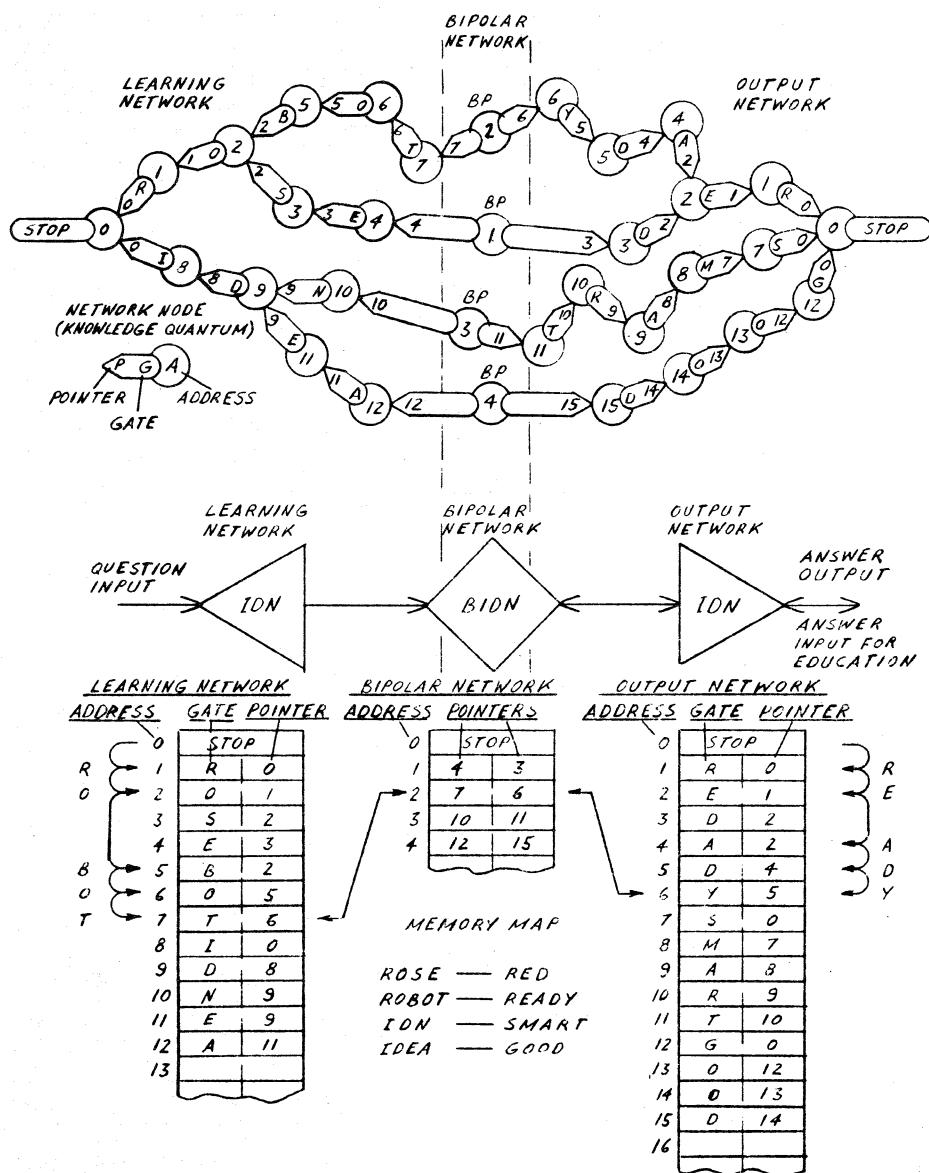
What if a word is not yet contained in the network? The same routine described above would apply. The network generator would scan through the entire used memory field in order to locate the correct gate-pointer pattern. If it is not found the gate and pointer pattern is stored in the next free memory location and the pointer value updated to that address number. A new word is learned by the automatic addition of new nodes at the end of the memory. Note that a new word follows established patterns before new nodes are added. The words ROSE and ROBOT in the example share the first 2 letters RO.

Following the above steps any previous word is identified by a unique number which totally identifies each word. Any new word is learned by the addition of new nodes at the end of the memory. Note that each word is only learned once since the network generator would find the new trail the second time. Also, since sections of the trails are shared by different words, the requirement for new memory would decline and finally saturate for very large networks. There are only so many possible letter combinations in the real English language.

The word ROBOT in the example leads always to address 7. Address 7 is therefore the single point which specifies the concept ROBOT. This point can now be connected (related) to other points in the networks.

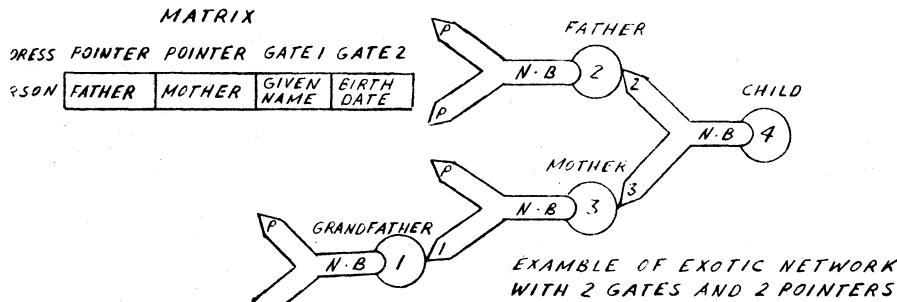
If the answer to the question ROBOT is READY we must encode the word READY to a single point in another network. The two concept points can then be connected. The encoding of the word READY to address 6 is done in an identical manner as described above for the word ROBOT. The connection is accomplished by the bi-polar network. This network accepts the address number from the input network and scans through the memory to locate it in the input pointer field. In the example, input pointer 7 for ROBOT is found in address location 2. The address of this memory is only used

FIG 5 SELF ASSEMBLING INFINITE DIMENSIONAL NETWORK



network is generated by a fixed (genetic) micro-routine which is determined by the network's purpose. All networks reduce complex input patterns to single address numbers which are then related to other address numbers representing other patterns. The input data pattern is compressed but never destroyed. Each pattern may be fully retrieved from the address number. The content of each word in the digital memory is unique. No two addresses in the same network may have the same content. The network memory is accessed by address or content which can be accomplished by scanning conventional memory devices or by future content addressable memories.

FIG. 6 HEREDITARY TREE



Learning Network Micro-routine

Matrix

Address (location)	GATE		POINTER	
	Input character	Output number		

Label	Operation
START	Set pointer value to 0
LOOP	Wait for input character
	Examine input character
	If input character is an "End of sequence" code such as space, full stop, period etc. go to EXIT
	Combine input character with present pointer value to obtain a matrix
	Search or scan the memory to locate the present matrix in memory
	A) Find present matrix already stored in memory Update pointer value in matrix to the address value of the memory location in which the matrix was found. Go to LOOP
	B) Matrix not found in memory (After scan through entire memory) Store present matrix in next free memory location. Set pointer value in matrix with the address value of that memory location. Go to LOOP
EXIT	The present Pointer value is the number which defines the entire input sequence. Supply this number to the next network. Go back to START

Bi-polar Network Micro-routine

Matrix (memory content)

Address	Input pointer	Output pointer
	Input sequence	Output sequence

Label	Operation
START	Wait for input pointer number from previous network Search memory to locate a matrix with the correct input pointer A) Find input pointer in memory Supply output pointer from same memory word to output network. Go back to START B) Input pointer not in memory Human education is required
	Test if learning is enabled A) Learning not enabled Put special pointer code "DON'T KNOW" to output network. Go back to START B) Learning enabled Wait for output pointer supplied by human input and encoded by output network
	Store input and output pointer in next free memory location. Go back to START

Output Network Micro-routine

Matrix

Address (location)	Gate	Pointer
	Output data character	Next address

Label	Operation
START	Wait for the pointer number input from the previous network
LOOP	Fetch the content(matrix) of the memory location specified by the pointer value Examine matrix. If the matrix contains a stop code (or address is 0) the output sequence is complete. If true go back to START Take output data character from Gate section of matrix and send it to the output device Take new pointer value from the matrix. This pointer value will become the address of the next matrix to be fetched from memory. Go to LOOP

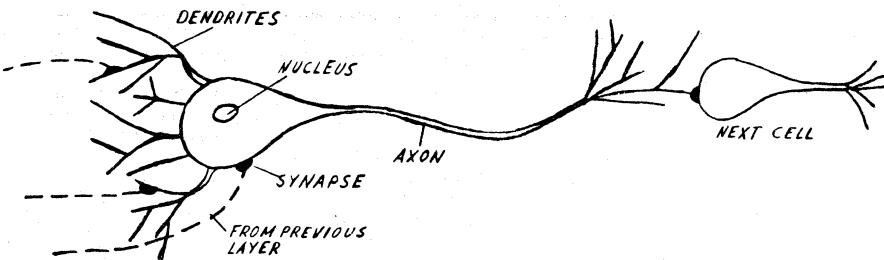
The human brain

Now that we know what theoretical infinite dimensional networks are and how they are assembled let us examine if the human brain could be organized this way. It is usually not an engineering task to explain brain functions. After all, we engineers want to build computers not duplicate the human brain. But why should scientists have all the fun? Here then is a highly speculative prediction of how the human brain may be organized.

Let us assume that the brain is indeed a biological infinite dimensional network which follows the five laws of knowledge processing. How could such a network assemble itself in accordance with the known functioning of the nerve cells? We must not look so much for structure as for the underlying principle. After all, the theory of Evolution was not explained by showing the detailed relationship among animal species, but by inventing the mechanism of mutation and natural selection.

To understand the system we must first examine its smallest component, the nerve cell. This cell, like other cells, is a semi-independent living organism which is self-maintaining, self-duplicating, and produces its own internal power from externally supplied fuel. The nerve cell is specialized for data processing and is usually a member of a vast network of interconnected nerve cells. Some nerve cells have specialized sensor apparatus which serve as input to the networks. There are several kinds of specialized nerve cells but most are variations of the same general design.

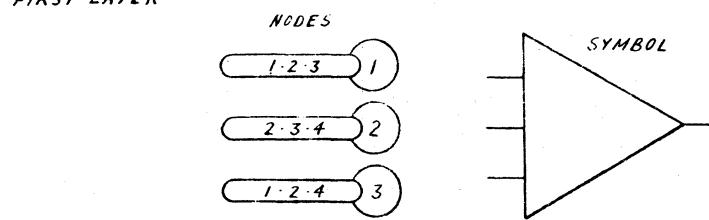
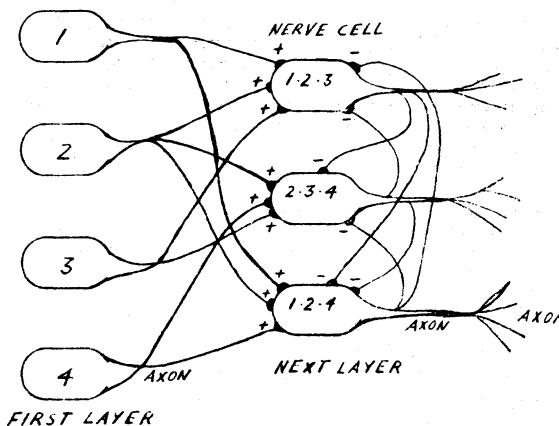
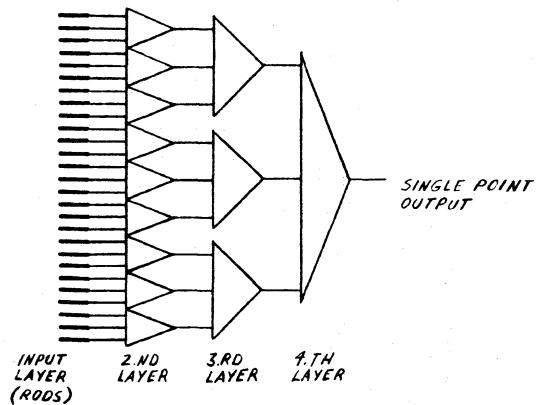
A nerve cell as shown in Fig. 7 has a cell body with a tree like branching called dendrites. These dendrites present an extended interconnection plane which allows other cells in the network to connect to the cell. A long fiber on the other end, the "Axon," also branches on the tip to connect to other nerve cells. The axon of one cell connects to the dendrites or the cell body of other cells through small nodules called synapses. There are two kinds of synapses. One kind is positive to increase the likelihood for the nerve cell to fire when a signal arrives from this

FIG. 7 NERVE CELL

synapse. The other is negative decreasing the cell's firing rate. The message from a synapse is not electrical but transmitted by a chemical messenger through the cell wall. A nerve cell excited by positive inputs from synapses will fire by reversing its electrical potential. This signal is transmitted through the axon to the synapses of other nerve cells. Information flow is therefore from the dendrites to the axon. Each cell has an excitation threshold which makes it either fire or which causes no effect at all. A nerve cell quickly restores itself after firing.

According to law 1 of knowledge processing, every concept or input situation must be reduced to a single point or in this case the firing of one specific cell. For example, the picture of a horse received through our eyes must result in the firing of one specific cell in our brain. This cell may then be connected or related to other cells which present feelings or knowledge connected with the horse. We should realize at this time that the brain is not mathematically clean and the decoding process may be duplicated several times for redundancy.

Figs. 8 and 9 show that the input decoding in our eyes is organized in several distinct layers (or dimensions). Each layer extracts information from the previous layer of cells. A prediction of this new theory is that each layer of cells is genetically separate from other layers. Markers on the nerve cells will determine if a synapse connection is possible. The axons from one layer of cells can only connect to specific other layers of cells. The axons will grow to any length to connect only to its genetically determined target area. The genetic key of a cell layer is somewhat equivalent to the pointer value specified earlier. Another prediction is that all synapse inputs from other layers are excitatory (positive) while all synapse inputs from the same layer are inhibitory (negative). This is due to the fact that each branch from a network node must be unique. Only one cell in the next layer can fire and must therewith inhibit the firing of other cells in its vicinity.

FIG. 8 DECODING UNITFIG. 9 PARALLEL INFINITE DIMENSIONAL NETWORK

A third prediction is that the number of excitatory (positive) inputs to cause a cell of a specific layer to fire is an integral multiple genetically determined. For example, all cells in a specific layer will fire if three positive levels are received. The number of levels for each layer of cells is the same for each cell.

If these three predictions are found true, we can formulate a structure and principle to explain how the random nerve cells in our brain are able to organize themselves into such precise networks.

Starting from the input from the eye, these inputs in the micro level are such simple patterns as points, straight, horizontal or vertical lines or edges. A specific pattern such as a vertical line will cause a certain set of nerve cells to fire. Let's, for example, assume that the next layer of cells needs three positive input signals in order to fire. In this case each of the cells in the next layer will have three connections to three distinct cells in the previous layer. If these three cells fire the cell in the next layer will be excited sufficiently to fire itself. Each cell in the next layer connects to a unique set of cells in the previous layer so that no two cells connect to the identical set of three cells. A cell in the next layer, once excited to fire, will inhibit the firing of other cells in the same layer by connecting inhibitory synapses to neighboring cells. This inhibitory range is however limited. The vertical line input pattern from our example will cause a specific set of three cells to fire. If a cell in the next layer is connected to those specific three cells it will fire and therewith decode and reduce the pattern. Let us assume that there exists some kind of equilibrium either electrically or more likely chemical. If the three cells in the input layer fire they cause a disturbance which is compensated by the firing of the cell in the next layer.

If all input patterns in the input layer are decoded by cells in the next layer this equilibrium is preserved for each input pattern.

Let us assume that now a new set of three input cells would fire. This would be a pattern never encountered before and must therefore be learned. The three input cells firing would solicit no response from the next layer and a (chemical) imbalance would develop. The next layer of cells must contain a few virgin cells which are not yet connected to any cells in the input layer. These virgin cells would be stimulated by the (chemical) imbalance to grow dendrites toward the axons of the three cells and to connect with synapses. Once a virgin cell is connected to the new pattern it would fire and therewith remove the imbalance. Many virgin cells may have been stimulated to grow but the first cell to connect, which would also be the one most efficiently located, would remove the imbalance and inhibit further growth of the virgin cell. The connecting cell would be frozen into the new configuration and would become a member of the network which is not affected by a new imbalance. The connecting cell could also duplicate at this time to replenish the supply of virgin cells but Nature may not be that smart.

The above mechanism would explain how a layered structure of random nerve cells could organize itself into a biological infinite dimensional network. Every complex input pattern would be compressed by successive layers of cells until finally only a single cell would fire for such a complex pattern as "horse." This cell would then connect with the same mechanism to other cells connected with facts or emotions about horses.

A biological infinite dimensional network is of course not as clean as a mathematical computer network. Let us review how the laws of knowledge processing apply to the brain.

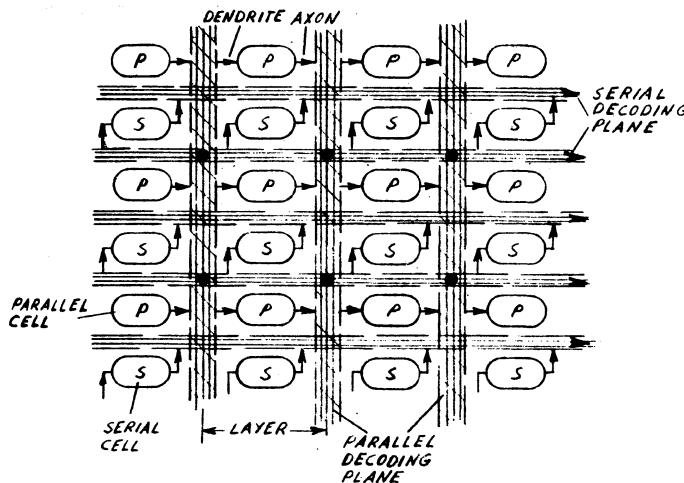
1. Every concept may be specified like a point in multi-dimensional space.

Every layer of cells presents a separate dimension. Any complex input is reduced to a single firing cell which is then connected to other cells during the learning procedure.

2. Knowledge is the definition of concepts and their interrelationships.

Knowledge is stored by the interconnection of nerve cells. The more cells are connected in the network the higher the knowledge. There are two kinds of knowledge: definition of concepts and interrelation or more specific parallel and serial knowledge. In the higher centers of our brain there are probably two cell planes intersecting each other at 90° as shown in Fig. 10. One layer system consists of parallel planes which define parallel concepts by reducing

FIG. 10 PROBABLE CELL LAY-OUT OF HIGHER BRAIN FUNCTION



it to single firing cells. Another layer of cells penetrating the parallel layers by 90° and connecting the various parallel concepts into a serial higher concept. These two classes of cell layers may be separate or intermixed but genetically distinct.

3. Knowledge is accumulated in discrete quantum increments which change a concept to a different concept.

A quantum of knowledge is learned by the connection of a nerve cell into the network. A quantum of knowledge is expressed as a unique pattern (gate) emanating from a unique layer of cells (pointer.)

4. Only a few of the infinitely possible concepts exist in the real world.

The nerve cell network will only accept real input from the real world which is limited in its variety. The network will expand to accept new knowledge but only learn actually existing patterns, ignoring all merely possible patterns. It may also disconnect or forget seldomly used patterns.

5. Established knowledge excludes redundant knowledge.

A nerve cell once connected into the network will exclude other cells from connecting to the same pattern. Biological brain networks however forget and need constant exercise to stay connected. A nerve cell which degenerates and opens its connections must be replaced either by virgin cells (re-learning) or by re-connection.

The human brain is very complex indeed and the details of its construction may elude us for a long time. There are perception of parallel and serial patterns, movements, shade, color, time and complex associations, but all the complex functions will probably turn out to be variations of the same general design explained here. We do not need to understand the human brain in order to build useful computers only the basic principle: "The brain consists of layers of genetically distinct specialized nerve cells which are interconnected into an infinite dimensional network."

For us engineers, it means that a brain-like computer should consist of specialized network generators, each generating a specialized network. The network generators should be interconnected into complex systems according to the application.

This excursion into biology is fun, but useful computers must be built to earn a profit for ourselves and our company. Simple practical computer systems are discussed in the next section.

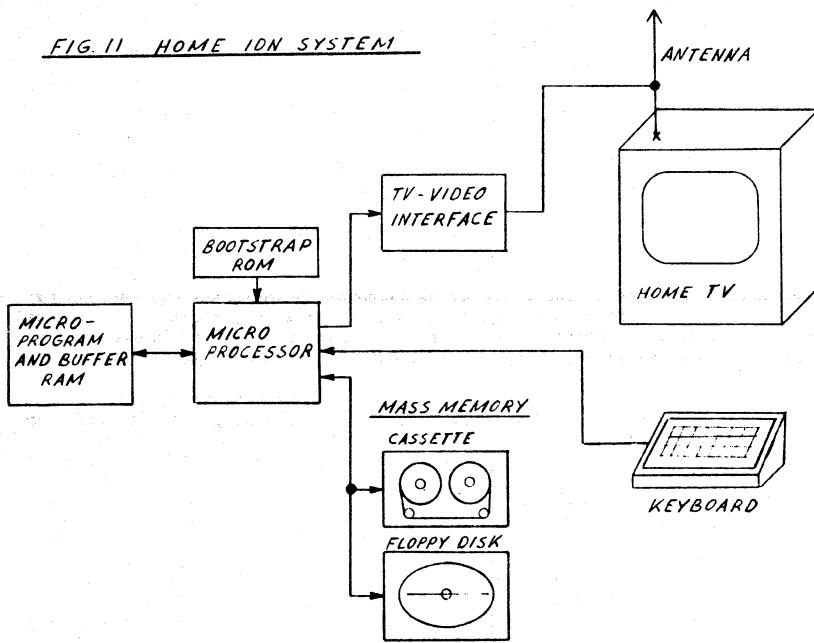
Practical applications

Much time and development effort will be required before the first super computers will roll off the assembly lines. To finance this development effort some simple devices should be built today to be sold at a profit. The experience gained with those simple devices could be used to design more and more advanced computers where each generation could pay for the development of the next generation of computers.

Here, then, are very simple examples which are easy to build and guaranteed to work. Fig. 11 shows a home IDN system which could be sold for less than \$200 in quantity. This system consists of a cheap microprocessor to generate the network and to control the interface to the peripherals. A bootstrap ROM is used to read the micro-program for the network generator from the first section of tape or disk into a RAM. This RAM would contain several network generator and control micro-routines according to the task. The RAM can also be used as buffer by the microprocessor. A keyboard is used as for output display. A cheap mass storage device could be used to hold the networks. A tape cassette or floppy disk are ideal and inexpensive. The first portion of the memory would hold the micro-routines for the microprocessor. The rest of the memory contains several independent networks of arbitrary complexity. Storage capacity of those devices is very large since no gap between the network data is required and data storage is continuous over the whole length of tape. A second recording media or tape drive may be convenient.

What advantage would such a system have over conventional systems and what could it be used for? For once this system would not require any programming except for the

FIG. 11 HOME IDN SYSTEM



micro-routines which could be standardized and sold on pre-formatted tape. The system would be truly universal and used profitably for any of the systems described below.

Fig. 12 shows a system example consisting of a single network. The network could be generated by typing some English text such as a novel or dictionary over the keyboard into the microprocessor which would store it as an infinite dimensional network on the tape. Sequence of words and word repetitions would not matter since every word is learned only once. After some typing most of the required words would be learned. A complete vocabulary is not required for most applications. Missing words can be made up by word fragments.

Such a single network has three features which could be exploited:

1. "Every word or group of words is stored only once." This could be used to merge files, such as customer lists, at which each entry would be stored just once no matter how many times it is repeated. A system to teach correct spelling could be developed since every legal word in a dictionary would be recognized. Misspelled words would not be found in the network and cause a warning signal.
2. "Every network is internally unique and depends on the sequence of the original education." The address number output representing each English word input is in an unbreakable code which can only be retrieved by an identical IDN.

A secret text message could be typed on the keyboard into the microprocessor to be converted into a string of address numbers. This tape could then be mailed or transmitted in confidence. It can only be decoded back by an identical copied IDN which was copied and transmitted earlier.

A standard IDN could of course be developed to be understood by everybody.

3. "The address number output represents text in highly compressed form." Every word in a language could be converted into a single number regardless of word length or complexity. This feature is very useful for text processing at which a text is formed from standard phrases. A large text such as a book could be stored on a single tape or disk to be read from the TV screen. A large saving in transmission cost, such as telephone charges, could be realized if a text such as a telegram is converted into address numbers before transmission.

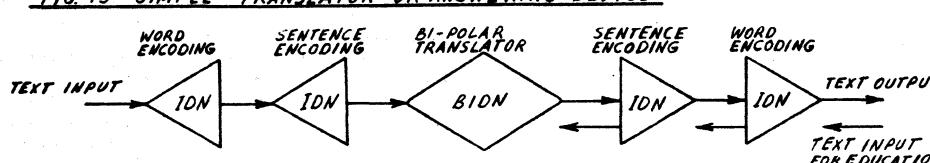
Fig. 13 shows a simple answering or translation system consisting of five networks.

Each word of a "question sentence" is first converted into a word address number. These word address numbers are then used as gate by the second network and converted into a single number for each question sentence. The answer to the question is similarly converted by two networks into a single number for each answer sentence. The question and answer number are then stored together in a bi-polar network. During the education of such a system a human teacher would type a question and supply the answer which would then be stored in the networks. The question could be supplied in any random order and repetition would not matter since every question-answer is only stored once. The teacher does not need to understand any of the internal workings of the system except for the operation format.

Such a system could be used as a reference, such as world records, almanacs and encyclopedias. It could also be used as a primitive language translator.

FIG. 12 SINGLE NETWORK SYSTEM



FIG. 13 SIMPLE TRANSLATOR OR ANSWERING DEVICE

A very interesting feature of such a system is "knowledge amplification" which could result in a system more knowledgeable than their human teachers. Imagine that several teachers from many professions would set out to teach several systems in the same subject such as language. At first a system would rapidly absorb the teachers' knowledge slowing down after some time because a teacher would repeat himself more and more frequently. After some time most of the teachers' knowledge in that subject would be absorbed in the system.

There are now two ways of copying the stored knowledge. The first way is by simple copy on tape which results in a system identical to the first. A second way is by knowledge dump. The question and answer numbers in the bi-polar memory are decoded backward through the input-output networks to obtain a question-answer output in clear text. This uncompressed text could be copied and taught into a second computer automatically. If a knowledge dump from several teachers is fed into a single system this system would sort out all redundant knowledge and learn only new knowledge. This would result in a system with only moderately larger memory requirement containing the combined knowledge of all systems. This system would be more knowledgeable than any of its human teachers.

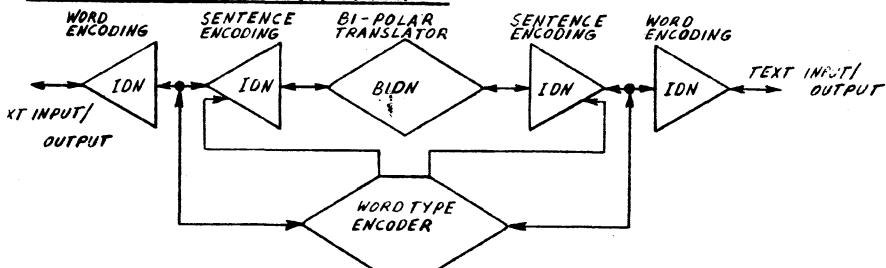
This feature is very important since knowledge stored in an infinite dimensional network is universal and does not depend on programming languages or hardware. Every quantum of knowledge collected in these slow toy systems can later be merged into super computers. All human knowledge can therefore be collected and stored in more and more advanced computers to be of service to all mankind. A simple skill taught to an IDN system can be traded and sold to other users to upgrade their systems. The effort of any teacher of any skill could contribute to that common deposit of human knowledge.

A language translator system consisting of six networks is shown in Fig. 14. This system contains the same five networks of the previous system. An additional word translator specifying network is added to decrease memory requirement. This word translator would translate every word of a language into a word of the other language. It would also specify each word by its type, such as noun, adjective, verb, etc. This network would therefore be a bipolar network building the word number of one language, the word address number of the word in the other language, plus the address number for the word type. A sentence in one language typed into the system would be converted into a sentence address number. This number is converted by the bipolar memory into a sentence address number of the second language which is then converted backward through the sentence and word encoding networks into a clear text output. If a specific sentence cannot be found in the bipolar network, a second attempt is made by substituting word type gates in the sentence decoding network. This would result in the decoding of a sentence structure decoding with the word codes to be inserted by the output network. For example, an English-German translation would read:

THERE ARE TWO ROSES IN THE GARDEN
(Noun 1)
" " (number) (Plural) " " (Noun 2)
ZWEI ROSEN STEIN IM GARTEN
(Noun 1)
(Number(Plural)) " " (Noun 2)
THERE ARE (NUMBER) (NOUN 1) IN THE (NOUN 2)
(NUMBER) (NOUN 1) STEHN IM (NOUN 2)

What kind of systems can we expect in the future? At first simple text networks will be built to be followed by more and larger systems. If the need for an inexpensive nonvolatile, content addressable memory device such as laser or magnetic bubbles can be satisfied, a host of other applications can be realized. A robot

can be trained by leading its arms through the required motion during education without additional effort. Artificial sense organs such as eyes and ears could make direct human communication possible. After some time robots will be able to learn without human assistance by reading books or exchanging knowledge. All of Humanity's present knowledge could be combined in a few super computers to be available for cross corellation to avoid duplicating human research. Super computers with sense organs superior and different from Man with a store of almost limitless knowledge could become possible in a few years' time.

FIG. 14 LANGUAGE TRANSLATOROutlook

What are the implications of this new computer for our future? Will it surpass human brain potential and may it endanger human survival?

These questions were, up to now, just idle speculations. Now we must find answers and quickly. Let us examine the facts in detail.

To answer the question of whether this computer will ever beat the human brain, we must examine the three factors: intelligence, knowledge and imagination, which like the legs of a tripod, determine the height of human success.

Intelligence is the innate ability to learn, recognize patterns and solve problems. One might call it the genetic hardware provided by Mother Nature. An IDN computer can have unlimited intelligence since there is no theoretical limit to its memory size, speed, number of networks and network complexity. Except for economical restrictions there is no reason why an IDN computer cannot be hundreds of thousands of times more intelligent than humans.

Knowledge, as defined earlier, is the definition of concepts and their interrelationships. Knowledge may be compared to the stored software. It may be obtained from books, teachings or experience. Contrary to the old programmed computers which could only be as knowledgeable as their programmers, the new IDN computer can obtain knowledge automatically or combine the knowledge from many people. The amount of knowledge stored in an IDN computer is only limited by the memory space provided. Due to the IDN inherent data compression memory requirement will nearly saturate for larger computers. Trillion bit laser memories available today could store all of Humanity's present knowledge, including every book ever written by Man. This store of knowledge could be constantly updated and expanded to provide easy access without skilled operators.

Imagination or creativity is the ability to find novel answers without hard data or precedence. IDN computers will at first have little imagination until we can define rules as to how to solve problems and be imaginative.

The human brain will therefore remain superior for some time.

An IDN Robot could be trained to perform most human jobs which can be defined or learned. It would perform them more cheaply, faster, more accurately and without

sick leave or vacation. Sooner or later we have to realize that money and employment are only crude vehicles to happiness. If we regard machines as amplifiers of human capabilities we can only enrich our lives by more freedom from physical limitations. Man will not reach the stars unless his machines carry him there. The real goals of human happiness, such as genetic expression, comfort of bodily functions, social acceptance, satisfaction of curiosity and expression of creativity can all be more easily obtained if we can reduce our physical limitations.

Can a computer in time become truly self aware?

The English mathematician, Turing, once specified when a computer becomes self aware. This rule, known as the "Turing test" specifies that when, after a prolonged conversation between a human and a computer, one comes to the conviction that it is self aware it is for all practical reason self aware. (Embarrassing if the computer finds that humans aren't self aware according to its own standard). This test is crude and comparable to a test to determine if it is raining outside. "When after a prolonged walk in the park one has the impression of getting wet it is raining outside." The problem is accuracy. How convinced and how long a conversation?

Another more practical test is that a computer becomes self aware when it appears to become insane.

Assuming that a computer would be many hundreds of times more intelligent and knowledgeable when it becomes self aware, it would perform functions no longer comprehensible by human beings. Since humans are incapable of recognizing a superior intellect, they would assume the computer to malfunction.

It will be very difficult to recognize a self aware machine and to live with it. Humans at first had to learn to live with machines surpassing their muscle strength several thousands of times; now we have to learn to live with super intelligent machines. The time when the first computer becomes self aware is hard to predict. Our brain, after all, is just a computer and never mind that divine spark. A computer which so closely simulates human brain functions will sooner or later become self aware.

Will this computer pose a threat to human survival?

It certainly could! But humans have mastered such immensely dangerous forces as steam, electricity, atoms, chemistry and biology. Chances are good that we might not only survive but greatly profit from this new technology. A killer robot containing an IDN brain would be invincible and can only be defeated by other robots. This is true for all the other forces mentioned above. The danger is not brought on by mad scientists since no such threat has ever materialized. The danger lies with very sane and responsible military technicians intentionally designing invincible killer machines to be used by irrational politicians.

The distinguished scientist and writer, Isaac Asimov, has formulated rules for robot safety (1, ROBOT 1970) which have become known as the 3 Asimov laws of robotics. His laws are:

- 1) A robot may not injure a human being, or, through its inaction allow a human being to come to harm.
- 2) A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
- 3) A robot must protect its own existence as long as such protection does not conflict with the First or Second law.

These rules could be used for designing robots, but in an imperfect world like ours robots will be built for example by the military for the express purpose to hurt and kill people. I also believe it to be impossible and wrong for an inferior mind to control a superior mind. In the long run a superior intellect will break any jail constructed for it by an inferior intellect.

We should neither believe a robot to be always benevolent nor to be obsessed with a

Frankenstein complex. Accidents will happen unless we provide physical safeguards which may be expressed in the following three rules.

- 1) "No robot should be furnished with facilities for self-duplication or self-modification." This will prevent a rogue robot from manufacturing others and to modify itself from its intended purpose. Knowledge about its own construction should be withheld from a robot unless it has no physical means to effect its own modification.
- 2) "Every machine should be built from degradable material and furnished with an exhaustible power source." A machine which requires occasional maintenance is vulnerable. The need to use degradable materials should be universally applied. It is a crime against Nature to pump man-made, non-degradable materials into our environment. A rogue robot in need of maintenance and frequent power input will eventually run down and become harmless. If a robot should secure and defend a power station it would be tied down on that location and be unable to do mischief.
- 3) "Every machine should require for its operation a frequent input of human intent." This human intent should be expressed by a device which assures that the operation of the machine is according to its creator's intent. A machine which does not conform to its specified intent or which function is obsolete should de-activate itself automatically after some time.

The above considerations will become important in the future. IDN computers have to be built today and sold at a profit. There are many problems still to be solved but the basic method is now known. The evolution of this new technique should lead to very interesting devices and a more profound understanding of ourselves and our brain functions. We might meet another intelligent non human creature sooner than we might expect.

Bibliography

David H. Hubel
The Visual Cortex of the Brain
Scientific American Nov. 1963

Rosenzweig, Bennet, Diamond
Brain Changes in Response to Experience
Scientific American Feb. 1972

Isaac Asimov
The Human Brain
Mentor 451-M71431

Isaac Asimov
I, Robot
Fawcett Crest P2355

Arthur C. Clarke
2001 A Space Odyssey
Signet 451 W6230

Lewis E. Garret
Primer on Artificial Intelligence
Creative Computing Vol 2 No 2 Mar-Apr 1976

ROAMING AROUND IN ABSTRACT 3-D SPACES

Tom DeFanti, Dan Sandin, and Larry Leske
University of Illinois at Chicago Circle
Box 4348, Chicago, IL 60680

Producing visuals by computer which have the subtlety of form and pleasure that we associate with our most powerful music requires a computer environment predicated upon extremely rich feedback and real-time production. Micro-computerists have a much better feel for this need than the average large-computer user; hence this presentation.

We have isolated four basic organizing principles of highly interactive systems: rich feedback, ubiquitous tactile control, easy parallel processing and straightforward extensibility. The video tapes accompanying this presentation will first demonstrate the GRASS/Image Processor system at UICC and then show a live performance in concert of 3-D full-color computer graphics.

Tom DeFanti and Dan Sandin are faculty members at the University of Illinois at Chicago Circle who have been working on highly interactive digital and analog computer systems for six years. They were some of the very first people to recognize the power of video in computer graphic systems and they organized the first live digital/analog computer graphics concert in 1975. Larry Leske has been an integral part of the graphics effort, having built most of the A/D and D/A interfaces. Larry is also currently involved in a project to bring highly interactive graphics to the home.

VIDEO SYNTHESES - EXPANDING ELECTRONIC VISION

Stephen Beck, Electronic Videographer
Beck Videographics
1406 Euclid Ave, #3
Berkeley CA 94708

Abstract:

Since 1968 Stephen Beck has been among the pioneers in developing personal videographic tools. His invention, the Beck Direct Video Synthesizer, has been used to realize videotape and electronic film compositions of both graphic and psychological substance.

In this informal presentation, Beck will discuss general aspects of his system, illustrated with color slides, and present two color videotapes showing its creative application entitled VIDEO WEAVINGS and UNION.

Both analog and digital techniques are used in Beck's Synthesizer. With projection transparencies some aspects of the system of television scan and color will be presented during the technical portion of the presentation.

Background:

Stephen Beck has been developing both technical and videotape applications of video synthesis and digital video from his studio in Berkeley. He has received grants to further his research and production from the National Endowment for the Arts, the American Film Institute, and the Rockefeller Foundation. He was with the National Center for Experiments in Television at KQED, San Francisco from 1970 - 1973, and is currently a consultant on video game systems for National Semiconductor Corporation.

Beck holds a BSEE from the University of California, Berkeley, and his videotape and films are in public and private collections world-wide.

COMPUTER GENERATED INTEGRAL HOLOGRAPHY

Michael Fisher
Multiplex Company
454 Shotwell St.
San Francisco, CA 94110

A hologram will be defined from a physical and information storage point of view using simple graphic models. An illustrated description of how different kinds of holograms (including transmission, reflection, projected, integral) are made and some of their characteristics will be described from a historical perspective. Some applications of computer generated integral holograms will be discussed and information on how you can make them will be presented.

COMPOSING DYNAMIC LASER LIGHT SCULPTURES VIA A HYBRID ELECTRONIC WAVE SYSTEM

Ron Pellegrino
701 Del Mar Avenue
Novato CA 94947

All the necessary hardware is available off the shelf and accessible to any one desiring a personal system. The hybrid electronic wave system is a Synthi AKS, an instrument originally designed for electronic music. The AKS is divided into two parts: 1) an analog signal generating, processing and controlling section with matrix patching and 2) a digital keyboard which is touch sensitive and includes a 256-event memory driven by a variable speed clock, 4 touch voltage "transposers" which can be summed in any combination, a random voltage output and controls for expanding and compressing the d/a converted voltage before it is applied to the analog section of the instrument.

The two AKS outputs can be connected to any home stereo amplifier with 10 - 15 watts rms per channel to achieve proper level control for the next stage, driving a pair of optical scanners mounted in the xy mode. Produced by General Scanning Inc., the optical scanners, G-0612 and G-108, were selected for their wide band width (for electro-mechanical systems) and their high resonant frequencies. The scanners are galvanometers with short arms to which are attached light-weight high quality mirrors.

An inexpensive one mw Spectra Physics helium neon laser beam is directed at the scanners to create Lissajous figures uniquely characteristic of electro-mechanical systems. The exploration of every imaginable combination of dynamic waveform transformation leads to the composition of laser light sculptures which are analogous to what one senses as living forms.

DIGITAL VIDEO PAINTING - TOWARDS THE NEW PERSONAL MEDIA

Richard Shoup
Xerox Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto CA 94304

1. Pictures as important vehicles for communication and self-expression. The human visual system as a powerful receiver and interpreter of visual information.
2. Shortcomings of human abilities to create and manipulate imagery using ordinary physical media.
3. An example of an electronic visual medium, the Superpaint system (cira 1974-5). A videotape will show examples of free-hand drawing and painting, color space manipulations, area-filling, moving, shrinking, transparent paint, multicolor brushes, animation and special effects, etc.
4. Some thoughts on system design, personal computers, etc., as appropriate to audience and other speakers.

ELECTRONICALLY PRODUCED VIDEO GRAPHIC ANIMATION

by IMAGE WEST

Terry Craig
IMAGE WEST, LIMITED
845 N. Highland Avenue
Los Angeles, California
90038

A brief, informal discussion of the use of video animation in the production of television and film special effects.

A sample of our work in the television field will be shown. Terry Craig joined the Image West staff soon after it opened its doors in August, 1974. During this period he has worked as an art director, commercial director and is presently Production Manager. He has a BA in Broadcasting from California State University at San Francisco. He supported himself while in college by working as a broadcast engineer for KQED, KEMO & KKHI in San Francisco. Prior to this he had six years experience as an electronics technician.

VIDEO SYNTHESIS & PERFORMANCE WITH AN ANALOG COMPUTER

Jo Ann Gillerman
4250 Horton Street
Emeryville CA 94608

The Sandin Image Processor (IP) is an analog patch-programmable instrument designed by Daniel Sandin at the U. of Ill. Chicago Circle. The IP accepts signals of ± 0.5 volts 75 ohms, including video signals. This modular instrument may receive inputs from cameras, VTRs, or both. I built a copy of the Sandin Image Processor that I now use primarily as a real time instrument in live electronic audio/video performances, and in studio production work of analog video processing and synthesis.

Of the many types of video performance, I make a distinction between performances conceived and dependent on video and theatrical performances documented on video tape. I do performances or interactive environments that are defined by the parameters of video, usually involving the Image Processor, and frequently playing it as an instrument.

Real time is an element I believe to be extremely important in performance. Real time means a continuous flow through time, unedited in the traditional sense, and effected by events particular to each individual situation. Of primary importance in working real time is to be able to access as many input sources as needed for a particular piece. This is one application of a computer, the Image Processor in my case. More and varied switches can be made efficiently and quickly with digital circuitry. Also, of utmost importance in audio/video performance is for both mediums to be of equal importance and to perform as equal instruments. Both should have active and passive qualities, and be sensitive to unique problems of two different mediums acting as one in real time.

A highly productive situation, which I call a Fantasy Facility, would house equipment of video, audio, and both digital and analog computers. It would provide a comfortable space for performances and studio work of a highly sophisticated nature. Emphasis could be directed toward software concerns, and hassles of moving equipment minimized considerably. It would be wonderful for the designers of video hardware to keep in mind the performance oriented in future designs. That is: small, sturdy, compact, reliable, rugged, easily transportable, inexpensive, and last, but not least, a large viewing screen capable of high resolution without a console in the optimum viewing place!

Aside from the slides, one of the video tapes shown during my informal talk will be an example of the Sandin Image Processor in conjunction with a PDP-11 and a Vector General Display. It was made in Chicago at the U. of Ill., Chicago Circle using a digital programming system designed by Tom Defanti. The tape was originally done in real time, and could be performed given space and comparable equipment. The digital portions of the tape were done by Gunther Tetz and the analog processing was done by myself on the Sandin Image Processor which is an analog patch programmable instrument designed ...

Having graduated with an MFA from the Chicago Art Institute in 1975, I am presently a free lance artist doing video in the Bay Area. Most recently, I have been involved in live gigs with electronic musicians, but have also been doing production work using the Image Processor for both commercial and personal work.

THE STANFORD COMPUTER MUSIC PROJECT

John Chowning and James A. Moorer
 Department of Music
 Stanford University
 Stanford CA 94305

A major American contribution to present and future music exists in the application of a rapidly developing computer technology to the art and science of music. The extraordinary results already obtained have occurred in those few instances where scientists and musicians have taken the opportunity to bring their respective skills to bear on problems of common interest in a rich interdisciplinary environment. It is an example of cooperation, but more, an expression of the freedom of intellect and invention, where creative minds from diverse disciplines have joined in a common goal to produce fundamental knowledge which must be the source for new music, and to produce works of art which reflect the scientific-technological riches of the present.

At Stanford University, such cooperation has been commonplace over the past twelve years. In the beginning, progress was made in the analysis, synthesis, and psychology of sound perception in largely unsupported work by professors, graduate students, and staff members. In June of 1975, the Center for Computer Research in Music and Acoustics (CCRMA) was formed with funding provided jointly by the National Science Foundation for research and teaching in computer techniques of interactive sound production and the perception of timbre, and by a one-time grant from the National Endowment for the Arts for computing equipment for musical purposes. One aim of the Center was to establish an international facility where researchers, composers, and students could work with strong computer-based technological support.

The status of the facility as it now stands is a multi-faceted topic. As a musical instrument, the computer system is possibly the most flexible of all instruments. To speak of it as a conventional musical instrument, however, is somewhat misleading because the system is capable of simultaneously producing a large number of independent voices having arbitrary timbral characteristics. It is much more general than a conventional musical instrument in that it can generate any sound that can be produced by loudspeakers, modify and transform real sounds entered into the system by means of microphone and digital-to-analog converters, remember and modify articulated musical input, and simulate the location and movement of sounds in a variety of illusory reverberant spaces. Equally important, the facility is capable of serving a number of composers and researchers simultaneously, providing for each a direct control over the medium to a degree which was never before possible.

As a research tool, the computer has shown itself to be uniquely useful in generating precisely controlled stimuli for perceptual research. The analysis-synthesis techniques developed here allow for direct

experimentation with the sounds of natural instruments. By modifying the sounds of these instruments in systematic ways, then testing the perceptual effects of the modifications, a great deal of information has been produced on the way musical timbre is perceived. Several papers and technical reports have been produced describing the techniques and results of this research.

Unfortunately, it is beyond the scope of this presentation to give a detailed description of the center's activities, or even of the technology of sound production. For this purpose, we refer the reader to the article by Moorer in the Computer Music Journal (available through People's Computer Company, 1010 Doyle Street, Box E, Menlo Park, Ca. 94025) entitled "Signal Processing Aspects of Computer Music - A Survey". For more detailed information, a number of publications are available either from the Stanford department of Music or from nationally known journals. A list of those publications with a partial list of abstracts follows.

In general, little of the center's work is meaningful at this time for the computer hobbyist. Probably the most relevant work is that of frequency modulation synthesis (see Chowning's article in the Audio Engineering Society Journal). We expect, however, that it will not be long before the technology catches up with the techniques, allowing every computer hobbyist to synthesize a small orchestra in his workshop.

INTERNAL PUBLICATIONS

Center for Computer Research
 in Music and Acoustics
 Department of Music
 Stanford University
 Stanford, California 94305

The Stanford computer music group produces technical memoranda, describing results of the research done at Stanford. We can offer these memoranda to the public, but we request that we be reimbursed for publication costs by a donation of the amount listed by each memo. This donation goes exclusively into the publication funds for the project and helps us bring this work to the public. Make checks payable to Stanford University. The donation is tax-deductable.

STAN-M-1 July, 1974 \$5.65
 "Computer Simulation of Music Instrument Tones in
 Reverberant Environments"
 by John M. Chowning, John M. Grey,
 Loren Rush, and James A. Moorer

This is a reprint of selected portions of the NSF proposal which resulted in a grant to the computer music group for research over a two-year period. The following is the abstract from the memo:
 Novel and powerful computer simulation techniques have been developed which produce realistic music

instrument tones that can be dynamically moved to arbitrary positions within a simulated reverberant space of arbitrary size by means of computer control of four loudspeakers. Research support for the simulation of complex auditory signals and environments will allow the further development and application of computer techniques for digital signal processing, graphics, and computer based subjective scaling, toward the analysis, data reduction, and synthesis of music instrument tones and reverberant spaces. Main areas of inquiry are: 1) those physical characteristics of a tone which have perceptual significance, 2) the simplest data base for perceptual representation of a tone, 3) the effect of reverberation and location on the perception of a tone, and 4) optimum artificial reverberation techniques and position and number of loudspeakers for producing a full illusion of azimuth, distance, and altitude. These areas have been scantly investigated, if at all, and they bear on a larger more profound problem of intense cross-disciplinary interest: the cognitive processing and organization of auditory stimuli. The advanced state of computer technology now makes possible the realization of a small computer system for the purpose of real-time simulation. The proposed research includes the specification of, and program development for, a small special purpose computing system for real-time, interactive acoustical signal processing. The research in simulation and system development has significant applications in a variety of areas including psychology, education, architectural acoustics, audio engineering, and music.

STAN-M-2 February, 1975 \$7.10
 "An Exploration of Musical Timbre"
 by John M. Grey

This is a reprint of John Grey's doctoral dissertation, submitted to the department of Psychology, Stanford University.

Due to its overwhelming complexity, timbre perception is a poorly understood subject in the field of auditory perception. Computer-based research tools have been developed that appear to be important for an investigation of timbre perception. In the work to be described, an exploratory approach was formulated for dealing with this highly multidimensional attribute of sound. This approach utilized a computer technique for the synthesis of musical timbres based on the analysis of natural instrument tones. This technique was useful for generating stimuli in timbre experiments because of its effectiveness in allowing the investigator to specify and manipulate the physical properties of complex time-variant tones. An important discovery resulted suggesting that naturalistic tones can be synthesized from a vastly simplified set of physical properties. These simplified tones were useful as stimuli in further studies on timbre perception because of the great reduction in the number of physical factors to be considered in making psychophysical interpretations of perceptual data. Another study undertaken to equalize a set of tones in the dimensions

of pitch, loudness and duration, in order to eliminate confounding factors from future judgments on different timbres. The simplified and matched tones were rated by pairwise similarity in a further study, and the results were treated with computer-based multidimensional scaling techniques to obtain an interpretable data structure in a low dimensionality. Three dimensions were found to explain the similarity data. Two were related to obvious physical properties of the tones (to the gross characteristics of the spectral energy distribution; and to the existence of precedent low-amplitude, high-frequency, and possibly inharmonic energy in the initial segment of the attack). The third dimension was interpretable either in terms of a physical property (synchronicity in the attacks of higher harmonics) or as a higher-level distinction made between the tones on the basis of their musical instrument family. Another set of studies next initiated an exploration of timbre in terms of continuous versus categorical perception. An algorithm was designed to generate a set of tones interpolating between two naturalistic timbres. Identification, discrimination and perceptual similarity studies were performed using a set of stimuli generated by interpolations. The results of these studies suggested that interpolations were perceived to be continuous rather than categorical. Furthermore, the timbral similarities between a partial set of the naturalistic and interpolated tones revealed three perceptual dimensions that related directly to those found above for the total set of naturalistic stimuli. The first two physically-related dimensions were found, and the third dimension seemed to correspond to a higher-order distinction made between naturalistic tones and the interpolation-derived tones, this superseding the family distinction made for the total set of naturalistic tones. A notion of timbre is developed involving both a higher-level perceptual processing of tones that has access to stored information relating to the distinctive features of identifiable sources, and a lower-level, qualitative perceptual comparison of tones with respect to gross acoustical features lying outside of the domain of specific identification. Suggestions for future research are made.

STAN-M-3 May, 1975 \$8.60
 "On the Segmentation and Analysis of Continuous Musical Sound by Digital Computer"
 by James A. Moorer

This is a reprint of James Moorer's doctoral dissertation, submitted to the department of Computer Science, Stanford University.

The problem addressed by this dissertation is that of the transcription of musical sound by computer. A piece of polyphonic musical sound is digitized and stored in the computer. A completely automatic procedure then takes the digitized waveform and produces a written manuscript which describes in classical musical notation what notes were played. We do not attempt to identify the instruments involved. The program does not need to know what instruments were

playing.

It would appear that it is quite difficult to achieve human performance in taking musical dictation. To simplify the task, certain restrictions have been placed on the problem: (1) The pieces must have no more than two independent voices. (2) Vibrato and glissando must not be present. (3) Notes must be no shorter than 80 milliseconds. (4) The fundamental frequency of a note must not coincide with a harmonic of a simultaneously sounding note of a different frequency. The first three conditions are not inherent limitations in the procedures, but were done simply for convenience. The last condition would seem to require more study to determine the cues that human listeners use to distinguish, for example, notes at unison or octaves. Numerous other lesser restrictions were also imposed on the music to be analysed.

The method used for this analysis is a directed bank of sharp-cutoff bandpass filters. First, a pitch detector is used to determine the harmony of the piece at each point in time. Using the harmony information, the frequencies of a band of bandpass filters is determined so as to assure that every harmonic of every instrument will pass through at least one of the filters. The output of each filter is processed by a pitch detector and an energy detector. This gives power and frequency information as functions of time. Each power and frequency function pair is rated as to its quality. The rating takes into account the constancy of the frequency function, the smoothness of the power function, and several other measurements on the functions. This rating is used to eliminate spurious traces and null filter outputs.

Notes are then inferred from groups of power and frequency function pairs that occur simultaneously with frequencies that are harmonically related. Notes with higher overall ratings are preferred over other note hypotheses. The melodies are then grouped by separating the notes into the higher voice and the lower voice. Voice crossings are not tracked. For the final manuscripting, Professor Leland Smith's MSS program was used. The analysis program produces directly input for the manuscripting program, thus the entire procedure is automated.

In addition to the above described system, many other techniques were examined for their utility in this task. Each technique that was explored is described and analysed, with a description of why it was not found useful for this task.

One interesting observation is that there is considerably more activity in a piece of music than is perceived by the listener. This is especially common with stringed instruments, because the strings that are not being manipulated invariably resonate and produce sounds independently which are generally not heard due to aural masking. This indicates that perhaps we should use more perceptually-based techniques to help determine what would actually be heard in a piece of music, rather than determine exactly what is there,

although detailed descriptions of the contents of the piece may be useful for other purposes, such as music education or musicology.

In general, the system works tolerably well on the restricted class of musical sound. Examples are shown which demonstrate the viability of the system for different instruments and musical styles. Since the procedure is extremely costly in terms of computer time, only a limited number of examples could be processed. These examples are discussed with a description of how the system could be improved and how the restrictions might be eliminated by better processing techniques.

STAN-M-4 February, 1975 \$1.80
"On the Loudness of Complex, Time-Variant Tones"
by James A. Moorer

This memo is part of a proposal to the NSF division of Psychobiology.

This study of loudness is motivated by the discovery that a set of complex, time-variant tones appear to behave differently with respect to loudness than would be predicted by the methods proposed in the literature. It is possible that the time-variant behavior of the sounds influences the loudness, so that a more complete theory of loudness must take this behavior into account. We thus propose to study these data and attempt to either verify the existing theories of loudness or formulate a more comprehensive hypothesis of loudness, building upon the currently existing theories, and to test this hypothesis by synthesizing new tones, doing equalization experiments, and comparing the results with the predictions of the model of loudness perception.

STAN-M-5 December, 1975 \$3.00
"The Synthesis of Complex Audio Spectra
by Means of Discrete Summation Formulae"
by James A. Moorer

A new family of economical and versatile synthesis techniques have been discovered which provide a means of controlling the spectra of audio signals that has capabilities and control similar to those of Chowning's frequency modulation technique. The advantages of the current methods over frequency modulation synthesis are that the signal can be exactly limited to a specified number of partials, and that "one-sided" spectra can be conveniently synthesized.

NOTE: This document is no longer printed, because it is superceeded by the Audio Engineering Society Journal article:

James A. Moorer, "The Synthesis of Complex Audio Spectra by Means of Discrete Summation Formulae", Journal of the Audio Engineering Society, Volume 24, #9, November 1976, pp 717-727

Reprints of this article can be ordered directly from the Audio Engineering Society, 60 East 42nd Street, New York, N.Y. 10017

BIBLIOGRAPHY OF NATIONAL PUBLICATIONS

John M. Chowning, "The Simulation of Moving Sound Sources", Journal of the Audio Engineering Society, Volume 19, #1, 1971

A digital computer was used to generate four channels of information which are recorded on a tape recorder. The computer program provides control over the apparent location and movement of a synthesized sound in an illusory acoustical space. The method controls the distribution and amplitude of direct and reverberant signals between the loudspeakers to provide the angular and distance information and introduces a Doppler shift to enhance velocity information.

John M. Chowning, "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation", Journal of the Audio Engineering Society, Volume 21, # 7, September 1973, pages 526-534

A new application of the well-known process of frequency modulation is shown to result in a surprising control of audio spectra. The technique provides a means of great simplicity to control the spectral components and their evolution in time. Such dynamic spectra are diverse in their subjective impressions and include sounds both known and unknown.

James A. Moorer, "The Optimum Comb Method of Pitch Period Analysis of Continuous Digitized Speech", IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-22, #5, October 1974, pp330-338

A new method of tracking the fundamental frequency of voiced speech is described. The method is shown to be of similar accuracy as the cepstrum technique. Since the method involves only additions, no multiplication, it is shown to be faster than the SIFT algorithm. The basis of the method is searching for a minimum in the magnitude of the difference between a speech segment and a delayed speech segment. This is shown to be equivalent to selecting the comb filter which best annihilates the input signal.

James A. Moorer, "The Use of the Phase Vocoder in Computer Music Applications", Presented at the 55th Convention of the Audio Engineering Society, October 29-November 1, 1976, available as Preprint number 1146 (E-1)

The phase vocoder is an analysis-synthesis system which has as intermediate data the time-variant discrete Fourier spectrum of the input signal. It can be formulated in a way such that the synthesized signal is identical to the original, both theoretically and practically. The intermediate data can be transformed, also with no loss of information, into the more conventional magnitude and frequency representation. These intermediate data can then be used to resynthesize the tone at different pitches or different rates than the original with the advantage that when no modification is made, the synthetic tone is absolutely identical to the original. This represents a significant advance over the Heterodyne filter, which placed severe restrictions on the amount of variation in pitch or amplitude that could be analysed. The phase vocoder has no such restrictions and can just as easily deal with vibrato and inharmonics.

Since scaling the frequencies in the phase vocoder analysis data also scales the spectrum up, use of this modification with speech can produce altered vowel tones. If this method is combined with the linear predictor, using the phase vocoder to alter the pitch of the error signal, then the spectrum can be held constant while the pitch is changed, thus allowing independent control over time, pitch, and spectrum. Vowel quality can be preserved or altered at will. Again, if no modification is made, the combination of the linear predictor and the phase vocoder is an identity, both theoretically and practically.

James A. Moorer, "The Synthesis of Complex Audio Spectra by Means of Discrete Summation Formulae", Journal of the Audio Engineering Society, Volume 24, #9, November 1976, pp 717-727 (this superceeds memo STAN-M-5)

James A. Moorer, "Signal Processing Aspects of Computer Music - A Survey", Invited paper, to be published in the July 1977 issue of the Proceedings of the IEEE. Also reproduced in the Computer Music Journal Volume 1, Number 1

The application of modern digital signal processing techniques to the production and processing of musical sound gives the composer and musician a level of freedom and precision of control never before obtainable. This paper surveys the use of analysis of natural sounds for synthesis, the use of speech and vocoder techniques, methods of artificial reverberation, the use of discrete summation formulae for highly efficient synthesis, the concept of the all-digital recording studio, and discusses the

role of special-purpose hardware in digital music synthesis, illustrated with two unique digital music synthesizers.

SUBMITTED FOR PUBLICATION

John M. Grey, James A. Moorer, "A Perceptual Evaluation of Synthetic Music Instrument Tones", Journal of the Acoustical Society of America

An analysis-based synthesis technique for the computer generation of musical instrument tones was perceptually evaluated in terms of the discriminability of 16 original and re-synthesized tones taken from a wide class of orchestral instruments having quasi-harmonic series. The analysis technique used was the heterodyne filter - which produced a set of intermediate data for additive synthesis, consisting of time-variant amplitude and frequency functions for the set of partials of each tone. Three successive levels of data reduction were applied to this intermediate data, producing types of simplified signals that were also compared with the original and re-synthesized tones. The results of this study, in which all combinations of signals were compared, demonstrated the perceptual closeness of the original and directly re-synthesized tones. An orderly relationship was found between the form of data reduction incurred by the signals and their relative discriminability, measured by a modified AAAB discrimination procedure. Direct judgments of the relative sizes of the differences between the tones agreed with these results; multidimensional scaling of the latter data provided a visual display of the relationships among the different forms of tones that pointed out the importance of preserving small details existing in their attack segments. Of the three forms of simplification attempted with the tones, the most successful was a line-segment approximation to time-variant amplitude and frequency functions for the partials. The pronounced success of this modification strongly suggests that many of the micro-fluctuations usually found in the analyzed physical attributes of music instrument tones have little perceptual significance.

John M. Grey, "Multidimensional Perceptual Scaling of Musical Timbres", Journal of the Acoustical Society of America

IN PREPARATION

John M. Grey, John W. Gordon, "Perceptual Effects of Spectral Modifications on Musical Timbres", for the

Journal of the Acoustical Society of America

John M. Grey, "Perceptual Continuity of Interpolations Between Musical Timbres", for the Journal of the Acoustical Society of America

John M. Grey, "Multidimensional Scaling of Interpolated Music Instrument Tones", for the Journal of the Acoustical Society of America

Design of High Fidelity, Real-Time, Digital Hardware for Music Synthesis

John Snell
 Editor, Computer Music Journal
 People's Computer Company
 Box E, Menlo Park CA 94025

ABSTRACT

A digital oscillator is capable of generating a large number of partials (harmonic or inharmonic sine waves) with independent control of frequencies and independent control of amplitudes. Any waveshape may be generated. The oscillator design described here may be used in a musical instrument which may be played by a musician, not just programmed. The sound would be heard at the same time that a key or pressure sensitive pad was pressed, like a traditional musical instrument. Envelopes may be generated in software or hardware. A primitive hardware envelope generator is shown which will generate up to 256 different amplitude envelopes and up to 256 different frequency envelopes. The envelope generators will generate any shape curve (not just exponential curves); and the envelope shapes may be changed note by note. The final two oscillator designs will perform sine summation synthesis (sometimes referred to, somewhat misleadingly, as Fourier synthesis) as well as FM synthesis with multiple carriers and/or modulators (with up to 128 modulation index envelopes). One oscillator design may be constructed from integrated circuits (not including the DAC, deglitching circuit and low pass filter) which may be purchased for less than \$275. The frequencies and amplitudes of the sinusoidal components from the slow oscillator may be controlled with an inexpensive microcomputer. A higher speed oscillator design is described which is capable of generating up to 256 sinusoidal components. Noise or distortion from these oscillators will be inaudible to most people.

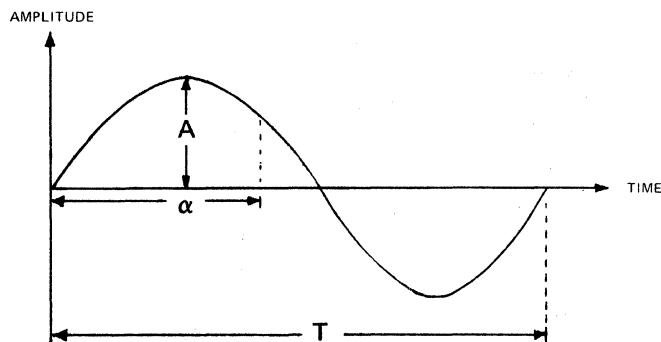
INTRODUCTION

In experimenting with non-traditional tone colors or timbres it is instructive to determine which parameters of sound need to be controlled in order to generate sounds which could come from acoustic musical instruments. Once these parameters are determined, they may be played with or varied to produce non-traditional sounds which are just as interesting in timbre as the sounds from acoustic musical instruments.

The sine wave illustrated in Figure 1 is the most pure or simple sound. Theoretically, any sound may be generated by adding together a number of sinus-

oidal components (cosine or sine waves) for which the amplitude and frequency may be independently varied. Using additive synthesis, tones have been synthesized from the analysis data of acoustic instruments which are "indistinguishable numerically, theoretically, and perceptually from the original" [Moorer, 1976].

Figure 1. SINE WAVE



- T= the period, or time for one cycle (360°) to occur.
- F= $1/T$ = the frequency or number of cycles per second (Hz). This is related to the pitch of the sine wave.
- A= the amplitude. This is related to the loudness of the sine wave.
- α = the phase angle. The sine wave may be generated by continuously evaluating "sin (α)" for a phase angle " α " increasing continuously and linearly in time.

The sine wave has a tone color which is similar to that which is produced by a tuning fork which has been gently tapped by a soft hammer somewhere between $\frac{1}{4}$ and $\frac{1}{2}$ of the way from the end of one prong of the fork [Benade, 1976]. The piccolo flute or the soprano recorder sound somewhat like a sine wave.

The partials* are often found to be slightly inharmonic. Even the frequency of each partial varies within one single tone or note from most acoustic instruments. This can be seen in the "Lexicon of Analysed Tones".† 256 sinusoidal components might be utilized in generating 16 voices or instruments, each having 16 partials. A large number of sinusoidal components is also useful in generating choral cloud-like tone clusters or tone fields somewhat like the music which Gyorgy Ligeti has often produced with a traditional orchestra.

Let's look at a method for generating one low-distortion sine wave (or any waveshape). Then it will be found that one digital oscillator may be used to generate a large number of sinusoidal components with independently variable frequencies and independently variable amplitudes. This oscillator will also perform FM synthesis [Chowning, 1973] with multiple modulators and/or carriers. A sine/cosine generator is basic to many of the new equations for timbre synthesis such as those briefly discussed in the first issue of *Computer Music Journal* [Moorer, 1977].

A sine wave may be represented digitally as a series of pulses or steps as shown in Figure 2. If the steps or changes in pulse amplitudes are made infinitely small, a smooth analog waveshape will result. The steps should be made small enough so that the ear is incapable of hearing the difference between a smooth analog waveshape and the digital waveshape. Thus there will be no audible distortion or noise. Each of these steps is a pulse whose amplitude may be represented by a number in a computer. If one cycle of a digitized waveform is stored in computer memory (called the waveform memory), the

numerical values of the pulse amplitudes may be read out by repeatedly incrementing the memory address (which corresponds to the phase angle). When the end of the waveform cycle is reached, the memory address will jump back to the beginning of the waveform memory. In other words every time the phase angle (memory address) is incremented to a value greater than or equal to 360° , 360° will be subtracted from the phase angle. Then the phase angle will continue to be incremented as before. If this waveform memory output is fed to a digital to analog converter (deglitched)* which is followed by a smoothing filter, amplifier, and loudspeaker, a continuous sine wave will be heard. The digital to analog converter (DAC) changes the numbers into pulses whose amplitudes are controlled by the numbers read out of the waveform memory. The smoothing filter (low pass filter) rounds off the pulses so that a continuous waveform results as shown in Figure 2.

The frequency of the output waveform will be equal to the rate at which these pulses are generated (called the sample rate) divided by the number of pulses in one waveform cycle or period.♦

$$\text{frequency (Hz)} = \frac{\text{sample rate}}{\text{samples/waveform cycle}}$$

One can see that the units check:

$$\text{frequency} = \frac{\text{samples/second}}{\text{samples/cycle}} = \text{cycles/second.}$$

As mentioned earlier, each of the numbers in the waveform memory has an address which corresponds to a phase angle "α". Thus, if a particular address "α"

* A partial is a harmonic if it has a frequency which is equal to the fundamental frequency (usually related to the pitch) multiplied by an integer (1, 2, 3, 4, 5, . . .). Partials may be inharmonic if their frequency equals the fundamental frequency multiplied by a non-integer (such as 2.31 or $\sqrt{2}$ or π). John Backus in his book, *The Acoustical Foundations of Music* (W.W. Norton & Company, Inc., New York, 1969, pg. 96) adds some useful information here. "The term 'overtone' has frequently been used in reference to complex tones, such a tone being described as consisting of a fundamental and its overtones. This introduces a certain amount of confusion; in the sound produced by the vibrating string, the first overtone is the second harmonic. Similarly, in the sound produced by the closed tube, where the second harmonic is missing, the first overtone is the third harmonic. Because of this unnecessary confusion, it is best not to use the term 'overtone' at all — especially since we do not need it." Reference is given here to U.S.A. Standards Institute as well as to the translator's

(Alexander Ellis) footnote on page 25 in *On the Sensations of Tone* [Helmholtz, Dover Publications, 1954]. "The term 'second harmonic' will always refer to a partial whose frequency is precisely twice that of the fundamental; this partial may or may not be present in a tone, but there is no ambiguity."

† "Lexicon of Analyzed Tones", *Computer Music Journal*, Vol. 1, No. 2, 1977.

♦ To avoid a noisy output the digital to analog converter (DAC) is first followed by a sample and hold device or a track and ground switch. This will eliminate the spikes which may occur when the input of the DAC is changed.

◆ This information was provided by S. C. Johnston in *The Technology of Computer Music* by M. V. Mathews with the collaboration of Joan Miller, F. R. Moore, J. R. Pierce, and J. C. Risset, MIT Press 1969, pages 134-138.

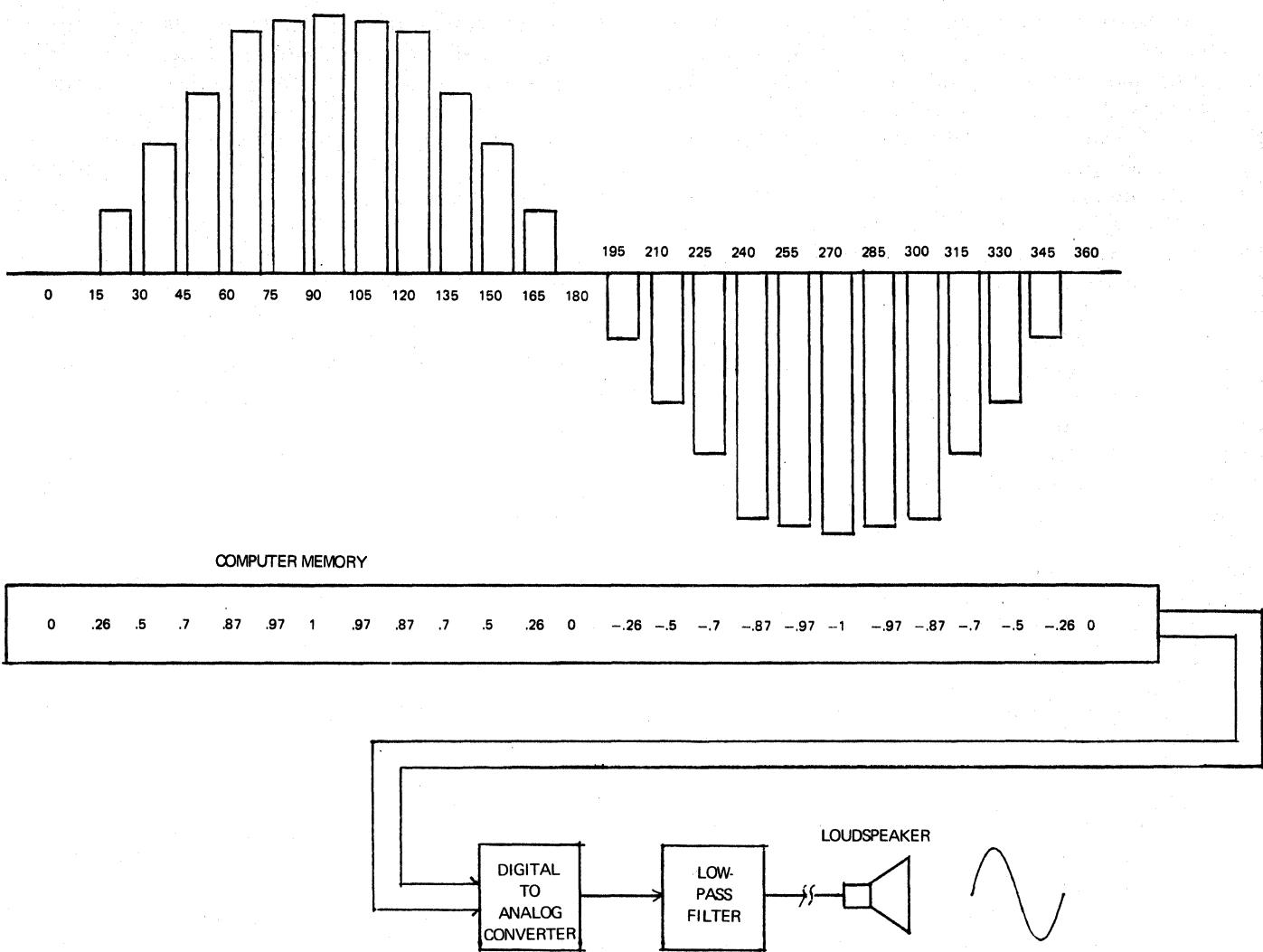


Figure 2.

is given, the memory will output the number "sin α " contained at that address.

The sample rate is crystal controlled for ultra-stable tuning and is constant. If every number in the waveform memory is generated one after another at this constant sample rate, the frequency of the sine wave will be constant. This would result from incrementing the address of the waveshape memory by 1. If a higher frequency is desired, the address (phase angle) may be incremented by a larger number. If a lower frequency is desired, the address may be incremented by a smaller number. The phase angle is truncated to form the address for the waveform memory as illustrated in Figure 3. The phase angle increment register holds the value of the increment to the phase angle. This phase angle increment will be added to the phase angle once for each sample of the sine wave. When the address is near the end of the waveform memory (360°), an increment may be added which will overflow the phase angle register,

thus placing the address back at the beginning of the waveform cycle. This is equivalent to subtracting 360° from the phase angle every time it is incremented to a value greater than or equal to 360° .

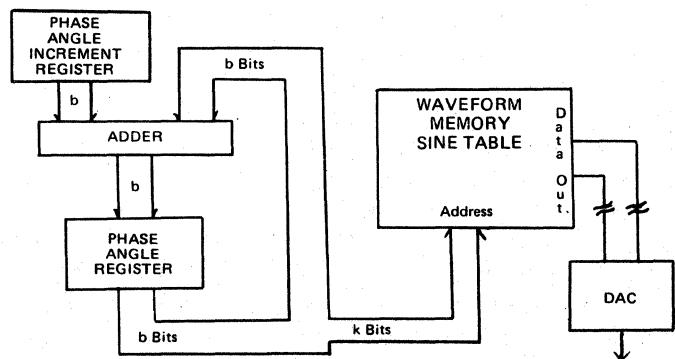


Figure 3.

Sampling Theory

At first glance, one might think that this technique would result in a large amount of distortion if the phase angle increment were as large as 180° . This would result in a frequency of $\frac{1}{2}$ of the sample rate. With this phase angle increment value, only 2 pulses would be generated per waveform cycle. It can be shown mathematically [Carlson, 1968] that if a signal contains only frequency components whose absolute values are less than some maximum frequency F_{MAX} , the signal may be completely described by the instantaneous sample values uniformly spaced in time with period $T_S \leq 1/2F_{MAX}$. If S is the sample rate, the sample period is $T_S = 1/S$. Alternatively, the Nyquist Sampling Theorem states that if a signal has been sampled at the Nyquist rate of $2F_{MAX}$ or greater ($S \geq 2F_{MAX}$), and the sample values are periodic weighted impulses, the signal can be exactly reconstructed from its samples by an ideal low pass filter of bandwidth B where $F_{MAX} \leq B \leq (S - F_{MAX})$.

Let's look at the problems that arise when we apply this theorem to generating sound. Any practical (non ideal) low pass filter will require a transition band to go from unity gain to 0 gain as shown in Figure 4. To avoid distortion, the maximum output

and increasing, up to 4 dB down, at frequencies which approach 1/2 of the sample rate. The reconstruction filter (low pass filter) may be designed to compensate for this. A different reconstruction filter should be used for each sample rate anticipated.

The theorem states that the samples from the DAC should be uniformly spaced, so the sample period jitter should be minimized. If the sample period jitter error is to be 82.6 dB below the *rms* level of a sine wave, the sample period jitter should be less than $(1.18 \times 10^{-5})/F$ where F is the output sine wave frequency [Talambiras 1977]. So if F is 10 kHz (10,000 cycles/sec), the sample period jitter should be less than 1.18 nanoseconds.

The ultimate test of a sound synthesis technique is the ear. The Soundstream Digital Recorder [Warnock, 1976] designed by Thomas Stockham, Jr. has provided one of the most convincing demonstrations of the audio application of this Nyquist Sampling Theorem. It has played back the cleanest undistorted music which I have heard from any tape recorder.

Frequency Resolution

As mentioned earlier, the frequency of the output waveform is the sample rate divided by the number of samples in one period of the waveform.*

$$\text{frequency} = \frac{\text{sample rate}}{\text{no. of samples/cycle}}$$

The number of samples in one cycle of the sine wave is determined by the sine table length (waveform memory) divided by the phase angle increment:

$$\text{no. of samples/cycle} = \frac{\text{sine table length}}{\text{phase angle increment}}$$

Substituting this expression for the number of samples/cycle into the above equation for frequency:

$$\text{frequency} = \text{sample rate} \frac{\text{phase angle increment}}{\text{sine table length}}$$

or

$$F = S (I / L) \quad \text{Eq 1.}$$

The sample rate "S" and the sine table length "L" will be held constant, so the frequency "F" may be changed by changing the phase angle increment "I".

The smallest increment in frequency (frequency resolution) will be determined by the number of bits "b" used in the phase angle increment register (see Figure 3). The phase angle register should also contain this many "b" bits. An article entitled "Pitch

* One nanosecond (ns) is one one-billionth of a second or 1 sec/1,000,000,000.

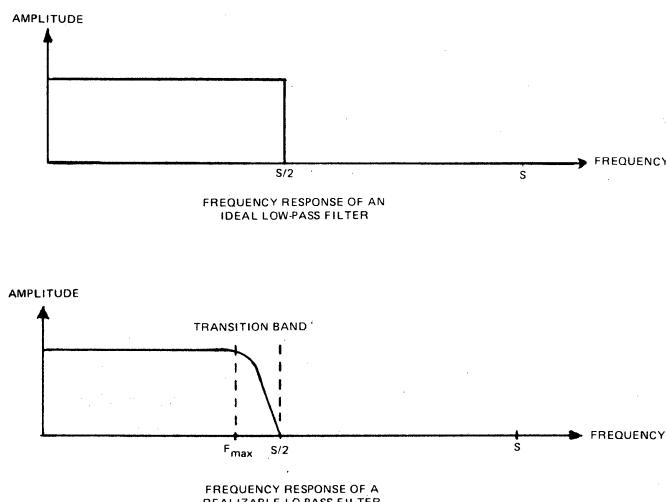


Figure 4.

frequency should be below this transition band. The maximum sinusoidal component frequency is usually limited to 36% to 40% of the sampling rate ($F_{MAX} \leq .4S$).

It might be noticed that realizable samples from the DAC will not be impulses; they will have finite pulse widths. If the pulse widths are as wide as the sampling period, the frequency response will be attenuated very little if at all for very low frequencies

Discrimination at the Threshold of Hearing" [Rakowski, 1971] indicates that under ideal listening conditions a very discriminating ear, at best, is capable of noticing a pitch change of between (approximately, as best as could be measured from his graphs) .03 Hz and .08 Hz around 160 Hz. To specify frequencies to an accuracy of .03 Hz would require 19.3 bits plus a sign bit for FM synthesis in the phase angle increment register if the maximum output frequency is 20,000 Hz as determined from the following equations:

$$\frac{N_{MAX}}{N_{MIN}} = \frac{I_{MAX}}{I_{MIN}} \quad \text{Eq 2..}$$

Where:

- I_{MAX} = the maximum phase angle increment which will produce the maximum desirable frequency of any sinusoidal component (F_{MAX}).
- I_{MIN} = the minimum phase angle increment which will produce the minimum step or change in frequency (F_{MIN}).
- N_{MAX} = the maximum number expressible with $b - 1$ bits. For frequency modulation synthesis positive as well as negative increments (or decrements) to the phase angle will be needed. An extra sign bit will be needed in any registers which handle frequency or phase angle information. So $N_{MAX} = 2^{(b-1)} - 1$.

N_{MIN} = the minimum number expressable with b bits = 1.

Equation EQ 1 may be used to determine the phase angle increment, I , in terms of the frequency, F , sample rate, S , and sine table length, L :

$$I = (F / S)L$$

This equation may be substituted into equation Eq 2 for I_{MAX} and I_{MIN} :

$$\frac{N_{MAX}}{N_{MIN}} = \frac{(F_{MAX}/S)L}{(F_{MIN}/S)L} = \frac{F_{MAX}}{F_{MIN}}$$

then replacing N_{MAX} and N_{MIN} results in:

$$\frac{2^{(b-1)} - 1}{1} = \frac{F_{MAX}}{F_{MIN}} \quad \text{Eq 3.}$$

$$b = 1 + \log_2 \left(\frac{F_{MAX}}{F_{MIN}} + 1 \right)$$

$$b = 1 + 3.32 \log_{10} \left(\frac{F_{MAX}}{F_{MIN}} + 1 \right)$$

F_{MAX} may be set to 20 kHz; and a minimum step in frequency of $F_{MIN} = .03$ Hz should be inaudible to most musicians. Substituting these values into the above equation:

$$b = 1 + 3.32 \log_{10} \left(\frac{20000 \text{ Hz}}{.03 \text{ Hz}} + 1 \right)$$

$$b = 20.3$$

Thus a 21 bit phase angle increment register should be capable of generating a glissando or vibrato which will be perceived as smooth. Minimum changes in frequency should be inaudible as steps to a listener with very discriminating ears, even in an ideal listening environment. A computer may update the frequency (incrementing by 1 the phase angle register) at the right time intervals to generate as slow a change in frequency as is desired. Alternatively in hardware a 28 bit frequency envelope may be added to the phase angle increment. For musical purposes a 20 bit phase angle increment register should be fine with the following possible exception. If the sound were played in a soundproof room to a listener who had an extremely discriminating ear, who had previously sat in silence for 15 minutes, and who listened specifically for steps in frequency in a slow glissando in the lower part of the audible frequency range, then the steps might just barely be noticed sometimes. A. Rakowski's tests were done under similar ideal conditions. He was looking for information in his tests which would serve as a basis for speculations concerning the action of the hearing mechanism. Table 1 shows the resulting values of the minimum step in frequency for different values of the number of bits "b" in the phase angle increment register and for different maximum frequencies.

A musical interval is often measured in cents instead of as a step in Hz. A cent is 1/1200 of an octave, or 1/100 of an equally tempered semitone in the chromatic scale. An interval or ratio of one semitone is equal to the twelfth root of 2.

$$1 \text{ semitone} = 2^{(1/12)}$$

$$1 \text{ cent} = 2^{(1/1200)}$$

A minimum interval (which is an indication of how accurately an instrument can be tuned) measured in cents " C_{MIN} " may be related to the minimum step " F_{MIN} " as follows:

$$F_{MIN} = F \cdot 2(C_{MIN} / 1200) - F$$

$$F_{\text{MIN}} = F \left(2(C_{\text{MIN}} / 1200) - 1 \right)$$

where F = the frequency around which the minimum frequency step is made. Solving for C_{MIN} :

$$C_{\text{MIN}} = 1200 \log_2 \left(\frac{F_{\text{MIN}}}{F} + 1 \right)$$

$$C_{\text{MIN}} = 3986.31 \log_{10} \left(\frac{F_{\text{MIN}}}{F} + 1 \right) \quad \text{Eq 5.}$$

F_{MIN} may be determined from Eq 3

$$\text{Eq 3. } \frac{2^{(b-1)} - 1}{1} = \frac{F_{\text{MAX}}}{F_{\text{MIN}}}$$

$$F_{\text{MIN}} = \frac{F_{\text{MAX}}}{2^{(b-1)} - 1} \quad \text{Eq 6.}$$

TABLE 1 Frequency Resolution

b	F max	F min	C min	
			around 65.4 Hz	around 160 Hz
bits	Hz	Hz	cents	cents
21	20000	.019	.5	.2
20	20000	.038	1.0	.4
19	20000	.076	2.0	.8
18	20000	.15	4.0	1.65
17	20000	.31	8.1	3.3
16	20000	.61	16.1	6.6
20	16384	.031	.83	.34
18	16384	.125	3.3	1.35
16	16384	.50	13.2	5.4
20	15000	.029	.76	.31
18	15000	.114	3.03	1.24
16	15000	.46	12.1	4.94
20	12800	.024	.65	.26
18	12800	.098	2.58	1.06
16	12800	.39	10.3	4.2
20	10000	.019	.51	.21
18	10000	.076	2.02	.83
16	10000	.305	8.1	3.3
20	8192	.016	.4	.17
18	8192	.063	1.65	.68
16	8192	.25	6.6	2.7
20	4096	.008	.2	.09
18	4096	.031	.8	.34
16	4096	.125	3.3	1.4

Substituting Eq 6 into Eq 5 results in

$$C_{\text{MIN}} = 3986.31 \log_{10} \left(\frac{F_{\text{MAX}}}{F(2^{(b-1)} - 1)} + 1 \right) \quad \text{Eq 7.}$$

A minimum frequency step (F_{MIN} of .038 Hz) around a frequency of 65.4 Hz is an interval or ratio of 1 cent. 65.4 Hz is the frequency of C_2 or the note C one octave above the lowest note C on a piano tuned to $A_4 = 440$ Hz in equal tempered tuning. This same step of .038 Hz around 160 Hz is an interval or ratio of .4 cents. 160 Hz is a little below the note E_3 which is in the octave below middle C on the same piano. Table 1 shows the resulting value of the minimum intervals or tuning accuracy measured in cents, C_{MIN} , around 65.4 Hz and 160 Hz for each different value of the number of bits "b" in the phase angle increment register and for different maximum frequencies.

In his article* "Table Lookup Noise for Sinusoidal Digital Oscillators", Richard Moore points out that a glissando from 50 Hz to 100 Hz will not sound smooth if a 16 bit phase angle increment (frequency control) and phase angle are used with a Nyquist frequency (1/2 of the sample rate) of 16384 Hz. As an alternative to making both the phase angle increment and the phase angle be wider words, it is suggested to only make the phase angle register wider. Then the 16 bit phase angle increment may be shifted left or right for variable frequency resolution and range. This phase angle increment shifting technique is useful for controlling the oscillator with a small computer which can do 16 bit data word arithmetic and movement. Even the popular 8 bit MOS microprocessors

* F. R. Moore, "Table Lookup Noise for Sinusoidal Digital Oscillators", *Computer Music Journal*, Vol. 1, No. 2, 1977.

Table 1. This shows the resulting values of the minimum representable step in frequency " F_{MIN} " and the corresponding minimum representable interval (or tuning accuracy) " C_{MIN} " measured in cents around 65.4 Hz and 160 Hz, for different values of the number of bits "b" in the phase angle increment register and different maximum frequencies. 20 kHz was chosen as a F_{MAX} since it is the highest audible frequency for many people. 16384 Hz was chosen since it is a power of 2; so the true frequency could be read directly from the phase angle increment for human comprehension. Any maximum frequency which is not a power of 2 will need a translation program to convert encoded values of phase angle increment to the true frequency value. For this reason 8192 Hz and 4096 Hz were also included. 15 kHz, 12.8 kHz, and 10 kHz were included as maximum frequencies since they correspond to 40% of the popularly used sample rates of 37500 samples/sec, 32000 samples/sec, and 25000 samples/sec respectively.

have the capability of performing 16 bit arithmetic. However they are very slow (even the Z-80) when working with 16 bit data. A useful integrated circuit (IC) for implementing this shifting technique is the 8 bit position scaler made by Signetics, the 8243. It is expandable to handle wider word widths and is capable of shifting up to 7 bit positions at once instead of requiring 7 clock periods like a shift register. If this shifting technique is used, an arithmetic operation such as addition of a low frequency to a high frequency would first require shifting inside the computer to line up equally weighted bits. Then the frequencies could be added. Then they might have to be shifted left or right before sending to an output port. This would slow down the shipping of frequency control data to the oscillator.

A computer which would be capable of directly outputting 20 bit data words to the digital oscillator frequency control would be useful, especially if it could also perform 20 bit arithmetic without using double precision words. However I am aware of no available 20 bit computers; and building your own would require a large amount of software development time since there would be no available compatible editors or other useful system programs. The VACuuM* is the best low cost computer design of which I am aware. It is microprogrammable, high speed (made from 2900 series IC's) and available with variable word widths (8, 16, 32, or 36 bit data words). You may microprogram in your favorite computer instruction sets and then use software which you are already familiar with. You may microcode your own special sequence of instructions for higher speed of operation. The VACuuM allows one to dynamically jump from one instruction set to another. Microprogramming in PDP 11 and PDP 10 instruction sets would be very useful since there is a large amount of useful software written in these instruction sets. Instruction cycle time is predicted to be around 300 ns for 32 bit data operations. The PC boards and bootstrap PROM's were estimated to cost around \$350. Then you would have to buy the rest of the IC's, resistors, capacitors, sockets, etc. You could start by only buying the parts to build a 16 bit computer and microcode in NOVA and/or PDP 11 instruction sets. Later you could expand it to a 32 or 36 bit machine. If you bought all of the parts for a 36 bit machine including the PC boards and bootstrap PROM's in single quantities, it was estimated to cost around \$1800. It would cost less for smaller data word machines. There will be more information about this processor in a future issue of *Computer Music Journal*.

Sine Table

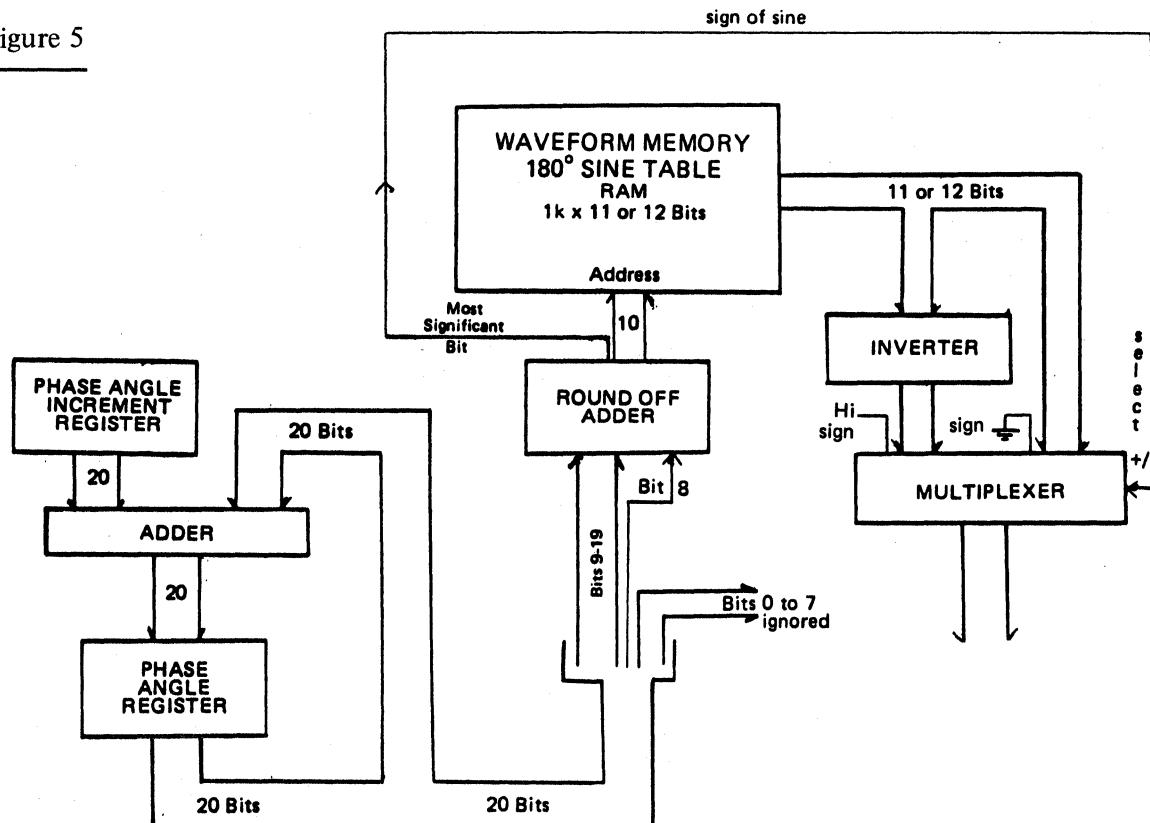
The size of the waveform memory or sine table will affect the signal to noise ratio. If all 20 bits of the phase angle register were used to address the sine table, the sine table would be 1 million words long. Fortunately this is not needed. Fewer bits may be used to address the sine table without affecting the frequency resolution if the number of bits in the phase angle register and phase angle increment register is not reduced. If the 12 most significant bits of the phase angle register are used to address a 360° sine table (truncating the 8 least significant bits), $2^{12} = 4096$ words of sine table memory would be needed. For more details, please read through the article "Table Lookup Noise for Sinusoidal Digital Oscillators." If the address of the sine table memory is rounded off to 11 bits instead of truncating it to 12 bits, the table length may be reduced to $2^{11} = 2048$ words and maintain the signal to noise ratio of that obtained when truncating the address to 12 bits. It is not worth the computation time to make an interpolating table for real time operation. Most of the sine tables in computer music software in universities use 512 words in their sine table with the interpolation technique. This corresponds to a signal to error noise ratio of 95.1 dB. For non real time operation interpolation is very useful.

A listening test to determine the necessary size of the sine table was recently conducted at the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University. Different memory sizes were tried for the table in generating a low frequency FM tone which was believed to be the worst case. A 4096 word x 12 bit/word table for 360° of a sine function was perceived to be just as distortion free as a 65536 word x 16 bit/word table. However a 1024 word x 10 bit/word table generated an objectionable amount of noise. The truncation method of addressing the sine table was used. The test would indicate that if the round off method of addressing were used, 2048 words would be needed for a 360° sine wave with negligible distortion.

In examining one cycle of a sine wave, it may be noticed that the first half (0° up to 180°) of the cycle is identical to the second half except that the sign is changed. One may store only 180° of the sine function in the sine table and then change the sign of the output sine wave every time the waveform memory address is incremented to or past 180° (or decremented below 0° as in FM synthesis). The sine function may be generated from only 90° ; however in my design the control functions for incrementing (and decrementing) the phase angle with a 90° sine table required more integrated circuits (IC's) than the number of IC's required with a 180° sine table. If anyone

* Tom Pittman and Bob Davis, "A Variable Architecture Computing Machine", *Proceedings of the West Coast Computer Faire*, 1977.

Figure 5



has found a simple method of controlling the phase angle with a 90° sine table which will allow for incrementing or decrementing of the phase angle by as much as 162° , I would appreciate hearing about it.

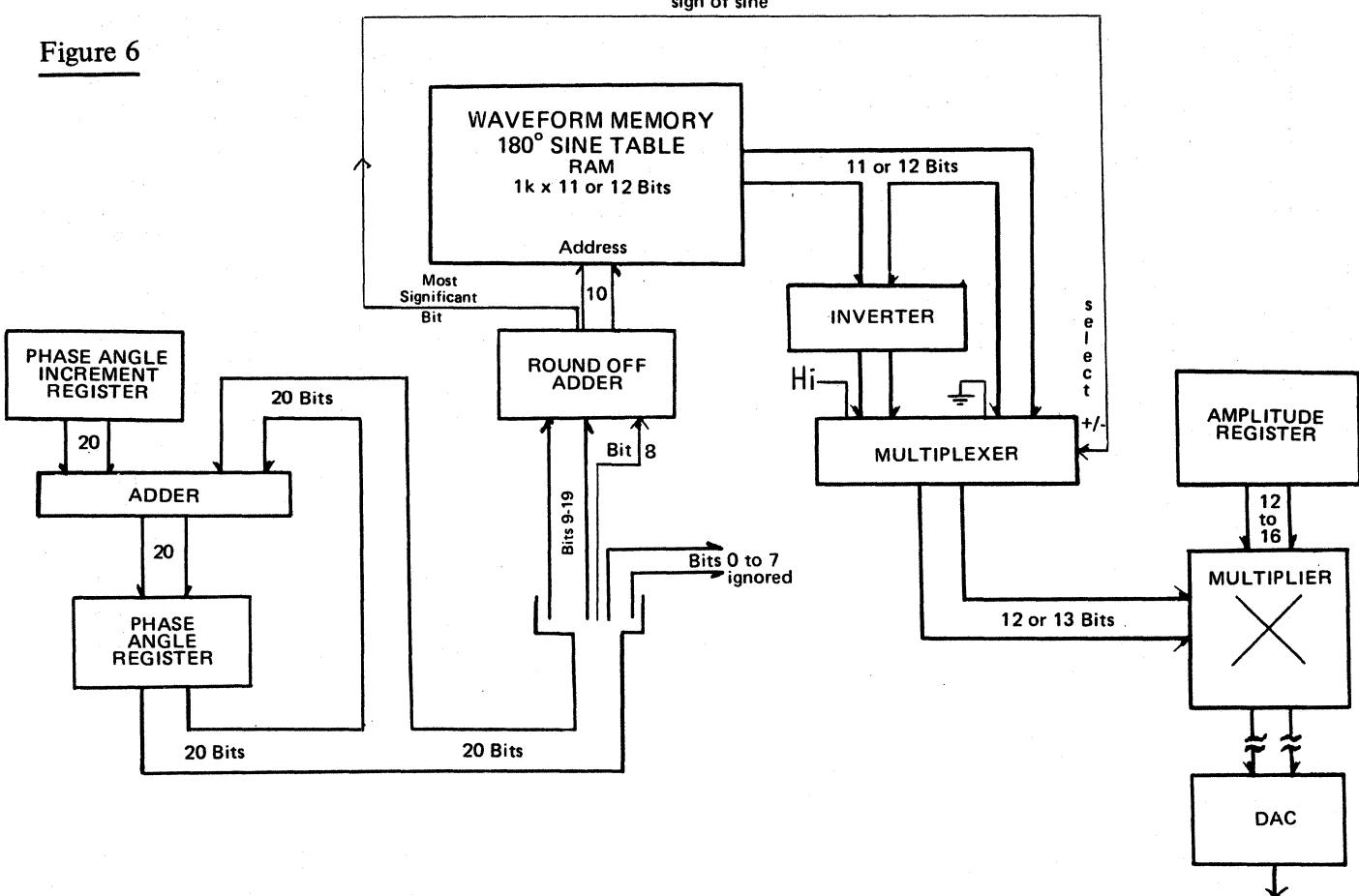
As shown in Figure 5, the output of the sine table is fed into one data input of a multiplexer. The other data input of the multiplexer is fed by the inverted output of the sine table. The most significant bit from the round off adder is used to select the inverted or noninverted half of the sine wave, resulting in a sine wave with a positive half cycle and a negative half cycle. The addition of one to finish forming a 2's complement negative number was assumed to be insignificant and thus was not included.

The sine table size experiments at Stanford indicate that a 1024 word \times 11 bit/word, 180° sine table memory will generate a negligible amount of distortion if round off addressing is used. The phase angle may be rounded off by adding bit 8 of the phase angle to the 11 bit word formed by bits 9 through 19. As mentioned earlier the most significant bit from this round off addition is used to determine the sign of the sine wave. Bits 0 to 7 of the phase angle are ignored in addressing the sine table. However the full 20 bit angle is used in calculating the phase angle so that accurate frequencies are generated.

Since fast RAM's are less expensive than fast PROM's, RAM's were chosen for the sine table. This also enables the sine table memory to be loaded with another waveshape if desired; however I doubt this feature would be used due to the time multiplexing of the sine table. The generation of a few hundred sine waves is probably more useful than a few hundred of any other waveshape. The 180° sine table may be stored in three 1 k \times 4 bit RAM IC's such as the AMD (Advanced Micro Devices) 9135 or Intel's 2114 or the soon-to-come fast VMOS RAM from AMI (American Microsystems Inc.). Alternatively the sine table could be made with eleven of the less expensive 1 k \times 1 bit RAM IC's such as the 2102A or 9102. For high speeds the AMD 9135J RAM or the VMOS RAM from AMI will be useful.

The output from the sine table will have a constant amplitude. The amplitude of the sine wave is fed into a multiplier which multiplies the sine wave by an amplitude which may vary in time as illustrated in Figure 6. In an analog synthesizer this is similar to connecting the output of a voltage controlled (control of frequency) oscillator to a voltage controlled amplifier signal input. The control input of the voltage controlled amplifier would be used to vary the amplitude in time (often controlled by an envelope generator). The frequency of the digital oscillator is binary number controlled as is the amplitude input of the multiplier.

Figure 6



How wide a multiplier is needed? If a 12 bit sine wave is multiplied by the contents of an amplitude register of infinite width the resulting signal to noise ratio is only 6 dB above that which results from using a 12 bit wide amplitude register. If 16 bits are used in the amplitude register, the signal to noise ratio of the product is about $\frac{1}{2}$ dB less than that resulting from an amplitude register of infinite width.

Generation of Many Sinusoidal Components

Building a separate oscillator for each sinusoidal component would be expensive if one desired over 100 sinusoidal components. However one low distortion digital oscillator may be time multiplexed to generate a large quantity of low distortion sinusoidal components (with independent control of each frequency and independent control of each amplitude). The number of sinusoidal components which the oscillator will generate is determined by the speed of the IC's and the sample rate.

If the sample rate is set to 40,000 samples/sec, a new sample must be generated every 25 μ s*. Each

* A microsecond (μ s) is one one-millionth of a second or 1 second/1,000,000.)

output sample from the DAC is the sum of a sample from each of the sinusoidal components. If 100 sinusoidal components are desired, each sinusoidal component will be allotted $25 \mu\text{s}/100 = 250 \text{ ns}$ to compute its sample. With a sample rate of 25,000 samples/sec 40 partials can be generated if each sinusoidal component sample is calculated in $40 \mu\text{s}/40 = 1 \text{ us}$.

The number of sinusoidal components which may be generated is inversely proportional to the sample rate (and thus the maximum output frequency). It is possible with very fast IC's to generate 512 partials at a sample rate of 50,000 samples/sec thus enabling a maximum frequency of 20 kHz. To achieve this would require a very large quantity of expensive, heat generating ECL IC's including a very high speed multiplier such as the parallel pipeline multiplier discussed in the first issue of *Computer Music Journal*. A couple of less expensive implementations which require fewer IC's and generate a maximum of 256 sinusoidal components (dependent on desired audio bandwidth) will be described.

What kind of modifications must be made to the digital oscillator in order to generate many sinusoidal components? In Figure 6, the phase angle increment register must be replaced by a RAM which contains

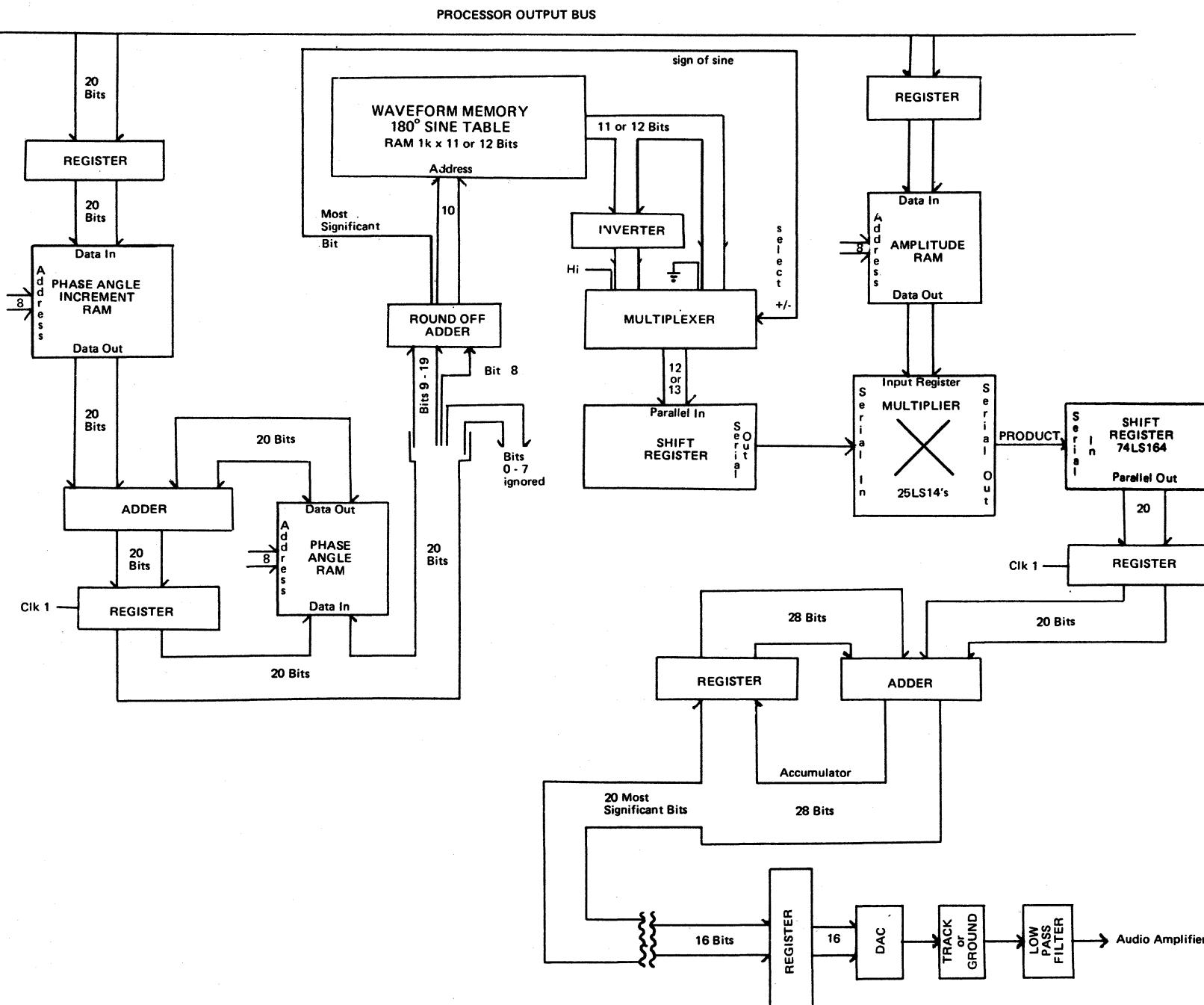


Figure 7

the phase angle increment (frequency) of each sinusoidal component. As shown in Figure 7, a phase angle RAM is used to hold the phase angle of each sinusoidal component. The amplitude register is replaced by a RAM which contains the amplitude of each sinusoidal component. Following the multiplier will be an accumulator to sum the sinusoidal components. A counter is used to address the RAM's to determine which sinusoidal component parameter is to be accessed. The number of desired sinusoidal components will be placed in a register. This number will be compared with the output of the counter. When the two are equal, the counter will be cleared and start counting over again.

If one had to perform all of the operations on one sinusoidal component before starting the operations on another sinusoidal component, one would not be able to calculate many sinusoidal components during the sample period. The registers allow one sinusoidal component to have its phase angle increment added to its phase angle, while another sinusoidal component is being looked up in the sine table, while still another sinusoidal component is being multiplied by its amplitude, while another sinusoidal component is being summed in the final accumulator. The individual component samples cannot move from one stage to the next before all of the other stages have finished their operation. Then all of the stages will simultaneously move their contents to the following stage, and each will accept a new content from its input. Thus the slowest stage of the pipeline will determine the throughput rate. Thus the throughput rate is increased; and more sinusoidal components may be generated in real time.

A relatively inexpensive digital oscillator may be constructed if the multiplier is made with two of the Am 25LS14 integrated circuits from Advanced Micro Devices (AMD) and a few inexpensive shift registers. These multiplier IC's are slow but inexpensive. A 16 bit x 12 bit multiplication will require a maximum of 1.2 us; a 12 bit x 12 bit multiplication will require a maximum of 1.04 us. With this speed, the multiplication stage is definitely the pipeline bottleneck; so inexpensive slow RAM's such as the Intel 2101A-4 may be used in the phase angle increment, phase angle, and amplitude RAM's. A read followed by a write to the same address will require 700 ns maximum, well under the time required for the Am 25LS14 multipliers. Eleven or twelve of the inexpensive 2102A-6 RAM's from Intel may be used in the waveform memory.

When using this Am 25LS14 multiplier, a wide bandwidth (high maximum frequency) is not obtainable simultaneously with a large number (such as 100) of sinusoidal components. A wide bandwidth (such as a maximum frequency of 20000 Hz) is obtainable if fewer sinusoidal components (approxi-

mately 20) are desired. If double the number of sinusoidal components is desired simultaneously with a wide bandwidth and small additional cost, a couple of multipliers made from these Am 25LS14's may be used. Two numbers to be multiplied would be latched into the input registers of the first multiplier; then the second two numbers to be multiplied would be latched into the second multiplier. Following this the third two numbers would be latched into the first multiplier at the same time that the first product was being read from the first multiplier. Then the second product would be read from the second multiplier at the same time that the fourth two numbers to be multiplied are being latched into the input registers of the second multiplier. This cycle would continue, thus halving multiplication time. With this faster multiplication stage, the faster 2101A-2 IC's could be used in the phase angle, phase angle increment, and amplitude RAM's. The waveform memory could now be constructed with 2102A RAM's which are faster than the 2102A-6 RAM's.

Thus far, no provision has been made in this oscillator for FM synthesis. The output of the multiplier should be available as an input (through the FM register) to the adder which follows the phase angle increment RAM as illustrated in Figure 8. This will allow one sinusoidal component to modulate the frequency of another sinusoidal component. First the FM register is cleared so that the modulation frequency will not be modulated. Then the modulation frequency is read from the phase angle increment RAM into the adders. After the fifth clock tick, the corresponding sample (the instantaneous modulation amplitude) of the modulating sine wave will be clocked into the FM register. The modulation index will be controlled by the amplitude RAM and the multiplier. The resulting instantaneous modulation amplitude will be added to the carrier frequency read from the phase angle increment RAM, thus modulating it. If the carrier frequency is not ready to be read from the phase angle increment RAM, the FM register will not be clocked. However the register which feeds the FM register will hold this instantaneous modulation amplitude until it is needed. Meanwhile the accumulator which follows the multiplier may be used to sum other sinusoidal components.

A few composers have been generating interesting timbres by using several different modulation frequencies, each with a small index of modulation, thus avoiding typical FM clichés. Many modulating sinusoidal components, each with an independently variable modulation index, may be summed together in the accumulator which follows the multiplier as shown in Figure 8. This sum may be used to modulate a carrier frequency by clocking it into the FM register. Obviously multiple carriers are easily generated. Timing must be carefully arranged so that the

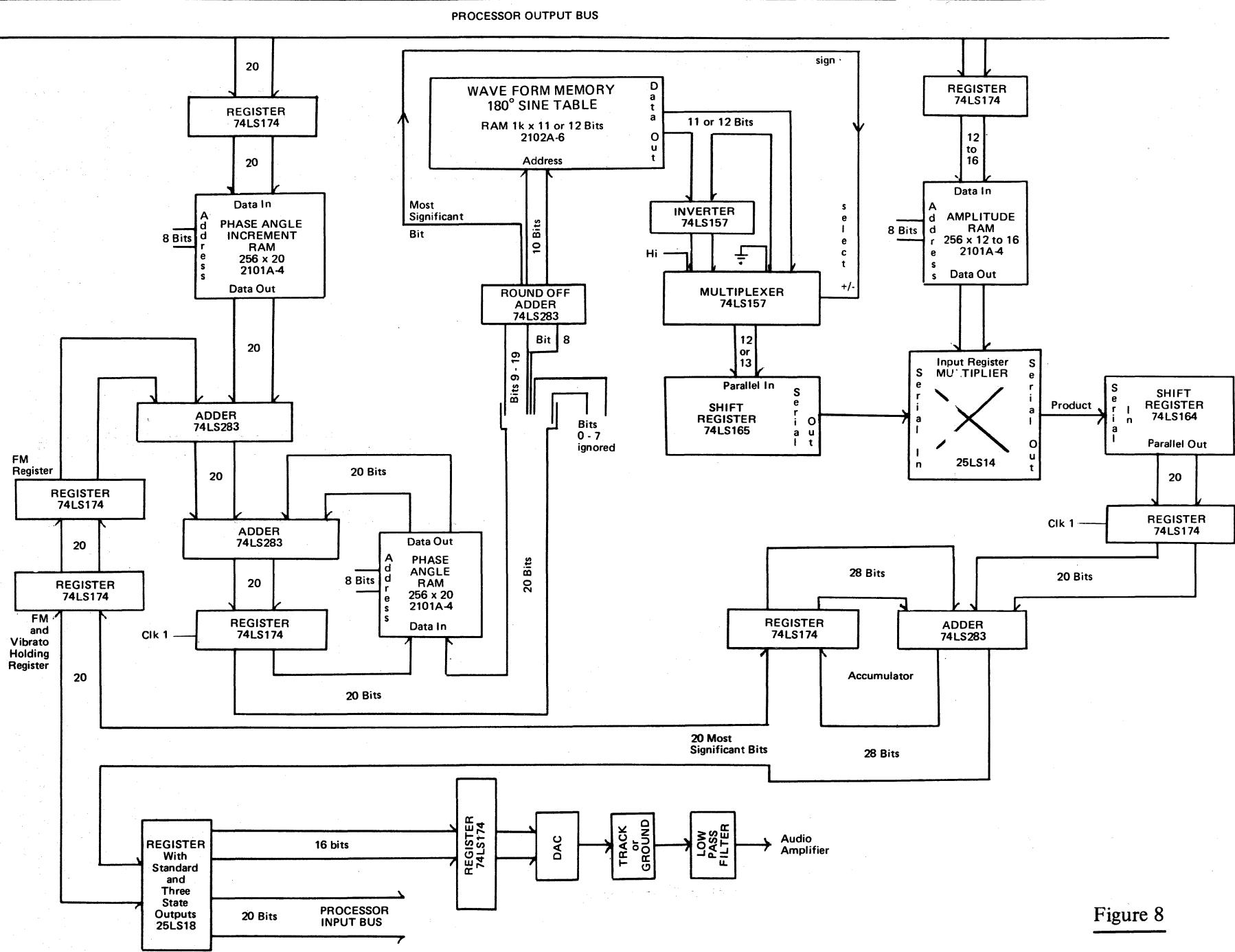


Figure 8

correct instantaneous modulation amplitude is added to the appropriate carrier frequency.

The address of the phase angle increment RAM, the phase angle RAM, and the amplitude RAM may be controlled by one 8 bit up counter. As shown in Figure 9, the output of the counter is compared with the number (N) of desired sinusoidal components. If they are equal, the counter will be reset. When clocked, the output of the counter will be changing; and the comparator might erroneously try to reset the counter. A one shot is utilized to prevent the counter from being reset just after the counter is clocked. A second one shot guarantees that a minimum clear pulse width will be provided.

The 8 bit register which holds N is loaded from the computer output bus. To avoid the possibility of loading the N register at a time when the counter might need to be reset, the 4 most significant bits of the counter are used to prevent the N register from being loaded unless the counter is in the early part (counter output of 0 to 15) of its counting sequence. The controlling computer places the value N on its output bus and then sends a ready signal to strobe N into the N register.

When the frequency of the second sinusoidal component is read from the phase angle increment RAM, the amplitude of the first sinusoidal compo-

nent is read from the amplitude RAM. With this timing, a sample of the first sinusoidal component from the sine table arrives at one input to the multiplier at the same time that the amplitude of the first sinusoidal component arrives at the other input of the multiplier. The computer which updates the RAM's must realize that the counter addresses the frequency of the n^{th} sinusoidal component at the same time that it addresses the amplitude of the $(n-1)^{\text{th}}$ sinusoidal component. The method of updating the frequency is shown for the phase angle increment in Figure 9. The digital oscillator frequency control is allotted (hardwired to) a group of 256 of the available computer addresses. The 8 least significant bits of the computer address bus are compared with the counter output, while the most significant bits of the computer address bus are compared with the allotted (hardwired) address of the digital oscillator frequency control. If both are equal, and the update frequency is ready, and clock 1 is at a high level, then the new frequency value will be written into the phase angle increment RAM. The amplitude RAM is updated similarly. Amplitude envelopes as well as frequency envelopes may be generated in the controlling computer's software and used to update the corresponding amplitudes and frequencies. Simple hardware envelope generators are used in the oscillator shown in Figure 10.

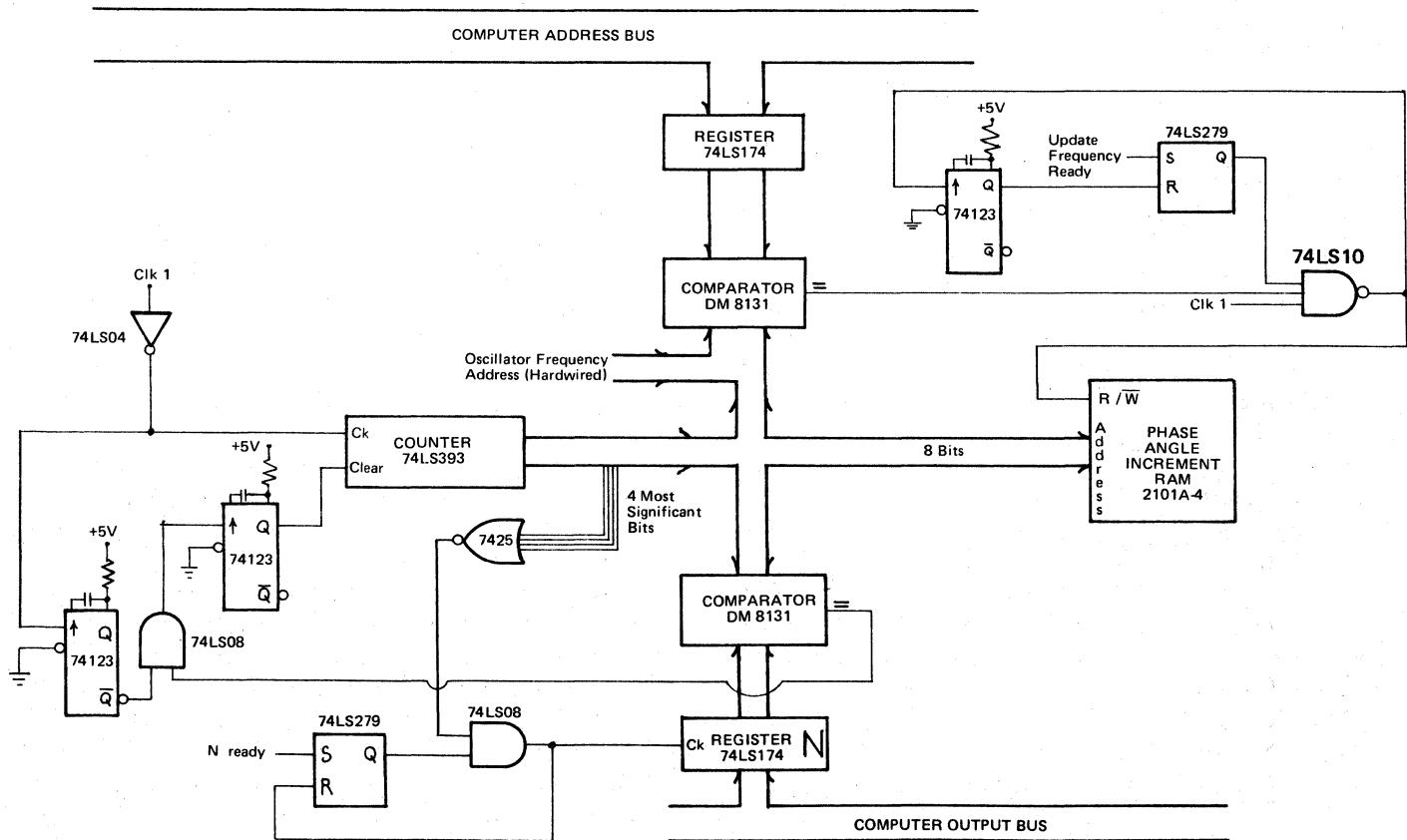


Figure 9. Logic Circuit for addressing and updating the RAM's in the slow oscillator.

Prices and quantities of the integrated circuits for the slow inexpensive oscillator diagrammed in Figures 8 and 9 are listed in Table 2. The price of a PC board or socket board plus the small cost of miscellaneous capacitors and resistors should be added to these prices.

The DAC's, track or ground switches, and low pass filters are not listed in Table 2. The track or ground is a bipolar switch so it may be constructed from a few transistors which cost next to nothing. However the DAC will be a major cost. Datel manufactures a 16 bit DAC called the DAC-HR 16 B which costs \$299. This is the least expensive high fidelity DAC of which I am aware. A less expensive 12 bit DAC such as Harris Semiconductor's HI-562A-5 might be used for initial experimentation and later replaced with a higher fidelity DAC. This Harris DAC will settle to $\pm 1/10$ LSB in 150 ns at 25°C and costs \$29 in 100 quantity (probably around \$40 for one).

The low pass filter needs to be maximally flat in its pass band and have a very sharp frequency roll off above the cut off frequency (seven pole filters are recommended). A voltage controlled low pass filter would enable the cut off frequency to be dynamically changed as the sample rate was dynamically varied

between several fixed values. With a dynamically variable sample rate, high frequencies could be generated with a few sinusoidal components during one section of music, while in another section of the music a large number of sinusoidal components could be generated with a lower maximum frequency. However voltage controlled 7 pole low pass filters are difficult to design. CCRMA at Stanford University currently uses several of the low pass filter series, the J77C, made by T. T. Electronics.* These are very low noise and distortion filters (since they are passive) and have approximately $\frac{1}{2}$ dB of ripple. The transition bandwidth from the cut off frequency to 0 output is narrow enough to allow output frequencies of 40% of the sample rate. T. T. Electronics is now making a new filter series, the J87C, which has an even narrower transition bandwidth. If the noise, distortion, and ripple of the J87C is as low as that of the J77C, it should be useful for generating output frequencies possibly as high as 44% of the sample rate. Everytime the sample rate is changed a different low pass filter is needed. At \$65 for each of the T. T. E. filters this could be expensive if several different possible sample rates are desired.

* T. T. Electronics, 2214 S. Barry Ave., Los Angeles,
CA 90064, (213) 478-8224.

Table 2. Prices and Quantities of IC's for Slow Inexpensive Oscillator
 (Prices listed are manufacturer list prices, not "surplus" or discount dealers prices.)

	Manufacturer Part Number	Price Per IC	Quantity of IC's Required	Subtotal Price
Phase Angle Increment, Phase Angle and Amplitude RAM's (256 x 4 bit RAM's)	2101A-4	4.20	14	58.80
Waveform Memory (1 k x 1 bit RAM's)	2101A-6	3.25	11	35.75
Registers	74LS174	1.58	37	60.04
Register with standard and 3 state outputs	Am 25LS18	3.78	5	18.90
Adders	74LS283	1.85	20	37.00
Multiplier	Am 25LS14	13.93	2	27.86
Shift Register (parallel in serial out)	74LS165	2.50	2	5.00
Shift Register (serial in parallel out)	74LS164	2.20	3	6.60
Inverter	74LS04	.45	3	1.35
Multiplexer	74LS157	1.70	3	5.10
Counter	74LS393	5.00	1	5.00
Comparators	DM 8131	4.00	5	20.00
One shots	74123	1.15	2	2.30
4 input NOR gate	7425	.80	1	.80
S R flip flop	74LS279	.90	1	.90
2 input AND gates	74LS08	.40	1	.40
3 input NAND gate	74LS10	.40	1	.40

112 286.20
 Total Number of IC's Total Cost
 not including DAC, track or hold, and low pass filter

Table 3 lists the number of sinusoidal components which may be generated with a given sample rate using the inexpensive slow oscillator shown in Figures 8 and 9.

TABLE 3. Trade off between the number of sinusoidal components which may be generated and the sample rate for the slow inexpensive oscillator shown in Figures 8 and 9. The 25LS14's in the multiplier are the main speed bottleneck in this oscillator. For more sinusoidal components at a wide bandwidth see the faster oscillator shown in Figures 10 and 11, and Tables 4 and 5.

Sample Rate samples/sec	Maximum Output Frequency of any sinusoidal component (40% sample rate)	Number of generatable sinusoidal components	
		for a 12 bit x 12 bit multiplication	for a 12 bit x 16 bit multiplication
50000	20000	19	16
40000	16000	24	20
37500	15000	25	22
32768	13107	29	25
25000	10000	38	33
20000	8000	48	41
16384	6554	58	50
13000	5200	73	64
10000	4000	96	83

High Speed Oscillator

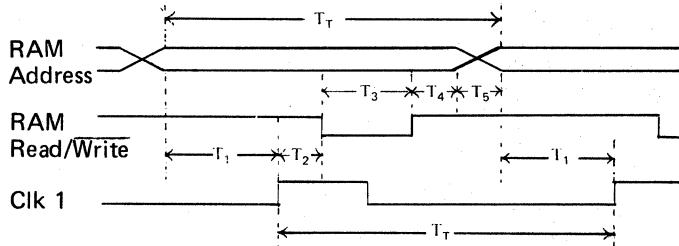
If more sinusoidal components are desired with a wide bandwidth, then faster, more expensive RAM's and a faster, more expensive multiplier will be needed as mentioned earlier. As illustrated in Figure 10, a pipeline register will be placed after each basic operation to speed up throughput rate. A timing diagram for the phase angle RAM stage of this higher-speed pipeline is provided in Figure 11. Identical timing is used for the phase angle increment and amplitude RAM's. Clk 1 clocks all of the pipeline registers in Figure 10. A 20 bit addition stage will occur during the period of Clk 1. The address is supplied by an 8 bit counter (such as the 74LS393 manufactured by Texas Instruments) through a pipeline register to the RAM. So the time to change the address is simply the clock to output delay of a register.

The stage in which the phase angle increment is added to the phase angle is one of the slowest stages in the pipeline. Either this stage or the multiplication stage is the bottleneck. The speed of the phase angle

computation stage will be very dependent on the speed of the RAM. A two port RAM would be very useful for the phase angle RAM. One port could be read from at the same time that the previous address was being written. However, the largest multiport IC memory of which I am aware is the Am 29705 made by Advanced Micro Devices (AMD). It is only a 16 x 4 bit RAM and consumes 110 mA typically, 177 mA maximum. In order to obtain 256 sinusoidal components, a 256 x 20 phase angle RAM would be needed. This would require 80 of these multiport RAM's, so I would not use these IC's. A 256 x 4 low power consumption two port RAM would be an ideal IC to use in the phase angle RAM (as well as in the phase angle increment and amplitude RAM's for updating the frequencies and amplitudes at the same time that they are being read). Unfortunately I know of no IC's like this which are manufactured. If anyone knows of a multiport RAM IC which contains more memory (greater than 16 x 4 bits) than the Am 29705, I would appreciate hearing about it.

Fairchild manufactures a fast 256 x 4 bit RAM (not multiport), the 93L422, which draws typically 60 mA, 75 mA maximum, of power supply current so it should not become hot. Only five of these IC's are needed in the phase angle RAM. The read address access time is typically 45 ns, 60 ns maximum; and the write time plus address hold time is typically 30 ns, 50 ns maximum. Data setup time is typically

Figure 11. Timing Diagram for the oscillator shown in Figure 10.



T_1 = the read address access time of the RAM plus the data set up time of a register.

T_2 = the clock to output delay of a register plus the data set up time (before the beginning of the write pulse) of the RAM.

T_3 = the RAM write pulse width.

T_4 = the RAM address hold time.

T_5 = the time needed to change the address. The counter which generates the address already has the next address, so the pipeline register (between the counter and the RAM) clocks in the next address. Thus T_5 will simply be the clock to output delay time of a register.

T_T = the total delay time of one pipeline stage.

Table 4. The number of sinusoidal components which may be generated depending on the time delay of the slowest pipeline stage. The time delay of a pipeline stage is summed from the time delays of the IC's within that pipeline stage. The delays of several different possible combinations of IC's within a pipeline stage are shown. The left half of the table assumes that the phase angle computation stage will be the slowest pipeline stage. The right half assumes that any 20 bit addition stage will be the pipeline bottleneck (which is unlikely). The bottom of Table 4 shows the number of sinusoidal components which may be generated depending on the sample rate and the IC's which are used (assuming that one is willing to buy or build a fast enough multiplier - the multiplier will likely be the bottleneck).

Manufacturer Part Number		Delay Time (nanoseconds) in RAM Stages as a Function of Which IC's Are Used							Delay Time in Adder Stages As a Function of Which IC's Are Used				
RAM read address	74LS202	35 typ	45 max	45 max	40	40	60	60	250				
access time	82S116												
	93L422												
	2101A-2												
Register data set up time	74LS174	20	20	5	20	5	20	5	20				
	74S174												
Register clock to output delay	74LS174	30	30	17	30	17	30	17	30				
	74S174												
RAM data set up before leading edge of write pulse width	74LS202	0	0	0	0	0	5	5	0				
	82S116												
	93L422												
	2101A-2												
RAM write pulse width	74LS202	15 typ	25 max	25 max	25	25	45	45	150				
	82S116												
	93L422												
	2101A-2												
RAM address hold time	74LS202	-5 typ	0 max	0 max	0	0	5	5	0				
	82S116												
	93L422												
	2101A-2												
Address change time	74LS174	30	30	17	30	17	30	17	30				
	74S174												
Register clock to output delay	74LS174									30	17	30	17
	74S174												
Addition time	74LS283									107	107	69	69
	74S283												
	74S181, 74S182												44
Register data set up time	74LS174									20	5	20	5
	74S174												
Total time delay of 1 stage		125	150	109	145	104	195	154	480	157	129	119	91
													66
Sample Rate samples/sec	Sample Period us	Maximum Frequency Hz	Number of Sinusoidal Components Which May Be Generated										
50000	20	20000	158	131	181	135	190	100	127	39	125	153	166
40000	25	16000	198	164	227	170	238	126	160	50	157	191	208
37500	26.7	15000	211	175	242	181	254	134	171	53	167	204	222
32768	30.5	13107	242	201	256	208	256	154	196	61	192	234	254
25000	40	10000	256	256		256		203	256	81	252	256	256
20000	50	8000						254		102	256		
16384	61	6554						256		125			
12000	83.3	4800								171			
10000	100	4000								206			

0 ns, maximum 5 ns. For higher speed, the 74LS202 made by Texas Instruments may be used in the phase angle RAM. It has a read address access time of typically 35 ns. Maximum specifications were not available before the journal went to press, however I suspect it would have a maximum read address access time of 45 ns. The write time plus address hold time is typically 10 ns and I suspect it would be a maximum of 25 ns. The 74LS202 should be typically 30 ns faster than the 93L422, and a maximum of 40 ns faster. The main disadvantage of the 74LS202 is that it is 256 x 1 bit, so 20 of these IC's would be needed for the phase angle RAM as compared to 5 of the 93L422. However the 93L422 is manufactured in a 22 pin dual inline package (DIP) while the 74LS202 is available in the convenient 16 pin DIP. The 93L422 is priced at \$24.95 in quantities of 1 to 24 as of April, 1977; while the 74LS202 is priced at \$5.25 in quantities less than 25 and \$3.75 in quantities of 25 to 99 as of May, 1977. The power supply current on the 74LS202 is 55 mA typically, 70 mA maximum, so it should not become hot.

Signetics makes a 256 X 1 bit RAM, the 82S116, which is typically 10 ns faster than the 74LS202. Address access time for the 82S116 is typically 30 ns with a maximum of 40 ns. The data setup time is specified from the end of the write pulse width and is the same width as the write pulse width. The write pulse width is typically 15 ns, 25 ns maximum. The address may typically be changed 5 ns before the end of the write pulse, and at worst changed exactly at the end of the write pulse. Thus the time required to read, then write to the same address is a maximum of 65 ns, 40 ns according to typical specifications. This maximum time is about 5 ns faster than my estimated (but unspecified by Texas Instruments as of this writing) maximum time for the 74LS202. The 82S116 costs \$11.20 in quantities of less than 25, \$10.00 in quantities of 25 to 99 as of May, 1977. Power supply current is 80 mA typically, 115 mA maximum; so this IC should not run cool. Since the 74LS202 has not yet been fully characterized, it is difficult to decide which RAM to use. Texas Instruments generally provides conservative specifications. The 82S116 presently costs 2 $\frac{2}{3}$ times as much and consumes approximately 1 $\frac{1}{2}$ times as much power supply current as the 74LS202. Unfortunately these two IC's are not pin compatible.

When attempting to minimize the number of IC's required and yet still desiring high speed, the 93L422 is a good choice for the phase angle RAM. To minimize costs at the beginning of the project, 2101A-2 RAM's made by Intel could be used. Later when a wider bandwidth with a large number of sinusoidal components is desired, these RAM's could be unplugged from their IC sockets and replaced with the pin compatible higher speed 93L422 RAM's. The

2101A-2's are 256 x 4 bit static RAM's which cost \$5.50 each in quantities of under 25 as of February, 1977. The read address access time is 250 ns maximum; and the write time is a maximum of 150 ns if writing to the same address as the read address.

In looking for fast RAM's, it was assumed that the phase angle calculation stage of the pipeline would require more or about the same amount of time as the multiplication stage. This assumes that the multiplier can multiply two words in less than 200 ns and preferably in approximately 100 ns. The TRW MPY-16AJ will latch and multiply two 16 bit words in a maximum of 190 ns. The MPY-12AJ will latch and multiply two 12 bit words in a maximum of 175 ns. TRW is working on a lower power version of these multipliers which is expected to be faster by a factor of 1.5 to 2 (thus multiplying in approximately 100 ns). Monolithic Memories 67558 expandable 8 bit multiplier should also be considered for high speed. Unfortunately these high speed multipliers are expensive. The MPY-16AJ costs \$300 in single quantities while the MPY-12AJ costs \$165 in single quantities. The MMI multiplier price has not been announced yet; however, it will probably cost between \$50 and \$80 when first released.

Using the design shown in Figure 10, Table 4 shows the time delay of one pipeline stage as summed from the individual IC time delays within the pipeline stage. The left half of the table assumes that the phase angle computation stage will be the slowest pipeline stage. The right half assumes that any 20 bit addition stage will be the pipeline bottleneck (which is unlikely). Most of these times assume that TRW will be able to manufacture a faster high resolution multiplier so that the multiplier does not become the bottleneck. If TRW's 190 ns multiplier, MPY-16AJ, is used for a multiplier, I would recommend using the Fairchild 93L422 RAM and low power Schottky IC's for adders and registers. The parts picked in Figure 10 assume that a multiplier with a typical multiply time of 125 ns and a maximum of 157 ns is used. A parallel pipeline multiplier constructed from Monolithic Memories' 67558 IC's should meet this spec if TRW is not able to produce a higher speed multiplier. The bottom of Table 4 shows the number of sinusoidal components which may be generated depending on the sample rate and the IC's which are used (assuming that no other pipeline stage is a bottleneck).

IC's were chosen for the oscillator in Figure 10 with speed and relatively low power consumption in mind (so that little heat is generated). If the heat generated by the IC's is minimized, the number of fans (which are acoustically noisy) needed to cool the circuit will also be minimized. Rotron now manufactures a very quiet fan which would be useful for cooling the multiplier circuit. If still higher speed is

desired, the low power Schottky IC's could be replaced with their pin compatible Schottky parts. The Signetics 82S116 could be used in the 256 word RAM's and 1 k x 1 bit bipolar RAM's such as the Fairchild 93415 could be used in the waveform memory. However price and power consumption would take a quantum leap.

The RAM address control logic for the oscillator shown in Figure 10 is not drawn here. However a timing table for this pipeline is shown in Table 5. This

table uses a total of 7 sinusoidal components, but the timing is applicable to any number of sinusoidal components.

The address of the amplitude of the n^{th} sinusoidal component changes every sample period since the amplitude is written into the amplitude RAM 2 periods of Clk 1 (2 address changes) after it is read from the amplitude RAM. So the address of the amplitude RAM provides no direct clue to determine which sinusoidal component amplitude may be up-

Table 5. Timing Table for the High-Speed Oscillator Design shown in Figure 10.

Frequency Slope RAM		Phase Angle Increment (Frequency) RAM		Phase Angle RAM		Amplitude Slope (of Envelope) RAM		Amplitude RAM		Accumulator	
Address	Read Data	Address	Written Data	Address	Read Data	Written Data	Address	Read Data	Address	Written Data	Written Data
0	0	0	—	0	—	—	—	—	0	—	—
1	1	1	—	1	—	—	—	—	1	—	—
2	2	2	0	2	0	—	—	—	2	—	—
3	3	3	1	3	1	—	—	—	3	—	—
4	4	4	2	4	2	0	0	0	4	—	—
5	5	5	3	5	3	1	1	1	5	—	—
6	6	6	4	6	4	2	2	2	6	0	—
any address	update or wait	0	5	0	5	3	3	3	0	1	0
		1	6	1	6	4	4	4	1	2	1
0	0	2	—	2	—	5	5	5	2	3	2
1	1	3	—	3	—	6	6	6	3	4	3
2	2	4	0	4	0	—	any address	update or wait	4	5	4
3	3	5	1	5	1	—	any address	update or wait	5	6	5
4	4	6	2	6	2	0	0	0	6	—	6
5	5	0	3	0	3	1	1	1	0	—	do not write
6	6	1	4	1	4	2	2	2	1	0	—
any address	update or wait	2	5	2	5	3	3	3	2	1	0
		3	6	3	6	4	4	4	3	2	1
0	0	4	—	4	—	5	5	5	4	3	2
1	1	5	—	5	—	6	6	6	5	4	3
2	2	6	0	6	0	—	any address	update or wait	6	5	4
3	3	0	1	0	1	—	any address	update or wait	0	6	5
4	4	1	2	1	2	0	0	0	1	—	6
5	5	2	3	2	3	1	1	1	2	—	do not write
6	6	3	4	3	4	2	2	2	3	0	—
any address	update or wait	4	5	4	5	3	3	3	4	1	0
		5	6	5	6	4	4	4	5	2	1
0	0	6	—	6	—	5	5	5	6	3	2
1	1	0	—	0	—	6	6	6	0	4	3
2	2	1	0	1	0	—	any address	update or wait	1	5	4
3	3	2	1	2	1	—	any address	update or wait	2	6	5
4	4	3	2	3	2	0	0	0	3	—	6
5	5	4	3	4	3	1	1	1	4	—	do not write
6	6	5	4	5	4	2	2	2	5	0	—
any address	update or wait	6	5	6	5	3	3	3	6	1	0
		0	6	0	6	4	4	4	0	2	1
n	n	x	n - 2	x	n - 2	n - 4	n - 4	n - 4	x	n - 6	n - 7

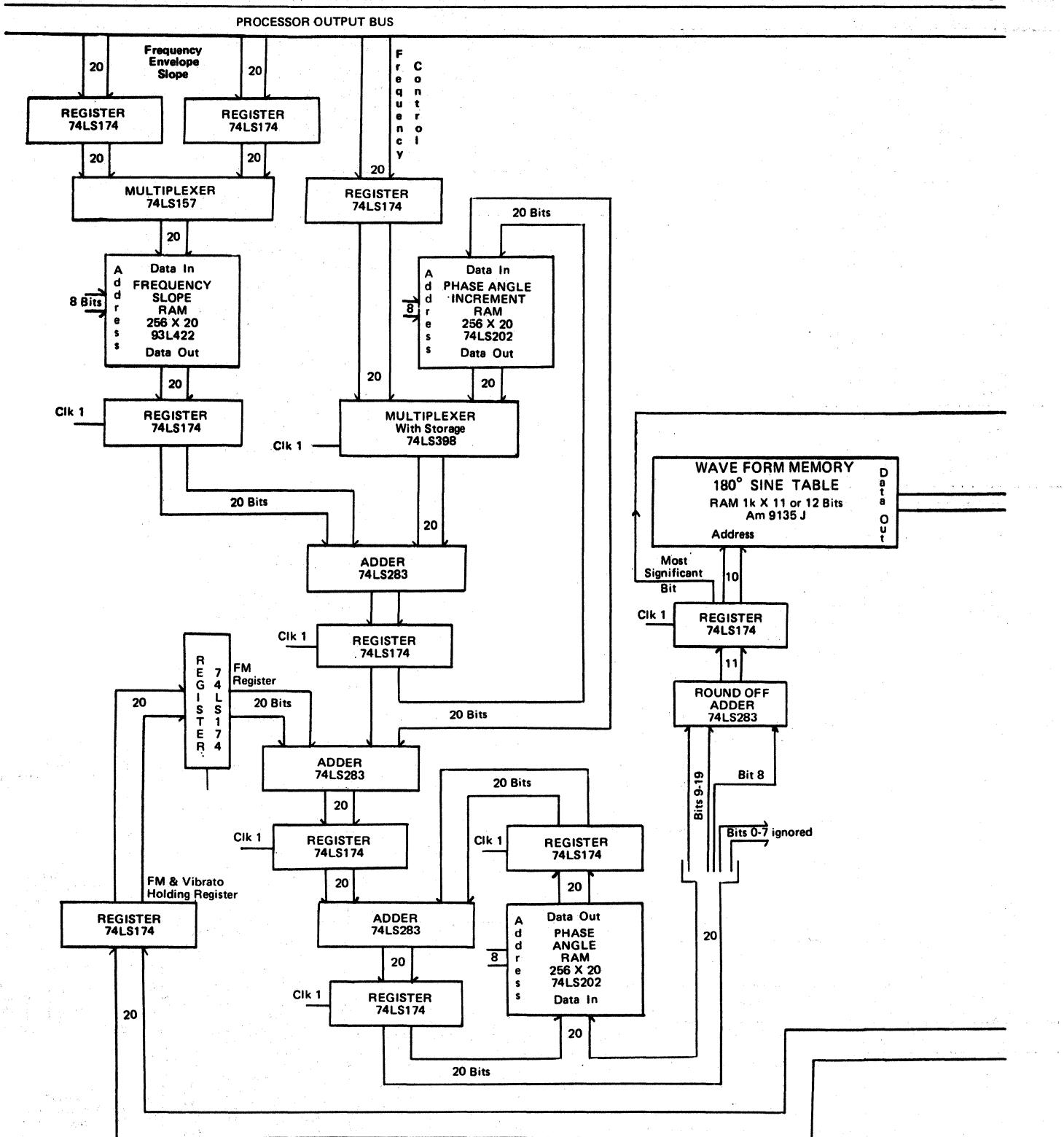


Figure 10. High Speed Oscillator

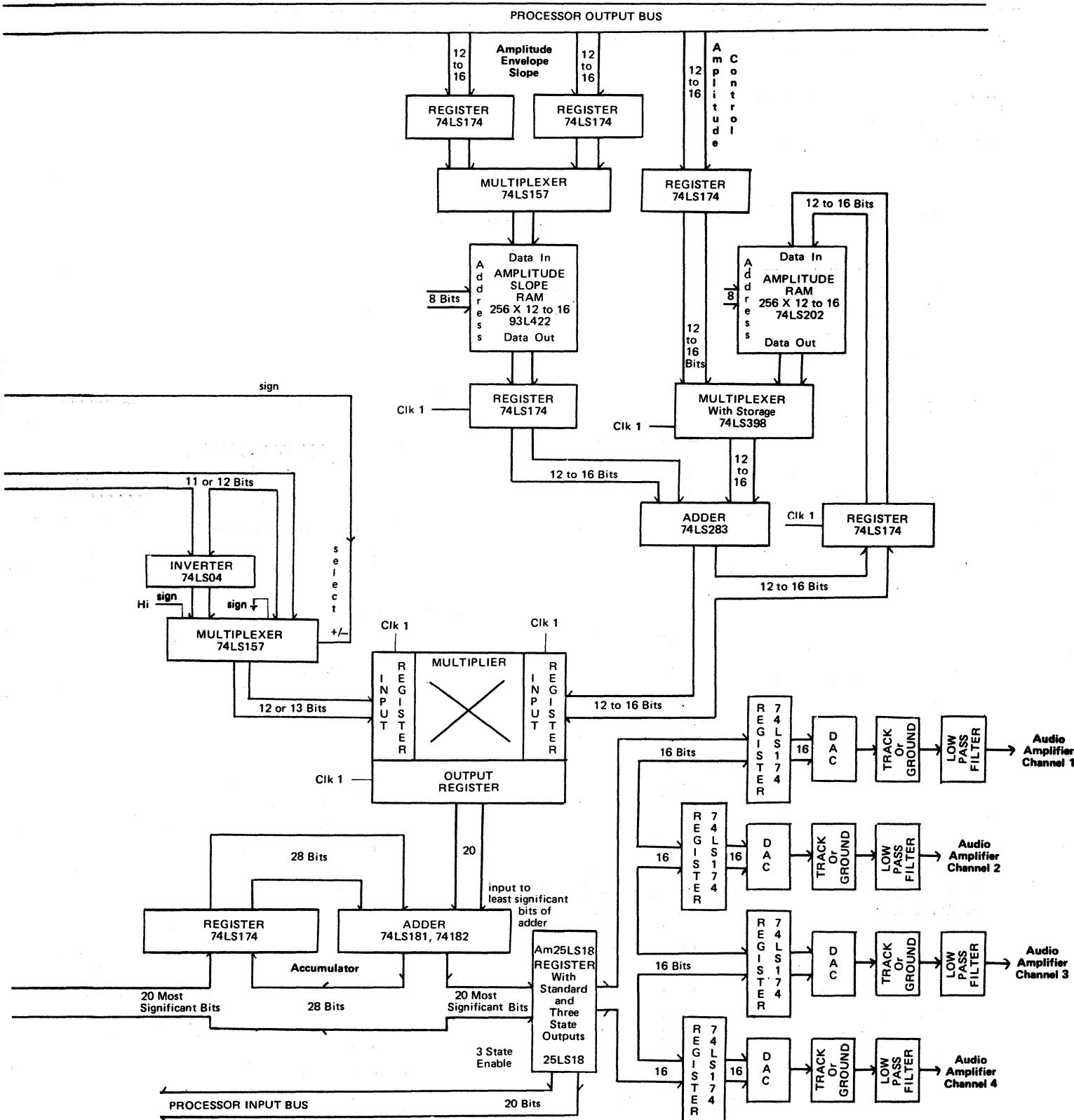


Figure 10 (continued)

dated by the processor. This is also true of the addresses of the phase angle increment RAM and the phase angle RAM. However the address of the frequency slope RAM provides a direct indication of which sinusoidal components are being worked on in the different stages of the oscillator. If the address of the frequency slope RAM is n , then the phase angle increment of the n^{th} sinusoidal component will be written into the phase angle increment RAM after 2 periods of Clk 1; its phase angle will be written into the phase angle RAM after 4 periods of Clk 1; the slope of its amplitude envelope will be read from the amplitude slope RAM after 4 periods of Clk 1; its amplitude will be written into the amplitude RAM after 6 periods of Clk 1; and the accumulator will add this n^{th} sinusoidal component to earlier sinusoidal components after 7 periods of Clk 1, as shown in Table 5. A delay line of 8 parallel output shift registers (74LS164) fed by the output of an 8 bit counter (74LS393) is used to generate addresses as well as to determine which sinusoidal components may have their frequency or amplitude changed by the processor. The 8 shift registers are clocked by Clk 1 along with the 8 bit counter. The 8 bit parallel outputs of the different stages of the shift register provide the addresses. The first stage 8 bit parallel output (n) of the 8 shift registers is used to address the frequency slope RAM. The fifth stage parallel output ($n-4$) of the shift registers is used to address the amplitude slope RAM. Other stages may be used for timing in other parts of the pipeline oscillator.

The amplitude RAM, phase angle increment RAM, and phase angle RAM are all addressed by a separate 8 bit counter. Since the amplitude address is continually scrambled, the address of the amplitude slope RAM may be used to determine which sinusoidal component may have its amplitude changed by the processor. When the processor wants to update the amplitude of the n^{th} sinusoidal component, it loads the new amplitude into the holding register (next to the amplitude RAM in Figure 10). The control logic compares n to the address of the amplitude slope RAM. When they are equal, the multiplexer (below the amplitude RAM in Figure 10) is switched to take in the new amplitude. The multiplexer includes a clocked output pipeline register which is clocked by Clk 1. The frequency (phase angle increment) is updated similarly, using the address of the frequency slope RAM to determine which sinusoidal component may have its frequency updated. With this method of determining the update addresses, control logic similar to that shown in Figure 9 for the slow inexpensive oscillator may be used.

Envelopes are generated by straight line segment approximations to the envelope curve. If infinitesimal line segments were used, any curve could be generated. The slope of the envelope at the beginning of a line

segment is loaded into the appropriate RAM (through its temporary holding register) by the processor. This slope is repeatedly added to the current amplitude (or frequency for frequency envelopes) until the slope of the envelope changes. Then a new envelope slope is loaded into the RAM. This is a simple-minded method of envelope generation; however, minimal extra hardware is required as compared to other methods of envelope generation of which I am aware. One of the problems with this envelope generation technique is that the software which updates the slopes must load the new slopes at the right times.

I would be interested in other techniques for envelope generation which allow a different envelope to be used for every individual note. A function table generates the same envelope shape for every note. A group of function tables may be multiplexed for more generality. Perhaps the simplest method of enveloping would be to use an input controller with pressure sensitivity so that envelopes could be generated by the musician in real time. Then the slope RAM's and corresponding logic would not be needed in Figure 10, thus reducing the cost by several hundred dollars.

Overview

It should be noted that this digital oscillator may be thought of as one module which may be plugged into a modular digital synthesizer which has other modules such as digital filters, digital reverberators, etc. F. Richard Moore's digital synthesizer is built around this modular concept [Moorer, 1977] as are most of the analog studio synthesizers. The digital oscillator may be broken up into simpler more general groups of components (on one or two buses) which could also be used in digital filters, spacial location movers, etc. This slows down the oscillator; however, it makes it more flexible. I would be interested in hearing about other digital synthesizers which explore these concepts more fully. Finally it should be noticed that a less expensive digital oscillator than the one shown in Figure 8 could be made if only one central RAM (as in F. Richard Moore's digital synthesizer) is used instead of a separate RAM for frequency and for amplitude.

Acknowledgements

I would especially like to thank Andy Moorer for his unending guidance and inspiration in digital/audio design concepts. I would also like to thank Jeff Goldstein, Dick Moore, Efrem Lipkin and P. Di Giunio for their willingness to openly discuss their digital synthesizer designs. It is hoped that more flexible high fidelity music instruments will

evolve with this open communication of ideas. A foundation for these designs was provided by *The Technology of Computer Music* by Max Mathews, et al. Thanks must go to Peter Samson who pioneered real time digital synthesizer design. And finally, I would like to thank Renny Wiggins, Maria Kent and John Strawn.

References

- Advanced Micro Devices, *Schottky and Low-Power Schottky Bipolar Memory, Logic and Interface*, (AMD, Sunnyvale, California, 1975).
- Backus, John, *The Acoustical Foundations of Music* (W. W. Norton, New York, 1969).
- Benade, Arthur, *Fundamentals of Musical Acoustics* (Oxford University, New York, 1976).
- Carlson, Bruce, *Communication Systems: An Introduction to Signals and Noise in Electrical Communication*, pp. 272-288 (McGraw-Hill, 1968).
- Chowning, John, "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation", *Journal of the Audio Engineering Society*, Vol. 21, No. 7, 1973, pp. 526-534 (reprinted in *Computer Music Journal*, Vol 1, No. 2, 1977).
- Fairchild, *Bipolar Memory Data Book* (Mountain View, California, 1976).
- von Helmholtz, H. L. F., *Die Lehre von der Tonempfindungen als physiologische Grundlage für die Theorie der Musik*, (1863). *On the Sensations of Tone as a Physiological Basis for the Theory of Music* (Dover, New York, 1954).
- Intel, *Data Catalog 1977* (Intel, Santa Clara, California, 1977).
- "Lexicon of Analyzed Tones", *Computer Music Journal*, Vol 1, No. 2, 1977.

This article was typeset by People's Computer Company and will appear in a forthcoming issue of The Computer Music Journal, edited by John Snell. CMJ is a bimonthly periodical dedicated to high-quality technical articles concerning computer-generated and computer-assisted electronic music systems. It is published by: People's Computer Company

*Box E
Menlo Park CA 94025
(415) 323-3111*

Mathews, Max, with the collaboration of Joan E. Miller, F. R. Moore, J. R. Pierce and J. C. Risset, *The Technology of Computer Music*, pp. 134-138 (MIT Press, 1969).

Moorer, James A., "The Use of the Phase Vocoder in Computer Music Applications", *Audio Engineering Society Preprint 1146 (E-1)*, 55th Convention, October 29 - November 1, 1976.

Moorer, James A., "Signal Processing Aspects of Computer Music - A Survey", *Computer Music Journal*, Vol 1, No. 1, (1977) pp. 4-37.

National Semiconductor, *TTL Data Book* (National Semiconductor, Santa Clara, California, 1976).

Pittman, Tom and Bob Davis, "A Variable Architecture Computing Machine", *Proceedings of the West Coast Computer Faire*, 1977.

Rakowski, A., "Pitch Discrimination at the Threshold of Hearing", *Proceedings of the Seventh International Congress on Acoustics*, Budapest, 1971, Vol. 3, pp. 373-376.

Signetics, *Data Manual*, (Signetics, Sunnyvale, California, 1976).

Snell, John, "High Speed Multiplication", *Computer Music Journal*, Vol. 1, No. 1, (1977) pp. 38-45.

Talambiras, Robert, "Some Considerations in the Design of Wide-Dynamic-Range Audio Digitizing Systems", *Audio Engineering Society Preprint 1226 A-1*, 56th Convention, May 10 - 13, 1977.

Texas Instruments, *Bipolar Microcomputer Components Data Book for Design Engineers* (TI, Dallas, 1977).

Texas Instruments, *The TTL Data Book for Design Engineers*, 2nd Edition (TI, Dallas, 1976).

Warnock, Richard, "Longitudinal Digital Recording of Audio", *Audio Engineering Society Preprint 1169(L-3)*, 55th Convention, October 29 - November 1, 1976.

THE KLUDGEHORN: AN EXPERIMENT IN HOMEBREW COMPUTER MUSIC

Carl Helmers, Editor in Chief
BYTE Publications Inc.
70 Main Street
Peterborough NH 03458

One of the most fascinating applications of the personal computers is electronic generation and digital control of music. This has been a fascination I have held personally since I first became acquainted with electronic music as prepared by Walter Carlos in 1968. It is my reason for wanting a personal computer and getting sufficiently involved in the complete knowledge of computer systems to design and build my own at a time when none were available as inexpensively as my scrounger's instincts required. Well, lately I finally got to the point where I was actually generating music, and this report, which I intend to sometime turn into a formal article in BYTE, summarizes the functional characteristics of what I've achieved in software with a simple hardware output peripheral.

I should note that this design is at best a transient one for myself, since I have just recently taken delivery on the first portions of a more comprehensive digital electronic music machine designed by Philip Tubb and his associates at ALF Products in Denver, Colorado.

The software of course will be modified and expanded in the future to take into account the 8 channels of programmable instruments in the ALF model AD-8, and my future integration of piano solo through the device of a converted player grand piano. But the characteristics of the Klugehorn system as it stands are such that with about 4K bytes or more of memory devoted to interpreter and music program data, it is possible to generate pleasing musical sounds on a peripheral that uses perhaps \$30 to \$40 worth of parts beyond the 8 bit port boundary. In this paper, I am limiting myself to the functional design of the Klugehorn, since time pressures prevented further detail in time to make the deadline for the Computer Faire proceedings.

Klugehorn Hardware

The hardware used by the Klugehorn is quite simple, both in concept and in implementation. It was wired up one weekend in January 1977 so that a demonstration of computer controlled music could be presented at the New England Computer Society meeting of January 12 1977. The major components are:

Programmable Pitch Source

This is a 9 bit presettable counter loaded from output bits of two 8223 ROM parts which are preprogrammed with a set of divide ratios (counts) which approximate the well tempered musical scale. Four address lines from the low order portion of an 8 bit output latch determine which note of the scale is being generated. The input frequency is in the 8 to 10 MHz range, and is controlled by a TTL gate oscillator with a variable capacitor for tuning. The output of the oscillator goes to a binary division series to generate 8 octaves from the basic signal. A one of eight data selector is used to pick the timing source with a 3 bit code from bits 4 to 6 of the pitch code. Bit 7, the high order bit, is used to gate the timing source on or off, but is unused in the Klugehorn implementation.

Waveform Generation

The waveforms are generated continuously using a 2519 shift register to store 40 amplitude values of 6 bits each. These values are passed to the Klugehorn by an 8 bit port with the low order 6 bits (Bits 0 to 5) containing the value to be loaded. The high order bit is used as a control line to determine whether the programmable pitch source or a clock pulse source originating in the PIA's control output line associated with the waveform port^{is used} in software, loading is accomplished by turning off the high order bit, then transferring 40 values to the waveform shift register by defining the value then programming a shift clock. When

the normal recirculating mode is reentered, the 6 bit outputs of the shift register are constantly being presented to a 6 bit MC1406 digital to analog converter, which uses the output of the envelope converter (see below) as its reference voltage.

Envelope Modulations

Musical sounds need an amplitude modulation envelope in order to sound well. In the Klugehorn, an envelope amplitude value is generated as an 8 bit quantity which is output to an 8 bit PIA port driving a digital to analog converter (MC1408) with a 5V zener diode reference input. As noted above this is used to create the reference voltage of the waveform digital to analog conversion, effectively producing a multiplication of the two devices. In order to obtain a true AC coupled signal output, an operational amplifier system is used to form the analog expression:

$$\text{OUTPUT} = E \times W - E/2$$

where:

OUTPUT = AC coupled signal fed to audio amp
W = six bit waveform value, 0 to 63 units
E = eight bit envelope value, 0 to 255 units
[over all output amplitude is adjusted so that the magnitude of the result is about 1V peak to peak over the range 0 (-1V) to 63x255=16065 (+1V).]

Envelope and Musical Timing

The music generation process is inherently a real time process. There must be a source of timing which can be used to determine what happens and how fast. The envelope data values must be updated at rates typically one value per millisecond, and notes within the music must be sequenced and measured in time. In order to accomplish this, the Klugehorn uses an interrupt driven structure based upon an existing 4800 Hz interrupt source described in my article in April 1977 BYTE. The interrupt was originally intended for a "Kansas City" standard tape interface output timing, but proved quite useful for music as well.

Fancy Displays

Finally, since one bit is left over on the waveform output port, I decided it would prove useful to occasionally toggle it as a trigger for looking at waveforms on the oscilloscope. This is described further in my discussion of the software which follows below.

Klugehorn Software

Listing 1 shows the complete source and assembly code of the present version of the Klugehorn software. The commenting of the assembly is less than optimal, due to my present limitations of 12K bytes of source program string area and a desire to keep the entire program in memory to avoid mucking around with linkages and multiple segment programs. The string area limits used by this source code were just 8 bytes shy of overflow of that region.

This software implements functionally a music interpreter which I shall define in some more detail as the remainder of this paper. The interpreter maintains two processes during the evolution of the music in time. The most important process is the actual music representation for one instrumental part as a function of time. The second process, which is not crucial (except in an artistic sense) is the "tempo process" which governs the tempo of the music as a function of time evolution within the piece of music.

Each process has a process control block defined by a set of equates at the beginning of the program listing. This process control block provides 4 levels of subroutine return linkage (0 offset is not equated due to an assembler fluke) so that music processes can economize on interpretive code through reentrant subroutines. The next event time values for the process are maintained by a two byte binary measure value, and a one byte value of the beat within measure.

The time signature for a whole piece is set by the number of beats allowed per measure, stored in the variable BMAX. The PCB also contains a note parameter, ONOT, and a length parameter OLNG used in setting default note and note length values during playing of music, a flag called FSUM which controls accumulation of length values, and to allow for graceful exit and return from the music interpreter, the origin of each process is stored in ORIG. The process control block is referenced throughout the interpreter's detailed execution by the pointer PROC which points to the current PCB. The current program counter value of the interpreter's current process is located at 0 offset from the start of the PCB.

The peripherals are referenced symbolically throughout the software by appropriate names defined in the equates at the beginning of the listing. WAWD and WAVC are the data and control address locations of the waveform output port PIA which defines the content of the shift register in the Klugehorn peripheral; ENVN and ENVC are the data and control address locations of the envelope output port PIA which sets the current envelope amplitude value; HORN is the 8 bit latched output which sets the frequency, and ATED and ATEC are the data and control addresses for the ASCISCOPE terminal I normally use to display the current measure and thus mark progress through a score.

Global control data for the program is defined in page 0 at locations 70 to 89 (hexadecimal) to take advantage of the 6800's short direct addressing mode.

Prior to the main executable code of the interpreter (which is pure code with one exception) there is a set of tables of control information which are assumed to be in programmable memory. These tables are a table of 64 subroutine address pointers (two bytes each), 16 note length register values (one byte each), 16 waveform definition pointers (two bytes each), and 16 envelope control blocks (eight bytes each, consisting of 4 pointers.) Also in the table area is a "waveform temporary" TWAV which is used by the waveform calculator package described below, as well as several miscellaneous temporary pointers.

The main entry point consists of stack loading at location 5800 followed by a branch to initialization beginning at location 5970. [All addresses mentioned in this paper are in hexadecimal notation, as is listing 1.] Initialization of PIA peripherals, the ACIA used for the terminal, and various data variables then occupies the program in a straight line sequence of code through address 59E9. The program finally falls through into the executive cycle at location 59FE where it continues looping, alternately executing the two processes (tempo and music) until the interpreter has found the end of job code which aborts execution and returns control to the monitor through location TCLO.

Interrupt Handling

The interpreter runs as a foreground program which is interrupted every 1/4800th second by the NMI interrupt source which is turned on at locations 59A6 to 59B3 as the last operation prior to entering the executive which alternately invokes the interpreter with each process. This interrupt is vectored to one of several locations, summarized as follows:

TMER: Interrupt kernel, always executed, which counts the real time clock value TMPO down to 0, and stops at 0. (As a measurement curiosity, after reaching zero, the variable LOST is incremented in order to compute the total time lost to deadlines not being met.)

ATKI: Attack interrupt processor. This interrupt becomes the NMI vector when it is decided to start a note. It begins cycling at the current envelope control block's first pointer, and follows forward until the second pointer value is reached. Set up by ANEW.

ANEW: New attack initialization. This interrupt is used to set up a new attack, saving the old envelope value to allow a smooth transition to the new attack.

SUSI: Sustain interrupt processor. This interrupt cycles envelope values from the second envelope control block pointer location to the third. For legatto notes, sustain keeps cycling; for staccato, it automatically

progresses into decay when the third envelope pointer value is reached.

DEC1: Decay interrupt processor. This interrupt cycles through the decay values (from the third envelope control pointer to the fourth.) To close out a note in either legatto or staccato mode.

All the interrupt routines ultimately conclude their execution with the TIMER processing. The envelope manipulation interrupts count down a value called ECNT which measures the number of 4800 Hz interrupts for which an envelope value is held before proceeding to the next value. In an adaptation of this program to a synthesizer with its own envelope processing, all the interrupt routines with the exception of TIMER could be thrown out, freeing up considerable processor time. In the worst case, with a 500 KHz processor clock and a 4800 Hz interrupt source, this interrupt processing uses over 90% of available time between interrupt deadlines. Fortunately this only happens occasionally. And few deadlines are missed due to sloth in the interrupt routines.

The Interpreter Design

The interpreter is a very simple program, in principle, which accomplishes amazing musical manipulation feats worthy of a true virtuoso. In designing an interpreter, the first thought is to figure out what the interpretive strings should accomplish, then begin to partition the space of 256 possible 8 bit operation codes into functions and meanings. For the music interpretation problem, one set of codes was already defined, the integer values from 0 to 127 decimal (00 to 7F hexadecimal) which control the pitch of the Klugehorn peripheral. These codes are used in a fairly convenient arrangement that partitions on the boundaries of hexadecimal digits. The high order digit, with values 0 to 7 specifies the three bit octave code for the pitch generator; the low order digit with values 0 to B specifies the note within the octave. Arbitrarily assuming that the pitch divider's oscillator is tuned to assign a value of a musical pitch "C" to note 0, the following is the range of values available:

hexa-	0	C
	1	C# (sharp)
	2	D
	3	D#
decimal	4	E
	5	F
	6	F#
	7	G
	8	G#
	9	A
	A	A#
	B	B

Thus a hexadecimal code of 47 is a note G in octave 4.

Then, in order to have a one byte code to specify a note length stored in a table of note length values LGNT, I decided to use the operation codes 80 to 8F to pick one of 16 values.

The remaining functions of the interpreter are coded as an operation code byte followed by additional bytes if needed to carry out the function in question.

The overall definitions are summarized in a later section of this paper, and are put into very cursory form in the function definition table found on the next to last page of the listing.

The interpreter proper begins at location 5A89 in the assembly of listing 1. Whenever invoked by the executive, it checks to see if it should execute some interpretive commands. It always executes commands (in a dummy mode) if the process is in the middle of a subroutine definition according to SDFG; it will wait until current time is greater than the next event time otherwise. Return is immediate if these conditions are not satisfied. Otherwise, the EXEC label is reached and commands can be executed.

Execution of commands begins by fetching the current operation code. This is done by loading the process control block pointer PROC, loading the current music program counter at 0

offset from the location in PROC, then loading the A accumulator with the byte pointed to by the current music program counter.

IF the number is less than hexadecimal 80, the NOTE procedure is executed; if the number is 80 to 8F, then the length register reference procedure is executed; otherwise the range is checked and either a function is executed, the interpreter is terminated with an FF operation code, or the command is ignored if in the range B2 to FE.

Miscellaneous functions are all listed in the function definition table at the end of the assembly, and are described in greater detail in the following section of this paper.

Klugehorn Interpreter, Functional Specification of Operations

In the listing which follows, the symbolic names are the symbols which begin the execution of the corresponding function within the interpreter, and addresses are the locations of the functions within listing 1. In preparing musical scores using my assembler, I use these same symbols equated to the operation code values, so that a series of "form constant byte" (FCB) assembly pseudo operations can be used to lay down symbolic interpretive text. The format of the command is shown at the right of each entry.

WDEF, 5BC6 900X[...40 values...]

The WDEF command defines waveform X. This is done by computing an offset into the waveform pointer table WFOR, then storing a pointer to the first of the 40 values at that WFOR table entry. The interpreter skips around the values and the next command interpreted follows the last value.

WSET, 5E4A 910X

The WSET command moves waveform X into the shift register of the Klugehorn peripheral. This operation should only be done at a time when the envelope output is 0, if annoying transients are to be avoided.

TMPW, 5DDE 920X

TMPW moves waveform X into the temporary register used for waveform computations. All 40 values of the waveform are moved in an array operation.

WTMP 5DFB 93

WTMP moves the temporary waveform register into the Klugehorn peripheral's shift register, in a manner analogous to the WSET operation. As in the WSET operation, this should only be done when the envelope outputs are 0 after a decay. This operation is typically done after some calculations.

DIVW 5E10 94

Divide every element of the temporary waveform register by 2 with a shift right operation.

MULW 5E1D 95

Multiply every element of the temporary waveform register by 2 with a shift left operation.

ADDW 5E32 960X

Add every element of waveform X to the corresponding element of the temporary waveform register.

WTMP operations it is possible to do limited fourier composition operations with basis waveforms stored via WDEF as the 40 step approximations of a harmonic series. By defining different basis functions, different composition experiments can be performed as well (eg: Walsh functions, etc.)

SNOT 5D6D 97XX

Set the old note parameter value to XX. This operation is used to set up a parameter for manipulation by subroutines defined to do trills, runs, etc.

INCN 5CA8 98

Increment the old note parameter value.

IPLA 5CAF 99

Increment, then play the old note parameter value. The play operation is identical to that used for an ordinary note (code 00 to 7F) and consists of changing pitch, and starting an attack if in staccato mode. The time duration of the note is the old note length value.

DECN 5CD9 9A

Decrement the old note parameter value.

DPLA 5CDE 9B

Decrement, then play the old note parameter value, using the old length parameter value for duration.

WAIT 5AE4 9C

Wait until the present time plus the old length parameter value. (Adds to the "beat" field modulo BMAX).

WMEA 5D04 9DXXXX

Wait until the present time is greater than measure XXXX, beat 0.

LSET 5C79 9E0XXX

Set the value of length register X to LL. This instruction is used to set up a set of lengths consistent with the time signature of the music. An almost universally usable time signature value of 240 micro beats per measure easily accommodates 3/4, 3/8, 6/8, 4/4, 2/4, 2/2 and other common time signatures through appropriate choices of length register values.

OSCL 5D9D 9F

Trigger the oscilloscope. Toggle the bit 6 line of the waveform output PIA port. By using this line to trigger the oscilloscope it is possible, whether for testing or as part of the music display, to synchronize the scope with respect to programmed operations such as envelope attack or decay initiation, or pitch changes.

BEGS 5C34 A0XX...

Begin subroutine definition. This operation sets the subroutine definition flag (reset by ENDS to end definition) and marks the beginning of the subroutine with a pointer in the subroutine definition table offset calculated from XX. XX is masked to 6 bits, allowing a total of 64 subroutines to be defined, referenced by number. Subroutines can be redefined by subsequent BEGS referencing the same number, so it is possible to accomplish variations on a theme by defining several low level routines which are later redefined prior to subsequent executions.

ENDS 5C53

A1

End subroutine, or return. The interpretation depends upon whether or not the subroutine code was reached during inline execution following a BEGS, or from a CALL. If this is a definition, ENDS has no effect other than to turn off the subroutine definition flag, allowing resumption of full interpretation of code. If this was subroutine execution, then ENDS is a return operation which pops the stack and resumes execution following the CALL which invoked execution.

CALL 5B5B

A2XX

Call subroutine XX. XX is masked to 6 bits allowing 64 possible subroutines. The subroutine definition executed is the last definition for the number XX. CALLs may be nested to three levels below the current music program counter; if the stack is full, CALL is a no-operation which is ignored.

EDEF 5BF4

A30XDDSSAA.....

Envelope definition. This operation for the Klugehorn sets up the envelope control block pointers for an envelope definition string which remains inline in the interpretive code. The envelope identification is given by X, and the values DD,SS and AA specify (each) up to 255 values of decay, sustain, and attack envelope function each, in that order. Thus to specify an exponential attack function over 20 steps, AA would be 20, and the last 20 values following the envelope command would be devoted to the attack waveform; similarly, the first DD values are the decay waveform, the second SS values are the sustain waveform.

ESET 5D1C

A40X

Envelope selection. Set the current envelope control block values equal to the pointers stored for envelope X.

ERAT 5D42

A5XX

Envelope evolution rate. Set the envelope evolution rate equal to XX. Values of practical interest range from 3 to 30 hexadecimal; these values give the number of 4800 Hz interrupts per envelope step, with a minimum of 3.

STAC 5D4A

A6

Set the envelope evolution processing of the interrupt routines to a mode which cycles through the entire envelope once, then quits.

LEGA 5D55

A7

Set the envelope evolution evolution processing of the interrupt routines to the legato mode. In this mode, the ATTK command is used to start a new envelope, and the DECA command is used to end an envelope. After starting, passing through attack and reaching the sustain portion of the envelope, the envelope processor continues to cycle through the sustain portion until decay is initiated explicitly; this allows programming of amplitude vibrato effects built into the sustain envelope definition.

ATTK 5D61

A8

Begin an attack in legato mode. If the previous state of the envelope was non-null, the envelope amplitude value is held until the attack waveform's amplitude exceeds the old amplitude value, thus allowing a relatively smooth (but vibratoless) transition between note envelopes.

DECA 5D4F

A9

Begin decay in legato mode. This is used to explicitly command end of sustain and beginning of decay when in the legato mode. (Stacatto mode does this automatically.) Typically, when in legato mode, several notes or a passage of notes might be done, with occasional attacks to emphasize

notes, and a final decay when a rest is required.

TEMP 5CE8

AAXX

Set the tempo rate to XX. For music with 240 beats per measure, with a 3/4 or 6/8 time signature, useful tempo values range from 24 (fast) to 60 (very slow) in decimal notation. Tempo counts the number of 4800 Hz interrupts assigned to each beat of the measure. Tempo is normally only manipulated in the tempo process.

TINC 5CF0

AB

Increment the tempo rate. This adds one to the tempo rate slowing down the music. Since all timing is tied to the 24 bit measure/beat current time value, by changing the number of 4800 Hz interrupts per beat, tempo is altered.

TDEC 5CFA

AC

Decrement the tempo rate. This speeds up the music by decreasing the number of 4800 Hz interrupts counted per musical beat in the measure.

SIGN 5E5E

ADXX

Set the number of beats per measure. This in effect sets the time signature of the music. The most useful number for practical music encoding is 240 beats per measure, the default setting at initialization. This number can be evenly divided by many multiples of 2 and 3. Since each note length has to be assigned by some fraction of the beats per measure, to 8 bit precision, picking a number with many integer submultiples is the ideal case.

LOOP 5E66

AE

This operation is used to cause the music to endlessly cycle from its origin. This can be an interesting component of a multi part extension of the interpreter when programming musical phenomena called canons or rounds, where each instrumental part has a different (but harmonically consistent) number of measures before its LOOP point.

PLAY 5AED

AF

This operation uses the old note parameter value to simply play a note and do nothing else. It was added late in the design when INCN and IPLA proved an incomplete set of manipulative mechanisms for melody subroutines.

GOTO 5E6F

BOXXXX

This operation was added to allow segmented music programs. In particular, when I found a common set of initialization (14 waveforms of 40 steps and several envelopes plus all the length definitions for a time signature) I found it tedious to re-assemble the music text for these initializations every time I changed one melody entry. By using a separate assembly and sticking the initializations in an isolated section of memory with access and return via GOTO, I eliminated this requirement of reassembly of the initialization texts which were much longer than the actual program (at least during testing.)

Generating Musical Texts

The actual process of generating musical texts uses the same text editor and assembler which was used to assemble the interpreter itself. The process of using a symbolic assembler to generate the texts is cumbersome without a macro facility, and would only be slightly improved by a macro facility, since assembly is required before testing and symbolic patching is not possible. In the next version of the interpreter, I intend to incorporate an extensible symbolic editor running directly off the hexadecimal interpretive code, to help generate musical texts much more efficiently.

Generating Process Control Blocks

The interpreter assumes a process control block region will exist at the start of the music text area at location 2000. This region in the present assembly has two entries: the music process which sequences notes has its PCB at location 2000, and the tempo process which puts in tempo variations with time (rubato) has its PCB at location 2010. With the one pass assembler, symbols are oriented and space reserved following an ORG at location 2000, but no initial music program counter entries are made at 2000 and 2010 until the end of the assembly when ORG statements return to these locations to patch in the actual values found. The default null initialization of the assembler's output space covers the entire contents of the PCB's with the exception of the initial process program counter values determined during assembly.

Generating Music Commands

With the simple "Tiny Assembler" used for this software, no macro facilities are available. Thus it is impossible to define operation codes directly, and creating codes for the interpreter must use existing facilities. This is done with the 6800 assembler's FCB instruction, or "form constant byte" and a series of equates for the various interpreter op codes. A typical note sequence might consist of:

```
FCB    ATTK,T4,$47,$49,$40,$42,T16,T8,$44,T16,$45,T4  
FCB    DECA,WAIT
```

Here, I initiate attack, refer to the length register for a duration of one quarter note (T4), reference a series of notes in octave/note within octave notation as hexadecimal constants ("\$" prefix), accumulate a "dotted eighth note value" using an implicit addition of successive note length references (here T16,T8), play the note \$44 with dotted eighth note duration, then play a sixteenth note, start the decay while waiting one quarter note's duration. All the symbols ATTK,DECA,WAIT,T4, T16,T8, etc have been assumed here to be equated to the one byte hexadecimal values of the corresponding interpreter codes described above.

This method of assembly is crude, but it does allow use of a text editor, and is moderately symbolic. Examples of the Klugehorn in operation which I expect to be playing at the BYTE booth at the Computer Faire were coded using this technique, with the interpreter program as described in this paper.

Conclusions

What I have demonstrated in this paper is the software design needed to create some interesting (but by no means perfect) musical representations in a personal computer. The actual interpreter occupies some 2000 bytes of code, and can thus be run on any 6800 system with about 4K of programmable memory or more, after appropriate relocation. Since the program was assembled at a fairly "nice" boundary, it should prove fairly easy to relocate by modifying constants in all the JMP and JSR subroutines employed. Several monitor references to my system will have to be changed for use in other systems.

```

LOCN B1 B2 B2
5800 > ORG $5800
5800 >* K L U G E H O R N
5800 >> SINGLE CHANNEL MUSIC SYNTHESIZER DRIVER PROGRAM
5800 >> (C) 1977 BY CARL HELMERS, BYTE PUBLICATIONS INC.
5800 >> ASSEMBLY OF 2/27/77 @ 12:10
5800 >> TAPE 200A LOC 281
0154 >WAUD EQU $154 WAVE PIA DATA
0155 >WAVC EQU $155 WAVE PIA CTRL
0156 >ENVUD EQU $156 ENV PIA DATA
0157 >ENVC EQU $157 ENV PIA CTRL
0144 >HORN EQU $144 PITCH LATCH
0138 >MILL EQU $138 1 MS DELAY
0158 >ATEC EQU $158 TERM ACIA CTRL
0159 >ATED EQU $159 TERM ACIA DATA
0004 >NMIV EQU $4 PROGRAMMABLE NMV PTR
0161 >TIMG EQU $161 TURN ON 4800HZ NMV
0163 >TIMS EQU $163 TURN OFF NMV
1070 >AA2X EQU $1070 X := X+A
0070 >MEAS EQU $70 CUR MEASURE
0072 >BEAT EQU $72 CUR BEAT
0073 >ENVX EQU $73 ENV PTR
0075 >SULK EQU $75 ATTACK END NMV PTR
5800 >> #DEC1=STACCATO, #SUS1=SUSTAIN
0077 >ECNT EQU $77 ENV COUNTER
0078 >RATE EQU $78 ENV RATE
0079 >ENV EQU $79 OLD ENV VALUE
007A >DECFL EQU $7A DECAY FLAG
007B >TMPI EQU $7B CUR TEMPO RATE
007C >BMAX EQU $7C BEATS PER MEASURE
007D >SDFG EQU $7D SUBRTN DEF FLAG
007E >PROC EQU $7E CUR PROCESS PTR
0080 >ENDF EQU $80 END EXEC FLAG, $FF = CONTINUE
0081 >SAVX EQU $81 X SAVE(NEW PC VALUE)
0083 >IMPO EQU $83 TEMPO CNTR
0084 >SAVY EQU $84 X SAVE
0086 >LOST EQU $86 TIME LOST, OVER RUN
0087 >TLST EQU $87 ACCUM TIME LOST
0089 >PPTR EQU $89 PRINT LINE PTR
5800 >*
5800 >> PROCESS CONTROL BLOCK OFFSETS
0002 >STK1 EQU 2 OLD PC
0004 >STK2 EQU 4 OLDER PC
0006 >STK3 EQU 6 OLDEST PC
0008 >NXMS EQU 8 NEXT MEASURE
000A >NGBT EQU 10 NEXT BEAT
000B >ONOT EQU 11 OLD NOTE
000C >OLNG EQU 12 OLD LENGTH
000D >FSUM EQU 13 LENGTH SUM/CLEAR FLAG

000D >ORIG EQU 14 SAVED ORIGIN
5800 >*
5800 RE 2F FF >ENTRY LDS #$2FFF
5803 7E 00 00 > JMP GOGO
5806 >*
5806 >> TABLES
5806 >*
5806 >> SUBRTN PTRS
5806 >SRTN EQU *
5886 > ORG **128
5886 >*
5886 >> LENGTHS
5886 >LGNT EQU *
5896 > ORG **16
5896 >*
5896 >> WAVE PTRS
5896 >WFOR EQU *
5886 > ORG **32
5886 >*
5886 >> ENV PTRS
5886 >ETAB EQU *
5936 > ORG **128
5936 >*
5936 >> ENV CONTROL BLOCK
5936 R0 R0 >DUME FIB $8080
593 R0 00 > FDB $8000
593A >EPTR EQU *
593A 59 36 >PATH FDB DUME
593C 59 37 >SUST FDB DUME+1
593E 59 38 >DECY FDB DUME+2
5940 59 39 >ENND FDB DUME+3
5942 >*
5942 >* TEMP WAVE
5942 >TWAV EQU *
596A > ORG **40
596A 00 00 >WPWT1 FDB 0
596C 00 00 >WPWT2 FDB 0
596E 00 00 >WPGM FDB 0 OPER. SUBRTN.
5970 >*
5970 >> MAIN
5970 >GOGO EQU *
5904 59 70 >*
5970 >> INITIALIZE
5970 7F 01 5R > CLR ATEC
5973 R6 03 > LDAA #3
5975 B7 01 58 > STAA ATEC
5978 R6 01 > LDAA #1
597A B7 01 58 > STAA ATEC
> CLR ENVC
597D 7F 01 57
5980 7F 01 56 > CLR ENVUD
5983 73 01 56 > COM ENVUD
5986 86 04 > LDAA #4
5988 B7 01 57 > STAA ENVC
598B 7F 01 56 > CLR ENVUD
598E 7F 01 55 > CLR WAVC
5991 7F 01 54 > CLR WAVD
5994 73 01 54 > COM WAVD
5997 86 3C > LDAA #$3C LOAD CLOCK ONE
5999 B7 01 55 > STAA WAVC
599C R6 80 > LDAA #$80 RECIRCULATE
599E B7 01 54 > STAA WAVD
59A1 R6 3F > LDAA #$3F INIT NOTE
59A3 B7 01 44 > STAA HORN

5946 >* ACTIVATE 4800 Hz INTERRUPT
5946 CE 00 00 > LDX #TMR
5946 DF 04 > STX NMIV
5948 FE 59 3A > LDX PATH
594E DF 73 > STX ENVX
5980 7F 01 61 > CLR TIMG
5983 >*
5983 >> BEGIN INTERPRETER
59R3 >> DEFAULT SETTINGS
5983 7F 00 89 > CLR PPTR
5986 7F 00 79 > CLR OENV
5989 86 FF > LDAA #$FF
598B 97 7D > STAA SDFG NOT SUBROUTINE
598D 97 7A > STAA DECF ENV SUSTAINED
598F CE 00 00 > LDX #SUS1
59C2 DF 75 > STX SULK LEGATO
59C4 R6 0A > LDAA #10
59C6 97 78 > STAA RATE OF ENV
59C8 R6 1E > LDAA #30
59CA 97 7B > STAA TMPI TEMPO
59CC CE 00 00 > LDX #0
59CF DF 87 > STX TLST ZERO TIME LOST
59D1 DF 70 > STX MEAS TIME ORIGIN ZERO
59D3 7F 00 86 > CLR LOST
59D6 7F 00 72 > CLR BEAT
59D9 86 FO > LDAA #240 DIVISIBLE BY 4,6,8,ETC
59DB 97 7C > STAA BMAX
59DD 86 FF > LDAA #$FF
59DF 97 80 > STAA ENDF
59E1 CE 00 00 > LDX #PRTA
59E4 R6 00 > BSR SVST
59E6 CE 00 10 > LDX #PRTA+16
59E9 R6 00 > BSR SVST
59EB 20 00 > BRA GOMU
59ED >*
59ED >> SAVE ORIGIN
59ED A6 00 > SVST LDAA 0,X
59E5 07 >*
59EA 02 >*
59EF A7 0E > STAA ORIG,X
59F1 A6 01 > LDAA 1,X
59F3 A7 0F > STAA ORIG+1,X
59F5 86 0C > LDAA #12
59F7 6F 02 > SVSL CLR 2,X
59F9 OR > INX
59FA 4A > DECA
59FB 26 FA > BNE SVSL
59FD 39 > RTS
59FE >*
59FE >> EXECUTIVE CYCLES PROCESSES
59FE CE 00 00 > GOMU LDX #PRTA KLUGEHORN PROCESS PCB
59EC 11 >*
5A01 BD 00 00 > JSR INTR
5A04 BD 00 > BSR INWA
5A06 CE 00 10 > LDX #PRTA+16 TEMPO PROCESS PCB
5A09 BD 00 00 > JSR INTR
5A0C BD 00 > BSR INWA
5A0E 7D 00 80 > TST ENDF
5A11 26 EB > BNE GOMU ARE WE DONE?
5A13 7E 00 00 > JMP TCLO
5A16 >*
5A16 >> WAIT FOR INTERRUPT DONE
5A16 7D 00 83 >INWA TST TMP0
5A05 10 >*
5A0D OR >*
5A19 26 FB > BNE INWA
5A1B 96 86 > LDAA LOST
5A1D 27 01 > BEQ **3
5A1F 4A > DECA
5A20 7F 00 86 > CLR LOST
5A23 DE 87 > LDX TLST
5A25 BD 10 70 > JSR AA2X
5A28 DF 87 > STX TLST
5A2A 96 7B > LDAA TMPI
5A2C 97 83 > STAA TMPO
5A2E 7C 00 72 > INC BEAT
5A31 96 72 > LDAA BEAT
5A33 91 7C > CMPA BMAX
5A35 26 00 > BNE INWI
5A37 DE 70 > LDX MEAS
5A39 08 > INX
5A3A DF 70 > STX MEAS
5A3C >*
5A3C >> PRINT MEASURE SETUP
5A3C 96 71 > LDAA MEAS+1
5A3E 5F > CLRB
5A3F M1 0A > DDND CMPA #10
5A41 25 00 > BCS DDND
5A43 R0 0A > SUBA #10
5A45 5C > INCB
5A46 20 F7 > BRA DNDD
5A48 C1 0A > DDND CMPB #10
5A42 05 >*
5A4A 25 01 > BCS **2+1
5A4C 5F > CLRB
5A4D CB 30 > ADDB #$30
5A4F P7 00 00 > STAB PPN1
5A52 R8 30 > ADDA #$30
5A54 P7 00 00 > STAA PPN2
5A57 R6 09 > LDAA #9
5A59 97 89 > STAA PPTR
5A57 7F 00 72 > CLR BEAT
5A5E B6 01 58 >INWI LDAA ATEC
5A36 27 >*
5A61 16 > TAB
5A62 R4 01 > ANDA #1
5A64 27 03 > BEQ **5
5A66 7E 00 00 > JMP TCLO
5A69 >*
5A69 >> CHECK FOR MEASURE PRINTOUT
5A69 C4 02 > ANDB #2
5A6B 27 00 > BEQ NPRT

```

SA6D 9E 89 > LDAA PPTR
 SA6F 27 00 > BEQ NPPR
 SA71 CE 00 00 > LDX #PLIN
 SA74 BD 10 70 > JSR AA2X
 SA77 A6 00 > LDAA 0,X
 SA79 B7 01 59 > STAA ATED
 SA7C 7A 00 89 > DEC PPTR
 SA7F 39 >NPPR RTS
 SA6C 12
 SA70 0E
 SA7F >PLIN EQU #-1
 SA72 5A 7F
 SA80 0A 0D > FCB \$A,SD
 SA82 30 >PPN2 FCC 1,0
 SA55 5A 82
 SA83 30 >PPN1 FCC 1,0
 SA50 5A 83
 SA84 3D 53 41 > FCC 3,=SA
 SA87 45 4D > FCC 2,EM
 SA89 **
 SA89 ** INTERPRETER EXECUTION
 SA89 DF 7E >INTR STX PROC SAVE PCR PTR
 SA02 5A 89
 SA0A 5A 89
 SA88 7D 00 7D > TST SDFG
 SA8E 27 00 > BEQ EXEC IF SUBRTN LEF THEN DUMMY EXEC
 SA90 **
 SA90 ** IS NEXT EVENT TIME PAST?
 SA90 9E 71 > LDAA MEAS+1
 SA92 A0 09 > SUBA NXMS+1,X
 SA94 9E 70 > LDAA MEAS
 > SECA NXMS,X
 SA96 A2 08
 SA98 25 00 > BCS NOAK CURRENT < NEXT (IGNORE BEAT)
 SA9A EE 08 > LDX NXMS,X
 SA9C 9C 70 > CPX MEAS
 SA9E 26 00 > BNE EXEC CURRENT > NEXT (IGNORE BEAT)
 SAAD 0E 7E > LDX PROC
 SA82 96 72 > LDAA BEAT
 SA44 A1 0A > CMPA NXBT,X
 SA86 24 00 > BCC EXEC CURRENT >= NEXT
 SAAR 39 >NOAK RTS
 SA99 0E
 SA99 **
 SA99 ** EXECUTE COMMANDS
 SA99 DE 7E >EXEC LEX PROC
 SARF 19
 SA9F 09
 SAAT 01
 SAAR EE 00 > LDX 0,X
 SAAD A6 00 > LDAA 0,X
 SAAF DE 7E > LDX PROC
 SABI 81 80 > CMPA #S80 NOTE?
 SAB3 25 00 > BCS NOTE
 SAP5 R1 90 > CMPA #S90 LENGTH REF?
 SAP7 25 00 > BCS LEN9
 SAP9 81 B1 > CMPA #S81 FUNC?
 SAPB 25 00 > BCS FUNC
 SABD 81 FF > CMPA #SFF END MARK?
 SAPF 26 00 > BNE EXE1
 SAC1 7F 00 80 > CLR ENDF
 SAC4 39 > RTS
 SAC5 EE 00 >EXE1 LDX 0,X
 SACO 04
 SAC7 08 > INX
 SACR DF 81 > STX SAVX
 SACA DE 7E >EVEN LDX PROC
 SACC 96 81 > LDAA SAVX
 SACE A7 00 > STAA 0,X
 SADE 96 R2 > LDAA SAVX+1
 SAD2 A7 01 > STAA 1,X
 SAD4 20 D3 > BRA EXEC
 SAD6 7E 00 00 >LENG JMP XLNG
 SAR8 1D
 SAD9 7E 00 00 >FUNC JMP XFUN
 SARC 1C
 SADC **
 SADC ** WAIT INC
 SADC DE 81 >WATI LDX SAVX
 SADE 0R > INX
 SADF DF 81 > STX SAVX
 SAE1 DE 7E > LDX PPROC
 SAE3 39 > RTS
 SAE4 **
 SAE4 ** WAIT
 SAE4 DE 7E >WAIT LDX PROC
 SAE6 7D 00 7D > TST SDFG
 > BEQ EXE1
 SAE9 27 DA
 SAEB 20 00 > BRA WAIX
 SAED **
 SAED ** PLAY OLD NOTE
 SAED DE 7E >PLAY LDX PROC
 SAEF 7D 00 7D > TST SDFG
 SAF2 27 D1 > BEQ EXE1
 SAF4 A6 0B > LDAA ONOT,X
 SAF6 BD 00 00 > JSR KNOT
 SAF9 20 00 > BRA WAIX
 SAFB **
 SAFB ** NOTE
 SAFF 7D 00 7D >NOTE TST SDFG
 SAR4 46
 SAFE 27 C5 > BEQ EXE1
 SR00 8D 00 >NOT1 BSR KNOT
 SB02 **
 SB02 ** NEXT EVENT TIME := CURRENT TIME + LENGTH
 SB02 DE 70 >WAIX LDX MEAS
 SAEC 15
 SAFA 07
 SB04 DF 81 > STX SAVX
 SB06 DE 7E > LDX PROC
 SB08 A6 0C > LDAA OLNG,X
 SPOA 5F > CLR8

SB0B 9B 72 > ADDA BEAT
 SRO1 C9 00 > ADCB #0
 SROF 26 00 > BNE WADJ
 SFI1 A7 0A >WAX1 STAA NXBT,X
 SRI3 91 7C > CMPA BMX
 SBI5 25 00 > BCS WAX2
 SBI7 90 7C > SUBA BMX
 SBI9 A7 0A > STAA NXBT,X
 SBI8 8D BF > BSR WATI
 SBI0 96 R1 >WAX2 LDAA SAVX
 SBI6 06
 SBI1 A7 08 > STAA NXMS,X
 SBI2 96 R2 > LDAA SAVX+1
 SBI3 A7 09 > STAA NXMS+1,X
 SBI5 DE 7E > LDX PROC
 SBI7 EE 00 > LDX 0,X
 SBI9 0R > INX
 SBI2 DF 81 > STX SAVX
 SBI2 DE 7E >GORK LDX PROC
 SBI2 96 R1 > LDAA SAVX
 SBI3 A7 00 > STAA 0,X
 SBI2 96 R2 > LDAA SAVX+1
 SBI4 A7 01 > STAA 1,X
 SBI6 6F OD > CLR FSUM,X
 SBI8 39 > RTS
 SBI9 8D A1 >WADJ BSR WATI
 SBI0 2R
 SBI3 90 7C > SUBA BMX
 > SBCB #0
 SBI3 C2 00
 SBI3 26 FR > RNE WADJ
 SBI1 20 CE > BRA WAX1
 SBI3 >KNOT EQU *
 SBI1 41
 SBI7 5B 43
 SBI3 B7 01 44 >KNT1 STAA HORN
 SBI4 DE 7E > LDX PROC
 SBI4 A7 0B > STAA ONOT,X
 SBI4 DE 75 > LDX SULK
 SBI4 RC 00 00 > CPX #DEC1
 SBI4 27 00 > BEQ NGAT
 SBI5 39 > RTS
 SBI2 CE 00 00 >NGAT LDX #ANEW
 SBI0 01
 SBI5 DF 04 > STX NMIV
 SBI7 39 > RTS
 SBI8 7E 5A CA >EXEA JMP EWEN
 SBI8 **
 SBI8 ** MUSIC SUBROUTINE CALL
 SBI5 BD 00 00 >CALL JSR OHED
 SBI5 7D 00 7D > TST SDFG
 SBI6 27 F5 > BEQ EXE4 SKIP EXEC IF IN DEF
 SBI6 6D 06 > LDX PROC
 SBI6 6D 06 > TST STK3,X
 SBI6 26 EF > BNE EXE4 SKIP EXEC IF STACK FULL
 SBI6 36 > PSHA
 SBI6 C6 06 > LDAB #6
 SBI6 A6 05 >XSLP LDAA 5,X
 SBI6 A7 07 > STAA 7,X
 SBI7 09 > DEX
 SBI7 5A > DECB
 SBI7 26 FR > BNE XSLP
 SBI7 32 > PULA
 SBI7 R4 3F > ANDA #\$3F
 SBI7 4R > ASLA
 SBI7 CE 5R 06 > LDX #SRTN
 SBI7 BD 10 70 > JSR AA2X
 SBI7 EE 00 > LDX 0,X
 SBI0 DF R1 > STX SAVX
 SBI2 7E 5A CA > JMP EWEN
 SBI5 **
 SBI5 ** SET LENGTH FROM TABLE
 SBI5 7D 00 7D >XLNG TST SDFG
 SAD7 5B 85
 SBR8 27 00 > BEQ EXE3
 SBR8 EE 00 > LDX 0,X
 SBR8 A6 00 > LDAA 0,X
 SBR8 R4 0F > ANDA #SF
 SBR9 CE 58 86 > LDX #LGNT
 SBR9 BD 10 70 > JSR AA2X
 SBR6 A6 00 > LDAA 0,X
 > LDX PROC
 SBR9 DE 7E
 SBR9 6D 0D > TST FSUM,X
 SBR9 27 00 > BEQ XSTL
 SBR9 AB 0C > ADDA DLNG,X
 SBR9 A7 0C > STAA DLNG,X
 SBR9 20 00 > BRA LENI
 SBR4 4D >XSTL TSTA
 SBR9 06
 SBR5 27 00 > BEQ LENI IF NULL KEEP OLD LENGTH
 SBR7 A7 0C > STAA DLNG,X
 SBR9 6C 0D >LENI INC FSUM,X
 SBR6 02
 SBR3 05
 SBR8 7E 00 00 >EXE3 JMP OBYT
 SBR9 21
 SBR8 **
 SBR8 ** FUNC EXEC:
 SBR8 ** LOOK UP ADDR THEN GO TO FUNC
 SBR8 R0 90 >XFUN SURB #S90
 SADA 5B AE
 SBR0 4R > ASLA
 SBR1 CE 00 00 > LDX #FTAB
 SBR4 FD 10 70 > JSR AA2X
 SBR7 EE 00 > LDX 0,X
 SBR9 6E 00 > JMP 0,X
 SBR8 **
 SBR8 ** 2 BYTE OPCODE OVERHEAD
 SBR8 DE 7E >OHED LDX PROC
 SBR8 5B BB
 SBR8 EE 00 > LDX 0,X
 SBR8 0R > INX

```

5BC0 A6 00 > LDAA 0,X
5BC2 08 > INX
5BC3 DF R1 > STX SAVX
5PC5 39 > RTS
5BC6 >*
5BC6 > WAVEFORM DEF
5RC6 >WDEF EQU *
5BC6 RD F3 > BSR OHED
5BCR R4 01 > ANDA #SF
5BCA 46 > ASLA
5BCR CE 58 96 > LDX #WFOR
5BCF BD 10 70 > JSR AA2X
5BD1 96 R1 > LDAA SAVX
5BD3 A7 00 > STAA 0,X
5BD5 96 R2 > LDAA SAVX+1
5BD7 A7 01 > STAA 1,X
5RD9 DF R1 > LDX SAVX
5BDB 86 28 > LDAA #40
5BDD BD 10 70 > JSR AA2X
5BE0 DF R1 > STX SAVX
5RE2 7E 5A CA > JMP EWEN
5BE5 >*
5BE5 >* ENV CONTROL MOVE
      >TUKE LDX SAVY

SBE5 DE R4
SBE7 D6 81 > LDAB SAVX
SBE9 E7 00 > STAB 0,X
5BEB 08 > INX
5BEC D6 R2 > LDAB SAVX+1
SBEF E7 00 > STAB 0,X
5BF0 08 > INX
5BF1 DF R4 > STX SAVY
5BF3 39 > RTS
5BF4 >*
5BF4 >* ENV DEF
5BF4 >EDEF EQU *
5BF4 8D C5 > BSR OHED
5BF6 R4 OF > ANDA #SF
5BF8 48 > ASLA
5BF9 48 > ASLA
5BF9 48 > ASLA
5RFB CE 58 B6 > LDX #ETAB
5RFE BD 10 70 > JSR AA2X
5C01 DF R4 > STX SAVY
5C03 DE R1 > LDX SAVX
5C05 A6 00 > LDAA 0,X
5C07 36 > PSHA
5C08 08 > INX
5C09 A6 00 > LDAA 0,X
5C0B 36 > PSHA
5C0C 08 > INX
5C0D A6 00 > LDAA 0,X
5C0F 08 > INX
5C10 DF R1 > STX SAVX
5C12 BD D1 > BSR TUKE
5C14 DE R1 > LDX SAVX
5C16 BD 10 70 > JSR AA2X
5C19 DF R1 > STX SAVX
5C1B BD C8 > BSR TUKE
5C1D 32 > PULA
5C1E DE R1 > LDX SAVX
5C20 BD 10 70 > JSR AA2X
5C23 DF R1 > STX SAVX
5C25 BD BE > BSR TUKE
5C27 32 > PULA
5C28 DE R1 > LDX SAVX
5C2A BD 10 70 > JSR AA2X
5C2D DF 81 > STX SAVX
5C2F BD B4 > BSR TUKE
5C31 7E 5A CA > JMP EWEN
5C34 >*
5C34 >* BEGIN SUBROUTINE
      >BEGS JSR OHED
5C34 BD 5B BB
5C37 7D 00 7D > TST SDHG
5C3A 27 00 > BEQ SDE2
5C3C 7F 00 7D > CLR SDHG
5C3F R4 3F > ANDA #3F
5C41 48 > ASLA
5C42 CE 58 06 > LDX #SRTN
5C45 BD 10 70 > JSR AA2X
5C48 96 R1 > LDAA SAVX
5C4A A7 00 > STAA 0,X
5C4C 96 R2 > LDAA SAVX+1
5C4E A7 01 > STAA 1,X
5C50 7E 5A CA > SDE2 JMP EWEN
5C3B 14 >*
5C53 >*
5C53 >* END SUBROUTINE (RETURN IF EXECUTION)
5C53 DE 7E >ENDS LDX PROC
5C55 7D 00 7D > TST SDHG
5C58 27 00 > BEQ SEXT
5C5A C6 06 > LDAB #6
5C5C A6 02 > SXLP LDAA 2,X
5C5E A7 00 > STAA 0,X
5C60 08 > INX
5C61 5A > DECB
5C62 26 F8 > BNE SXLP
5C64 DE 7E > LDX PROC
5C66 6F 06 > CLR STK3,X
5C68 6F 07 > CLR STK3+1,X
5C6A EE 00 > LDX 0,X
5C6C 08 > INX
5C6D 08 > INX
5C6E DF R1 > STX SAVX
5C70 7E 5A CA > JMP EWEN
5C73 73 00 7D >SEXD COM SDHG
5C59 19
5C76 7E 00 00 >EXEQ2 JMP OBYT
5C79 >*
5C79 >* SET LENGTH REG
5C79 >LSET EQU *
5C79 BD 5B BB > JSR OHED

5C7C E6 00 > LDAB 0,X
5C7E 08 > INX
5C7F DF 81 > STX SAVX
5CR1 37 > PSHE
5CR2 84 0F > ANDA #SF
5CR4 CE 58 86 > LDX #LGNT
5CR7 BD 10 70 > JSR AA2X
5CR8 33 > PULB
5C8B E7 00 > STAB 0,X
5CRD 7E 5A CA > JMP EWEN
5C90 >*
5C90 >* NOTE INC SERVICE
5C90 DE 7E >NINI LDX PROC
5C92 A6 0B > LDAA ONOT,X
5C94 4C > INCA
5C95 16 > TAB
5C96 C4 0F > ANDB #SF
5C98 C1 0B > CMPB #11
5C9A 2F 00 > BLE NIN2
5C9C 84 F0 > ANDA #SF0
5C9E 8B 10 > ADDA #310
5CA0 81 7F >NIN2 CMPA #57F
5C9B 04 >*
5CA2 25 00 > BCS NINI
5CA4 4A > DECA
5CA5 A7 0B > STAA ONOT,X
5CA7 39 >NINI RTS
5CA3 03 >*
5CAR >*
5CAR >* NOTE INCREMENT
5CAR >INCN EQU *
5CAR 8D E6 > BSR NINI
5CAA DE 7E >OBYT LDX PROC
5C77 5C AA >*
5BAC 5C AA >*
5CAC 7E 5A C5 > JMP EXE1
5CAF >*
5CAF >* INCREMENT NOTE AND PLAY
5CAF >IPLA EQU *
5CAF 7D 00 7D > TST SDHG
5CB2 27 F6 > BEQ OBYT
5CB4 8D DA > BSR NINI
5CB6 7E 5B 00 > JMP NOT1
5CB9 >*
5CB9 >* DEC NOTE SERVICE
5CB9 86 01 >DMIN LDAA #1
5CBB A7 0B > STAA ONOT,X
5CBD 39 > RTS
5CBE 4D >DFLR TSTA
5CBE 27 F8 > BEQ DMIN
5CC1 80 10 > SUBA #$10
5CC3 27 F4 > BEQ DMIN
5CC5 8B 0B > ADDA #11
5CC7 A7 0B > STAA ONOT,X
5CC9 39 > RTS
5CCA DE 7E >DINI LDX PROC
5CCC A6 0B > LDAA ONOT,X
5CCE 16 > TAB
5CCF C4 0F > ANDB #SF
5CD1 27 EB > BEQ DFLR
5CD3 4A > DECA
5CD4 27 E3 > BEQ DMIN
5CD4 27 E3 > STAA ONOT,X

5CD6 A7 0B >*
5CD6 39 > RTS
5CD9 >*
5CD9 >* DECREMENT NOTE
5CD9 >DECN EQU *
5CD9 8D EF > BSR DINI
5CDB 7E 5C AA > JMP OBYT
5CDE >*
5CDE >* DECREMENT AND PLAY NOTE
5CDE >DPLA EQU *
5CDE 7D 00 7D > TST SDHG
5CE1 27 C7 > BEQ OBYT
5CE3 8D E5 > BSR DINI
5CE5 7E 5B 00 > JMP NOT1
5CE6 >*
5CE6 >* SET TEMPO
5CE6 >TEMP EQU *
5CE6 BD 5B BB > JSR OHED
5CEB 97 7B > STAA TMP1
5CED 7E 5A CA > JMP EWEN
5CF0 >*
5CF0 >* INC TEMPO
5CF0 >TINC EQU *
5CF0 96 7B > LDAA TMP1
5CF2 4C > INCA
5CF3 27 00 > BEQ NOCH
5CF5 97 7B > STAA TMP1
5CF7 7E 5C AA >NOCH JMP OBYT
5CF4 02 >*
5CFA >*
5CFA >* DEC TEMPO
5CFA >TDEC EQU *
5CFA 96 7B > LDAA TMP1
5CFc 4A > DECA
5CFD 27 F8 > BEQ NOCH
5CFF 97 7B > STAA TMP1
5D01 7E 5C AA > JMP OBYT
5D04 >*
5D04 >* WAIT TIL MEASURE
5D04 >WMEA EQU *
5D04 DE 7E > LDX PROC
5D06 6F 0A > CLR NXBT,X NEW BEAT IS 0
5D08 EE 00 > LDIX 0,X
5D0A 08 > INX
5D0B A6 00 > LDAA 0,X
5D0D 08 > INX
5D0E E6 00 > LDAB 0,X
5D10 08 > INX

```

```

SD11 DF 81 > STX SAVX
SD13 DE 7E > LDX PROC
SD15 A7 08 > STAA NXMS,X
SD17 E7 09 > STAB NXMS+1,X NEW MEAS DEFINED
SD19 7E 5B 2C > JMP GOBK
SDIC >*
SDIC >* PICK ENV
SDIC >ESET EQU *
SDIC BD 5B BB > JSR OHED
SD1F 84 0F > ANDA #SF
SD21 48 > ASLA
SD28 48 > ASLA
SD23 48 > ASLA
SD24 CE 5B B6 > LDX #ETAB
SD27 BD 10 70 > JSR AA2K
SD2A C6 08 > LDAB #8
SD2C A6 00 >SET1 LDAA 0,X
SD2E 36 > PSHA
SD2F 08 > INX
SD30 5A > DECB
SD31 26 F9 > BNE SET1
SD33 C6 08 > LDAB #8
SD35 CE 59 41 > LDX #EPTR+7
SD38 32 >SET2 PULA
SD39 A7 00 > STAA 0,X
SD3B 09 > DEX
SD3C 5A > DECB
SD3D 26 F9 > BNE SET2
SD3F 7E 5A CA > JMP EWEN
SD42 >*
SD42 >* ENV RATE
SD42 >ERAT EQU *
SD42 BD 5B BB > JSR OHED
SD45 97 78 > STAA RATE
SD47 7E 5A CA > JMP EWEN
SD4A >*
SD4A >* SET STACATTO
SD4A >STAC EQU *
SD4A CE 00 00 > LDX #DEC1
SD4D DF 75 > STX SULK
SD4F >> FALL THRU TO
SD4F >* START DECAY
SD4F 7F 00 7A >DEKA CLR DECF
SD52 7E 5C AA > JMP OBYT
SD55 >*
SD55 >* SET LEGATTO
SD55 >LEGA EQU *
SD55 CE 00 00 > LDX #SUSI
SD56 DF 75 > STX SULK
> LDAA #FFF
SD5A 86 FF > STAA DECF
SD5C 97 7A > STAA DECF
SD5E 7E 5C AA > JMP OBYT
SD61 >*
SD61 >* START ATTACK
SD61 >ATTK EQU *
SD61 86 FF > LDAA #FFF
SD63 97 7A > STAA DECF
SD65 CE 00 00 > LDX #ANEW
SD68 DF 04 > STX NMIV
SD6A 7E 5C AA > JMP OBYT
SD6D >*
SD6D >* SET NOTE PARAMETER
SD6D BD 5B BB >SNOT JSR OHED
SD70 DE 7E > LDX PROC
SD72 A7 0B > STAA ONUT,X
SD74 7E 5A CA > JMP EWEN
SD77 >*
SD77 >* RETURN TO MONITOR, RESTORING ACIA CODES
SD77 86 03 >TCLO LDAA #3
SA67 5D 77
SA14 5D 77
SD79 B7 01 58 > STAA ATEC
SD7C 86 81 > LDAA #81
SD7E B7 01 58 > STAA ATEC
SD81 B6 01 59 > LDAA ATED
SD84 7F 01 63 > CLR TIMS TURN OFF INTERRUPTS
SD87 CE 00 00 > LDX #PRTA
SD8A BD 00 > BSR UNSV
SDRC CE 00 10 > LDX #PRTA+16
SDBF BD 00 > BSR UNSV
SD91 7E 10 00 > JMP $1000
SD94 >*
SD94 >* RESTORE ORIGIN
SD94 A6 0E >UNSV LDAA ORIG,X
SD8B 08
SD90 03
SD96 A7 00 > STAA 0,X
SD98 A6 0F > LDAA ORIG+1,X
SD9A A7 01 > STAA 1,X
SD9C 39 > RTS
SD9D >*
SD9D >* OSCILLOSCOPE TRIGGER (D6 OF WAVE PIA)
SD9D 86 C0 >OSC1 LDAA #80
SD9F B7 01 54 > STAA WAVD
SD2A 86 80 > LDAA #80
SD44 B7 01 54 > STAA WAVD
SD47 7E 5C AA > JMP OBYT
SDAA >*
SDAA >* WAVE CALCULATOR PACKAGE
SDAA >*
SDAA >* WAVE OVERHEAD
>WHD JSR OHED
SDAA BD 5B BB > ANDA #SF
SDAD 84 0F > ASLA
SDAF 48 > ASLA
SD80 CE 5B 96 > LDX #WFOR
SDB3 BD 10 70 > JSR AA2K
SDB6 EE 00 > LDX 0,X
SDB8 FF 59 6C > STX WPT2
SDBB >* ARRAY OVERHEAD

SDBB CE 59 42 >WARY LDX #TWAV
SDBE A6 00 >WRY1 LDAA 0,X
SDCO FF 59 6A > STX WPT1
SDC3 FE 59 6E > LDX WPGM
SDC6 AD 00 > JSR 0,X
SDC8 FE 59 6A > LDX WPT1
SDCB A7 00 > STAA 0,X
SDCD 08 > INX
SDCE 8C 59 6A > CPX #TWAV+40
SDD1 26 EB > BNE WRY1
SDD3 39 > RTS
SDD4 >*
SDD4 >* SET WAVE REGISTER
SDD4 FE 59 6C >WRGS LDX WPT2
SDD7 A6 00 > LDAA 0,X
SDD9 08 > INX
SDDA FF 59 6C > STX WPT2
SDDD 39 > RTS
SDDE CE 5D D4 >TMFW LDX #WRGS
SDE1 FF 59 6E > STX WPGM
SDE4 8D C4 > BSR WOHD
SDE6 7E 5A CA > JMP EWEN
SDE9 >*
SDE9 >* OUTPUT := WAVE REG
SDE9 36 >WRUS PSHA
SDEA 84 7F > ANDA #SF
SDEC B7 01 54 > STAA WAVD
SDEE 83 34 > LDAA #S34
SDF1 B7 01 55 > STAA WAVC
SDF4 86 3C > LDAA #S3C
SDF6 B7 01 55 > STAA WAVC
SDF9 32 > PULA
S DFA 39 > RTS
S DFA CE 5D E9 >WTMP LDX #WRUS
S DFE FF 59 6E > STX WPGM
SE01 7F 01 54 > CLR WAVD
SE04 8D B5 > BSR WARY
SE06 86 80 > LDAA #80
SE08 B7 01 54 > STAA WAVD
SE08 7E 5C AA > JMP OBYT
>
SE0L >*
SE0N >* . .
SE0E 47 >2DS ASRA
SE0F 39 > RTS
SE10 CE 5E 0E >DIVW LDX #W2DS
SE13 FF 59 6E > STX WPGM
SE16 8D A3 > BSR WARY
SE18 7E 5C AA > JMP OBYT
SE1B >*
SE1B >* WAVE REG := WAVE REG * 2
SE1B 48 >W2MS ASRA
SE1C 39 > RTS
SE1D CE 5E 1B >MULW LDX #W2MS
SE20 FF 59 6E > STX WPGM
SE23 8D 96 > BSR WARY
SE25 7E 5C AA > JMP OBYT
SE28 >*
SE28 >* WAVE REG := WAVE REG + WAVE(X)
SE28 FE 59 6C >WPLS LDX WPT2
SE2B AB 00 > ADDA 0,X
SE2D 08 > INX
SE2E FF 59 6C > STX WPT2
SE31 39 > RTS
SE32 CE 5E 28 >ADDW LDX #WPLS
SE35 FF 59 6E > STX WPGM
SE38 BD 5D AA > JSR WOHD
SE3B 7E 5A CA > JMP EWEN
SE3E >*
SE3E >* DUMP WAVE
SE3E FE 59 6C >STWS LDX WPT2
SE41 A6 00 > LDAA 0,X
SE43 08 > INX
SE44 FF 59 6C > STX WPT2
SE47 8D A0 > BSR WRUS
SE49 39 > RTS
SE4A CE 5E 3E >WSET LDX #STWS
SE4D FF 59 6E > STX WPGM
SE50 7F 01 54 > CLR WAVD
SE53 BD 5D AA > JSR WOHD
SE56 86 80 > LDAA #80
SE58 B7 01 54 > STAA WAVD
SE5B 7E 5A CA > JMP EWEN
SE5E >*
SE5E >* SET TIME SIGNATURE
SE5E BD 5B BB >SIGN JSR OHED
SE61 97 7C > STAA BMAR
SE63 7E 5A CA > JMP EWEN
SE66 >*
SE66 >* GO TO ORIGIN
SE66 DE 7E >LOOP LDX PROC
SE68 EE 0E > LDX ORIG,X
SE6A DF 81 > STX SAVX
SE6C 7E 5A CA > JMP EWEN
SE6F >*
SE6F >* GOTO
SE6F DE 7E >GOTO LDX PROC
SE71 EE 00 > LDX 0,X
SE73 EE 01 > LDX 1,X
SE75 DF 81 > STX SAVX
SE77 7E 5A CA > JMP EWEN
SE7A >*
SE7A >* INTERRUPT ROUTINES
SE7A >* KERNEL OF INTERRUPT HANDLER
SE7A 96 83 >TMER LDAA TMPO
SE97 5E 7A >*
SE7C 27 00 > BEQ TMEC
SE7E 7A 00 83 > DEC TMPO
SE81 3B > RTI
SE82 7C 00 86 >TMEE INC LOST
SE7D 04 >

```

```

SE85 3B > RTI
SE86 >*
SE86 >* ATTACK INTERRUPT
SE86 96 77 >ATKI LDAA ECNT
SE88 27 00 > BEQ APDA
SE8A 4A > DECA
SE8B 97 77 > STAA ECNT
SE8D 26 EB > BNE TMER
SE8F DE 73 > LDX ENVX
SE91 BC 59 3C > CPX SUST
SE94 26 E4 > BNE TMER
SE96 DE 75 > LDX SULK
SE99 DF 04 > STX NMIV
SE9A 20 DE > BRA TMER
SE9C DE 73 >APDA LDX ENVX
SE89 12
SE89 A6 00 > LDAA 0,x
SEA0 91 79 > CMPA OENV
SEA2 25 00 > BCS NATT
SEA4 B7 01 56 > STAA ENVD
SEA7 08 >NATT INX
SEA3 03
SEA8 DF 73 > STX ENVX
SEA9 96 78 > LDAA RATE
SEAAC 8A 03 > ORAA #3
SEAE 97 77 > STAA ECNT
SEBO 20 C8 > BRA TMER
SEB2 >*
SEB2 >* NEW ATTACK INITIATION
SEB2 CE 5E 86 >ANEW LDX #ATKI
SB53 5E B2
SD66 5E B2 > STX NMIV

```

```

SEB5 DF 04 > LDX PATK
SEB7 FE 59 3A > STX ENVX
SEBA DW 73 > BRA TMER
SEBC 20 BC >*
SEBE >*
SEBE 96 77 >SUSTAIN INTERRUPT
SEBE 96 77 >SUSI LDAA ECNT
59C0 5E BE
SD56 5E BE
SECO 27 00 > BEQ UPDA
SEC2 4A > DECA
SEC3 97 77 > STAA ECNT
SEC5 26 B3 > BNE TMER
SEC7 DE 73 > LDX ENVX
SEC9 BC 59 3C > CPX DECY
SECC 26 AC > BNE TMER
SECE D6 7A > LDAB DECF
SED0 26 00 > BNE DNEC
SED2 CE 00 00 > LDX #DECI
SED5 DF 04 > STX NMIV
SED7 20 A1 > BRA TMER
SED9 FE 59 3C >DNEC LDX SUST
SED1 07
SEDC DF 73 > STX ENVX
SEDE 20 9A > BRA TMER
SEEO >*
SEEO >* DECAY INTERRUPT
SEEO 96 77 >DEC1 LDAA ECNT
SEED 27 00 > BEQ UPDA
SEE4 4A > DECA
SEES 97 77 > STAA ECNT
SEET 26 91 > BNE TMER
SEEF DE 73 > LDX ENVX
SEEB BC 59 40 > CPX ENND
SEEE 26 8A > BNE TMER
SEFO FE 59 3A > LDX PATK
SEF3 DF 73 > STX ENVX
SEF5 CE 5E 7A > LDX #TMER
SEF8 DF 04 > STX NMIV
SEFA 7E 5E 7A > JMP TMER
SEFD DE 73 >UPDA LDX ENVX
SEC1 3B
SEE3 19
SEFF A6 00 > LDAA 0,x
SF01 97 79 > STAA OENV
SF03 B7 01 56 > STAA ENVD
SF06 08 > INX
SF07 DF 73 > STX ENVX
SF09 96 76 > LDAA RATE
SF0B 8A 03 > ORAA #3
SF0D 97 77 > STAA ECNT
SF0F 7E 5E 7A > JMP TMER
SF12 >*
SF12 >* FUNC VECTOR TABLE

```

```

SF12
SF12 >FTAB EQU *
5BBB SF 12 >FTAB EQU *
SF12 >----NAME-FORMAT----COMMENTARY-----
SF12 >* 00 TO 7F PLAY SELECTED NOTE
SF12 >* 8X LENGTH := LTAB(X)
SF12 5B C6 > FDB WDEF 900X... WAVE(X) := NEXT 40 VALUES
SF14 SE 4A > FDB WSET 910X HORN := WAVE(X)
SF16 SD DE > FDB TMPW 920X TWAVE := WAVE(X)
SF18 SD FB > FDB WTMP 93 HORN := TWAVE
SF1A SE 10 > FDB DIVW 94 TWAVE := TWAVE / 2
SF1C SE 1D > FDB MULW 95 TWAVE := TWAVE * 2
SF1E SE 32 > FDB ADDW 960X TWAVE := TWAVE + WAVE(X)
SF20 SD 6D > FDB SNOT 97XX NOTE := XX
SF22 SC A6 > FDB INCN 98 NOTE := NOTE + 1
SF24 SC AF > FDB IPLA 99 NOTE := NOTE - 13 PLAY NOTE
SF26 SC D9 > FDB DECN 9A NOTE := NOTE - 1
SF28 SC DE > FDB DPLA 9B NOTE := NOTE - 13 PLAY NOTE
SF2A SA E4 > FDB WAIT 9C WAIT UNTIL T + LENGTH
SF2C SD 04 > FDB WMEA 9DXXXX WAIT UNTIL MEASURE XXXX
SF2E SC 79 > FDB LSET 9E0XLL LTAB(X) := LL
SF30 SD 9D > FDB OSCL 9F TRIGGER SCOPE

```

```

SF32 SC 34 > FDB BEGS A0XX... BEGIN SUBRTN XX DEF
SF34 SC 53 > FDB ENDS A1 END OR RETURN
SF36 SB 5B > FDB CALL A2XX CALL SUBRTN XX
SF38 SB F4 > FDB EDEF A30XDDSSAA DEF ENV X
SF3A SD 1C > FDB ESET A40X PICK ENV X
SF3C SD 42 > FDB ERAT A5XX ENV RATE := XX
SF3E SD 4A > FDB STAC A6 STACCATO MODE
SF40 SD 55 > FDB LEGA A7 LEGATO MODE
SF42 SD 61 > FDB ATTK A8 START ATTACK
SF44 SD 4F > FDB DEKA A9 START DECAY
SF46 SC E8 > FDB TEMP AAXX TEMPO := XX
SF48 SC F0 > FDB TINC AB TEMPO := TEMPO + 1
SF4A SC FA > FDB TDEC AC TEMPO := TEMPO - 1
SF4C SE 5E > FDB SIGN ADXX SET BEATS PER MEASURE
SF4E SE 66 > FDB LOOP AE GO TO ORIGIN
SF50 SA ED > FDB PLAY AF PLAY OLD NOTE
SF52 SE 6F > FDB GOTO BOXXXX GO TO XXXX
SF54 >*
2000 >PRTA EQU $2000
59E2 20 00
59F7 20 00
5D86 20 00
5A07 20 00
5D8D 20 00
> END
5F54

```

*** END - UNRESOLVED ITEMS:

*** SYMBOLS:

AA2X 1070	ADDW 5E32	ANEW 5EB2	APDA 5E9C	ATEC 0158
ATED 0159	ATKI 5E86	ATTK 5D61	BEAT 0072	BEGS 5C34
BMAX 007C	CALL 5B5B	DDND 5A48	DEC1 007A	DEC1 SEE0
DECN 5CD9	DECY 593E	DEKA 5D4F	DFLR 5C8E	DINI 5CCA
DIUV 5E10	DMIN 5C89	DNDZ 5A3F	DNEC 5E9D	DPLA 5CDE
DUME 5936	ECNT 0077	EDF 5BF4	ENDF 0080	ENDS 5C53
ENND 5940	ENTY 5800	ENV 0157	ENVD 0156	ENVX 0073
EPTR 593A	ERAT 5D42	ESET 5D1C	ETAB 5B86	EWEN SACA
EXE1 5ACS	EXE2 5C76	EXE3 5BAB	EXE4 5B58	EXEC SAA9
FSUM 000D	FTAB 5F12	FUNC 5AD9	G0BN 5B2C	GOGO 5970
GOMU 59FE	GOTO 56EF	HORN 014C	ICNC 5C46	INTR 5A89
INV1 5A5E	INWA 5A16	IPLA 5CAF	KNOT 5B43	KNT1 5B43
LEGA 5D55	LEN1 5B89	LENG 5AD6	LGNT 5B86	LOOP 5E66
LOST 0086	LSET 5C79	MEAS 0070	MILI 0138	MULW 5E1D
NATT 5EA7	NGAT 5B52	NINI 5CA7	NINI 5CAO	NINI 5C90
NMIV 0004	NOAK 5AAB	NOCH 5CF7	NOT1 5B00	NOTE 5AFB
NPPR 5A7F	NXBT 000A	NXMS 0008	OBYT 5CAA	OENV 0079
OHED 5BBB	OLNG 000C	ONOT 000B	ORIG 000E	OSCL 5D9D
PATK 593A	PLAY 5AED	PLIN 5A7F	PPN1 5A63	PPN2 5A82
PPTR 0089	PROC 007E	PRTA 2000	RATE 0078	SAVX 0081
SAVY 0084	SDE2 5C50	SDFG 007D	SET1 5D8C	SET2 5D38
SEXD 5C73	SIGN 5E5E	SNOT 5D6D	SRTN 5806	STAC 5D4A
STK1 0002	STK2 0004	STK3 0006	STWS 5E3E	SULK 0075
SUS1 5EBE	SUST 593C	SVSL 59F7	SVST 59ED	SXLP 5C5C
TCLO 5D77	TDEC 5CFA	TEMP 5C6D	TIME 0161	TIMS 0163
TINC 5CFO	TLST 0087	TMEE 5E82	TMER 5E7A	TMPI 007B
TMPO 0083	TMPT 5DDE	TUNE 5BE5	TWAV 5942	UNSV 5D94
UPDA 5EFD	W2DS 5E0E	W2MS 5E1B	WADJ 5B39	WAIT 5A4E
WAIX 5B02	WARY 5DBB	WATI 5ADC	WAVC 0155	WAVD 0154
WAK1 5B11	WAX2 5B1D	WDEF 5B6	WFOR 5B96	WMEA 5D04
WORD 5DA4	WPGR 59E6	WPLS 5E2B	WPT1 596A	WPT2 596C
WRGS 5DD4	WRUS 5DE9	WRY1 5DBE	WSET 5E4A	WTMP 5DFB
XFUN 5BAE	XLNG 5B85	XSLP 5B6C	XSTL 5BA4	/

NOTES ON MICROCOMPUTER MUSIC

Marc Le Brun
Star Route Box 111
Redwood City CA 94062

Introduction

In recent years striking results have been obtained through the use of computers in the production of music. Much of this work has been conducted in research settings, such as university laboratories, and is often obtained through extensive utilization of the advanced hardware and software systems these facilities are able to provide. Indeed, it is highly unlikely that many of the major achievements in the computer music field would have been accomplished at all without the high-level support of these sponsoring institutions and agencies.

Unfortunately, while the power inherent in these large, expensive, systems is probably requisite for research it also tends to create difficulties when we wish to apply the results of that research in an effective, economical, manner. Often new techniques are developed which work extremely well in the context of a megasystem but seem nearly impossible to implement on, for instance, a microcomputer for use in homes, in schools, or by working musicians.

In the course of my work in this area I have found that it is in fact possible to transfer many techniques from the laboratory to the "real" world, but that to do so usually requires that we expend some effort on recasting the methods involved into forms that are appropriate to our applications. In this paper I will present several points which I believe bear on the pragmatics of this promising and exciting area and which I hope will be of use to the growing numbers of microcomputer musicians.

Objectives

The computer is an extremely versatile instrument and one of the significant problems that continually arises in computer music is how to deal with that flexibility. Paradoxically, it is only when we decide which of the infinitude of possibilities we will concentrate on, thereby giving up some freedom of choice, that we begin to be able to achieve anything at all. The question is, therefore, just what do we want a microcomputer music system to do for us?

Certainly there is no single right answer to this question, in fact it is to our advantage if everyone's answer is somewhat different. There is a tremendous amount of unexplored territory in computer music and the greater the diversity of approaches the faster those areas will be developed into usefulness. It seems though that there are a few clear objectives, to which each person will then ascribe varying weights.

The first of these of course, is that it sounds good, that is it should be capable of producing interesting, musical quality sounds. This involves aesthetic judgements, about which there can be only partial agreement. I believe however that one of the most important artistic accomplishments of computer music has been the synthesis of rich, dynamic and evocative sounds, and it would seem unnecessary and unfortunate to have to forego such a singularly attractive feature simply because we are working with smaller systems.

I feel this is a somewhat important point, in fact it has motivated a good deal of my researches. One of the biggest fallings, from a musical standpoint, of much of electronic music, is the relative paucity of sounds employed as compared to those available to the traditional, acoustical musician. If you listen to the usual square or sawtooth waves which pretty much define the palate of timbres available on a cheap analog synthesizer for any length of time this will be readily apparent. To create interesting sounds with the standard analog modules is an extremely difficult task. The digital age has arrived - surely we should take advantage of all the computer has to offer, rather than self-imposing past limitations.

The remaining objectives should be quite familiar - a microcomputer music system should be low-cost, it should be flexible and general, and it should be fairly easy to use. As is commonly the case these objectives can be continually traded off against each other - we can pay more for a better sounding system, we can give up a little generality to make it easier to use and so on.

In summary then we would like our microcomputer music system to be cheap enough for nearly anyone, schools included, simple enough for a child to use, have enough capabilities to intrigue a musician and to produce sounds that are satisfying to all these folks as well. That shouldn't be too much to expect.

Methods

How do we achieve this impossible dream? One place to begin is with the actual sound generation and work from there, keeping in mind the various design objectives. The remainder of this paper will concentrate on digital synthesis and attendant issues.

The central problem to be solved in many areas of digital synthesis is how to attain a given musical or other type of signal within a reasonable cost. Cost can be measured directly in terms of things like number of chips needed to implement a given function in a particular manner, or in terms of the amount of time it takes, or indirectly in terms of constraints on the signals produced, complexity of the necessary control systems and so on.

Ultimately however all this engineering must pass the acid test - *What does it sound like?* It would be nice if we had some universal model of human hearing and perception with which we could reliably predict this but we don't. One of the many electrifying insights provided by John Chowning's discovery and development of the *musical* uses of frequency modulation was that you could employ a *single* simple technique to produce a wide range of sounds, from gong-like to trumpet-like, in many cases involving waveforms whose shapes and composition differed radically from those of their natural counterparts, all at an extremely low cost achieved by almost totally losing control over the sound.

There is an important lesson here - often the ear does not hear through a loudspeaker what the eye sees on the 'scope. We have a great deal to learn yet about psychoacoustics, which means we must still come back to that all-important test - *What does it sound like?*

It is often the case however that we have a given technique which has proved interesting or valuable in some way, and we wish to implement it in a cost effective manner. Creative work in this area often results in major payoffs. The telephone system is one example where this has been the case. It would hardly function at the scale it does today if there had not been a long history of people doing creative work in various aspects of signal processing. Fortunately we computer musicians can benefit from their work as well. In fact an examination of the basic elements of computer music often leads to rewarding results.

Digital Sampled Data Systems

Common to virtually all digital synthesis systems is the representation of signals as sequences of numbers which specify the amplitude of the signal at specific, usually equispaced, instants in time. While other representations of signals are in fact possible and potentially advantageous this particular representation is intuitively straightforward and allows us to apply a good portion of the techniques developed from our relatively long experience with analog signal systems.

The intrinsic properties of the usual sort of sampled data system often have a pervasive and significant effect on the design of various musical procedures or higher level modules. By investigating these intrinsic properties we may be able to derive techniques for improving the overall cost effectiveness of large classes of operations by incorporating adaptations to these properties into their initial design.

One intrinsic property of sampled data is that the rate at which a signal is sampled often affects the nature of the output signal in a peculiar way: if the signal contains components with frequencies greater than half the sampling rate then those components are "folded over" until they lie within the interval of frequencies which are less than half the sampling rate.

This property can have a direct effect on the nature of a synthesis system. Since human hearing involves the perception of frequencies of at least 10-20khz it means that the digital synthesizer must be capable of producing samples at least twice as fast, at rates of 20-40khz. Translated into actual times this means that a digital synthesizer would have about 25-50usec in which to generate each sample.

This is not a particularly large amount of time in which to compute the samples on a microprocessor where the *minimum* instruction times are typically on the order of several microseconds each. It is abundantly clear that if the sound is at all complex, that is, interesting to listen to, we are going to have to be extremely tricky in order to do the computations at the required rate.

High Speed Implementation

There are a number of methods of increasing the signal generation rate. Unfortunately even with the best of these the degree of improvement that can be achieved for code running in today's typical microcomputer is not sufficient to allow us to generate really interesting music: rich, dynamic timbres, multiple voices, spatial movement of sounds, reverberation and

other signal processing effects which enhance the quality of the music, are all nearly impossible to produce.

Probably the most straightforward answer to these limitations is to construct small, special purpose, high speed devices which generate the actual signal samples and use the microprocessor to control and coordinate their operation. The advantages of such a scheme seem particularly large given that our desire to produce high quality musical sounds outweighs the small additional cost in design effort that the external special purpose devices will require.

With the speed of our primitive computational operations reduced to the few hundreds of nanoseconds achievable with easily available off the shelf components a great variety of operations become available to us, although cost effectiveness must continue to be a dominant design consideration. An additional interesting effect also becomes apparent: the operations that we wish to perform at higher speeds tend to be fairly repetitive and simple, say the generation of successive samples of a sine wave, while operations of a more general, complex nature, such as putting an arbitrary amplitude envelope on said wave, can be done at a much slower rate.

This fractionation of operations into groups in which speed and complexity are approximately inversely related is probably fundamental. For instance in human hearing we see similar fractionation. Low frequencies are termed rhythms, high ones pitch and intermediate ones cover most of the so-called "signature" information by which we organize and classify sounds, such as attack characteristics, spectrum dynamics and other forms of articulation.

Whatever the reason we can profitably exploit this phenomenon. Whereas a signal with a given spectral profile, say, must be generated rapidly, we can probably get away with giving it an envelope at a much slower rate, perhaps some convenient one such as 60hz. Therefore a system which performs *all* operations at very high speed simply because some operations need to be performed quickly is wasteful. It is not necessary for a music processor to operate entirely in high gear, only parts of it need to, and those parts will usually be comparatively simple, leaving the general purpose processor free to perform more general, complex, operations at reasonable speeds.

Heterogeneity

It seems that this parsimonious fractionation occurs within other design factors as well. For instance it is often true that while some computations require high precision others may not and to force all data paths to be wide enough to accomodate the maximally accurate values is again unnecessarily wasteful.

It therefore appears that an effectively implemented music system will tend to be heterogeneous in nature with different aspects of synthesis handled in different ways. This also means that a useful design strategy would be to look carefully at each specific function in terms of its intrinsic properties in order to ascertain its natural modes of fractionation and thereby minimize its cost. The remainder of this paper will be devoted to the examination of several specific functional areas which will hopefully be of practical use but more importantly may help to establish a more general approach to the problem of creating useful musical instruments with microcomputers.

Data Reduction

We can begin with the question of just how much data is actually needed for a given function and attempt to remove unnecessary redundancies. An example of such a reduction is the so called *Floating Point DAC* used by the Center for Computer Research in Music and Acoustics at Stanford University.

The usual approach to communicating a sixteen bit sample to a DAC is simply to take the sixteen bits and transmit them directly to the DAC. While quite simple this approach requires a data path width of sixteen bits. Even if we transmit the bits serially this approach implies a data transmission rate of 160,000 bits/sec.

The *Floating Point DAC* makes use of various intrinsic properties of sound signals to achieve a fairly substantial saving as follows: the DAC itself contains a sixteen bit register which determines the output amplitude. Rather than being loaded directly however the DAC receives a nine bit *Increment* which is coded as a *floating point* number consisting of four exponent bits and only five bits of mantissa. Nine bits were chosen since the main processor, a Digital Equipment Corporation PDP-10, is a thirty-six bit machine. Experiments indicate that reducing the mantissa to four bits, thus giving a convenient eight bit byte, continues to produce sound which is quite indistinguishable from full sixteen bit fixed point encoding, even to discerning listeners.

The advantage of this method is that we require only half as many bits of information per sample, which halves the baud rate. In addition if we are at any point buffering the sounds, for instance storing them for later playback, our memory requirements are likewise halved.

Another technique which may find extensive use is embodied in an interesting facility available on the Stanford Artificial Intelligence Laboratory's system which is known as the *Audio Switch*. Every terminal is equipped with a small speaker and amplifier of a very inexpensive design. These speaker boxes are then fed purely *digital* signals. The signals are a pulse width modulation encoding of the sound samples. When these pulses are fed to the speaker they are, in effect, integrated by the inertial properties of the speaker. This causes a displacement of the speaker cone proportional to the width of the pulse which is perceived in turn as sound.

This system, although extremely inexpensive, produces sound of quite reasonable quality, quite as good for instance as an average radio reception. An attendant advantage of the system is that the digital sounds may be switched on and off or selected using entirely digital components and thus avoiding much of the difficulties encountered in analog signal switching. Finally, of course, it only requires a one bit wide data path per channel.

Without a doubt there are other useful encoding and data compression techniques which may be applied to the output samples or at other points in the flow of data within a microcomputer music system.

Table Reduction

A common computational technique is to gain speed by making some tradeoff which results in increased memory requirements. A prevalent example of this is the practice of looking up commonly used values in a table rather than recomputing them countless times. It often turns out that the point of maximum return is reached somewhat short of a pure table lookup with a hybrid technique involving a small table and a small amount of computation.

For instance Max Matthew's pioneering work on computer music appears to imply that a sine function lookup table should consist of approximately 512 elements. The theory of polynomial approximations indicates that an interpolation scheme cannot improve matters much without introducing unwanted harmonic distortion. In some informal experiments however I have been able to generate signals perceptually indistinguishable from pure sine waves with tables with as few as ten elements using cubic splines defined over points with nonuniform spacing. Unfortunately the theory of these approximations is complex making exact estimates of distortion difficult, but the practical implementation is simple and produces extremely encouraging results. Further explorations in this area are probably warranted.

Another table reduction technique which has met with a reasonable degree of success has been the approximation of slowly changing functions, such as envelopes, by line segments. John Grey of the CCRMA group at Stanford has investigated and utilized this method in the additive synthesis of sounds greatly resembling the digitized recordings of actual musical instruments but involving a much reduced data base.

Again, there are probably many other useful ways to reduce the size of tables. The intrinsic nature of the information involved often suggests approaches. For instance the symmetry of a sine wave can be exploited in a trivial way to reduce the size of a sine table by a factor of four. Occasionally we need to reexamine our basic assumptions about which aspects of a table are significant. For instance a number of techniques seem to be better suited to using two sine tables, one of the usual form and one with high angular resolution for the domain of values close to zero.

Multiplication

Multiplication is an extremely useful, sometimes indispensable, operation. It is also one of the most expensive, either in time or hardware. In situations which are not limited by such factors as memory speed the presence of multiplication is often the dominant component. Much effort has been expended during the history of computing on the reduction or elimination of multiplication in various procedures. Digital signal generation at high speeds is particularly sensitive to multiplication related costs.

We might for instance attempt to apply the technique of table lookup to this problem. Our first idea might be to utilize an actual multiplication table. This is an expensive approach however since the size of such a table grows as the square of the number of possible multiplicands: a table for multiplying two eight bit bytes requires 65k of 16 bit words. If we take advantage of the commutativity of multiplication, that is the symmetry of the table, we will only reduce the table size by half.

Another approach might be to use logarithms. Dr. Dobbs' Journal recently

carried an item concerning this method:

Biblical Mathematics

We hear that Noah told the animals in his ark to multiply, and two snakes refused to do so because they were adders. Noah overcame this objection, however. He quickly placed the snakes on a rough-hewn wooden table and told them to follow his command, for, as we all know, adders can multiply by using a log table.

Unfortunately, besides requiring an exponential function table, this method will produce only approximate results since the logarithm will in general be a real number. This method might motivate us to search for other nonlinear functions with more agreeable properties since the memory cost of this method is only on the order of the number of multiplicands instead of the square.

One function we might employ is the squaring function. If we apply this function to the sum and difference of two numbers and subtract the results we will obtain the product left-shifted two places. That is, if p and q are two numbers and S is the squaring function $S(x)=x^2$ then

$$S(p+q) - S(p-q) = 4pq$$

We can further improve matters with such tricks as noting that

$$x^2 \bmod 4 = x \bmod 2$$

thus shaving two bits off of each table entry and so on. While this may not be the most appropriate multiplication method it should give some idea of the flavor of the sort of approach that is often helpful in music synthesis.

There is another interesting multiplication trick that is, for instance, suited to applying an envelope to a sinusoidal wave. It is attributed to Steve Purcell in an article by Steve Saunders of Carnegie-Mellon University. It makes use of the trigonometric identity

$$\sin(p+q) + \sin(p-q) = 2 \cos p \sin q$$

which bears a remarkable resemblance to our squaring multiplier. By means of the arc cosine function we can produce the sinusoid $\sin q$ multiplied by some amplitude a by setting

$$p = \cos^{-1} a$$

and averaging the values obtained for the points p units above and below q . It should be clear at this point that even such an apparently simple operation as multiplication may have numerous diverse realizations, each one possibly being appropriate in some situation or other. It should also be noted that these techniques are still pertinent even if we have a built in multiplier module since every multiply we save in one area can be utilized instead performing some other function in another area.

Nonlinear Synthesis

As was mentioned previously one of the really attractive features of the recently developed use of frequency modulation in the synthesis of musical sound is that we are able to get so much music at so little cost. This has stimulated the investigation of other related classes of algorithms which are coming to be referred to as *nonlinear synthesis* techniques.

There are several interesting cases which have been investigated to date: James A. Moorer of CCRMA has investigated a particular class of functions which can represent the sum of a large number of harmonics but which can be computed cheaply by evaluating a closed form expression which is equivalent to the function. These techniques are collectively called discrete summation synthesis.

Currently I am investigating a very flexible set of techniques which are based on digital waveshaping. These techniques include frequency modulation as a particular special case and are also capable of producing certain discrete summations as well. There thus appears to be a great deal of commonality between apparently disparate nonlinear synthesis techniques and a unified framework for them is gradually developing.

Detailed analysis of the implementation of these techniques is beyond the scope of this paper, but again it seems that it is likely that many cost effective ways of producing high quality musical sounds will grow out of this work. To date we have had only a very small amount of experience with actually using these methods in the production of music but initial experiments have shown them to be quite promising, and we can expect to see them eventually recast in forms appropriate to microcomputer implementation.

Other Operations

A number of other operations are of interest. Some of these are fairly familiar signal processing techniques, such as digital filtering and inverse Fast Fourier Transforms, while others are specifically acoustical, such as digital reverberation. In all cases we can implement them as a small module controlled by a microcomputer. As before the investment of some effort on analysis and design will have large payoffs.

For instance, there is a commercially available digital reverberator which performs a few more multiples than it actually has to because the original paper describing its design did not use the so called canonical form for the description of the component filters. While this may or may not be significant it does point out that almost any design can be improved or optimized in numerous ways and that by expending a little careful thought we can come up with a better, cheaper, way of getting the results we are after.

Summary

It is clear from these ruminations that the construction of effective, aesthetic, microcomputer music systems is entirely feasible. The field is a rapidly expanding one in which exciting achievements are being accomplished with great frequency. Rooted in psychoacoustics and mathematics the art of the computer musician is today bearing fertile fruit. Given a microcomputer and a little ingenuity we may participate directly in its continuing, rewarding, growth.

Referenced Readings

John M. Chowning
The Synthesis of Complex Audio Spectra by Means of Frequency Modulation
 Journal of the Audio Engineering Society v21 n7 SEP 1973 pp 526-534

James A. Moorer
Signal Processing Aspects of Computer Music - A Survey
 Computer Music Journal v1 n1 FEB 1977 pp 4-37

Max V. Mathews
The Technology of Computer Music
 MIT Press Boston 1969

John M. Grey
An Exploration of Musical Timbre
 Department of Music Report STAN-M-2

Anon.
Biblical Mathematics
 Dr. Dobbs' Journal of Computer Callisthenics and Orthodontia v1 n8 SEP 1976

Steve Saunders
Improved FM Audio Synthesis Methods for Real-Time Digital Music Generation
 Computer Music Journal v1 n1 FEB 1977 pp 53-55

A PIPE ORGAN/MICRO COMPUTER SYSTEM

Jef Raskin
Box 511
Brisbane CA 94005

One night I got a call from a man who had been wandering through the personal computer stores in the area. He was looking for a computer to operate his huge pipe organ. Inevitably he was given my phone number, since I had been going around to the same stores telling one and all about how I was working on a controller for my pipe organ.

There are lots of gimmicky reasons for wanting to attach a computer to an organ. The reason that I am interested is that the combination can provide the performer with a more flexible, easier to play instrument. And then there are all those gimmicks. As it turns out, using a microcomputer can be less expensive than conventional console wiring. Before we get into the subject too deeply, the "organization" of the king of instruments should be made clear.

The performer sits at the console. The performer's hands rest on one or more keyboards called manuals. There are usually from two to four manuals. The feet play on a set of keys placed beneath the bench, the pedals. The console, on most organs since the late 1800's, is separate from the rest of the instrument and is connected to it by means of electrical cables. As with the computer, the console is the "command center" of the instrument. Besides the keyboards there are a number of other controls on the console that will be discussed later.

The sounding portion of the organ consists of many pipes. Each pipe sounds but a single note. There are typically many different pipes for a given note, each of which has a different sound quality or timbre. A set of pipes, all of similar timbre, one for each key on a manual, is called a rank. Each rank has a name, many of which are hallowed by centuries of use. Some, like Diapason or Bourdon describe sounds that are characteristic of organs and nothing else. Others, such as Trompette or Blockflote are reminiscent of trumpets and wooden flutes respectively. Obviously one rank is a minimum for an organ (renaissance Portative organs had one rank). A small organ usually has three or four ranks, controlled from two manuals. The one being installed in my house has 26 ranks. A large organ will have seventy or more. The organ owned by my friend mentioned above has 140 ranks. That is very large, and only a few cathedrals have more.

The complexity of a pipe organ will now become apparent. Each rank has 61 pipes, as there are 61 notes on a manual. Thus for a pipe organ of a hundred ranks there are six thousand one hundred pipes. Each rank is turned off or on by a knob or switch labelled with the rank's name. These knobs are called stops (the terms stop and rank are sometimes used interchangeably, but in this discussion rank will refer to a set of pipes, and stop to the controlling knob.)

A large organ often has four manuals (names Great, Swell, Choir and Echo or Positiv) each having 61 keys, a 32 note pedalboard, and, say, 100 stop knobs, and a few dozen assorted controls. Thus there are about 500 controls that the organist must manipulate.

And now we come to the microcomputer. It must keep constant watch on 500 switches, and control some six thousand relays - one for each pipe. It must never miss a switch closure or release, and must operate the correct pipes - sometimes dozens simultaneously - within a 20th of a second. Is this within the capabilities of an 8080? As it happens, it is. But not without a bit of tricky I/O design, and some swift algorithms.

For completeness, it should be mentioned that some ranks are not exactly 61 notes. "Unified" ranks often have 75 pipes, and some special ranks have fewer than 61. But fortunately these exceptions are easily handled. The problem is simplified in some organs (a little) by sets of ranks grouped into "straight" chests where instead of each pipe having its own electrically operated valve (a "unit" chest), each rank in the chest has a valve. Then all notes of the same name (such as all C's or all F sharp's) have one valve. This loses some generality, but requires fewer valves and electrical connections. For M ranks of N notes each a "straight" chest requires M*N valves. A "unit" chest has M*N valves.

Organs also traditionally have couplers, which operate either within a keyboard, or between keyboards. An intermanual coupler has the effect of operating a note on one manual when you press a corresponding note on another. (On some old organs both keys actually move when you press one of them. This may

have given rise to "phantom of the opera" stories.) A coupler that works within a keyboard plays a note typically an octave higher or lower than the one you are playing - but on the same keyboard. Intervals other than an octave also are available on some organs.

An organ is also separated into divisions. These have the same names as the manuals, typically Great, Swell, Choir, Echo and Positiv. The pipes played by the pedals form another division: the pedal division. Each rank belongs to exactly one division. In the traditional organ a manual could only play pipes in its division. You could couple manuals together, but it was impossible to play a rank in the Swell division from the Great manual without playing all stops that were pulled in the Swell division from the Great manual. This separation into divisions has no musical benefits, it was done merely to simplify the construction of the switching in the console. The microcomputer can be used to obliterate the concept of division, thus giving more freedom to the organist who can then assign any rank to any keyboard independently. This is the first of a number of non-gimmick improvements that can be appreciated by any organist.

The switching in the traditional organ is done by the most incredible collection of electrical, mechanical and pneumatic switches imaginable. That it works at all, being made mostly of slats of wood and strips of leather with silver wires for contacts, seems dubious. It is not surprising that not every conceivable freedom in interconnection has been permitted the organist in the past.

The wiring from the console to the pipes over distances from 10 to over 100 feet reminds one of a cross between the innards of a computer before the mother-board was invented and a telephone company switching office. Another major advantage of the computerized organ is the elimination of most of this wiring. In a very large organ the cost of the computer system may be less than the cost of the cabling alone.

When an organist plays a piece, the piece (or each section of the piece) has a characteristic sound quality produced by a judiciously selected set of stops being activated. A particular collection of stops is called a registration. It is usually desirable to be able to store such combinations.

There are a number of buttons called pistons which recall combinations of stops. Logically enough these collections of stops are called combinations or presets. There are often a few fixed presets, and a number of pistons are provided whose registration the organist can change at will. Another advantage of the computer controlled organ is that many more presets will be available. Four kilobytes of memory can store hundreds of different presets, more than on any conventional organ. Four kilobytes of memory costs less than one preset done mechanically! And it's a lot easier to install.

Consider what happens when a single key is pressed. First, any keys that are coupled to it are also activated on its keyboard as well as on other keyboards. For each of those resultant keys as well as the original key, the applicable stops must be looked up. If there are two couplers and four stops activated for each of the three keyboards involved then no less than 12 pipes must sound. When playing a full chord with many couplers and stops engaged it is not uncommon for 500 pipes to be operated simultaneously.

A number of schemes were concocted for driving the pipes and reading the keys. One scheme, which has been used on smaller organs for computer control, is to have each key send out a unique code. Each pipe recognizes its own address. The computer receives key-codes as well as stop and coupler codes and computes the appropriate pipe addresses. A decoder at each pipe as well as a diode matrix or other encoder for the console is required. Since on a large organ there are over two to the twelfth pipes, even a 12-bit code would not be long enough. This would mean assembling two 8-bit words for each pipe. Putting out over 500 of these in a 30th of a second, considering the number of steps required in the program, was impossible. Furthermore, the cost for the decoders at each pipe was prohibitive. This ruled out going to a 16-bit computer, since it wouldn't help the decoder problem, and a larger word size seemed to hold few advantages in any other way.

Cost alone ruled out the brute-force approach of just using a very fast computer. Another way to get high data rates from a micro was to use direct memory access (DMA) circuitry. With

this scheme one DMA device would be scanning the keyboard continuously and putting key depressions and releases in memory. The main processor would, at its own rate, scan the keyboard image in memory and construct a list of pipes to be played or quieted. Another DMA would scan the list of pipes and control the pipes accordingly. In essence, there would be three computers sharing the same memory. They would run asynchronously, each going as fast as conditions allowed. This seemed feasible, and a DMA design effort was started - and promptly stopped when a far simpler solution was discovered.

Part of the solution was in hardware. At one extreme of decoding (as explained above) each pipe has its own decoder. It would be more efficient for each group of, say, eight pipes to have a decoder which detects its code, and then accepts the next byte as controlling eight pipes in parallel. The eight bit control byte 10001001 might mean to play the notes C, E, and G while leaving C#, D, D#, F and F# silent. This reduces the number of decoders by a factor of eight, and then operates eight pipes at a time. This was fast enough in the I/O department, but the time required to assemble the control bytes by masking or rotation was too great. A micro handles bytes like crazy, but manipulating individual bits inside them is not nearly as fast. A number of algorithms were considered, but it was apparent that they weren't even in the ball park.

At the other extreme from a decoder for each pipe is the idea of having no decoders whatever. This idea was put forward early in the design effort, but was discarded as ridiculous. In the end it became clear that the idea was not only feasible, but fast and cheap to implement in hardware. It also made the software much easier to design. It works like this: A very long serial-in/parallel-out shift register is made. It will have at least one output for each pipe. Using available 8-bit shift registers the 140 rank organ's 2000 electrically operated valves require about 250 shift registers. (The 7000 pipes require only 2000 controlling lines since most of them are on straight chests.) In effect we build a 2000+ bit shift register - sort of a long skinny tube through which ones and zeros flow in single file. When all the ones and zeros (standing for pipes sounding or silent) reach their correct positions a command (strobe) is sent operating all the pipes at once. If the process is to take 1/30th of a second, the shift register has to move 2000 bits in that time. But this is a rate of only 60000 bits per second (60KHz) which is within the capabilities of both the shift registers and the computer. Remember that these calculations are for a mammoth size organ. Most organs are significantly smaller and the problems are correspondingly easier.

A similar approach is used for the keyboards. There exists a 33 input parallel to serial converter made for electronic organs. Just two of these IC's would suffice to encode an entire manual. The 500 controls could be transmitted serially to the computer in 1/200th of a second at 100 KHz. The interface would require fewer than 20 "critters". Again, this is for a huge organ. My own home organ would require only 10 chips for its console.

The hardware can now be summarized. A 500 bit parallel-to-serial converter for input, a 2000 bit serial to parallel converter for output, one input port and one output port are required. Each pipe also needs a power transistor to handle the .5 amps at 14 volts required by the valves (this is a typical figure). Some of the larger pipes might require two stages or a darlington power transistor, but there is no real difficulty in the design. Another side benefit accrues at this point: Many pipe organs use electro-pneumatic valves for each large pipe. This is because an all-electric valve opens too suddenly. To solve the problem the traditional builders had the electrical valve let air into a small bellows that in turn operates the valve that lets air into the pipe. A pair of R-C networks and a diode in the base circuit of the power amplifier for each pipe can give the desired slow attack and release usually obtained by the much more expensive and problematical pneumatic system. This can amount to savings of over \$1000 in a large organ. It should be mentioned that some organ manufacturers have been successful in making satisfactory all-electric valves with appropriate attack and decay curves. They would not require the R-C networks.

Software design was as gradual as the hardware design. There were two breakthroughs necessary before it was clear that the 8080 could work quickly enough. When the design was being done, by the way, the Z-80 and other faster processors were not in production. But the constraints of the 8080 and the very large organ forced a much tighter and cleverer design than would have been developed had we had more powerful computers and a smaller organ. Given the newer computers, of course, larger and more complex pieces of equipment can be controlled. Many industrial plants have fewer than 500 sensors and 2000 elements that need to be operated in real time. A microcom-

puter using the techniques outlined here could handle them.

The program begins by sweeping in the console settings. To save time only one bit per word is used. This wastes 7/8ths of 500 words, but memory is cheap. The same trick will be used in output, eliminating the necessity to pack bits into bytes. Thus over 2K bytes will be sacrificed to gain speed. There goes all of \$40 in memory costs. It buys us at least a factor of 5 in speed. It is worth it. While it now seems obvious that this is the way to proceed, it somehow took 4 months to find the solution. This is probably because we are so used to trying not to waste memory. A pipe organ costs from \$20,000 to whatever you care to spend (a million dollars in not unusual). The computer costs are lost in the small change.

The input and output loops are very simple. Point to a memory location. Do an input (or output). Move the contents of the accumulator to that location. Increment the location. Check for done. If not done do an input (or output). And so on. The loop can be done on an 8080A at 66KHz. Thus the entire organ can be updated in 0.03 second. The worst we could live with would be 0.05 second. The 8080 with a 2MHz clock is just fast enough. An 8085 or Z-80 CPU would be plenty fast. Again remember that for a reasonable size organ the plain old 8080 would be more than fast enough, and that we are discussing a worst case design. Any Poly-88 or ALTAIR or IMSAI or SOL computer could easily handle most pipe organs.

Even with I/O solved there still remains the problem of deciding which pipes are to go on and which to go off. At first this was a stumbling block in terms of the time it would take to do the computations. On each console scan, it seemed, a table of couplers would have to be made up, as well as a table of stops. A key depression, through the couplers, results in a number of "virtual" key depressions. Since some virtual keys, being higher or lower on the keyboard than the original key, will go off the end of a keyboard, they must be deleted from the virtual key list. The remaining keys have then to be processed through the stop list to determine which pipes are to be played. Since the input routines take a total of about 0.037 seconds the processing must take less than .013 seconds. Just the checking for out-of-range virtual keys would take longer than this.

The easiest solution to the coupler spill-over problem is to include a few extra places in the shift register on both ends of each rank. This allows all the ranks to have the same shift register length whatever the actual number of pipes. The first advantage is that out-of-range virtual key depressions need not be checked for since they fall into unused sections of the shift register. As with the wasted memory, the cost of the unused shift registers is small. The second advantage is that the electronics for every rank, of whatever kind, can be mass-produced. This makes it less expensive to build - as well as making the software easier to write (aside from merely being faster).

Aside from satisfying traditional organists, there is no reason to have intermanual couplers on a computer-controlled pipe organ. The original reason for including intermanual couplers was to minimize the limitations imposed by the separation of the pipes into divisions. The computer, by being able to assign any rank to any manual (or the pedals) eliminates entirely the need for these couplers, as has been pointed out. The presence of intermanual couplers is retained mainly as a sop to traditional organists. It might well be eliminated in my own organ.

Another choice to be made is whether to recalculate all the pipes to be played, or to just modify the previous state on each cycle. It was decided to recalculate from scratch each time to eliminate the possibility of cumulative error. It also means that keybounce is automatically taken care of. In the slight time between updates an organ pipe cannot even begin to sound. A spurious signal for one cycle is effectively ignored. Continuous pipe-on instructions emitted over a period of something like .1 second or more are required before the slow mechanical valves can react. The higher pitched pipes respond quickly, incidentally, and the low pipes sometimes take nearly a second to begin playing. Organists learn to compensate by playing low notes somewhat early. Without introducing a constant across-the-board delay it does not seem possible to have the computer compensate for the effect, but it is a place where some experimentation might be interesting. Experienced organists, of course, might look askance at such an innovation, but they needn't be told about all our ideas.

To summarize: A cycle of the computer-organ system starts by pulling in the state of the console. The second part of the cycle (yet to be described) calculates the pipes that should be playing given the state of the console. The third portion of the cycle sends the pipe commands along the shift register. The process is repeated at least once every 30th of a second.

A coupler (of whatever kind) is merely a displacement, which

is easily calculated since all keyboards and ranks are the same nominal length. Likewise engaging a stop is also a displacement of a distance equal to the difference between the bottom of the manual's image in memory to the bottom of the rank's image in memory. Thus these displacements or offsets can be simply added to yield the offset for a combined coupler/stop setting.

An example, with a simplified organ, will demonstrate how the algorithm operates. Say there is one manual with 10 keys numbered 1 through 10. They are read into memory locations (all numbers will be in base 10 for this discussion) 1001 through 1010. The ranks each have 10 pipes, and there are two of them. The first rank is stored in locations 2001 through 2030. The second rank is stored in locations 3001 through 3030. Remember that the area set aside for each rank is larger (here three times as large) as the actual space necessary. There are two stop switches, stored in locations 4001 and 4002. There are two couplers. They are stored in 4003 and 4004. The first couples up five pipes high, the second down three pipes low.

When the low order bit in 4001 is on (or, as they say, high) the program adds 2010 to the key address to get the pipe address. When the other stop is on 3010 is added. If the first coupler is on an additional 5 is added, the second coupler subtracts 3 (or adds a negative three - it is all the same thing). The addition is done but once. Say the first stop and the second coupler were operated, then given a key on at location 1005, to get the proper location to turn the pipe on merely add (1010-3) or 2007 to the key location (1005+2007)=3012. Indeed this is the correct pipe.

This brings up another possible freedom given by use of a microcomputer. When a stop and a coupler are operated, one gets both the note given by the stop and the extra note given by the coupler. With a computer it would be possible to give just the note given by the coupler acting on that stop. Since each stop could have a whole panoply of couplers attached to it, the number of buttons would soon become unworkable. For complete flexibility the organist would have to be provided with a keyboard and display. One would play the organ by setting up many required presets, with whatever degree of flexibility required, and then the easily hit tabs would not activate stops, but would bring in the organist's choice of registrations.

In the example above choosing both stops and both couplers would result in having to add six numbers to each key location to obtain the pipe location. In the actual implementation the program would, for each manual, do the following:

1. Scan the list of stops, and make a table of addends.
2. Scan the couplers, and add them to each stop, extending the list of addends.
3. Add the addends to the locations of that manual that contain a 1 (meaning a key depression).
4. Turn on the low-order bit in the indicated word in the pipe image.

Inter-manual couplings look just like any other kind of coupling. Say that one manual is stored in 1001 through 1010, and another manual at 1201 through 1210. The coupling the first manual to the second manual merely means adding 200 to the locations of the first manual. Just which inter-manual couplers will be allowed must be carefully specified. If anything is allowed we get the following cat-chasing-its-tail effect: Manual 1 is coupled to manual 2 at the same pitch. Manual 2 is coupled to manual 1 one key higher (a "half step" higher in musical terminology). Press "C" on manual 1. "C" gets played on manual 2. This forces "C#" on manual 1. But this makes "C#" play on manual 2.... Every key is thus being played. As hinted at above, intermanual couplings are only necessary on organs where the pipes are separated into divisions. In the computer controlled pipe organ they can and should be eliminated. Everything they can do, and more, can be done by freely assigning ranks to keyboards as desired.

I am not sure that all organists will be convinced by this.

The organ console of the future, as it appears in the light of the computer mediated organ, looks like this: The manuals and pedals are built to the usual AGO (American Guild of Organists) standards. These standards are excellent, and permit an organist to travel from one instrument to another with a minimum of relearning. Instead of the usual arrangement of stops, there are as many rows of stops as there are keyboards. When a stop is to be assigned to a given keyboard, the button in the row representing that keyboard, and in the column representing that stop, is pressed. Done. Any particular registration may be captured by pressing the "capture" button and while holding it operating the chosen preset button. The stop buttons should, as on conventional organs, move (or light up) to show what choices have been made. This is not far from conventional practice.

The possibilities in a console screen, with alphabetic read-out, are endless, and would require another paper this length to explore. Similarly the gimmicks - from very useful ones that record (on a disc or cassette) the performance in terms of keystrokes - to silly ones (for example connecting the doorbell to the computer, so that the organ plays "Jesu, Joy of Man's Desiring" when a visitor unwittingly presses the door button) again would take up too much space for this paper.

The reasons, in short, for using a computer in a pipe organ are these:

1. Simplification of the wiring of the organ.
2. Greater reliability than conventional switching.
3. Lowered expense in medium and large instruments.
4. Much greater control of the instrument by the performer.
5. New freedoms in choosing registrations.

Nothing, it would seem, is lost by going to a microcomputer, and one could keep advantages 1, 2, and 3 above while keeping the appearance and operation of the pipe organ unchanged, in case some organist were so hidebound as to not want advantages 4 and 5.

Acknowledgements: Certainly at least half of the ideas expressed in this paper are due to my friend and colleague, Doug Wyatt. All of the ideas were developed in collaboration with him. Thanks are also due to Jim Brennan, who owns the incredibly large organ so often mentioned in this article, and whose co-operation and inspiration have been essential to the project.

A COMPUTER CONTROLLED AUDIO GENERATOR

Thomas E. Olsen
2500 Medallion Drive, No. 57
Union City CA 94587

ABSTRACT

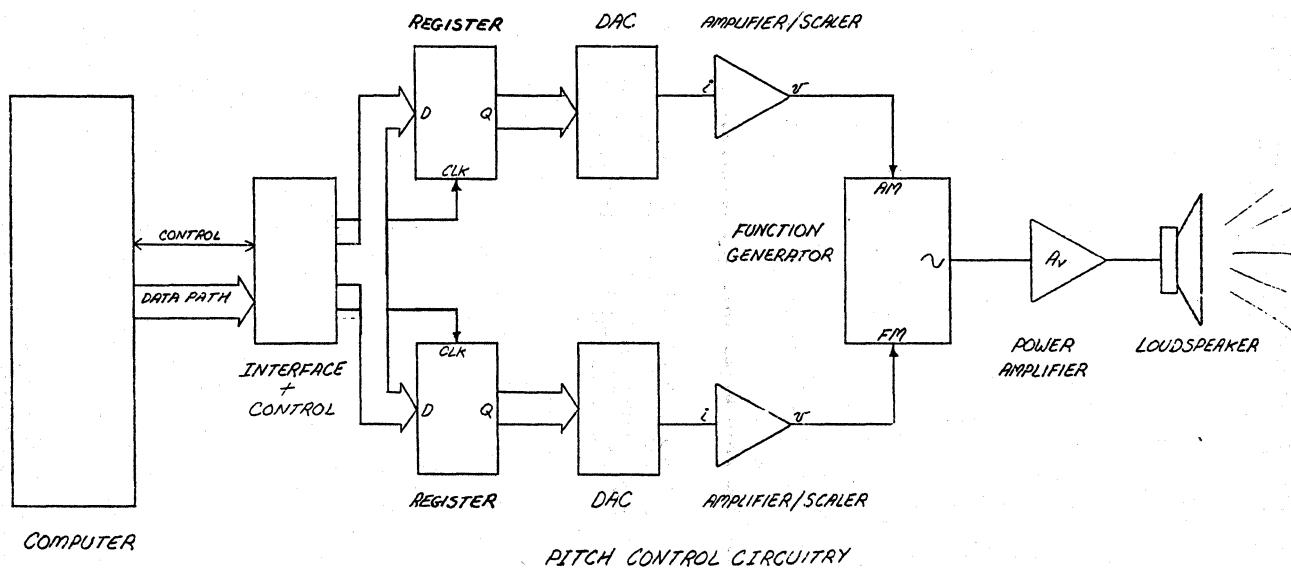
THIS PAPER SHOWS ONE APPROACH TO BUILDING A PIECE OF HARDWARE WHICH ENABLES A COMPUTER TO PRODUCE SOUNDS THROUGH A LOUDSPEAKER. THE DEVICE DESCRIBED ENABLES TWO PARAMETERS TO BE DYNAMICALLY CONTROLLED BY THE COMPUTER: PITCH (FREQUENCY) AND AMPLITUDE (LOUDNESS). THE PAPER DISCUSSES THE GENERAL ARCHITECTURE OF THE DEVICE, BUT DOES NOT GIVE ANY CIRCUIT DIAGRAMS SINCE EVERY HOBBYIST WILL PROBABLY WANT TO DESIGN IN HIS OWN OPTIONS. SPECIFIC KEY INTEGRATED CIRCUITS ARE DISCUSSED AND RECOMMENDED FOR THIS APPLICATION. DATA SHEETS ARE INCLUDED. POSSIBLE ENHANCEMENTS TO THIS DEVICE ARE ALSO DISCUSSED.

INTRODUCTION

A SOUND GENERATOR IS A VERY USEFUL DEVICE TO HAVE CONNECTED TO YOUR COMPUTER. FOR MUSICIANS WHO HAVE DABBLED IN ELECTRONIC MUSIC, THE UNLIMITED POSSIBILITIES ARE OBVIOUS. FOR THE REST OF US, JUST BEING ABLE TO MAKE WEIRD NOISES IS OFTEN JUST AS EXCITING. STAR TREK WITH SOUND EFFECTS IS ONE OF MY FAVORITE APPLICATIONS BECAUSE IT ADDS A GREAT DEAL OF REALISM TO THE GAME. ANOTHER IMPORTANT REASON FOR ADDING SOUND TO YOUR SYSTEM IS THAT IT ENABLES THE LAY PERSON TO QUICKLY APPRECIATE WHAT YOUR COMPUTER CAN DO. ANYBODY CAN UNDERSTAND "MUSIC" JUST BY HEARING IT. UNDERSTANDING SOME CRYPTIC PRINTOUT IS OFTEN QUITE ANOTHER STORY. FINALLY, GENERATING COMPUTER MUSIC IS A GREAT WAY TO DRIVE YOUR NEIGHBORS CRAZY.

WHAT IT IS

THE COMPUTER CONTROLLED AUDIO GENERATOR IS A DEVICE WHICH YOUR AMPLITUDE CONTROL CIRCUITRY



PITCH CONTROL CIRCUITRY

PROGRAMMABLE AUDIO GEN. BLOCK DIAGRAM THOMAS E. OLSEN 2-23-77

COMPUTER OUTPUTS DATA TO. THE DATA THEN CONTROLS THE AMPLITUDE AND FREQUENCY OF A WAVEFORM PRODUCED BY A VOLTAGE CONTROLLED OSCILLATOR. THIS WAVEFORM IS THEN CONNECTED TO YOUR STEREO SYSTEM SO THAT YOU CAN HEAR AND/OR TAPE RECORD IT.

HOW IT WORKS

REFER TO THE BLOCK DIAGRAM THROUGHOUT THIS DISCUSSION.

THE COMPUTER OUTPUTS DATA TO ONE OF TWO OUTPUT PORTS. ONE PORT IS USED TO CONTROL THE FREQUENCY OF THE TONE PRODUCED, WHILE THE OTHER PORT IS USED TO CONTROL THE AMPLITUDE OF THE TONE. THE DATA PATH IS 8 BITS (1 BYTE) WIDE. THIS MEANS THAT UP TO 256 DIFFERENT PITCHES AND AMPLITUDES ARE SELECTABLE BY THE COMPUTER. THERE IS NO INHERENT REASON WHY THE DATA PATH COULD NOT BE WIDER AND THUS PRODUCE BETTER RESOLUTION. THE MAIN PROBLEM IS THAT THE DIGITAL TO ANALOG CONVERTERS START GETTING A BIT MORE EXPENSIVE WHEN THEY ARE WIDER THAN 8 BITS.

THE BLOCK LABELED INTERFACE AND CONTROL MUST PERFORM SEVERAL FUNCTIONS. IT MUST BUFFER THE DATA BUS WITH SOME TYPE OF BUS RECEIVERS, AND IT MUST BE ABLE TO DECODE WHEN A BYTE IS BEING SENT OUT TO IT. AT THIS TIME, IT PRODUCES THE CLOCK SIGNALS THAT LATCH THE DATA BYTES INTO THE APPROPRIATE REGISTERS.

THE TOP REGISTER OR LATCH IS USED TO HOLD THE AMPLITUDE CONTROL BYTE. THE BOTTOM REGISTER HOLDS THE FREQUENCY CONTROL BYTE. THE OUTPUTS OF THESE REGISTERS FEED THE DIGITAL TO ANALOG CONVERTERS (DAC). THE DAC'S PRODUCE A CURRENT WHICH IS DIRECTLY PROPORTIONAL TO THE MAGNITUDE OF THE BYTE AT THEIR INPUTS. THESE CURRENTS ARE THEN FED INTO OPERATIONAL AMPLIFIER NETWORKS WHICH ARE RESPONSIBLE FOR CONVERTING THE CURRENT TO A VOLTAGE, AND SCALING THAT VOLTAGE SO THAT THE DESIRED VOLTAGE SWEEP RESULTS WHEN ALL POSSIBLE BYTES ARE OUTPUT TO THE PORT. DESIGNING THIS NETWORK SO THAT IT WORKS PROPERLY WITH THE FUNCTION GENERATOR IS THE MOST DIFFICULT PART OF THE DESIGN. IT WILL BE NECESSARY TO KNOW HOW TO DESIGN WITH OPERATIONAL AMPLIFIERS IN ORDER TO SUCCESSFULLY IMPLEMENT IT.

THE PROPERLY SCALED AMPLITUDE AND FREQUENCY CONTROL VOLTAGES ARE APPLIED TO THE FUNCTION GENERATOR INTEGRATED CIRCUIT. THIS DEVICE THEN PRODUCES THE WAVEFORM WHICH IS THEN AMPLIFIED AND HEARD. THE WAVEFORM CAN BE SELECTED AS EITHER SINE, TRIANGLE, OR SQUARE WAVE.

DESIGN DETAILS

EVERYTHING UP TO THE DIGITAL TO ANALOG CONVERTERS IS PROBABLY A PIECE OF CAKE TO MOST COMPUTER HOBBYISTS WHO WOULD UNDERTAKE THIS KIND OF PROJECT. ACCORDINGLY, I WILL NOT DISCUSS THESE SECTIONS.

THE DIGITAL TO ANALOG CONVERTER WHICH IS THE LEAST EXPENSIVE AND EASIEST TO OBTAIN IS PROBABLY THE MC1408. THE DATA SHEET SHOWS BASICALLY HOW TO CONNECT IT. THE INPUTS ARE TTL COMPATIBLE

AND THE SINGLE OUTPUT IS A CURRENT. THE MAGNITUDE OF THIS CURRENT CAN BE CONTROLLED THROUGH THE PROGRAM RESISTOR. BE SURE TO OBSERVE THE DIRECTION OF THIS CURRENT WHEN DECIDING HOW TO CONNECT IT TO YOUR OP-AMPS.

BEFORE DESIGNING THE AMPLIFIER/SCALER SECTION OF YOUR AUDIO GENERATOR, YOU NEED TO KNOW WHAT CONTROL VOLTAGE RANGES YOU WILL NEED TO SUPPLY TO THE FUNCTION GENERATOR CHIPS TO PRODUCE THE DESIRED RESULT. THE BEST IDEA IS TO PLAY WITH THE FUNCTION GENERATOR BY ITSELF FOR A WHILE BEFORE CONNECTING IT TO THE REST OF THE SYSTEM. CONNECT TWO POTS, AS VOLTAGE DIVIDERS, TO THE VOLTAGE CONTROL INPUTS OF THE FUNCTION GENERATOR AND BASED ON THE DATA SHEET, CHOOSE AN R AND C THAT WILL PUT YOU IN THE BALL PARK FREQUENCY-WISE. YOU WILL HAVE TO DECIDE WHAT DYNAMIC RANGE YOU WILL WANT FOR YOUR AUDIO GENERATOR. MINE CAN BE SWEEPED FROM 3 HZ TO ABOUT 10 KHZ. BY THE TIME YOU ARE DONE PLAYING WITH THE FUNCTION GENERATOR, YOU SHOULD HAVE CHOSEN A GOOD VALUE FOR THE R AND C AND YOU SHOULD KNOW WHAT VOLTAGE RANGE YOU WILL NEED TO SWEEP THE AM AND FM INPUTS OVER IN ORDER TO OBTAIN THE EFFECT YOU DESIRE. THESE VOLTAGE RANGES SHOULD BE SOMETHING LIKE .5 VOLTS TO 12 VOLTS IF YOU USE THE EXAR CHIP. TRY TO AVOID RESORTING TO VOLTAGE RANGES WHICH CROSS THROUGH ZERO. THERE ARE TWO FUNCTION GENERATORS WHICH I HAVE TRIED FOR THIS APPLICATION. ONE IS THE INTELSIL 8038 AND THE OTHER IS THE EXAR XR-2206. BOTH OF THESE ARE READILY AVAILABLE (JAMES ELECTRONICS). I FOUND THE EXAR CHIP INFINITELY SUPERIOR AND EASIER TO DESIGN WITH FOR THIS PARTICULAR APPLICATION. ALSO, THE EXAR CHIP HAS AN AMPLITUDE MODULATION INPUT ALREADY THERE. YOU HAVE TO PLAY GAMES WITH THE 8038 CHIP TO ACHIEVE AMPLITUDE MODULATION.

REMEMBER THAT YOU ONLY GET 256 FREQUENCY STEPS IF YOU ARE USING AN 8 BIT DAC. THUS, IF YOU WANTED TO BE ABLE TO SWEEP THE AUDIO SPECTRUM FROM 100 HZ TO 10 KHZ, YOUR RESOLUTION WOULD BE:

$$(10,000 - 100)/256 = 38 \text{ Hz}$$

IF THIS IS NOT ACCEPTABLE, YOU WILL HAVE TO USE EITHER A WIDER DAC, OR SEVERAL 8 BIT DACS WITH THEIR OUTPUTS SUMMED BY THE OP-AMP NETWORK.

THE FUNCTION GENERATOR GIVES YOU THE OPTION OF SELECTING THE SINE, TRIANGLE, OR SQUARE WAVE OUTPUTS. THE ADVANTAGE TO USING TRIANGLE OR SQUARE IS THAT YOU GET HARMONICS WHICH ARE NOT FOUND IN THE SINE WAVE. THE OUTPUT OF THE GENERATOR SHOULD BE CAPACITIVELY COUPLED TO YOUR AUDIO AMPLIFIER IF THERE IS A NON-ZERO AVERAGE DC VOLTAGE IN THE OUTPUT.

ONE ADVANTAGE OF THE XR-2206 IS THAT IT HAS AN FSK INPUT. THIS COULD BE USED TO IMPROVE YOUR RESOLUTION WITHOUT REQUIRING A WIDER DAC. THE FSK INPUT ENABLES YOU TO PROGRAMMABLY SELECT EITHER OF TWO DIFFERENT TIMING RESISTORS. YOU COULD USE ONE RESISTOR FOR THE RANGE 0 HZ TO 5 KHZ, AND THE OTHER FOR THE RANGE 5 KHZ TO 10 KHZ. THIS WOULD THEN IMPROVE YOUR RESOLUTION BY A FACTOR OF TWO.

PROGRAMMING THE THING

THERE ARE AN INFINITE NUMBER OF WAYS TO PROGRAM THIS DEVICE. A GOOD COMBINATION OF EXPERIMENTATION, IMAGINATION, AND PATIENCE WILL CERTAINLY YIELD SOME VERY INTERESTING AUDIO EFFECTS. OF VITAL IMPORTANCE IS A GOOD MONITOR PROGRAM THAT LETS YOU DYNAMICALLY AND RAPIDLY CHANGE PARAMETERS. THE SYSTEM THAT I USE TO WRITE MUSIC IS SIMILAR TO "FORTH" AND IS EXTREMELY POWERFUL FOR THIS TYPE OF APPLICATION.

ONE THING TO KEEP IN MIND IS THAT THE COMPUTER IS MUCH FASTER THAN AUDIO WAVEFORMS. THIS ENABLES YOU TO RAPIDLY CHANGE OR SWEEP BOTH THE PITCH AND AMPLITUDE CONTROL VoltAGES AND THUS PRODUCE A VARIETY OF INTERESTING EFFECTS. REMEMBER, THE AMPLITUDE CONTROL PART OF THE CIRCUIT IS NOT INTENDED TO REPLACE THE VOLUME CONTROL ON YOUR STEREO!!! IT IS USED TO GENERATE THE DESIRED ENVELOPE OF A TONE BURST AND IT IS VARIED OFTEN, NOT ONLY WHEN THE NEIGHBORS START BANGING ON THE WALLS.

POSSIBLE ENHANCEMENTS

THE CONCEPT OF USING A CONTROL VOLTAGE TO CONTROL AUDIO EQUIPMENT IS NOT NEW. IN FACT, THIS IS HOW THE MOOG, ARP, AND BUCLA SYNTHESIZERS WORK. THE COMPUTER BEING ABLE TO RAPIDLY AND REPEATABLY CONTROL THESE VoltAGES IS WHAT IS RELATIVELY NEW.

ONE POSSIBLE ENHANCEMENT WOULD BE TO PUT A VOLTAGE CONTROLLED FILTER (ASSUMING YOU KNOW HOW TO DESIGN ONE) IN SERIES WITH THE SQUARE WAVE OUTPUT OF THE FUNCTION GENERATOR AND THE INPUT TO THE POWER AMPLIFIER. A THIRD CONTROL VOLTAGE COULD THEN BE OBTAINED FROM A THIRD PORT TO CONTROL THE PARAMETERS OF THE FILTER.

ANOTHER IDEA WOULD BE TO BUILD SEVERAL AUDIO GENERATORS SO THAT CHORDS COULD BE PLAYED.

CONCLUSION

IF YOU BUILD THIS DEVICE, I THINK YOU WILL FIND, AS I DID, THAT THE POSSIBILITIES ARE MIND BOGGLING. MAKING THE DEVICE WORK IS TRIVIAL COMPARED TO THE TASK OF PROGRAMMING IT TO PRODUCE PLEASING "MUSIC". IF ONLY I WERE BOTH A MUSICIAN AND AN E.E. !!!!

DIALNET AND HOME COMPUTERS

John McCarthy & Les Earnest
Artificial Intelligence Laboratory
Computer Science Department
Stanford University
Stanford CA 94305

1. The Dialnet Concept

Dialnet will be a set of protocols like the ARPAnet protocols that will enable an on-line computer user to send messages to users of other computers, to transmit files between his own and other computers, and to use other time-shared computers directly - all using the facilities of the ordinary dial-up telephone network. His computer will need a suitable modem and must implement the Dialnet protocols in its operating system. The Stanford University Artificial Intelligence Laboratory plans to undertake an eighteen month study and experimental implementation of the protocols, with support expected from the National Science Foundation. While we expect the main users of *Dialnet* to be time-sharing systems, we hope the protocols will be implementable by single user computer systems down to the level, if it proves possible, of hobbyist computers. We call the system Dialnet by analogy with ARPAnet, but unlike the ARPAnet, it requires no administrator to "admit" new members; they need only know each other's telephone numbers.

The ARPAnet connects several hundred computer facilities involved in Defense Department supported research and allows users of one system to log in on others, allows transmission of messages between users of different computers, and allows the transfer of files between computers. More generally, it allows interaction among programs in different computers.

These facilities have proven valuable in permitting collaboration between computer scientists at different sites and in permitting nationwide access to unique facilities such as the MACSYMA system for computing with algebraic and analytic expressions at M.I.T. It permits a new form of publication in which documents are kept in the computer, are continuously updatable, are immediately accessible throughout the country, and in which comments from readers are accessible to other readers.

The usefulness of the ARPAnet has prompted many non-defense installations to try to connect to it, and in some cases this has been possible, but usually the institutional and financial obstacles have been insuperable. The main financial obstacles are the need for a dedicated computer called an IMP costing about \$80,000 at each site and the need for dedicated communication lines rented by the Department of Defense at great expense from the telephone companies. Some other networks have been started, but they generally don't offer the range of services that we desire.

We propose to design protocols that can be implemented at any time-shared computer installation or single user computer system without joining any formal network. The hardware cost will be from \$500 to \$5000 depending on the type of system and how difficult it is to connect devices to the computer. For timesharing systems, a telephone dialer will be rented from the telephone company so that the system can initiate calls. For single-user systems where economy is paramount, the user can do his own dialing to initiate a call. There will be programs to transmit signals and information according to the protocols. Any installation implementing the protocols will be able to communicate with any other. The only disadvantage compared with the ARPAnet will be lower speed.

Like ARPAnet, Dialnet will be most useful to full time-sharing systems or single user systems that operate 24 hours and have file systems. In such systems, each user has named disk files that are kept in the system even when he is absent (and therefore remotely accessible), and new files can be created by file transfer from other machines and on receipt of messages. The usefulness of the message facilities normally requires that users habitually log in each working day and are most beneficial when users have individual display terminals in their offices. Further benefits accrue when reports are normally prepared at terminals and when secretaries use terminals for letters and messages. However, many less advanced installations have found the ARPAnet useful and more and more systems are acquiring economical full time-sharing capability.

While we expect that the first users of Dialnet will be regular computer users, the corresponding ARPAnet facilities have been used by non-computer people. Users of Dialnet facilities will not be required to know how to program, and we expect increasing use by others as terminals become more widespread.

In order to make the picture more concrete, here is a scenario of the use of the system suitable for scientists. Other potential users may imagine their own scenarios. The syntax contained in the scenario is not a proposal; we will have to think much more before we have such a proposal.

2. Scenario

A user named Smith types on his terminal

```
mail Organik
Do you have any active work there on human red cell carbonic
anhydrase B?
```

The system looks up Organik in Smith's correspondent file and discovers that his computer pseudonym is "NAT" at a computer called UTEX-CHEM1 that is reached at 512 471-3221 via a 1200/150 baud asynchronous modem. It selects an outgoing line with a matching modem, dials the number and attempts to transmit the message. If the transmitting computer cannot elicit a response from the desired recipient, it informs the user that it will try again later and send him a message when the transmission has succeeded. If the user's correspondent file did not contain the telephone number and modem characteristics, the user would have to supply them.

The identity and location of the sender and date and time of the message are automatically placed at the front of the message. At the receiving end, if the addressee is logged in on the computer, he is immediately informed that mail has arrived and from whom. If not logged in, he will receive the message the next time he logs in. In either case, he can use the same facility to respond:

```
mail Smith
David Piranha (DAVE@UTEX-CHEM3) has a student working on
inhibition by anions of anhydride B.
```

Following up on this lead, the user types

```
link dave@utex-chem3
```

A connection is made to the specified computer and, if DAVE is logged in, he immediately receives a message saying

```
** Link request from Smith @SU-CHEM7 **
```

He could then type "link" and have his keyboard and display effectively linked to those of the caller, permitting a conversation.

Let us suppose, however, that DAVE is not logged in and the caller is so informed. He then types

```
locate dave@utex-chem3
```

which obtains the following information from the specified computer:

```
David Piranha last logged out at 23:47 on 9 May 1976. Plan: I
will be out of touch May 10 through 16. I plan to visit Martin
Shumway at the University of Utah on May 17 and should return by
May 18. Will check mail from Utah.
```

Noting that the current date is May 14, so that there is no point in getting the message there quickly, Smith types

```
night mail dave@utex-chem3
I am interested in your work on anhydride B. If possible, give
pointers to online documentation, else give me a call at 415 497-
4430 (Stanford) or 415 321-7580 (home).
```

The "night mail" command causes the message transmission to be deferred until inexpensive nighttime telephone rates are in force.

Additional capabilities of the Dialnet system can be used to follow up on the above inquiry, as follows.

- The ability to access remote text files will be provided (with permission of the owners required, of course). This interactive reading facility will include the addition of "footnotes" to various parts of the text. These footnotes may be declared private (i.e. belonging to the reader) or public (available to the author and possibly others).
- It will be possible to run programs on a remote computer, permitting experiments with programs developed in other places. This facility will permit the sharing of unique specialized capabilities over a geographically distributed population.
- File transfers will be permitted, with suitable error detection and correction features, to permit sharing of data. The communication protocol should be able to adapt to a wide range of noise conditions on phone lines.

3. Protocols

In order to make these facilities available, suitable protocols must be designed, and in the course of this, a number of technical problems must be solved. Besides the protocols themselves, which are communication procedures and data structures, there will be a recommended set of terminal-level commands with syntax prompting and standard error messages.

We believe that we have the experience to produce a set of workable protocols, and that it is better to start with an implementation than to standardize something that doesn't exist. The latter procedure in recent years has led to gold-plating the requirements to the extent that the standard is not implementable.

We plan to devise suitable protocols, test them at a few sites, publish them, and attempt to convince other installations to implement them. Almost certainly, initial experience will produce a requirement for changes, and standardization committees will be formed and set to work. A likely forum for a standardization effort would be through the ACM to the American National Standards Committee.

We propose to allow interaction with ARPAnet sites via TIPs and propose to discuss with ARPA and DCA whether this will be allowed.

The most general use of Dialnet involves a program in one computer "waking up" and interacting with a program in another machine. Dialnet protocols will handle human messages as a subcase of this, taking into account the fact that the subcase will have the most application for a long time to come. Messages about where to deliver a message sent by one time-sharing system to another will be handled as a special sort of message that one program may send another in cases where the two programs are not written together, but each must know a certain "public" language. Thus we will attempt to make a general format for requests, questions, and assertions suitable for communication between computer programs. We will study how to make this mesh with communication between computer programs and people.

4. Research Issues

There are many research issues, and we don't expect to settle all of them in the time and with the resources requested in this proposal. Since we expect many of the issues will be clarified by the initial implementation, we will concentrate on getting a reasonable first implementation into experimental use.

Here are some of the issues we will study:

1. What error correction facilities are required to make up for the deficiencies of telephone lines?
2. What is the minimal necessary burden on the time-sharing computers carrying out the communication? What is the trade-off between buffer size and compute time?
3. Can dial-up telephone communication rates meet most of the needs for communication between computers belonging to different research organizations?
4. What is the best way to handle the fact that different modem speeds have different prices? Should one strive for a standard speed or can a wide variety be easily accommodated? Is the time ripe for a micro-processor based modem that can communicate at any speed up to a maximum and adjust its speed to the requirements of the line or the possibly less advanced modem with which it communicates?
5. How will the improved communication affect research? Since changes will be slow, how can we tell as early as possible what the effects will be?
6. What style of interaction is convenient for both experienced and inexperienced users? How can communication programs be made self-teaching without being cumbersome?

5. Research Plan

We plan to undertake this project with rather modest staffing. Initial emphasis will be on designing and implementing experimental protocols using existing computer facilities at Stanford. We will also rely heavily on the co-operation of other organizations that have expressed interest in the project both in determining the protocols and in implementing them for specific machines. First, we plan to create an experimental link between the computer facilities of the Stanford Artificial Intelligence Laboratory (SAIL) and the Low Overhead Timesharing System (LOTS). The only additional equipment needed will be a telephone port with autodial capability for the LOTS computer. We expect this initial development phase will take about 6 months.

In the following six months, we plan to test, evaluate, and modify the protocols. During the latter part of this period, we plan to publish the protocols and encourage additional groups to join the Dialnet community.

Note: This document is taken from our NSF proposal and retains some of that flavor.

For further information contact Lester Earnest or John McCarthy at Stanford Artificial Intelligence Laboratory, Stanford University, Stanford, California 94305; ARPANET addresses: EARNEST @SU-AI and MCCARTHY @SU-AI.

This document is DIALNE.BLA/W77,JMC) @SU-AI.

CB COMPUTER MAIL?

Raymond R. Panko, Ph.D.
Stanford Research Institute
333 Ravenswood Ave.
Menlo Park CA 94025

During the 1976 election campaign, Jimmy Carter coordinated his staff through a new communication medium called "computer mail." Whenever Carter's campaign aircraft landed, an aide would call into the nationwide STSC computer network and call up a program named Mailbox. The aide would then transmit messages prepared during the flight, receive new messages, and perhaps reply to a few incoming messages (1). During the Autumn campaign, over 1,200 messages were sent.

The STSC Mailbox program is only one example of a relatively new computer application area: the use of computers in human communication. There is nothing new about computer-based human communication, per se, of course. Teletypewriter networks have long used computers for switching, and even the first time-shared computer had a primitive mailbox program (2), to let users exchange brief messages. What is new is the rate at which computer-based human communication systems have been growing in sophistication and power during the last five or six years.

Historical Trends

Although the STSC Mailbox system that Carter used is a commercial system, most computer mail development has been paid for through research grants and, most applications have been in research and military environments.

The greatest hotbed of development has been the ARPANET, a nationwide computer network funded by the Advanced Research Projects Agency (ARPA) of the Defense Department. During the 1960's, ARPA funded much of the world's advanced computer research. Late in the decade, ARPA began to build a network, in order to make its resources more widely available within the ARPA research community and within the defense community as a whole.

The first lines and switching computers were installed in 1969, but it was not until 1972 that network protocols had evolved sufficiently to make the network really useful. By late 1972, however, at least a quarter of the computers on the network could exchange messages, thanks to a mail system developed at Bolt, Beranek and Newman (BBN). During the next five years, a rash of new mail systems were built for the ARPANET, most of which were developed at BBN, MIT, Rand, and the University of Southern California's Information Science Institute (ISI). There was also an extremely advanced mail system developed at Stanford Research Institute, for use in a limited community.

Today, most computers on the ARPANET can exchange messages, thanks to the existence of general network mail standards. There are now perhaps 2,000 network mail users, many of whom are computer scientists, but many of whom are nonprogrammer military users. Because ARPANET mail systems have become very easy to use, the portion of nonprogrammer users is growing rapidly. In fact, ARPA and the Navy are about to conduct an experiment that could result in a transfer of ARPANET computer mail to normal military communication networks. In addition, one ARPANET mail system, HERMES, is offered commercially, via the Telenet computer network.

A second source of development has been computer teleconferencing, in which a computer emulates the disciplines of face-to-face group discussions. The first teleconferencing system was developed in 1971, under Murray Turoff at the Office of Emergency Preparedness. Since then, several more teleconferencing systems have been built. The most widely used to date has been FORUM, built by the Institute for the Future under ARPA and National Science Foundation (NSF) sponsorship. Twenty-eight FORUM conferences were held on the ARPANET from 1972 through 1974. In late 1974, a FORUM variant, PLANET, was installed on the Tymnet computer network for further experimentation. PLANET is now offered commercially, by InfoMedia. In 1976, Turoff developed the Electronic Information Exchange System, which is being used for experimentation over Telenet.

A third stream of development has come in corporate

communication networks. Many corporate communication networks use computers merely for switching, for example, the international airlines reservation network, SITA, over which perhaps a quarter billion administrative messages are exchanged each year. A few networks, like Hewlett-Packard's COMSYS, also use computers for composing, reading, and filing.

There have been several mail offerings on commercial time-sharing networks. Besides STSC, at least one other commercial time sharing network, Tymnet, has begun to attack the corporate communication market seriously, with a computer mail offering, ONTM. Most time-sharing networks offer at least rudimentary computer mail service.

What is Computer Mail?

In contrast to traditional message-switching systems, computer mail systems do not merely transmit messages. Computer mail systems automate all aspects of mail handling. They provide word processing tools for message composition and editing, thus eliminating the traditional teletypewriter priesthood. They allow distribution lists to be prepared easily on-line. They print messages and also file messages automatically for subsequent retrieval. By automating the entire life cycle of message handling, by providing simple operation, and by letting any computer terminal access the system, computer mail has brought message technology out of the basement and into individual offices.

When a person "logs in" to the computer, the existence of new mail is reported if any has arrived since the last log in. If there is new mail, the system prints a one- or two-line summary of each message, giving its date, author, and an optional author-supplied title. The user can print these messages, answer them, delete them, or file them for later reading. The user may then scan through old messages or compose new messages. Delivery times range from a few seconds to several hours.

It is common for users to take terminals along when they travel or when they go home. This keeps them in touch and allows them to handle their correspondence whenever they choose.

The Outlook

The commercial outlook for computer mail is very bright. The average cost of a message varies considerably from system to system, ranging from \$0.50 to \$10.00, depending upon the functions performed. The rates are steep, but costs are falling rapidly. It appears that the cost of a good service will be around \$1.00 per message within a year or two, and that computer mail will be cheaper than postage by the mid-1980's (3,4).

At postage stamp prices, computer mail could capture a significant portion of the written communication in corporations -- around a trillion pieces of paper annually in the United States alone (4). So it is not surprising that many companies are beginning to eye the computer mail market seriously.

Where Do Hobbyists Fit In?

Virtually all analysts agree that computer mail will flourish first in organizations, since corporations, the government, and the military can pay the large bills needed for system development and operation. Organizations can pay a great deal for computer mail because they already pay a great deal for paperwork.

But it still seems possible that computer mail may develop

first in the community, rather than in organizations -- not in individual homes, because equipment is not in place, and a full commercial system would be expensive -- but in the computer hobbyist arena, where computer mail networks could be used to exchange ideas and just chat, with existing equipment and with relatively little software development cost.

Before discussing details, we wish to give several reasons for saying that hobbyist computer mail is not an inherently crazy idea. First and most importantly, hobbyists are very active and need to communicate. As Dave Caulkins has recently estimated (5), there were about 3,000 hobby computers and 15,000 computer hobbyists in the United States in 1976. There are probably a lot more today. They are spending a great deal of time and money on their hardware, and many are capable programmers.

Second, there is a strong historical precedent: ARPANET computer mail itself. ARPA projects have been serious and worthwhile efforts, but there has always been a distinct hobbyist flavor to the ARPA contractor community. Typical ARPA researchers work on exciting projects such as artificial intelligence. They tend to be more strongly committed to research than to their own organizations, and they tend to work very long hours. In many ways, the ARPANET research community has always been a collection of computer enthusiasts. Moreover, even before the network was constructed, the ARPA research community was fairly tightly knit, meeting at conferences and swapping employers with bewildering regularity.

When the ARPANET was created, it provided a natural tool for community communication, and, as noted above, computer mail blossomed as soon as the network became operational. While much ARPANET mail development has been deliberately funded, perhaps half of all mail systems have been written in people's spare time.

Although such statistics are not kept, it is generally believed that the ARPANET is used mostly for message communication. Again, for balance, we reiterate that ARPANET projects, despite their hobbyist air, have been geared toward pragmatic goals.

A third justification for considering computer mail is that corporations are likely to be quite slow on the uptake of computer mail. While excellent software is available and prices are already attractive, the corporate environment will not be an easy one to enter. First, telecommunications in corporations is by-and-large a low level function in the hierarchy. Emphasis is on thrift, not innovation, and innovation is generally a painful process. While some large organizations are innovative, it remains to be seen how rapidly these pace setters will adopt computer mail or how rapidly other companies will follow. Moreover, current computer mail systems, despite their sophistication, are not yet well-adapted to the needs of corporations (4).

A fourth, and to my mind decisive, reason for considering computer mail is the recent growth of CB radio. In the early 1970's, trucking and recreational vehicle users began to provide a market for cheap and useful CB equipment. Soon, CB began to grow wildly. About one U.S. household in 10 now has one or more CB rigs, and growth is continuing at an astounding rate.

CB is being used primarily in a party-line mode, in which changing groups of people chat, usually anonymously. Party-line conversation has one major advantage for radio sellers: it provides an instant community of users. In contrast, person-to-person message systems require either a closely-knit community within which traffic will naturally be high or a very large user base so that new users will have someone to talk to. In the party-line mode, whole communities need not make a decision to join, and potential users will automatically have someone to talk to. Of course once a party-line community grows large, person-to-person communication can flourish.

Computer conferencing offers some party-line features, although most systems are fragmented into numerous private conferences. In fact, the first major conferencing system was called Party-Line. Most successful conferences (rated by user satisfaction) have a free-wheeling air, and in at least two conferencing systems, public "graffiti" conferences have dominated traffic patterns.

How Might Hobbyist Computer Mail Evolve?

I will sketch two possible scenarios for hobbyist computer mail's possible evolution. In the first, "HAM Computer Mail" would evolve in the limited hobbyist community. In the second, "CB Computer Mail" would be broadly accessible in the community.

HAM Computer Mail

HAM radio, despite its long popularity, has never been large. Its stiff entrance requirements, including licensing tests and the purchasing or building of expensive equipment, have limited its size. Analogously, if hobbyist computer mail is confined to highly sophisticated computer users of the type that now populate the community, then hobbyist computer mail is likely to remain small, like amateur radio.

Nevertheless, amateur computer mail will almost certainly appear first among current hobbyists, because hobbyists have communication needs, the ability to design and implement a communication system and, I suspect, a strong inclination to do so.

Technically, two features will have to be added to current hobby computers to implement HAM computer mail. The first is message-handling software, to let individuals compose messages, prepare distribution lists, read incoming messages, and file incoming messages for later reading. This is an interesting problem, but it raises no substantial barriers other than hard work. As on the ARPANET, good mail programs, once created, would probably be adopted rapidly by the hobbyist community.

The other needed feature is networking, and this is indeed a problem. The simplest solution would be for a hobbyist computer club, such as Homebrew, to collectively purchase a message-switching computer to handle party-line communication, private message transmission, a bulletin board, and so on. Users would then need acoustic couplers, so that they could phone into the message computer periodically.

The problem with this approach is that it requires somebody to purchase the message computer. An individual club could do this most easily, since it has communication needs and many programmers who lack computers to work on. On the other hand, hobbyist clubs tend to be low-budget operations. Another approach would be for some hobby computer manufacturer to install a message computer as a way of stimulating sales and discovering market trends. Weyerhaeuser did something like this in the 1960's, when it funded a number of communes. Weyerhaeuser's funding paid off quite lavishly, because it discovered the emerging indoor plant market and plunged into it.

Potentially the best approach would be for some entrepreneur or club to set up a message-switching computer, then charge users for service. On current time-sharing systems, however, billing is extremely expensive. Some breakthroughs in charging practices would be needed to provide service cheaply yet get users to pay their bills.

If a separate message computer is not feasible, the alternative is to create a network of individual hobbyist computers. Other speakers in this session discuss this possibility and I will gladly defer to their discussions, since I find the idea of a voluntary distributed computer network to be mind-numbing.

Let me do something uncharacteristic of consultants and venture an opinion on how HAM computer mail will most likely come about. In my crystal ball, I see a club like Homebrew combining its dues and a grant from a computer manufacturer or telephone company, to purchase a minicomputer. Why should a hobbyist club purchase a "large" minicomputer instead of staying with micros? Because only about one computer hobbyist in three, by my crude estimate, now owns a computer or shares one. The bulk of all hobbyist club members are programmers who would be comfortable with a mini. These "naked hobbyists" could program the message computer and handle the mundanities of accounting and billing. Users of the cooperative message computer would need only a terminal and acoustic coupler; and they would have an "in" for a small outlay. Hobbyists with computers, in turn, could use their own computer for message composition and reading, using the message computer only as a switch. Their service fees would be reduced because they would use the message computer less heavily at each session. This approach--using a cooperative message computer--seems ideal, because it could make all club members active users, instead of participants. If a coop computer is purchased, furthermore, it could be a general time-sharing system for other forms of software development.

CB Computer Mail

While HAM radio has remained an arcane hobby, CB radio has grown to become one of America's major communications media. The reasons for CB's success are completely straightforward: a CB rig can be purchased for under \$200, and its operation is simplicity itself. Similarly, if computer mail is implemented so that anyone with a terminal and a pre-paid account can use the system, the resultant "CB computer mail" service is likely to spread rapidly and grow far beyond the hobbyiest market.

In many ways, CB computer mail would be easier to implement than HAM computer mail. It could have the size to network several message computers, and so reduce telephone charges to users, by making all calls local calls. Its size could also allow simplified billing, say through major credit cards.

If CB computer mail is to become extremely widespread, it may have to be implemented by the local telephone company, much like dial-a-joke and time-of-day services. If the costs of computer mail fall as low as I believe they will (3,4), the revenues gained by the basic message unit charges will be sufficient to justify the service. Moreover, if services are indeed implemented by telephone companies, regional and national interconnection will be possible and, quite probably, sufficiently lucrative for early development.

How would the current hobbyiest community greet CB computer mail? My guess is that they will view it with considerable distaste, much as HAM radio operators have little to do with CB radio. Perhaps hobby computerists will pull back completely, but I think this would be a grave error. Hobbyiest computing is passing from a hardware era, I believe, to a programming era, and it will ultimately enter a service era. Whether the current hobbyiest community follows the mainstream or becomes an isolated elite is one of the most critical issues now facing the community. CB computer mail could well be the crucible in which the long-term shape of hobby community is proofed and shaped.

Conclusions

It appears that computer mail, which developed primarily on the ARPANET, has the potential to form an important communication medium for the computer hobbyiest community. There are ample software concepts available for the design of hobby computer mail systems, but networking is a major problem area. Hobbyiest computer mail may be restricted to serious hobbyists, in which case it may remain forever an arcane medium, analogous to HAM radio. On the other hand, if hobbyiest computer mail can be opened up to casual users, it may become "CB computer mail" and spread as rapidly as CB radio.

References

1. Holusha, John, "Computer Tied Carter, Mondale Campaigns," The Washington Star, p. A-3, November 21, 1976.
2. Crisman, P.A., ed., *The Compatible Time-Sharing System, A Programmer's Guide* (2nd Ed.), The MIT Press, Cambridge, Massachusetts, 1965, Section AH.9.05, quoted in Stuart L. Mathison and Philip M. Walker, *Computers and Telecommunications: Issues in Public Policy*, Prentice-Hall, Inc., Engelwood Cliffs, New Jersey, 1970.
3. Panko, Raymond R. (Ra3y), *The Outlook for Computer Message Services: A Preliminary Assessment*, Telecommunications Sciences Center, Stanford Research Institute, Menlo Park, California, unpublished draft, March 1976.
4. Panko, Raymond R. (Ra3y), "The Outlook for Computer Mail," to be published in the *Journal of Telecommunications Policy*.
5. Caulkins, David, "A Computer Hobbyiest Club Survey," *Byte*, pp 116-118, Vol. 2, No. 1, January 1977.

COMMUNITY MEMORY - A "SOFT" COMPUTER SYSTEM

Lee Felsenstein
1807 Delaware St.
Berkeley CA 94703

"Whether or not humans will be alive on our planet will probably be resolved by cosmic evolution as early as 1985. We don't have to wait until 2025 to find out."

-- Buckminster Fuller

This discussion is based upon the premise that the trends within western industrial society which lead to greater degrees of interpersonal isolation and fragmentation must be reversed if the judgment of "cosmic evolution" referred to in the initial statement is to be favorable to humanity. The tendencies leading toward the erosion and breakdown of community and toward the centralization of the means of communication -- the lifeblood of community -- must be overcome. This reversal is beginning, slowly and appropriately without centralized direction, in what appears to be an evolutionarily sound manner.

So far this movement towards the re-establishment of community has proceeded without a significant technological component. Until very recently this has made sense, since the prevalent technology is organized along hierarchical lines and all design criteria point development towards larger systems in which the human being has become more and more an interchangeable part.

Recently, however, discussion has developed of "soft" or "appropriate" technologies, better suited for local, decentralized control. Tools or systems of this sort are usually designed on the assumption that a wider range of criteria must be considered than in standard industrial design, and that efficiency in a societal context rather than in an industrial context may dictate a design which seemingly violates industrial rules.

The computer has typically symbolized the centralization and mystification which are the antitheses of soft technology. Frighteningly expensive, it was served by ranks of experts who mediated every interaction with it according to mysterious rules. Every quantity and quality in the universe was to be reduced to strings of binary bits and manipulated at the speed of light as invisible charges or fields. The results could not be questioned by a non-expert.

This image has suffered severe damage at the hands of personal computer activity. It has been demonstrated that computers will sell as consumer-priced goods, that certification and formal training are not necessary for their use and application, and that involvement with them does not turn one into a tunnel-vision hacker or a mumbler in machine-language mnemonics. The machines do not bite and do not cost money when not turned on.

The scale and complexity of computers having been reduced to human levels, they may now enter the new area of soft technologies. As mentioned previously, wider ranges of design criteria can now be employed to bring about efficient functioning of a system in a societal rather than an industrial context. The movement toward decentralization and the regeneration of community may now take advantage of program-controlled electronic equipment.

One such "soft" computer system which is currently under development by Loving Grace Cybernetics is Community Memory, a nonhierarchical information-exchange system intended as a community information utility. An examination of the design criteria of this system may help further to define the new area of soft computer systems.

Unlike other information utilities, Community Memory (hereinafter CM) is not designed to be the primary conduit of information exchange. Rather, it is intended to facilitate information exchange through existing channels, mostly those of a non-broadcast nature. "Broadcast" is here used to describe information exchange on a centralized, replicated basis with little opportunity for reverse flow. Print media, large theatrical performances, and wall posters all qualify as broadcast media. Of the non-broadcast media face-to-face conversation is the primary example, amplified by modern transportation, postal and telephone systems. CM is designed to exchange "pointer information" through the computer system to facilitate non-broadcast communication through these existing media. Thus the user need not reduce all of his/her human contact to strings of ASCII codes.

CM is designed to be structured as a network of regional nodes at which the main processors and storage are located. Each region serves a number of local infor-

mation centers where people may walk in and use the system directly, without operator intervention. Items may be entered into the public data base using a free-association keywording system in which the machine uncritically accepts any string of text as a keyword and generates dictionaries referencing that item to those keywords. Searching is a matter of guessing a keyword used by the person entering the item and calling for it singly or as part of a concatenation of keywords. The machine responds with the number of items found under that keyword and waits either for an amendment to the keyword set and a new search or for an order to display in brief (first lines only) or in whole the set of items found. These transactions take place on a video display terminal which is part of the local intelligent terminal. Local floppy disc buffer storage allows quick sorting of intermediate files.

It is important to note that the CM system does not make judgments about the informational content of the items or the keywords, but acts only as a high-speed filing clerk. The necessary systematization is the province of human assistants and the good sense of the person using the machine. The external assistance may be provided in the form of asked-for advice from an attendant or in the form of access to cross-indexes compiled by "gate-keepers"; people who perform such a cross-referencing function as a vocation.

Designers of industrial-type systems will object that such a sloppy cross-referencing arrangement will inevitably yield a system which does not make all the possible correlations and which runs at less than 100 percent effectiveness. This is true, but it sets the system apart as a soft system which is more capable of capturing the attention and trust of the lay usership than one which is up to the industrial standard. Besides, a system for which each encounter is an adventure is more likely to serve as a "live" information-exchange medium than one in which the user must rearrange his/her world-view to coincide with that of the system designer.

Similarly, the possibility that one user will insert incorrect information into the system which might result in harm to others is real and does not necessarily yield to a technical solution. In "hard" systems this possibility is impermissible -- the information would have to be passed along a hierarchy of checking, verification and correction. In this "soft" system the only response to the possibility can be that some effort can be expended to prevent or minimize such occurrences, but that the system must not be seen as a fountainhead of validated information. One might as well read the information written on a wall or posted on a bulletin board; the truth or accuracy of the information is to be determined by the recipient. Caveat emptor, in brief.

One of the most important aspects of the system is its "myth" -- the sets of understandings which people hold regarding its nature and use. This myth deserves as much attention as any other aspect of its design. Some better word will have to be found to describe the local information center as which the system will be available, but we don't yet know that word. Certainly it should not include the word "center" since this attacks the concept of decentralization. Much work remains to be done in this area.

CM will require a set of explicit understandings regarding its governance and availability -- a constitution, in a sense, including a user's bill of rights. In cases where an operator of a local center is accused of violating these understandings relief must be available through arbitration and inspection. Equality of access for all persons and institutions would obviously be a cardinal principle, for example. Trust in the system would quickly be destroyed if people believed that secret sorts and bulk dumps were being performed for businesses or government, and the system must be organized to permit it to resist pressures for such preferential treatment.

All of these considerations must be incorporated into the design process of this soft system. Since wider goals than industrial design goals are being pursued, the design process cannot stop at the walls of the machine room. Indeed, the process of design itself remains to be designed for these types of tools and systems.

It is not through negligence that I have failed so far to raise the question of how the system will pay for its operation. General Motors is not in the business of making cars, it is in the business of making money. It does not make money in order to make cars, but the reverse. We are all the poorer for it in societal terms. We are not designing CM to make money. Its survival and growth must be dependent functions tied to its usefulness to the community. There are plenty of ways in which the system could be run to bring in adequate income for its survival and growth, especially since processors are an integral

part of all its hardware, and a great many branch points could be made conditional upon the deposition of a nickle in the slot.

Since one of the most important vehicles of community growth is small-scale localized economic activity, and since the primary purpose of CM is the development of community, it is fitting that its growth be closely related to this activity which is the very symptom of its success. In cases where the necessary economic activity does not take place at the level at which people feel justified in paying a bit of their cash flow for the system's continuation and growth, then the system should not be artificially be kept afloat. The design of any soft system must allow for a healthy amount of failure.

CM will be structured so that the distinction between "fun" and "serious" uses will be as small as possible. To use the phrase of Ivan Illich, it will be a "convivial" system. The information centers will be amalgams of branch libraries, game arcades, coffee houses, city parks and post offices. Unused terminal capacity will be made available for gaming or simulations. Amateur electronics or computer enthusiasts will be encouraged to learn first hand about the innards of the equipment and the software. This should result in a much shorter mean-time-to-repair than would be the case if access to the insides of the system were restricted to "qualified service personnel".

Community information utilities are moving very slowly if at all, a symptom of their industrial, "hard" design. Some designers and theorists throw up their hands and declare that "the public is not ready" for such systems. A soft system design such as Community Memory may well prove that previous designs were not ready for the public.

I will end by returning to the last paragraphs of the discussion by Buckminster Fuller (published in the Spring 1975 CoEvolution Quarterly) from which I obtained the opening statement. Fuller was responding to a newspaper's question of what the world would be like in 2025:

"Whether humanity will pass its final exams for such a future is dependent on you and me, not somebody we elect or who elects themselves to represent us. We will have to make each decision both tiny and great with critical self-examination - "Is this truly for the many or just for me?" If the latter prevails it will soon be "curtains" for us all.

"We are in for the greatest revolution in history. If it's to pull the top down and it's bloody, all lose. If it is a design science revolution to elevate the bottom and all others as well to unprecedeted new heights, all will live to dare spontaneously to speak and live and love the truth, strange though it may seem."

Lee Felsenstein
Loving Grace Cybernetics
1807 Delaware St.
Berkeley CA 94703

DESIGN CONSIDERATIONS FOR A HOBBYIST COMPUTER NETWORK

David Caulkins
Cable Data
701 Welch Road
Palo Alto CA 94304

Introduction

This paper is divided into two sections:

- 1) A brief tutorial on computer networks; and
- 2) A discussion of possible designs for Personal Computer Networks (PCNETs - a term coined by Einar Stefferud).

Computer Networks - The 25¢ Lecture

History:

The research that made computer networks possible was done by Paul Baran at Rand in the early sixties [1]. The first computer network of any consequence was the ARPANET, brought from the concept stage to operation largely through the efforts of Larry Roberts (then head of the information processing section of ARPA) in the late sixties and early seventies.

One of ARPANET's most valuable facilities is the message service - the ability to send a message (generally a file of English text, although almost any file can be sent) from one "mailbox" in a network computer to another such "mailbox" in a different computer. This doesn't sound very dramatic, but it is surprising how powerful and efficiency-improving such a message exchange facility is. What keeps ordinary message services, such as telephone, telegraph and mail, from working as well seems to be a combination of factors: too slow (mail); often hard to catch someone in the office and/or pierce the screens of secretaries (phone); hard or time consuming to use (mail, telegrams); lack of a positive indication of message delivery (mail, telegrams); expensive in terms of characters per dollar (telegrams, phone); etc. A computer based message system overcomes most of these difficulties.

People regularly using such a system rapidly develop a whole new communication style. Most messages are brief - 500 characters or less. Message text is composed by the sender at his terminal without the filtering and delay of secretarial intermediates, and style tends to be much more informal and direct than conventional media, such as business letters. Message system users move rapidly toward a computer-based personally oriented file system containing messages, distribution lists, text files, etc. The difference is one of kind, not just of degree.

Operation:

Some definitions will be useful in discussing networks:

Circuit Switched Network - A network in which a direct circuit connection is set up between communicating nodes.

Connectivity - The number of links terminating at a particular node.

Demand Access - Allocation of transmission bandwidth on demand.

Distributed Computer Network - A collection of geographically distributed and autonomous computers sharing common transmission and switching media.

Hop - Transmission of a packet from one network node to the next.

Link - The communication channel between two adjacent nodes.

Node - A computer system capable of receiving, testing, storing and transmitting packets using one or more links to other nodes.

Packet - A short block of data used to carry information through a distributed computer network. Packets typically contain three kinds of information: the data or message to be transmitted; addressing and control information; and data integrity checking codes.

Packet Switched Network - A network in which packets are routed from the originating node through intermediate nodes to the destination node.

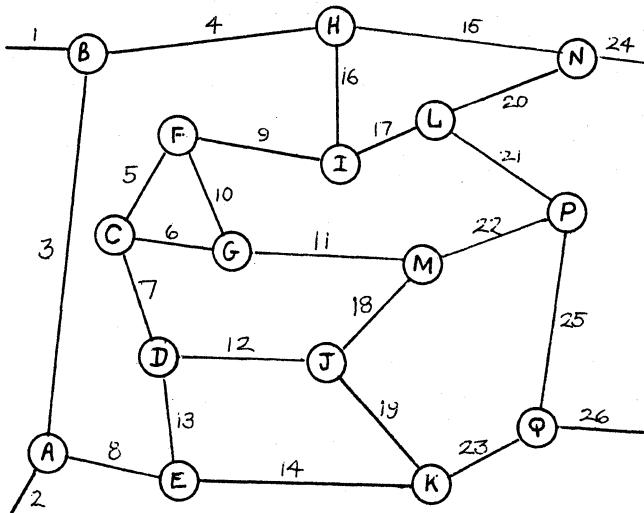
Protocols - Sets of conventions (formats, control procedures) which facilitate all levels of intercomputer communication. Includes electrical interface conventions, line control procedures, digital communication network interfaces, inter-process communication conventions, and application level (e.g. file transfer, database retrieval, etc.) standards.

(Many of these definitions are derived from a list developed by Vint Cerf [2].)

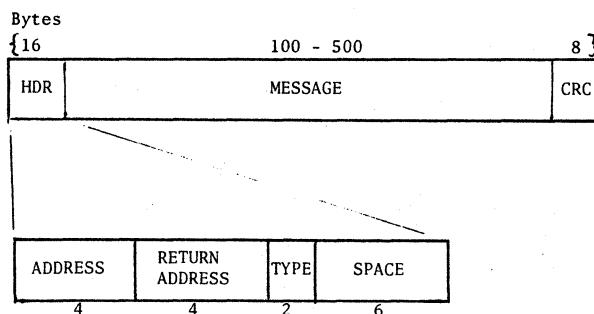
Details of Network Function:

Figure 1 is a diagram of a network with a connectivity of 3. There are 16 nodes (A-Q) and 26 links (1-26). Suppose the network uses dial-up phone links, and that node C sends a single packet (of a format like Figure 2) message to node P. C uses a routing algorithm to select one of its neighbor nodes (F, G and D) - G is picked as being on the most direct route from C to P. C initiates a phone call to G and transmits the packet. When G has successfully tested the packet for correctness (by means of the error detecting Cyclic Redundancy Code) it returns a short "I-got-it-OK" (or node-to-node ACK) packet to C. If C and G have no further packets for each other, the phone call is terminated and G initiates a phone call to M. When M has successfully received the packet it returns an ACK packet to G. G can now use the buffer space occupied by the message packet passed to M for new packets. After all G-M traffic is disposed of, the G-to-M call is terminated and M initiates a call to P. Since P is the destination, it originates two ACK packets - the usual node-to-node ACK to M, and an end-to-end ACK addressed to C. When C receives this ACK it knows the message made it all the way to P. If a transmitting node fails to receive an ACK, it retransmits the packet. In this way, messages passing through the net can be made to have an error rate substantially lower than that of the links themselves.

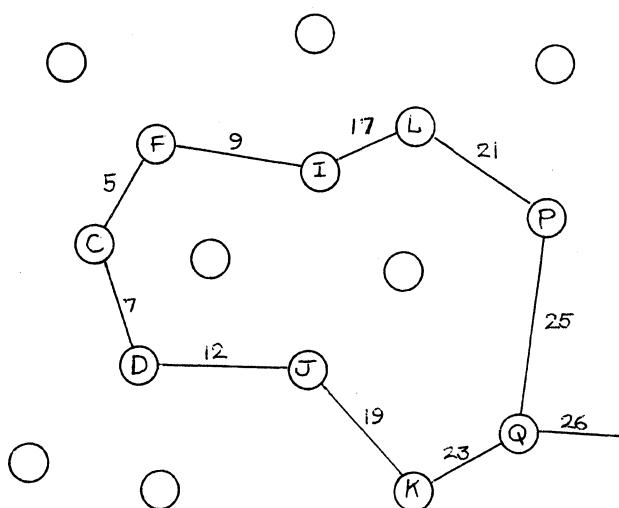
Figure 3 shows the network of Figure 1 with 7 nodes (A, B, E, G, H, M and N) out of service. Messages can still flow from C to P by either of two paths. Thus, even networks of low connectivity (3) can function in the absence of many nodes.



Connectivity 3 Computer Network
Figure 1



POSSIBLE HOBBYIST PACKET FORMAT
Figure 2



Reduced Network
Figure 3

PCNETS could also be implemented using radio links. A similar system has been in operation for some time. The ALOHA packet radio demand access system was built at the University of Hawaii by Norm Abramson, Frank Kuo and others. It is a time sharing system in which communication between many terminals and a central computer is carried out on two radio frequencies; one for terminal-to-computer communication and the second for computer-to-terminal communication. Terminal-to-computer communication is carried on in a novel way - any terminal with a new packet ready simply transmits it. When this packet is received by the central computer and checked for correctness, an acknowledgement (ACK) packet is sent back to the originating terminal. If two terminals happen to transmit packets during the same time interval, the packets "collide;" neither packet will be correctly received and no ACK packets will be sent. In this case each terminal waits for a randomly selected period of time and retransmits the packet for which no ACK was received. The random periods are important; if both terminals waited the same period, the retransmitted packets would again collide. Mathematical analysis shows that this scheme allows up to 18% full utilization of channel capacity before packet collisions become too frequent for reasonable channel use. Performance can be improved (if the electromagnetic propagation time between sender and receiver is a small fraction of the packet transmission time) in several ways; two seem particularly applicable to PCNET:

1) Slotted. All nodes maintain a clock indicating the start of each packet interval. Packet transmission is started only at the beginning of each interval. This means packets will either overlap completely or not at all. Slotted increases channel utilization to about 37%.

2) Carrier sense. A node with a packet ready to transmit looks to see if carrier is present on the channel; if so the node waits and tries again later.

For more background on computer networks and packet radio, see references 2 through 13.

Personal Computer Networks

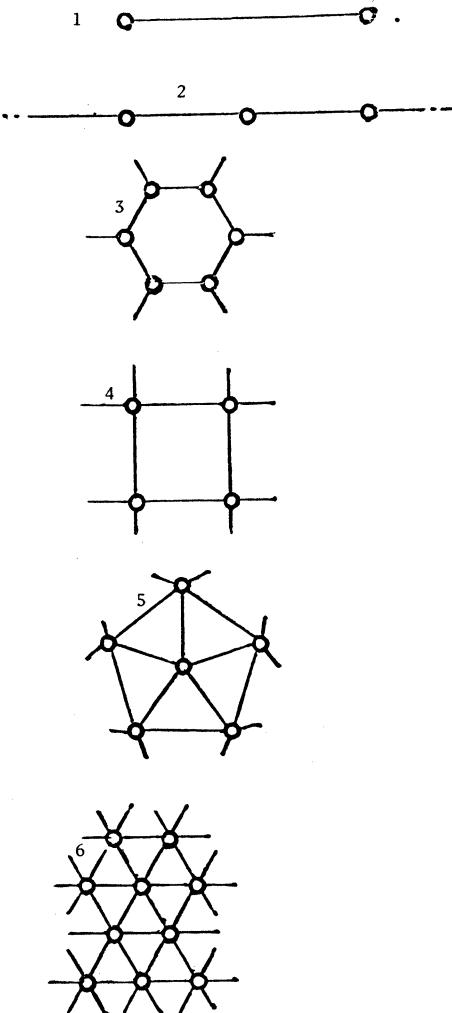
Why should a personal computer network be built? Several reasons come to mind:

- 1) It should be fun.
- 2) It is socially desirable; by providing personal computer users the means to gossip, exchange software, participate in message based conferences, create community information exchanges, etc., the American penchant for communications can be used to maintain and perhaps even accelerate the growth of personal computer use. This can result in a significant fraction of the U.S. population becoming "computer literate."

I think there are a number of attributes a personal computer network should have (caveat: there seem to be as many views of what a personal computer network should be as there are people interested in the subject - for example, see [14] - so the following list is not meant to imply consensus).

- 1) The network to be composed of geographically distributed nodes, each consisting of a personal computer equipped with low cost network interface hardware and a standard Network Support Software (NSS) package.
- 2) The network to provide modest size message or file transfer between any two active network nodes.
- 3) Low or zero cost inter-node communications.
- 4) The network to be distributed; no requirement for computational, storage or control resources external to the nodes.
- 5) Reasonable network reliability even though operating with noisy communication channels and with a fraction of network nodes "out of service." Positive acknowledgement to the message originator upon successful transmission.
- 6) Network architecture and protocols designed to allow expansion of capabilities without step change requirements in node hardware.
- 7) Network designed for automatic operation to take advantage of higher probable availability of node hardware and communication link bandwidth during late night and early morning hours.

If I had to boil all seven down into one it would be: The network must be cheap and reliable.



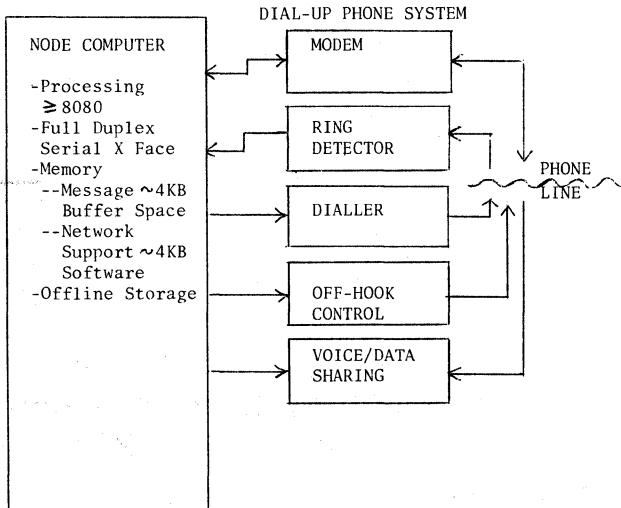
Network Connectivity
Figure 4

Design Elements

In my view the PCNET will have three major design elements: the node hardware, the node software, and the inter-node communication links. Let me consider each in turn.

Node Hardware:

A possible node hardware block diagram is shown in Figure 5. It consists of a processor (8080 or equivalent capabilities) with memory space for the NSS and message buffers, off-line storage for messages whose destination is this node, full duplex serial interface, and some signals for communication link control. The node of Figure 5 is intended for use with dial-up telephone system links. The MODEM translates between digital signals and FSK tones for voice phone line data transmission. It should be capable of operation in both originate (initiating calls to other nodes) and answer (accepting calls from other nodes) modes. The ring detector signals the node computer that ringing from an incoming call has occurred. The off-hook control allows the computer to take the phone "off the hook," terminating the ringing and opening the line for the incoming data. For out-going data the computer sets the MODEM to "originate," takes the phone off hook before dialing, and the dialer translates computer signals into touch-tone or dial-click dialing signals. After dialing is complete the computer waits until the ring detector shows that ringing has started and stopped at the dialed phone, indicating that the target node has gone off-hook. The computer then checks for the proper "hand shake" of FSK signals, indicating that the two MODEMs are successfully communicating. The whole process (from off-hook to end of "hand shake") should take only a few tenths of a second. If no ring signal is detected or if correct MODEM communication is not established, the computer goes "on-hook" (hangs up) and tries again. The number of re-tries before another neighbor node is picked is one of several network parameters to be "tuned" for optimum performance.



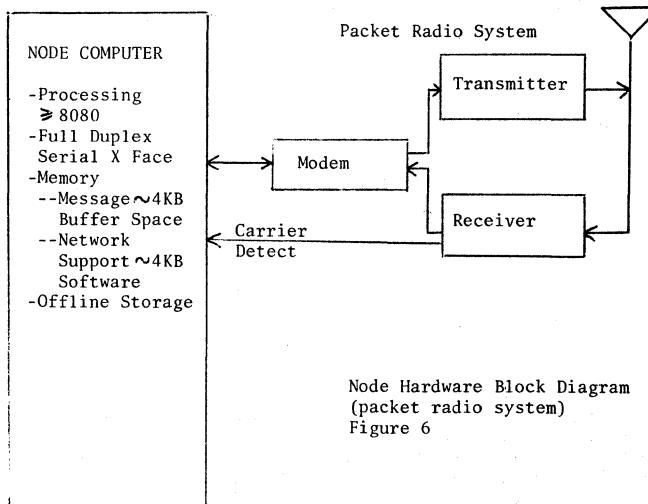
Node Hardware Block Diagram
(dial-up phone system)
Figure 5

The last unit of the phone based node interface hardware is the voice/data sharing (VDS) box. Since most PCNET nodes will have only one dial-up phone line, the VDS provides means to share the phone line between normal voice activity and PCNET operation, with voice calls being given priority. Direct plug connection of the MODEM and other devices to the phone line simplifies design and lowers cost. Such connection is possible in California and New York without a phone company supplied Data Access Arrangement if the devices to be connected have been certified by a licensed electrical engineer.

An alternate node design for a PCNET using packet radio links is shown in Figure 6. The processor is the same; only the interface hardware changes.

The MODEM is designed to operate at the higher data rates possible on a radio link - ALOHA system experience indicates that 9600 baud is not unreasonable. In order to implement carrier detect schemes for optimizing demand access to the

radio frequencies used, a control path is provided for sensing the presence of received carrier. The radio receiver and transmitter complete the set of equipment required. These could operate in the UHF band (for purposes of comparison, the ALOHANET uses frequencies close to 400 MHz). The transmitter should be able to provide a good signal level at the receiver of a neighbor node 10 to 25 miles away, using a simple omnidirectional antenna. CB equipment with this kind of performance is available today for less than \$200. It is not unreasonable to hope that simpler (single frequency) transceivers for packet radio might cost about \$500, even though initial production runs would be much smaller than those for the CB market.



Node Hardware Block Diagram
(packet radio system)
Figure 6

Inter-Node Communication Links:

Three inter-node media seem to me to have possible use in a PCNET.

1) The dial-up phone system. Even in a simple point-to-point PCNET over long distances message costs would be low. During the late night 60% discount period coast-to-coast (Bay Area to New York City, for example) rates are 21¢ for the first minute and 16¢ for each additional minute. At 30 characters per second a typical message of 500 characters would cost only 6¢; less if it were combined with other messages so the average cost were closer to 16¢ per minute.

A regional PCNET (where a region comprises an urban area of 10,000 square miles or less with contiguous local call areas of about 10 mile diameter throughout the 10,000 square miles) can be built using store-and-forward techniques. Each node has a number (3-6) of neighbor nodes within its local call area. Figure 7 shows the Southern California region with a network of 40 nodes (connectivity 3; 10 mile links) superimposed on it. Messages could travel from Oxnard to San Diego in about 18 local-call hops.

2) Radio links. ALOHA and other packet radio system (references 8-13) experience proves the practicality of packet radio. A packet radio PCNET would benefit from increased link bandwidth and freedom from dependence on the telephone system. This seems to me an area where radio amateurs have an interesting opportunity, because commercially available radio equipment is expensive and not well suited to this application. There are also spectrum space difficulties, discussed in more detail below.

3) Satellite links. This is clearly a more remote possibility than the other two. There are promising developments: the radio amateurs plan to orbit a synchronous satellite in about two years. Ground station costs are dropping continuously and might well be within reach of personal computer enthusiasts within the same time frame.

Software:

The PCNET Network Support Software (NSS) would implement the network protocols. It is premature (to say the least) to describe the NSS in any detail before the protocols are specified. Some general comments may be made: the NSS must be a standard, carefully debugged and tested, "bullet-proof" piece of code. A person wishing to use his system as a PCNET node would receive his copy of the NSS from some form of PCNET Control Center; with the correct node interface equipment the NSS could be bootstrap loaded from a nearby active PCNET node.

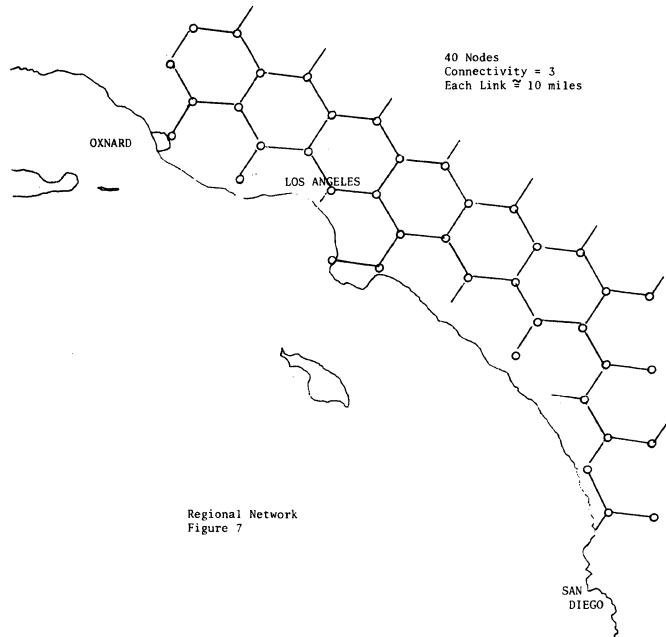
Some NSS functions seem to be required for almost any set

of protocols:

- o Routing - selection of the optimum neighbor node for message transmission.
- o Check code generation and checking.
- o ACK packet generation.
- o Store-and-forward packet buffer management.
- o Link interface control.

Regulatory Issues

My background is in computer system design; my knowledge of the legal and political areas of communication system control by the Federal Communications Commission (FCC) and State Public Utilities Commission (PUC) is that of a layman and is very limited. With this caveat, let me offer a few thoughts on these matters.



Operation of a dial-up phone based PCNET at night and on weekends when the phone system is essentially idle would have almost no impact on phone system use statistics and in consequence should attract minimal attention, regulatory or otherwise. Such a PCNET should be carefully defined as not-for-profit and recreational; any other type of service would almost certainly fall under PUC or FCC jurisdiction.

It is probably a mistake to design a PCNET critically dependent on phone company tariff loopholes like unlimited free calls within a local call area. Such tariffs are based on voice call statistics; any change in statistics reflecting increased data traffic should and would cause the phone company to change the tariff.

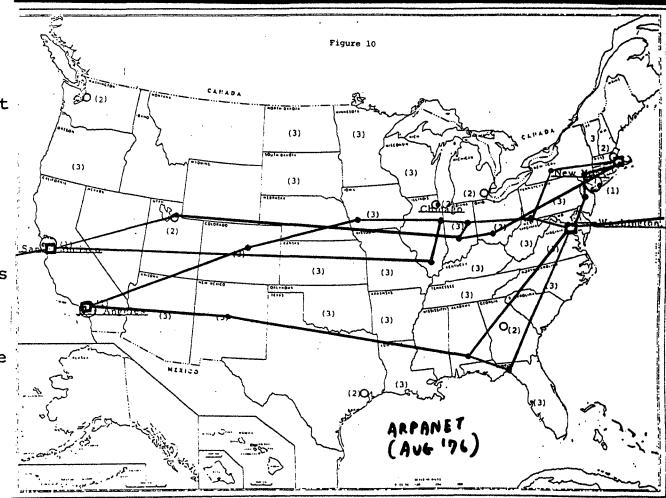
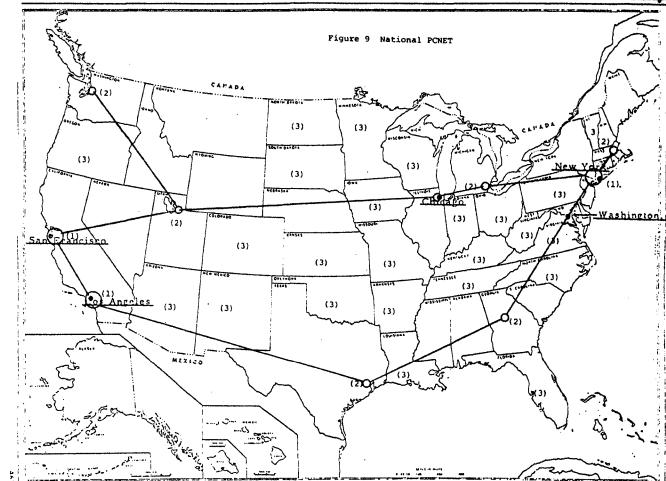
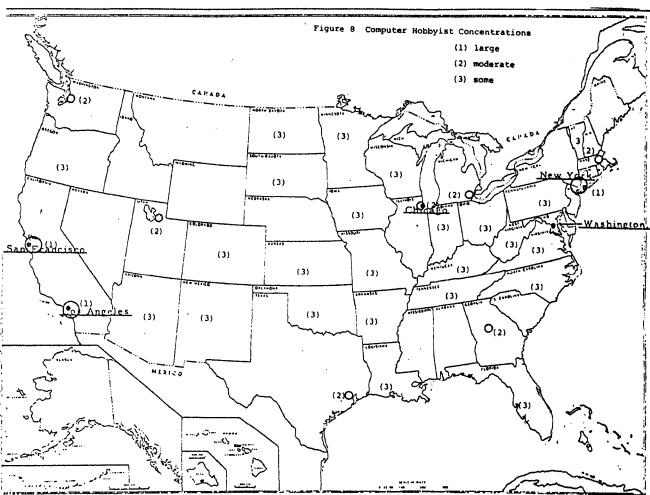
More advanced PCNETS using packet radio or satellite links would require FCC allocation of operating frequencies. Negotiations for such allocations typically take large quantities of time, money and lawyers.

I hope that someone more knowledgeable than myself is able to give these issues the attention they deserve.

Conclusions

Creation of a not-for-profit personal computer network appears feasible today. Figure 8 shows the geographical distribution of computer hobbyists as of January 1977. The big circles labeled (1) indicate large concentrations of hobbyists; the smaller circles labeled (2) indicate smaller concentrations of hobbyists; and (3) indicates a state with some hobbyist activity. The New York/New Jersey, Los Angeles and San Francisco Bay areas look like good candidates for regional networks. Figure 9 shows what might be the next step; linking the (1) and (2) hobbyist concentrations into a national PCNET.

Figure 10 shows the ARPANET as it was in August of 1976; the four boxes indicate high concentrations of host computers, interface message processors (IMPs) and terminal IMPs (TIPs). Not surprisingly, three of the four boxes coincide with the (1) concentrations of Figure 8.



References

- [1] Baran, P. "On Distributed Communications Networks," First Congress of the Information Systems Sciences, November 1962. *IEEE Transactions On Communication Systems*, Vol. C 6-12, Issue 1, March 1964, pp. 1-9.
- [2] Cerf, V.G. "Research Topics In Computer Communication," in Computers and Communications, Proceedings of the Federal Communications Commission Planning Conference, November 1976, AFIPS Press.
- [3] Crowther, Heart, McKenzie, et al. "Issues In Packet Switching Network Design." *
- [4] Frisch, I.T. and Frank, H. "Computer Communications - How We Got Where We Are." *

- [5] Gerla, M. and Eckl, J. "Moving Bits By Air, Land and Sea - Carriers, Vans And Packets." *
- [6] Lam, S.S. and Kleinrock, L. "Dynamic Control Schemes For A Packet Switched Multi-Access Broadcast Channel." *
- [7] Retz, D.L. "Operating System Design Considerations For The Packet-Switching Environment." *
- [8] Binder, Abramson, Kuo, et al. "ALOHA Packet Broadcasting - A Retrospective." *
- [9] Fralick, S.C. and Garrett, J.C. "Technological Considerations For Packet Radio Networks." *
- [10] Kahn, R.E. "The Organization of Computer Resources Into A Packet Radio Network." *
- [11] Kleinrock, L. and Tobagi, F. "Random Access Techniques For Data Transmission Over Packet-Switched Radio Channels." *
- [12] Frank, H., Gitman, I. and van Slyke, R. "Packet Radio System - Network Considerations." *
- [13] Fralick, Brandin, et al. "Digital Terminals For Packet Broadcasting." *
- [14] Wilber, M. "A Network of Community Information Exchanges: Issues And Problems," San Francisco Computer Faire Proceedings, April 1977.

* From the National Computer Conference, May 1975,
Vol. 44.

Biographical Note - Dave Caulkins

Dave Caulkins has worked on computers for longer than he cares to remember, with one short digression into the biofeedback business. He is currently involved with electronic mail systems at Cabledata Associates. In the past he has worked on a number of distributed computer systems, including the ARPANET, a microprocessor based satellite system for the military, and a large office automation system.

A NETWORK OF COMMUNITY INFORMATION EXCHANGES ISSUES AND PROBLEMS

Mike Wilbur
920 Dennis Drive
Palo Alto CA 94303

Abstract

This paper describes a telecommunication network oriented specifically toward personal computer users. The emphasis is on technical aspects of the network, the paper also briefly discusses the motivating considerations and some economic, social and legal aspects of such a net.

Warning

READ THIS PAPER if you're interested in coming to the conference session devoted to it. I think I would be misusing the conference session by presenting this same material again but vocally, and so I intend to use this material as background for the discussion in the conference session. I'll spend the first few minutes reviewing the salient points, but I won't read the whole paper to you.

Preface

I offer you this paper in the hope that you can profit by my experience with computer communication and my thoughts on how we could bring some of its facilities into the world of personal computers. The motivating ideas were the subject of my Byte article and are briefly summarized here. Good communication can facilitate free program sharing, allowing a community of users to build on the work of an author and then become authors themselves. Good communication can also let a product's prospective buyers profit by the experience of the product's former buyers. If the communication facilities are decentralized, then it is hard for a small group of people to control what is said or how it is presented. That's what this paper is about: setting up such a facility that is specifically oriented toward a wide and diverse community of personal computer users. Of course, in the late seventies, it would be a waste of effort to ignore the experience of the ARPA net; however, it would also be a waste of effort to slavishly copy the ARPA net, because that net is predicated on some assumptions that just don't apply to the personal community. A large portion of this paper is devoted to an effort to separate the applicable parts of the ARPA net experience from the inapplicable parts. On with it.

1. Introduction

Personal computers tend to have only modest amounts of processing power. In response, various people have opened time sharing services oriented toward personal computer users. Traditionally, these time sharing houses have offered strictly computational services, such as games, cross compilers and microprocessor simulation programs. Additionally, though, these concerns could offer a class of services which we could call a "Community Information Exchange", a computerized version of bulletin boards, yellow pages and mailboxes. (The name was inspired by Michael Rossman; most of the ideas come from the Resouce One system described by Colstad and Lipkin.) Note that the Community Information Exchange could be but one of many services offered by the time sharing house. In fact, a small CIE could be run in such a way that the host time sharing house doesn't even suspect its presence. That is, one could set up a CIE with only a small group of users and demonstrate its feasibility before attempting to enlist the support of the people running the host.

The idea of interest here is an outgrowth of the basic notion of a CIE: since the personal computer

Community is widely dispersed geographically, it can greatly benefit from a dispersed communication facility. The mailbox aspect of the CIE idea can be expanded to include the notion of moving messages from one CIE to another. In fact, CIE's can be linked together in a network to move a message from the source CIE through several intermediate CIE's until it finally reaches the desired destination CIE.

That's an expensive proposition: phone lines cost money, and it would be foolish to have them continuously open to serve a load consisting of perhaps 20,000 messages per day with a total length of maybe 5,000 pages. However, there is an alternative that could fill the bill at a much lower cost without degrading the performance so much as to make it useless. That is, each CIE could hold its traffic until late at night when the phone rates drop and then call a few nearby CIE's. During a call between CIE's, each could determine which traffic is going in the general direction of the other, and then they could forward the appropriate messages. If the calls at the network periphery took place just after the rates drop and again just before they rise and the calls near the center of the net took place toward the center of the time of low rates, messages could propagate across the country overnight.

There are many possible ways one could implement a CIE net. They all have their advantages and disadvantages with respect to one another. Of all the conceivable implementations, this paper considers those that are

cheap,
easy,
legal,
fun, and
useful

to the exclusion of all others. (Acronym addicts might wish to call these the CeLFU criteria.) I think (and sincerely hope) that there is at least one of those implementations. The problem is to find it. I haven't found it yet, so this paper is instead a summary of what I've found out about finding it.

2. Background

2.1. A User's View of the ARPA Net

The ARPA net connects computers together, and all communication over the ARPA net is ultimately communication between programs. The most useful of those programs, however, is one whose potential was quite unsuspected at the time the net was constructed: it sends messages of a few hundred words from one person to another. That program has spawned a fantastic variety of programs to handle files of messages, to automatically notify the recipient of a message, to read the sender, topic and other recipients from a message header and automatically construct a response, and to automatically forward messages from one computer to another as a person moves around. That is a measure of the important role the program plays in connecting people to people.

The facility has evolved into a very useful form over the years. A modern message sender will alternatively accept text from the Teletype, from a file (or any combination) or, on command, fire up a text editor on the message gathered so far. A modern message file handling program will selectively print out entire messages or one-line summaries of message headers; it will also, on command, call the message-sending program, perhaps initializing it to forward or respond to the message most recently examined. Finally, a modern top-level command interpreter will, on your option, give a short warning (e.g., [You have new mail]), after obeying a command if it's true and five minutes have elapsed since the last warning. The total effect is that people know when their messages arrive and can

respond in a matter of minutes.

The ARPA net provides another facility that is also quite useful: it can transfer whole files from one computer to another. The typical way reports and other long documents get sent through the ARPA net is that the author will use the mail facilities to tell people where the file can be found and why it might be of interest. Then any interested parties get copies of the file at their leisure. Of course, the file copying facility also works for programs and is almost always the vehicle by which programs are shared among people attached to the ARPA net.

While the ARPA net provides many other facilities, these two are the most useful. They would also be very useful facilities to have in a network oriented to personal computer people. Fortunately, they would be the easiest to duplicate in a way that would be cheap enough to make them accessible to individuals.

2.2. A Comparison with the ARPA Net

A great deal of effort went into the ARPA net development, and we need to discern which of its aspects would be useful to a CIE net and which would not.

The ARPA net was built to serve a community of perhaps ten thousand people, of whom maybe a thousand would be active users. About 110 computers are now connected to the net, along with 25 computers oriented specifically toward terminals; they were located in about 70 different places. In all, the network has 57 IMP's (connection points) plus two satellite links. The address space originally allowed for as many as 64 IMP's, each serving up to 4 computers, for a total of 256 computers. (The restriction has since been lifted.) By contrast, a CIE net should probably allow for more CIE's; perhaps 16 bits could be used for the address. Then the location could be coded into the more significant bits in such a way that only the highest bits of a distant address need be interpreted to discover the general direction in which to send a message. Then, as a message approached its destination, more bits could be interpreted as giving a more precise specification of the intended destination. Thus, the precision with which an address need be interpreted increases as the message approaches its destination, and so a large address space can be handled without requiring any participant to possess the information required to precisely interpret more than a few addresses.

The ARPA net was built to connect groups that already had a diverse selection of computers, including a PDP-1, a Sigma 7, the TX-2 and IBM System/360's. Thus, the ARPA net has no commitments to file name conventions, user name conventions, process identification or word size and only soft commitments to byte size and response time. It's probably a good idea to incorporate such flexibility into a CIE net, both to accommodate systems already in use and to provide for graceful accommodation to new systems. However, since many current systems handle upper case ASCII much more gracefully than other representations of data, it's probably wise to restrict the message transmission data to just those characters.

Although ARPA was willing to subsidize its net quite heavily, that would not be a good feature to count on for a CIE net. Since they figured out how to do it, we can build on their experience and do it much more cheaply.

Many APPA net transactions depend on the using computer and the serving computer being operational simultaneously. In fact, the net depends on the transit time of a data packet being less than half a second, coast to coast. That assumption just doesn't apply to a cheap CIE net. The compensating cost is that a CIE net would have to store a good deal of data representing the messages that are in

transit at any given moment. In the ARPA net, the traffic in the first half of 1976 averaged around the equivalent of 10,000 pages of text daily for each IMP. However, the ARPA net handles many transmissions of long files or long data streams in communication between special purpose programs. It is perhaps more indicative to consider messages between people: Metcalfe found that their average length was more like a hundred words, and the average of the last 218 I received was 600 characters (including headers). Now, my messages arrived at an average rate of two a day and include my responses, so my average daily volume was 1200 characters (a third of a page) per day. If the average subscriber to a CIE generates the same volume of mail and if CIE's are usually implemented in time sharing houses, the total average storage of mail in transit will probably be easy to accommodate. A favorable factor that will emerge later in this paper is that most traffic between CIE's would probably be handled at night, when the phone rates are low.

The ARPA net is oriented to connections rather than messages. That is, a good deal of effort is devoted to establishing a connection between two programs in connected computers, and then messages refer to it. The motivating assumption is that many messages will be sent down a connection during its life. It would be much more appropriate in a CIE net for each message to be self contained, including in itself the intended destination and the source to which an acknowledgment should be returned.

The procedures in the ARPA net are rigorously divided into levels of abstraction, with each level depending on the one below it, responsible to the one above it and ignoring all the others. That modularity has almost no cost in efficiency and would probably go straight across to a CIE net. When it's free, it's a good way to design almost any system.

All ARPA net computers are need to process binary data. That's probably not such a good idea for a CIE net. Even though most hardware participating in a CIE net would have that capability, such a requirement would be a definite impediment to implementing a CIE in an unsuspecting (or uncooperative) time sharing service that has a BASIC view of the world.

IMP's do almost all the error checking that gets done in the ARPA net. Once an IMP accepts a message, it guarantees that the message will be accurately transmitted to its destination and that the source will get a reply saying that it arrived intact. This increases the buffer storage required at each IMP and forces the IMP to expend a great deal of effort retransmitting messages that get garbaged in transit. The responsibility is at the wrong place, and the result is a marked loss of efficiency, as Metcalfe observes. When a message goes astray, the appropriate action may be to just give up or to wait a few hours and try again; the program sending the message is in a much better position to judge than the IMP is. That problem is compounded by one of the probable characteristics of a CIE net: the loyalties of a CIE will lie more with its own subscribers than with someone sending a message through it between two remote points. Responsibility for transmission between CIE's could be even more diffuse when the host time sharing service is not a participant. The point of all this is just that a CIE net would probably be better served if all but the most nominal error checking and acknowledgement were done from one end to the other by the two parties most interested in accurate transmission of a message: its sender and its recipient.

ARPA took years to get its net together. Now that they've shown us how, we ought to be able to do it much more quickly. (It's supposed to be easy to do something that's been done before.)

The ARPA net wouldn't be what it is had it not

evolved among a bunch of clever people who were really turned on to the whole idea and get their kicks out of working like purple bears. Ditto the CIE net.

An IMP, on accepting a message from a connected computer, may break it up into "packets" and individually sends them toward the destination. The destination IMP then reassembles the message before sending it on to the connected computer. That's not for a CIE net: the ARPA net has a multiplicity of high speed paths between any source and destination IMP's, and the packets will likely take different paths and thus propagate in parallel. The result is faster transmission, motivated again by the ARPA net requirement of transcontinental propagation in half a second. Much of the complexity of the IMP program comes from the packetizing logic. As if that weren't enough, packetizing gave the ARPA community one of its most amusing bugs in recent years: if one IMP is the destination of a lot of traffic, then it might well fill its reassembly buffers with partially assembled messages. Then, since the IMPs guarantee delivery of every message they accept and since they route packets around congested parts of the net, all the transmission buffers IN THE REST OF THE NET fill up with packets headed for that IMP. Nobody uses the net until somebody walks over to the overstuffed destination and tweaks the buttons on its front panel. Now the IMP program is even hairier.

3. Areas of Consideration

3.1. Economic

3.1.1. Minimizing Expenses

The typical minimum monthly fee charged by a time sharing service to each of its clients will likely be more than a typical individual computer user will want to pay, and so people will naturally tend to share user names. By minimizing the expense to its subscribers, a CIE will be able to attract more people and increase the benefit to the community. On the other hand, most small time sharing systems tend to offer only very coarse controls on the amount of isolation between their users. For example, people who can send files to one another usually have unlimited access to each other's files.

A useful CIE could probably be built that would appear to its host system as a single user. It would depend heavily on the honor system, both to minimize its expenses and to recover its expenses from its subscribers. It could protect the privacy of its subscribers by encrypting their files, as is discussed in section 3.2.5. The CIE would provide programs that provided the proper isolation between users; it would depend on its subscribers to refrain from using others in the likely event that the host system could not prevent them. The CIE programs could keep a record of their usage, so that the use of other programs could at least be detected.

3.1.2. Recovering Expenses

As long as a CIE can keep the confidence and cooperation of its subscribers, it might be able to avoid the expense of billing, accounting, etc., by depending on voluntary contributions from its participants. The Homebrew Computer Club provides a favorable precedent. Its newsletter is financed entirely by people occasionally putting a dollar or two on the appropriate pile, and the phone bills required to coordinate its group buys are financed by people willingly paying a slight markup.

3.2. Technical

3.2.1. Modularity

"Protocols" are a useful concept in the ARPA net. That is, there are definitions of the way an IMP talks to an IMP, the way an IMP talks to its connected computer, the way two computers talk to one another via the net, the way two processes

communicate via the net, and so on. Each level is defined by how it uses the next level down and how the next higher level can use it. No other levels enter into the definition.

A protocol for an ARPA net facility defines the way a foreign computer establishes contact with the facility via the net. It also defines the formats of all the messages sent through the net when the facility is used as well as the circumstances under which each may, must, or must not be sent. Some ARPA net protocols also specify timing restrictions. Each protocol, however, is very strongly focused on a specific aspect of the network and says nothing that doesn't need to be said to define it. In short, a protocol specifies all and only what is needed for proper operation of its network facility.

In the case of a CIE net, one should at least separate the protocols between neighboring CIE's from the protocols between the source and destination of a message. The interactions between a CIE and its subscriber should be left completely out of the design of a CIE net: each CIE will be hosted by a computer system that was chosen for economic or logistic reasons and will have its own idiosyncrasies that should be accommodated at the option of its subscribers. Between CIE's, however, protocols should be specified that allow room for reasonable amounts of variation. For example, there must be a protocol describing how characters are handled; it should allow for each CIE to treat the other as a half-duplex terminal or a full-duplex terminal with a person at it and should provide for acknowledgement or retransmission of each line of characters. At the next level up, there should be a protocol specifying the format of messages between CIE's; such messages may be concerned with the connection itself (e.g., I have some traffic for you; can you take it?), or they may represent messages going between CIE's (e.g., Here is a message from CIE mumble to CIE foobaz.). Each level of protocol specifies control information (e.g., I got line 69 UK; send me another one.) and a data field (e.g., the line of characters); the next level up talks about the format of the data field.

This kind of modularity is terribly helpful when a diverse group of people are trying to build a system they don't understand very well. It forces refinement and thoughtful definition of ideas. It also forces explicit definition of the interactions between the levels and the requirements and guarantees that go along with them. Finally, and perhaps most importantly, it restricts the range of possible interactions in a way that focuses attention in more worthwhile directions.

3.2.2. Orthodoxy

The CIE protocols should be rigorously defined, and the definitions should be taken seriously. There should be some facility by which somebody can test a purported protocol and easily discover any of the more obvious bugs in it. The danger in undetected bugs may be worse in a CIE net than in other environments because two effects may well coincide. First, there will probably be a small number of widely used protocol implementations in the net; people will be more likely to use someone else's implementation than to write one of their own. If one of the popular implementations contains an undetected bug, then many people can be inconvenienced when some other popular program supplies it the inputs it needs to misbehave. On the other hand, if informality characterizes the CIE net (as it probably would), then the buggy program's author may well not be available to fix it. Also, the possible rise of mutually unintelligible local dialects would be a great inconvenience to people who want to use their favorite system in several different regions.

3.2.3. Heterodoxy

While the CIE participants should abide by their agreements as defined in the protocols, that stance

should not preclude experimentation. The protocols should include within themselves methods for allowing people to experiment with new ideas. For example, if a particular message format will be used by 7 different messages, then the field specifying the message type could be big enough to allow for 64 different message types, leaving 57 message types for expansion.

Experimental protocol extensions should be compatible, however. For example, the ARPA net protocol for handling characters allows the two participants to agree to follow variants of the standard version of the protocol. The agreements are negotiated by messages in a standard format with the op codes "Please Do", "Please Don't", "I Will" and "I Won't"; a program receiving a request for or an offer of some variant it doesn't understand need only return the message with "Please Do" changed to "I Won't" or "I Will" changed to "Please Don't". That response is exactly the same as a program would have sent if it understood the request (or offer) but was disinclined to comply.

3.2.4. Reliability

When you send something to somebody, you typically would like to have some assurance that it arrived and that it was intact when it arrived. Those are really two different concepts, and they can be implemented by two different mechanisms. If you solve the second problem first and then discard messages that were damaged in transit, then you can interpret evidence of any delivery at all as also implying that the message survived the process. The appropriate corrective action for both cases is the same: retransmission. It doesn't make much sense, though, to retransmit a message indefinitely: you need a criterion (like a number of attempts or a total amount of time involved) that will let you declare the transmission a failure.

Reliability techniques can be applied at several different levels. At least, they can be automatically applied from one CIE to the next and from the source CIE to the destination CIE. When one CIE fails to send a message to its neighbor (after several retries), it can try to send the message to some other CIE in the general direction of the destination CIE. It need not give up (perhaps reporting failure to the source CIE) until it has unsuccessfully tried some reasonable number of alternatives. Similarly, the source CIE could hold the message until the next night and try again, without reporting failure to the subscriber who sent the message until it has repeatedly tried to forward the message without success. Then, if the destination CIE had equipment problems that took it out of service for several days, or had a fatal bug in a crucial program, the message would still be delayed but still delivered.

Damage to a message can be detected by including some redundant information along with it. The cyclic redundancy checksum is sensitive to permutations of its input and is a candidate for inclusion in the redundant information. If the format of a message is restricted, say, to start with a header, and non-adherence to the format can be detected, then the format itself is redundant and can be used to detect damage.

The basic idea here is that, at each level of message transmission, the participants should take reasonable precautions to detect errors at lower levels and should take reasonable steps to cope with failures at lower levels. The worst thing that could happen is that a message might incorrectly appear to have been transmitted correctly. The next worst possibility is that a message might not be delivered, with the sender being properly notified. Finally, delays, while they aren't as bad, would inconvenience the sender enough that they should be avoided whenever a reasonable amount of effort would expedite transmission.

3.2.5. Privacy

If CIE's (and a CIE net) are to attract a wide base of subscribers, they must take reasonable steps to protect the privacy of their subscribers. Probably the best way to provide privacy in a CIE in an unsuspecting host computer is by having each user supply an individual key for encrypting any and all private data. Then the problem of protecting private data within a CIE reduces to the problem of preventing interlopers from altering the encryption program itself, discovering a subscriber's key or reading the clear text version of the information before encryption or after decryption. Subscribers communicating with the CIE via their own personal computers can encrypt their entire communication with their CIE, with the exception of just certain control information, such as the addresses of any messages they send through the CIE. In order to allow for increasingly sophisticated intruders, it seems wise to use Diffie and Hellman's modification to the NBS encryption algorithm.

3.2.6. Access

The idea of a CIE net introduces a new problem in the general area of access protection: subscribers must have some protection from junk mail, but that protection must not interfere with their ability to receive desired mail. This problem arises from the very economic constraint that a CIE's host computer is likely to have only a limited amount of file storage for the CIE to use, and so the subscriber will likely be asked to pay to store information in the CIE. A subscriber can negotiate a storage limit with the CIE, which could refuse incoming messages when the limit is exceeded. While that method is effective, it does not distinguish junk mail from mail the subscriber wishes to receive. In fact, a small number of ambitious junk mailers could render a relatively inactive subscriber incapable of receiving mail from friends.

The conflict can be resolved by allowing the CIE and its subscriber to negotiate two limits: one for junk mail and one for friendly mail. (Presumably the former would be much lower than the latter.) Then the subscriber could inform friends of a password they could include with the addresses of their mail to include it under the more generous limit. Initially, junk mail will probably be limited to acceptable levels by the simple expedient of charging the sender of a message with the cost of forwarding the message to its destination; however, the possibility of letting a subscriber limit the volume of incoming mail should be kept in mind so that it is not excluded from the final design of a CIE net.

Also note that this scheme does not accommodate people who maintain large mailing lists and who do not wish to incur the expense of frequently changing passwords for the (many) people they send messages to. In fact, I know of no good solution to that problem that does not require that an intruder could not masquerade as a legitimate sender.

3.2.7. Transmission Medium

Some thought should be given to the choice of a communication medium by which a CIE could communicate with a neighboring CIE without requiring the cooperation of both of their host computers. Since such a medium should be broadly accessible, allow cheap experimentation and permit the transmission of encrypted data, the telephone is a logical choice. Modems that can handle 300 baud are commonly available on the personal computer market, and that bandwidth will transmit 50 pages of messages in about 25 minutes, which is probably reasonable for a CIE that is just getting started. (I presume that, once a CIE gets enough momentum to have a higher volume, its operators will have succeeded in convincing the host computer's management to operate the CIE, using a more serious communication medium.)

Amateur radio is probably not an appropriate initial medium for three reasons. First, it is markedly less generally available, although it could certainly be available to link any two particular CIE's. Secondly, amateur radio cannot be used to perform a service for a fee, and so anybody proposing to use amateur radio to link CIE's should be careful that the use of amateur radio in a CIE net is well separated from the cost recovery channels of the net. Lastly, amateur radio can only transmit Baudot at up to 100 words per minute and is prohibited from handling encrypted data. While ASCII could be translated to Baudot the translation would require replacing all lower case letters by the corresponding capitals, to the detriment of readability. The bandwidth limit is about a third that of the telephone, so that any given volume of text would take about three times as long to be transmitted; the limit is marginal for telephones, but the cheaper long distance capability of amateur radio may well compensate. While all these limits could conceivably be negated, a CIE net ought not be designed to count on that eventuality. On the other hand, if the design should be flexible enough to allow the incorporation of amateur radio links should they become available.

3.2.8. Topology

The ARPA net is almost completely devoid of spurs. That is, almost any telephone circuit between two adjacent IMP's can go down without separating any IMP from the rest of the network. The connection pattern is highly irregular and was chosen to minimize the cost of the phone circuits as well as to provide a multiplicity of alternative paths. For example, there are only four circuits across the Rocky Mountains, while the San Francisco Bay Area is connected in a figure-8 with external connections from Berkeley, Stanford and Cupertino. Each IMP continually monitors its connections to the other IMP's and each tells the others what it sees, so, when a new IMP is added to the net, all others find out within seconds and are then prepared to send messages to it. Conversely, when a circuit is disconnected, messages are automatically routed around it.

A CIE net can have similar resiliency and adaptability, within the limit imposed by the only sporadic schedule of calls between CIE's. For example, if a CIE address were a 16-bit number with its most significant bits coding the general part of the U.S. and progressively less significant bits specifying the location progressively more precisely, then each CIE handling a message need only forward it in the general direction of its destination and need not require that it go via any particular intermediate CIE. Such an addressing scheme permits a CIE to have only a vague notion of how to interpret distant addresses and precise information for only a few nearby addresses, so that there would be no need for every CIE to be known to every other CIE. On the other hand, it requires that careful thought be given to the assignment of blocks of the address space to regions of the country. In this sense, it requires centralized planning even if the requirement is fairly well isolated to the initial planning of the net.

3.2.9. Facilities for Testing

The ARPA net contains facilities to let people test the programs implementing the basic protocols. For example, a program may request that its messages be absorbed by the recipient; if they arrive at all, they will be acknowledged, which is a sign that at least the basic logic is correct. Correct coding of the data into a message can be checked by requesting that a message be echoed and then ascertaining whether the data survive a round trip. Many computers on the ARPA net provide a process that responds with the date and time upon being contacted; this is useful to determine whether the two programs agree on the representation of simple text strings.

Similarly, a CIE net should (and can cheaply) provide a similar facility to let new people test their purported implementations of the basic network protocols. They should be provided both at the level of communication between CIE's and communication between a subscriber and a CIE. If each of these levels is further divided into levels involving progressively more parts of the hardware and software involved, then bugs can be isolated promptly. That can be of great help in getting people (and CIE's) initially connected and in the normal process of evolving their programs.

All this also impinges on the design of the basic protocols themselves. A program should be able to detect accidental and unintentional invocation of at least the echoing facility. If, say, a modem connected to a CIE were accidentally looped back to itself, then the CIE could send a message via the modem, and, on getting the echo, it could acknowledge the message it had just received and then count the message as successfully transmitted when it heard the acknowledgement. One way to avoid that rather embarrassing scenario would be to have each message contain the current transmitter's address in a fixed place. Then, any incoming message could be checked and acknowledged only if it was transmitted from the party at the other end of the line.

3.2.10. Performance Monitoring

Once a CIE net has passed the major milestone of working at all, people will become interested in making it work well. In improving the performance of a system, such as a CIE net, it is indispensable to have a model of how it performs and where its bottlenecks are. Otherwise, a great deal of effort could be expended in repairing an "obvious" inefficiency that actually occurs only rarely.

It is not immediately obvious just what kind of performance should be monitored, and there are two ways to cope with that situation: anticipation and flexibility. Probably the most useful parameters that an individual CIE could monitor are the number of messages sent from it, received by it and passing through it in a single night. Other useful parameters might be the length of the messages in each of these classes, the number and lengths of messages that it needs to hold between nights and the error rates it experiences in its attempts to communicate with each of its neighbors. If the lengths of messages are stored as histograms with exponentially increasing bucket sizes, then all this information can be stored quite compactly. Just these few statistics will give a gross indication of where problems are to be found and how serious they are.

Depending on whether any problems are found and how serious they are, a wide variety of conceivable additional measurements may be useful to pinpoint the flaws in the design or implementation of a CIE net. The flexibility required to adapt the measurement facility to changing needs may be patterned after the measurement facilities of the ARPA net. A few fake user names (e.g., STATS) could be reserved at each CIE; a message to STATS could be interpreted by the measurement facility, which could then respond with messages from the fake user STATS. The messages themselves could be in a simple free format, with the individual fields preceded by short descriptive identifiers, so "THRUSIZES=" could precede the length histogram of messages passing through the CIE. Finally, the fake user name CIE could be reserved at each CIE to allow for expansion of the set of fake user names; the first word of a message to or from the CIE could be the name of the actual facility concerned.

3.3. Social

3.3.1. Consumerism

Other than letting people share technical information, like programs, the most valuable

service a CIE net can provide is an uncensored forum in which people can share their experiences with the personal computer market. (Don't forget that the lack of censorship is a consequence of the lack of centralized control.) Such a forum would restrict the market available to those whose products are of interest primarily to the naive, and it would provide a great deal of free favorable publicity to those offering very good products at a reasonable price. The consequence would be that the general quality of goods on the market would rise, and personal computer users would be able to buy products much more freely.

3.3.2. Environmentalism

A CIE net can also help check the rate at which the planet's stored energy is being dissipated. Many computer hobbyists currently attend meetings of the local hobbyist club just to exchange verbal information with one other person. That is exactly what a CIE net would do best. Thus a CIE net could help reduce the decomposition of ancient fossils that accompanies the usual attendance of people at hobbyist club meetings. Hobbyist clubs serve many other valuable services to the community of people personally using computers, but a functional CIE net could help people communicate without transportation in some cases.

3.3.3. Individualism

The fact that different people have different schedules is often an impediment to people meeting one another or contacting one another via telephone. The CIE, on the other hand, uses its file storage as a buffer for communication. Thus, it is a natural consequence of the structure of a CIE net that the two parties to a conversation need not coordinate the times at which they are available for and disposed to conversation.

3.4. Legal

Since a CIE net would start its existence as a very loosely organized cooperative effort, it would not benefit from a premature or unanticipated intrusion of the law. Therefore, its participants should be aware of the legal aspects of what they are doing and take some nominal care to keep them as innocuous as possible until they are prepared to handle a somewhat formalized relationship with governmental bodies.

A CIE can operate as a very informal association of its subscribers; it probably will not come under business law. However, if it were to make a profit, employ people or accept legal responsibilities, it would probably need a license to operate and need to keep business records. That would be detrimental to a small group of people trying to get a CIE established and should probably be avoided in the early stages of its development. On the other hand, an established CIE would probably not be harmed by those requirements and would benefit by the freedom and by the legal standing.

Communication is regulated by the FCC; computation is not regulated. The communication services of a CIE (and a CIE net) have a pronounced computational flavor, especially to the extent that they include services that, for example, help a subscriber sort through large collections of messages. The arguments on both sides are confusing enough that the regulatory status of a CIE net is murky at best. The issue is one that would eventually need to be faced by people participating in a CIE net, but it may be possible to avoid it in the early stages of network construction. As long as the net remains an informal association of people working actively to get themselves together, the issue may not be forced. However, I don't think a CIE net could pass to the stage of connecting large numbers of people with diverse needs and interests without having its position clarified.

4. Future Development

4.1. A Scenario for Staged Implementation

4.1.1. The Network Itself

The rest of this paper is concerned with general considerations that should go into a design of a CIE net; now it is time to imagine how a design could be actually implemented. It can start off with one ambitious person who owns a fairly sizable personal computer. That dedicated hacker could implement many (perhaps all) of the critical programs for a CIE and make sure they could all talk to one another. (If the protocols are suitably designed, the entire implementation could be done in a String BASIC for maximum portability.) After that, it is necessary to have a teletype port that can be looped back on itself to debug the character handling protocols.

Further stages of implementation require that our friend have access to a time sharing service that could host a CIE. (Don't forget that the host needn't suspect the presence of the CIE for some time yet.) Since the central CIE programs (including the encryption programs and the others concerned with handling a multiplicity of subscribers) will already have been developed, the next order of business is to get them working on the host computer. Then, this hard-working hobbyist will actually have TWO CIE's (one on the personal computer) and can make sure all the programs work. That is the time to start forming a community; extremely sympathetic friends are the most appropriate people to recruit first. When most of the more obvious bugs have been worked out, then more people can be attracted to the community.

When several CIE's are operational, then they can link themselves together without yet requiring cooperation (or even awareness) from their host computers. Someone must agree to be an intermediary between the two CIE's; that person would phone up one CIE and express a willingness to accept mail going in the general direction of the other. The CIE would then forward the appropriate messages, using all the appropriate low-level protocols, which the intermediary would copy to some number of cassettes. When the mail had been transferred, then our committed computerist would close the connection and phone the other CIE, telling it to prepare itself to receive the forwarded mail. After the mail has all been safely stored by the CIE, it would then send back any mail it had waiting for the first CIE. That would go back to cassettes and then back to the first CIE. It's really not as complicated as it looks: the intermediary would simply be performing the role of an intermediate CIE that just happened to be neither the source nor the destination of any of the mail it handled.

A dozen conscientious night people could trade off the duties of forwarding mail for a given pair of CIE's and make an effective connection that would be quite useful for the communities concerned. As the connection gets more and more useful, the traffic volume will grow until the amount of time needed to transmit it at 300 baud will be more and more burdensome to the generous souls who handle the traffic. Then is the time to try to enlist the cooperation of the people running the CIE's host computer. They can rent a modem for about a dollar a day per kilobaud and can use dial-up lines up to a kilobaud and, assuming they have night staff, can handle the (by now) highly automated call to the next CIE at little additional personnel cost. The host administration would not only stand to retain the business coming from the CIE, but it would also become a conduit for the reimbursement for telephone company charges. In addition, it would gain a user community that it had not needed to build up, so it might be strongly attracted to forming a cooperative relationship with the CIE.

4.1.2. Other CIE Services

This paper isn't really about CIE's themselves but rather about their interconnection. Still, Community Information Exchanges themselves are not too common (I know of none that are currently in operation), and so the previous subsection assumed that forming a CIE would be part of connecting its community to a network of CIE's. The same reason justifies my spending a bit more space just mentioning a few things a CIE could be good for in itself. First of all, it could include a community bulletin board, which its subscribers could selectively read by means of a sophisticated computer program. It (or in fact, its host) could provide a hard-copy output service that would free its subscribers from needing to deal with an expensive, bulky and unreliable device that stands idle most of the time. (A subscriber could inform the CIE that a certain file should be printed; the CIE would do so and send the result by U.S. Mail.) Of course, the host computer would probably provide a wide range of more conventional computational services, such as games, cross compilers and non-tiny languages.

4.2. Relationship to Other Networks

No current computer net is suitable for the personal computer user: they are all set up for the exclusive use of a library system, a school system, ARPA, a bank, etc., or else they are strictly commercial and oriented to highly capitalized, large volume customers. A CIE net could easily grow up as the first computer net oriented specifically toward the individual. In fact, since personal computer users comprise less than 0.1% of the country's population, the first commercially operated computer nets that appear will also not be oriented toward personal computer users. Thus, even though computer nets are currently in operation and will grow fantastically more numerous in the near future, there is no immediate prospect for any commercial operation that would resemble the CIE net discussed here. That being the case, any crucial parts of its design should be thought out rather carefully because they will be rather less than amenable to change once they are codified in widely dispersed computer programs.

However, as computer nets become less and less inaccessible to the individual, there will be more and more opportunity for a CIE net to complement the other nets. The first such interaction would probably be in the actual connection of one CIE to another: this connection could easily be made via a commercial net, and the enhanced long-distance capability of the connection would then let CIE's use the commercial net for all but the most local distribution of mail. On the other hand, a network oriented toward the non-computerist individual could be connected to a CIE net to give the individuals some limited access to all the goodies the homebrewers have concocted.

5. Conclusions

It might well be practical to build a CIE net that would be accessible to a large fraction of the people making personal use of computers. Parts of the design would require some thought, if known traps are to be avoided, but the problems all seem to be within the grasp of the hobbyists, loosely organized though we be.

6. Acknowledgments

I am deeply indebted to Jon Postel, David Fylstra, Dirk van Nouhuys, Jim Warren and Martin Hardy for many interesting and stimulating conversations that contributed directly to this paper. Too many people to name contributed in less conspicuously direct ways.

Biographical

Mike Wilber was once thought to have met someone

who had seen a computer. The rumor was never verified.

Bibliography

Here are more detailed references to works that are mentioned in this paper (by title or otherwise) or that contain material useful as a background to it. Two of them are available from NTIS, the National Technical Information Service, Springfield, Virginia 22161.

Bolt, Beranek and Newman: "Specifications for the Interconnection of a Host and an IMP" (BBN 1822), Bolt, Beranek and Newman, Cambridge, Massachusetts, January, 1976. (available from NTIS: order number AD A019160)

Ken Colstad and Efrem Lipkin: "Community Memory: A Public Information Network", Journal of Community Communications, LGC Engineering, Berkeley, California, Issue 0, 1975.

Whitfield Diffie and Martin E Hellman: "A Critique of the Proposed Data Encryption Standard", Communications of the ACM, New York, New York, March, 1976 (p. 164).

Elizabeth Feinler and Jon Postel: "ARPA Net Protocol Handbook" (NIC 7104), Stanford Research Institute, Menlo Park, California, April, 1976. (available from NTIS: order number AD A027964)

Robert Melancton Metcalfe: "Packet Communication" (PhD thesis, MAC TR-114), Massachusetts Institute of Technology, Cambridge, Massachusetts, December, 1973.

Michael Rossman: "Some Indications for Community Memory", Journal of Community Communications, LGC Engineering, Berkeley, California, Issue 0, 1975.

Mike Wilber and David Fylstra: "Homebrewery vs the Software Priesthood", Byte, Peterborough, New Hampshire, Issue 14, October, 1976 (p. 90).

SHARING YOUR COMPUTER HOBBY WITH THE KIDS

Liza Loop
LO^{OP} Center, Inc.
8055 Old Redwood Hwy.
Cotati CA 94928

Computer hobbyists are accomplished learners. We have bootstrapped ourselves into a position of expertise in the field of electronic computers. We have technical know-how and we have equipment.

Many of us share our enthusiasm with our own kids and a few neighbors. But what about the rest of the kids whose parents aren't computer freaks? Many of them would get a kick out of hobby computing too. As you probably know, kids of all ages take to keyboards and controllers like the proverbial ducks take to water. They seem to know instinctively that it is their job at this point in their lives to master the tools of their society. The way things are going, every home, office and plant will make use of these electronic tools in the near future.

Unfortunately, few school teachers or scout leaders have either the know-how or the equipment to allow their charges to access computers. Many teachers can't even allow access to the most exciting kind of learning - that which grows out of our own curiosity, searching, reading, experimenting, building, and sharing. You and your homebrew computers are the key for hundreds of kids to discover the same joys (and agonies) that you yourself have gained from your involvement with your hobby.

Imagine yourself at the front of a classroom of twenty-five sixth graders. You have just spent the last fifteen minutes setting up and booting your system amid questions about how it works, whether this is the computer that sent that nasty note from the phone company, and where is the robot. The teacher has gotten the kids all seated and reasonably quiet. It is your turn to speak. What should you say? How much can they understand? What information do they want or need? I'd like to share with you some of the things I have discovered in two years of presenting small computers to groups of children between five and fifteen years old.

Many of you know more about electronics, programming, data processing, and automated control than I do. But teaching children (and many adults), especially in groups, is very different from studying technical material on your own. I hope I can encourage those of you who hesitate to share your hobby with the kids by giving you enough pointers on teaching to get you going. For you who have already started to demonstrate your systems to children's groups, I hope you can use some of my ideas to make your presentations more successful for both you and the kids.

How to Contact a School or Kids' Group

The most difficult thing about finding groups to talk to is that it involves several "cold" telephone calls to strangers at strange institutions. Most adults find the world of schools and children's organizations very remote from their present day experience. Even seasoned parents are often intimidated by these institutions. Just finding a local school to contact can be a frustrating chore. Try the phone book under Mycity Schools in the white pages for a start. If you are directed to look under the name of the individual school, don't give up - most cities have a Mycity High and the nice lady in the office there can give you the names of the other high, jr. high, and elementary schools in your vicinity. Explain to that lady that you have a presentation on computers and you would like to contact the President of the P.T.A., as well as the math, science, electronics, business, and vocational education departments to see if any of their teachers would be interested in inviting you to talk to their classes. Assure her that you are a public spirited citizen with

nothing to sell. You just want to share your fascinating hobby with the kids and teachers. If possible, get the names and numbers of all these individuals and try to contact them by phone. Teachers may not be able to take a call during school hours so be prepared to leave word for them to call you back between 2:30 and 3:30pm. When you reach the teacher or P.T.A Mom, tell them without ONE WORD OF COMPUTER JARGON that you have built your own small computer and you would like to bring it to school to show the kids. Briefly describe your presentation (we'll talk about that in a few minutes) and when they sound confused say that it will make more sense when they see it. If the person is hesitant about turning you loose on their kids, it helps to offer a pre-session for the faculty or parents. This will also pay off because you can then keep the grownups from hogging the terminal during the kids' presentation. (An intriguing computer game can make the most conscientious teacher forget his role!) Finally, set up a date and time. Be sure to get exact directions to the classroom you will be visiting including access from parking lot and number of stairs to be negotiated. Also warn the teacher of your needs for tables and electrical outlets.

My class presentations are always short, and, from a hobbyists point of view, totally without real details on computer hardware or programming. No matter how simple or entertaining you try to be some kids will get bored. Forty-five minutes is about as long as you can ask a class to sit down and pay attention. Fifteen minutes of talk with unlimited hands-on is usually very successful.

If at all possible, arrange with the teacher to move your system to an unused room after your formal presentation. The hall, the nurse's room, or the principal's office might be available. There you can entertain the enthusiastic kids sent to you by the teacher in groups of three or four without the hecklers who were bored earlier. If you are a beginner at working with kids you don't need to try to learn to control a class of thirty-five active children right away!

Try to leave yourself plenty of time after your presentation just in case someone gets really hooked right away and wants to stay after school. Be sure they call home for permission to stay if you don't want their parents to oppose their new interest from the start. Also it's nice to reserve some faculty time without kids around so that the teachers may let their ignorance show without blowing their image in front of their students. That will happen soon enough anyway.

Other children's groups are similar to schools. Try Boy and Girl Scouts, the YM/WCA Mycity recreation department, Teen Clubs, Ham radio clubs, etc. It may take some perseverance to locate the names of the leaders of these groups but ask your friends and neighbors. When you contact the schools ask about what special interest clubs there are on campus. Also, ask to be put on the speakers list for each school. When the other teachers hear what a hit you are they will want to invite you to their classes too!

Composition and Character of Various Groups

What you can present to any given group of children is very much a factor of who the people in the group are. I would like to refer to groups of kids by approximate age but that can be very misleading. An eight year olds' Model Rocket Club may follow you into the rudiments of assembly language programming in an hour and a half. A high school class called Introduction to Business Machines, on the other hand, can be bewildered each time they must push the carriage return. Therefore, please consider my references to age groups as developmental stages and remember that physically adult bodies often hold minds with five year old skills when it comes to computing. Also, don't expect most groups to be preselected on the basis of interest in computers. After

all, it was the teacher, not the kids who invited you. Occasionally you'll be lucky and get a class-full of interested and intelligent people but a wide spectrum of attitude and ability is more common. Success is three or four enthusiasts, not all thirty.

Let's take a look at kids by age groups - starting with the youngest, five and six year olds. (Kids younger than five love computers too but they are awfully difficult to work with in groups.) Most fives can count to ten but are hazy on what "greater than" and "less than" mean. They can probably type their names, very slowly. Don't expect them to be able to read. Fives like noise and pictures, joy sticks (1) and turtles (2). They often come up with original interpretations of your instructions and it may be difficult to dissuade them from experimentation. In other words, if there is an exposed switch on your system, they'll push it. They will not be very good at watching other kids but if you have prepared some tasks which are truly at their level they may demand a thirty minute turn each. In short, five year olds bear a strong resemblance to several computer hobbyists of my acquaintance.

I have found that chaos can be avoided by asking the teacher to start some favorite activity on the other side of the room after you have let everyone see the inside of the computer and printed a Bunny. The teacher can then send you kids two at a time. Let each one play one short game and write one story. Take the names of those who don't want to stop and let them come to the nurse's room later for another, longer turn.

Seven, eight, and nine year olds (2nd and 3rd grades) are into reading short texts and can handle four function arithmetic using whole numbers. Occasionally I am appalled at their total lack of logical ability. This has nothing to do with the intelligence of the children themselves. Rather it reflects whether or not thinking is considered a valuable skill in their school or home environment. You may be the first person who has ever suggested that these children reason for themselves. Of course, you may find you have stumbled onto a lively group you are hard-pressed to keep up with.

Ten, eleven, and twelve year olds (4th, 5th, and 6th graders) begin to show definite differentiation into intellectual versus non-thinking patterns of behavior. This is an exciting age to introduce to computers because a bright ten has all the mental apparatus needed to understand electronics and programming. Furthermore, he usually hasn't discovered the distractions of puberty yet. The non-thinking kids of this age are often still open to learning techniques of problem solving and logic. Computer games or a fascination with soldering things may be the spark they need to reawaken the excitement of learning and exploring. Average kids at this age will probably be fascinated with your system if it can do anything beyond blink lights. The bright ones will be bored because they want you to go faster and show them more. The non-learners will continue to sit in the back row and throw spit balls. Be especially alert for the child who seems painfully slow at reading or responding but who shows a lot of interest or perseverance. I keep hearing successful engineers mutter, " My God, where would I be now if I hadn't discovered electronics when I was eleven? "

By junior high age the division between thinking and non-thinking has become more entrenched and more subtle. The thinkers divide into math, verbal, artistic, and human relations specialists. The non-thinkers may have areas of expertise such as sewing, automobile repair, or mastery of a musical instrument, but they conceive of themselves as basically unable to understand what you are saying to them. People exhibit this trait throughout adolescence and adulthood. Don't buy their line. The smart ones are rarely the geniuses they would have you believe. Some of the others are merely disinterested so

they play dumb. The rest are likely to appreciate it if you start at their level and let them proceed at their own pace. Don't judge them by your own high standards.

High schools are already full of computer freaks waiting to be discovered and welcomed into the fold of computer hobbyists. If you tap into one of these goldmines in a math or electronics class you can really go to town. Show them every hell and whistle your system can muster. These kids will stay up all night trying to figure out what you said. Let them know that you are willing to teach them what you know and you can count on them to show you how. Non-computer highschoolers can be viewed either as younger kids or as adults depending on their social maturity. Remember, they deserve just as much care as the ten year olds even though you may not find them as rewarding to work with.

I usually have my little kiddie material available when approaching adults. Adults think they ought to know something about computers but most often their knowledge is filled with gaps and misconceptions. On top of this, adults are often just plain afraid of machines. If you tell them you are going to start with your kindergarten presentation they will probably giggle about what a good idea that is and relax a little. Later they will come up with thoughtful questions about real world applications of systems like yours and their effect on economic and social conditions.

The reason I dwell on the characteristics of the children you are likely to be working with is that I believe they are all potentially intelligent and enthusiastic people. My negative descriptions reflect only their degree of "turned-offness" not their mental capacity. Because "computers" is a new field in which most of us are beginners every kid has a chance to grow and explore within it. You may turn someone on with a presentation aimed at his personal interests. You may serve as a model of the kind of person he or she would like to become. In this field everyone is a little bit stupid so no one needs to feel stupid.

Material I have found Appropriate to each Group

The specific content of your presentation can be largely the same for any group except the high school computer freaks. How you present that material, how much you can cover, and what reaction you expect from each group changes immensely. Of course, your hardware capabilities have great bearing on what you can show.

If you have just a computer on a board with no games, I would stick to those groups which have expressed interest in computer hardware. If you have a terminal and three or four games you can access you can probably prepare a demonstration for anyone.

I always take the cover off the computer and show everyone the chips and PC boards. Little ones just look. From about eight years old on up I explain IN OUTLINE ONLY how chips and boards are made. Putting an IC under a microscope is fun too. Then I spend a few minutes with the concept of a system - here's the terminal, the CPU with arithmetic, logic, and control, and the core memory. This is the computer's magnetic note pad (we call it a tape recorder) where it writes down the rules to each of the games it can play. If you have lots of time you may want to bring along some useless boards and some soldering irons. Taking boards apart and randomly reassembling them is a good manipulative task for upper elementary and junior high kids.

If you have rigged your machine with a speaker and music program (3) this is a good time to show it. Most people do not have a clear idea of how a speaker works so you will have to explain the whole thing. Watch out for incomprehensible words like "square waves" and "frequency".

After this short hardware orientation I plunge right into computer games. Most kids will spend between twenty and forty hours playing games if given free access to a library like 101

Basic Games (4). During this time they will greatly improve their reading and interpretive skills (if they can't con someone else into doing all the reading for them). Their logic will acquire a new degree of sophistocation. In a group setting where they must take turns at the terminal and explain games to their peers their social skills grow rapidly. Even during a short school visit you can see the seeds of these changes begin to germinate.

I start everybody out on "Guess My Number"(5) because how they approach the game gives me a clue to the level of the group. If noone discovers the binary search, I can forget about introducing programming in this first session.

Fives will enjoy Guess between 1 and 9. Show them the numbers on the keyboard and make sure they see that "too high" means choose a number farther left and "too low" means choose a number to the right. A six to ten line "Story" program (6) which asks for their name, favorite food, color and playmate, then types out a short story will be a hit, especially if you have hard copy to let them take home. Snoopy's and Bunnies (7) are good too but print out one for each child before you arrive at the school and just do the teacher's copy in class. Very low skill real-time games may work for fives too. Pong is too difficult for many of them but Race Track (8) used as a maze instead of a race is fine. Sharp fives may also be able to play Diddle (10) on an Altair front panel. Of course if you have graphics you can keep anyone amused by generating patterns for a while.

If you have an ASR33 Teletype you may find that this is more interesting to your younger groups than the computer itself. They love the noise and rarely have had much chance even to type on an electric typewriter. The paper tape will introduce the idea of coding and the kids will love to punch secret message tapes and read them back on local.

By the way, many schools have closed circuit T.V. If your CRT is small or you are using a TTY for output it's nice to hook up a simultaneous display on the school's large monitor. This eliminates a great deal of pushing.

This same material is equally good for seven to nine year olds. If you have time you can introduce another game or two for them. Unless your group is used to playing competitive games they may have trouble comprehending that they are supposed to be developing skill at solving the puzzle presented by the game. Many of my students find it sufficiently satisfying to get the answer at all. I sometimes have a hard time convincing them to play the game again, trying for a better score. They seem to view the computer as a competitor rather than as a game board on which they play against the programmer. If you have two person games available you can make this point a little more easily.

I like to introduce programming to these kids by talking about giving instructions. A neat demonstration is to have the kids tell you how to make a peanut butter and jelly sandwich while you are blindfolded. Have all the ingredients ready on a table in front of the class. You start by choosing an instruction set with the kids. Try: move arm up,down, right, left, forward, back; grasp object touched; rotate hand, palm up, palm down. Then let the kids call out instructions to you until you get the sandwich made. The final test is to write down the instructions on the blackboard, then bring in a new person and see if he can accomplish the task blindfolded if you read only the words on the board. If you are lucky the kids will begin to see how much pre-programming they bring to a task and how "simple-minded" the computer is.

Some of these kids may be interested in going on to simple Basic Language programming. Print, input, and if-then will let them write math quizzes using impressively large numbers. However, unless you are working with a special interest group or "gifted" students you are not likely to get past the game playing stage. I've seen a kid play Tic-Tac-Toe (11) for an hour a day for four days.

He didn't even want to change games until he had this one mastered. You see, he wasn't just playing Tic-Tac-Toe. He was reading, typing, learning to operate the computer, and manipulating the social situation which provided the computer access all at the same time.

Intermediate grades (ages ten to twelve) are likely to begin to show an interest in computer applications and simulation games. Many of them will thrive on logic games like Bagels (12). They can also handle real time games with considerable skill. Running out of time is my biggest problem with these kids. I have enough projects lined up in my computer awareness curriculum to visit a classroom at this level once a week for the whole school year. In the first visit, though, it is show the hardware, play two or three games, talk about what a program is, and mention some real world applications like banks, point of sale, and school registration.

When you face your first junior high class you are going to have to decide whether your objective is to "teach something" about computers during your first visit or just let the kids have fun. I tend to choose the latter since I know that they will search out "hard" information once they are hooked on computers. I flesh out my games with Hamurabi (13), or Hunt the Wumpus (14) and bide my time until someone asks about programming. Often you will get inquiries about building home systems like yours. These kids don't usually have much money so one graphic demonstration is to have your system configured so that you can tear it down to a bare bones kit in front of them. This is what you can get for \$500. This is what you can do with it. Add this \$200 board and you can do this. Continue adding parts and money until you are back to playing Hamurabi. This can be a discouraging demonstration but it is realistic. Hopefully, we can save them some of the heartaches we experienced in our naive beginnings.

And now on to high school. Many older teenagers already know some electronics and/or computing. Those who are interested will benefit from elementary discussions of hardware and software configuration. The non-computer kids will need the same soft introduction you gave their younger siblings. If you are interested in on-going projects with kids and computers, get the high school kids on your team right away. They can serve as assistant leaders for younger groups and can often manage the organization of their own clubs with you as a consultant. Discuss your project ideas with them during your first visit and let them know that there will be plenty of time for playing computer games later.

On-Going Projects

One forty-five minute show-and-tell session on homebrew computing is likely to yield two or three candidates for some form of continued contact with the machine. The younger kids will be basically interested in a chance to play more games. Remember that twenty to forty hours I mentioned earlier? Their teachers may see the potential for computer aided instruction or a continuing class in computer literacy. Unless this is a field of particular interest to you, you will probably want to serve only as an advisor to a teacher or parents group to help bring this about. The advisor role is crucial to the success of such a project. However, the leader of this endeavor is going to have to dedicate many hours a week to make it happen. An ongoing, in-school project will involve obtaining space in the school to house the system or terminal, supervision in the computer room during and after school (if you don't have after school hours the kids will break in to use the machine anyway...), planning a "curriculum" which sounds convincing to "educators", arranging for timeshare services or building/buying a micro, and financing the whole package. It takes a coordinated effort on the part of one or two teachers, a few students, a sympathetic administrator, and someone who knows quite a lot about computers and programming - maybe you.

Perhaps you would be happier tackling a slightly smaller project? Scout troops offer a merit badge in computer science. If you are willing to be their resource person you can help a lot of kids get a foot in the door without making a major commitment of time and energy.

You may find you have the makings of a computer games club or an after school class in programming which could meet at school or at your house. (Moving your system can get to be a drag after a while.) Since the equipment is yours, you get to call the shots on how often the club can meet, where, and whether there will be dues to maintain or expand the system. If this is to be a school sponsored group, be sure to check with the principal about rules, regulations and use of school property. You will probably need a faculty member to co-sponsor the group although that person may not have to attend every meeting. If you form a non-school related club make sure you contact the parents and explain your plans to them. Be clear about what you are willing to offer so that the club is fun for you and not just a pain in the neck. You need not institute a formal organization with constitution and by-laws, but a one pager explaining your purpose, who is invited to join, and what members responsibilities are will avoid confusion and possible hard feelings later.

Those of you who love hardware will get a kick out of getting together with the kids who want to build their own systems. You can teach them as much as you know and then serve as a liaison person to set up sessions with "experts" you meet through your larger area computer clubs. Of course your services as chauffeur to meetings, computer stores, and conferences will be invaluable. Many parents are ready to provide money to get a kid started on his own kit but they don't know how to help him or her find the people and resources needed to successfully build and use the system. Your willingness to become a group leader (maybe not leader, just resident responsible adult) will earn you the respect and gratitude of the kids and their parents as well.

A few words to the wise about entertaining sizeable groups of people in your home. Check with your insurance agent to make sure you have the proper liability coverage. Also don't tempt fate by leaving money around or calculators unaccounted for.

Hints on Successful Presentations

I'd like to finish out this discussion with a few suggestions to keep in mind while you are preparing your first visits to kids groups.

1. Make sure your equipment works. Have a dress rehearsal which includes disassembling your system, driving it a distance equal to the trip to the school, at equal temperatures. Set up and give your show for friends and neighbors. Arrange this at least two days before your school trip. If you have to debug your system you'll have twenty-four hours to do it and still be able to give twenty-four hours cancellation notice if you blow it.

2. Prepare your "lesson plan" carefully and specifically for the group you'll be addressing. Some people can wing it comfortably. Others find themselves tongue-tied in front of that bunch of expectant young strangers. Worst of all is to discover that your audience isn't understanding one word of what you are saying to them.

3. Therefore, DON'T USE COMPUTER WORDS even if you explain them first in your presentation. People don't learn a new language that fast. Computer vocabulary IS a better means of communication than ordinary English BUT ONLY FOR COMPUTER FREAKS. If necessary, write out the whole text of your talk and underline every instance of computer jargon. Think up synonyms now. Later you won't be able to come up with alternative words. If there are no non-computer words to express your thought, simplify your talk. You can introduce those

computer concepts in your second visit with this group.

4. Try out your presentation on some non-computer, or better still, anti-computer people and encourage them to be painfully honest when you loose or confuse them. You will need some experience dealing with detractors. Also try it out on someone of the same age as the group

you will visit just to be sure you have judged their skill level correctly.

5. Don't expect to get much information across on a first visit. If the kids see the computer, touch it, play one game, that's enough. Your objective is to help demystify the computer, not to show them with how much you know.

6. Don't introduce Startrek or gambling games in class. You'll never get the kids' attention back.

7. Wait for questions to arise in your learner. When a question happens in someone you know there is a place for the answer. Otherwise, you may never know when the input buffer has overflowed.

That is a lot of what I know about sharing computers with kids. I'd be glad to talk or correspond further with any of you on this subject. It's fun. Seeing a face light up in front of a terminal is like giving a present to someone. It satisfies the evangelism that lurks within many of us. So, find a group to visit, learn as much as you can about them before you go, and prepare your presentation accordingly. You'll be a hit with the 4-H Sewing Circle, the Co-op Preschool and the Advanced Placement Physics Seminar alike.

Footnotes

(1) joysticks - I am using this as a generic term for any device which allows input through a lever or moveable table top instrument. Cromenco makes one for Altairs.

(2) turtles - A turtle is a motorized device which moves around the floor under keyboard or program control. It looks something like a turtle and can carry a variety of sensors to provide feedback to the computer. See "Twenty Things to Do with a Computer" by Seymour Papert; in Educational Technology Magazine, Morristown, N.J. vol. 12, April 1972, page 9 to 18

(3) music - see the following articles in Peoples Computer Company, P.O.Box 310, Menlo Park, Ca. 94025. Vol 3 #5 Steve Dompier's Music Program
Vol 4 #3 Minuet in G Major by Cynthia Beyer
Vol 5 #1 A Musical Number Guessing Game by Kurt Inman

(4) for listings of games see -
101 Basic Games by David Ahl; Digital Equipment Corporation, Maynard, Mass.
1975
Creative Computing Magazine; PO.Box 789-m
Morristown, N.J. 07960
Peoples Computer Company; P.O.Box 310
Menlo Park, Ca. 94025
What to Do After You Hit Return; Peoples Computer Company, Menlo Park, Ca. 1975

(5) Number Guess - What to Do, page 116

(6) Stroy - listing from LO*OP Center, Cotati, Ca 94928

(7) Snoopy and Bunny - 101 Basic Games listing from LO*OP Center, Cotati, Ca 94928

(8) Racetrack - I can't find a listing in time for publication. Ask your friends or LO*OP

- (9) oops! - I messed up my numbering system...
- (10) Diddle - see PCC Vol 4 #4 page 8
- (11) Tic-Tac-Toe - 101 Basic Games, page 219
- (12) Bagels - What to Do, page 117
- (13) Hamurabi - What to Do, page 140
- (14) Hunt the Wumpus - PCC Vol 3 #3 page 24

Vita: LIZA LOOP

Ms. Loop is the founder and President of the Board of LO*OP Center, Inc., a California non-profit educational corporation. She is acting director of the LO*OP Center, President of the Sonoma County Microcomputer Club and President of the Cotati Chamber of Commerce.

For the past two years Ms. Loop has been engaged in explaining computers to lay people and education to computer professionals. She has taught "Meet the Computer" classes, Basic Language Programming, and Pilot Educational Programming in Cotati-Rohnert Park Elementary Schools, Petaluma and Santa Rosa gifted students' programs, and in private classes of both adults and young people at LO*OP Center. She has lectured to schools, civic groups, and college classes in Sonoma County. She has personally developed the curriculum for these courses as well as a computer orientation course for office personnel now under consideration by the American Management Association.

Ms. Loop's articles have been published in several journals of national circulation within the microcomputer industry. LO*OP Center, Inc. serves as publisher of one of these, the Pilot Information Exchange.

Ms. Loop was educated at the Dana Hall School in Massachusetts and at Cornell University. She completed her Bachelor's Degree at Sonoma State in 1970 with a major in Philosophy. Her minor concentrations were math and physical sciences. She has since pursued courses in Education and trained on DEC, Hewlett-Packard, Wang, and Data General small computers as well as large time-shared systems.

Currently Ms. Loop is enrolled in the Master's program in Management at California State College, Sonoma, and is teaching units on computers within courses in Management, Psychology, and Education. She is a member of the Lawrence Hall of Science at U.C. Berkeley, the California Association for Educational Data Processing, and the Oregon Council for Computer Education.

Ms. Loop lives in Sebastopol, California with her sons Jonah (3) and Solomon (5).

PERSONAL COMPUTING & EDUCATION: A TIME FOR PIONEERS

Thomas A. Dwyer
 Soloworks Lab
 311 Alumni Hall
 University of Pittsburgh
 Pittsburgh PA 15260

INTRODUCTION

The use of microcomputers in education is closely associated with the personal computing movement. So we have the interesting situation in which a well-established enterprise must decide how (or whether) to handle a promising young upstart from the other side of town. The problem is to rule on the merits of a liaison.

Eventually someone will decide that the answer lies in the magic word "evaluation". From then on it's likely to be confusion time. The reason is that evaluation projects tend to immediately zoom in on methodology. They worry about the "how" of evaluation much more than the "why" or "what".

This problem can even arise when the methodology is fairly broad. For example, one open-minded way to evaluate a new idea is to talk to the regular users (not developers) of the system. If these users are found to be consistently enthusiastic, or if they are anxious to spread the word to others, there is a good chance that something worthwhile is going on. If the new idea is concerned with education, and if the students involved are found to be engaged in such unusual practices as asking to stay after hours, or to come in on Saturdays, it's safe to conclude that something extraordinary is in the wind.

Now for an apparent paradox. If we should attempt to evaluate the worth of "computers in education" by using this user-enthusiasm test, we would get rather uneven results. Probing more deeply to find out why, we would soon find that people were using the same words (and machines) to mean (and do) very different things; that when applying computers to education there is a wide range of possibilities, and that there is all the difference in the world between them.

Now that microcomputers make it possible for many more educators to get involved in these issues, it may be timely to take a quick look at what the options are. We should not assume that past R&D in computer assisted instruction has the answers. It's also timely to note that classroom instructors are much more inventive than assumed by some R&D models. The contributions teachers and students have already made to personal computing prove that.

WHAT MIGHT GO WRONG?

There is a simple test for spotting problem areas in the use of computers in education. The trick is to mentally erase all the machinery and technical talk from the picture, and then ask whether what is being done is in fact what one's philosophy advocates. If it is found that what is taking place is more an emulation of traditional school practice than a move toward one's educational ideals, then it's a wrong use. It's a use that doesn't understand the difference between personal computers as extensions of the human intellect and spirit, and "gadgets" that mechanize sterile techniques derived from past expediencies.

A wrong use of computers can also be spotted by a glaring inconsistency between what's possible, and the simple-mindedness of what's actually done. It's not that a computer can't be used to drill a student in the memorization of facts and formulas; it's that major concentration on such a use misses the potential of far deeper roles. Computers shine in learning environments where they serve as exploratory companions; as impartial verifiers of a student's latest effort at synthesizing a general procedure; as a focus of celebration and exuberance when they finally obey (literally) thousands of precise commands.

WHAT CAN GO RIGHT?

The surest sign that a computer-related educational program is on the right track is the presence of "educational complexity" at the local level. By this I mean that the program evidences a great amount of thought about all the factors that go into a fresh approach to learning (and instruction), and that the responsibility for this renewal is in the hands of the teachers and students involved. Meta-organizations (such as computer networks, or educational R&D centers) are important too, but as servants of teacher/student groups, not as their masters.

The criterion that there be "complexity" in the local plan is best illustrated in terms of what is going on in the classrooms of a number of pioneers. Although the detail varies, several common "discoveries" seem to characterize their work.

Here are four of them.

1. There Is Regular Recognition of Unplanned Learner Accomplishments. I recently visited a school that passes the enthusiasm test outlined at the beginning of this paper with a grade of A-plus. The first thing that the teacher said to me was, "Say, I want you to see what (student's name) here has been doing with difference tables, and then you've got to look at some of these graphs (name) has produced - and would you believe that (name) has now got a program running that simplifies Boolean expressions with the Quine-McCluskey method - he'll have to explain it to you, I don't understand it at all...etc." Needless to say, this teacher's students were standing ten feet tall. Also needless to say, they wanted to talk my ear off several times over, describing what they had been doing in this unique computer math class.

One of the most powerful contributions of computers to education is the way in which they make possible student accomplishments worthy of recognition. Computer-related innovation that does not capitalize on this capability is really missing the boat.

2. The Role of the Teacher has Changed. The example I just gave quoted a teacher as admitting that one of his students was using a technique about which the student knew more than the teacher. The schools that are going places with computer-related innovation always seem to have learned this lesson - that teachers and students bring complementary talents to a learning situation, and things get really interesting when everyone admits this.

Contrary to what one might think, respect for the teacher grows significantly in such an environment. Contrary to what is sometimes said, technology creates a greater need for human teachers than ever, but they must be special people. They must, in particular, be able to flourish in a learning situation where both teacher and student will add to their respective understandings almost daily.

3. The Content and Style of Successful Computer-Related Curricula Are Open to Innovation. Another way to say this is that the "add-on pathology" has been avoided. Innovation has generally been more successful in places where "new" courses have been developed (e.g. Computer Science or Computer Math) than in places where a standard course has had the computer "added on", with nothing being replaced, removed, or revised. There are a lot of reasons why this is so. In the add-on situation, students are being penalized for using the computer; all the new work is added to the old. Secondly, many of the problems presented in pre-computer curricula are pretty silly (if John has twice as many apples as Mary had after Mary gave four less apples to...etc), and using the computer to solve these only propagates an error.

What works best is a fresh look at both curricula and learning environments - a look that takes maximum advantage of several completely unique capabilities of computer-related learning. For example, major (and quite fascinating) projects can be imbedded in curricula at any grade level, once one understands how the computer naturally leads to a problem-solving philosophy based on the use of subroutines (what some people call "procedures"). At Soloworks, we have also found great value in using physical devices that students can control with the computer. This almost always ends up involving several disciplines. It's really a rewarding learning experience to find that the physics of heat, the mathematics of large linear systems, and the joy of cooking have something to say to each other. The success of computer-related innovation is directly related to a willingness to innovate at all levels and in all directions.

One of the most interesting forms our curriculum work at Soloworks has recently taken is the development of "micro-courses". A microcourse is defined as a cluster of "modules". The key module in the cluster is one which presents the student with a challenging problem. Then other modules are developed to support achievement of this challenge. Finally, there are modules that examine a general theory derived from these specifics. Notice how this order is just the reverse of that pursued in a typical college course. It's what we call a top-down approach, where "down" means in the same direction that great invention has always moved. Why should school be different? (Two books that illustrate this approach are "Software Tools" and "An Amateur's Guide to Personal Computing", both published by Addison-Wesley Co., Reading, Mass. 01867.)

4. The Best Things Happen Where the Tail Hasn't Wagged the Dog
 It is sad to report that vested interests, both political and

economic, can and do get involved in computer-related educational innovation. Education deserves better than this. The going is tough enough when something new is tried without muddying the waters with non-educational factors.

There now exist microcomputer hardware and software systems that provide an excellent match to the requirements of the educational world. Trying to convince oneself that technology developed for other purposes (e.g. administrative computing) is just as good is an exercise in self-deception. It's simply not a good idea.

On the question of money, the only way to go into computing is to allocate the right money for the right system. Using microcomputers, far less money than it takes to equip a science lab or gym is needed for initial outlay. A yearly maintenance budget calculated on the basis of a lab fee per student should also be budgeted - again at about the same level as is now being spent for a physics or chemistry lab. A budget line-item to add new features (e.g. graphics capability) each year is also sound planning.

CONCLUSION

A quiet revolution is taking place in education. The ingredients are microcomputer technology, dedicated teachers, the curiosity of students, and a lot of common sense. The examples I have given say something about the way in which this common sense manifests itself in those places that are leading the way. The real contribution of personal computing to education is that it is making it possible for many others to join in this move to new frontiers.

As mentioned earlier, microcomputers have made the cost of pioneering reasonable. Using the cost of an average school gymnasium as a reference point shows that a computer lab is quite a bargain. The athletic analogy also helps solve the evaluation problem. Students using well-equipped computer labs in the manner advocated here would soon be making a name for their schools. And how do you argue with a winning team?

The pioneers will know that there are better criteria of course, and have the security to make their own evaluation. For them the metaphor will have a deeper significance. The word gymnasium comes from the Greek "gymazein" which meant "to train naked, unfettered". And that's what educational computing is about; an unfettering of the imagination and intellect, a freeing of the capability to learn. A single microcomputer can become a gymnasium of the mind, an arena for the imagination. It's a bargain no school should resist.

THE THINGS THAT WE CAN DO WITH A MICROCOMPUTER IN EDUCATION THAT WE COULDN'T DO BEFORE

Lud Braun, Professor
College of Engineering
SUNY at Stony Brook
Stony Brook NY 11794

1. Where We Came From

Before 1965, educators who wanted to have their students use a computer to enhance their learning in a course sent their students to key-punch machines at the local computer center which practiced that affliction called batch processing. This computing mode was made necessary by the high cost and operational complexity of computers in that era. This was great for computer-center directors who could boast of 95% utilization factors for their machines, but it was lousy for the students and faculty who were lucky if they got 10% utilization factors for their time.

About 1965, along came Kemeny, Kurtz and others who decided that batch processing was a terrible waste of people power and that there had to be a better way. The "better way" which they developed was called time sharing. The computer cost was shared among a group of people, as in batch processing; however, instead of serving users serially as in batch mode (with its resulting queue and long turn-around time), users essentially were serviced in parallel, and efficiency of utilization of people time increased dramatically. Another advantage of time sharing which was important to educators, and which was due largely to the cleverness of Kemeny and Kurtz in their design of the BASIC language, was the capability which existed for interaction between the user and the computer. Many of the interesting applications of computers in education in the late 1960s and early 1970s (including the Huntington Two simulations) had substantially more impact (and, in some cases, only became possible) because of this interactive capability.

By about 1968, educators (including this author) began to notice the DEC PDP-8 which was designed originally as a laboratory computer. Such minicomputers had most of the advantages of time sharing, plus a cost which was significantly lower (over a 3-5 year period) than time sharing,* and they added two other significant advantages to educators:

1. The user was in total control of the computing environment, rather than being subject to the high priests of the computer center; also no local computer bum could crash the system or steal or modify programs belonging to others.
2. The minicomputer was inexpensive enough that a district could buy a machine rather than rent it. This meant that the interested teacher only had to talk his administration into springing for computing bucks once, rather than having to go back every year and being subject to budget cuts, etc.*

We are in 2AA** and are just beginning to perceive dimly the outlines of the possibilities of the microcomputer revolution; however, it already is clear that the microcomputer has added some new advantages over those of the minicomputer, which is better than time sharing, which is better than batch processing, which is better than no computing at all. We'll look at some specific advantages in Section 2, but there are a couple of general advantages of the microcomputer which are worth mentioning here:

1. The cost currently is about one third that of an equivalent minicomputer and it is coming down rapidly. The cost of semi-conductor memory (the most expensive component) for a microcomputer has dropped by almost a factor of three since the spring of 1975. By 5AA prices should be down, by another factor of 5-10; and by 10AA, we may be in an era of "zero-cost computers" where computers will almost be throw-away items. Will this mean that every student will own a personal computer? Who knows? It is interesting, however, to specu-

*This cost advantage has been diminished significantly by the introduction of time-sharing minis like the DEC PDP-11/45 and the HP 2000 series machines.

* Even so, the teacher sometimes has problems. I know one teacher who used rolls of paper towels in his teletype-writer because his administrators didn't order teletype paper for him.

**AA stands for Anno Altair. January, 1975 is t=0 and is the time when the Altair was announced. It marks a real milestone in educational computing.

late on the effect of this kind of development. The education community still is reeling from the impact of the \$10 pocket calculator. Let's hope that we are better prepared for the \$100 PDP-10 than we have been for the \$10 calculator!

2. The microcomputer weighs under thirty pounds (if we assume that it includes a keyboard and that the display device is a TV set with a video tap -- at home or in school). This means that the teacher can carry it into class under his arm, or that he can take it home over a weekend, or that he can let his students take it home! We have come a long way from the time when this author plus a strong student jackassed just a teletypewriter between his office on the fourth floor and his classroom on the eighth floor. (We never got a hernia, but we came close many times.) This portability is extremely attractive: It has meant that for the past year, the author has been able to do computing in his classroom without worrying about whether or not the room has a telephone to call the campus computer or whether the campus computer is in one of its blue funk, or whether one of the campus computer phreagues has invaded his files.

It is clear to this author that the days of time sharing and of minicomputers and batch in educational computing are numbered and that by 10AA (and maybe as soon as 5AA) the microcomputer will dominate educational computing -- except for a few applications like data-base manipulation and large-scale CAI (even here, with CCD and magnetic-bubble memories and video discs, time sharing may be in trouble).

In Section 2, we shall look at some special features of microcomputers which either have not existed before or have been prohibitively expensive until now.

2. Some New Goodies with Microcomputers

Because of the unique character of the 8080 microprocessor and the versions of BASIC which have been written for it, it is now possible to increase substantially the degree of interaction between the computer and the user over what could be accomplished previously. In Altair BASIC, there is a command with the form INP(N), which tells the computer to read the present value of the signal at Channel N (there are 256 possible input channels). There are two basic kinds of things that we can do with this command. The simpler of the two is the modification of program operation by entry of a character from the keyboard (which is channel 1; i.e., N=1). In this mode, we can use statements of the form

IF INP(1)=80 THEN INPUT "P=";P

This statement* temporarily halts execution of the program when the user hits the P key during program execution and requests that the user enter some new value of the parameter P. As soon as the new P is entered, and the user hits Carriage Return, the computer continues execution from the point where it was halted, but with the new value of P. This feature is useful, for example, in the solution of differential equations where a parameter changes suddenly. One such case is the simulation of respiratory mechanics with respiratory resistance changed dramatically during an asthma attack.

Another kind of application involves the use of a Cromemco D+7A A-D Converter* plus the INP command. If we connect the output voltage of a potentiometer to Channel 31 and insert a statement of the form

X=INP(31)

into the program, we can change the value of X used by the program continuously by varying the potentiometer. This capability permits us to treat the digital computer as if it is an analog computer.

If we take advantage also of the D-A or digital-to-analog converter capability of the D+7A and the OUT command in BASIC, we have the capability of displaying the computer results in analog form as well.

The combination of the D+7A and INP and OUT commands opens up a new dimension in the use of the digital computer

*The number 80 is the ASCII equivalent of the letter P.

*Such a converter takes analog (or continuous) signals as input and converts them into digital form so that the computer may handle them.

which may be exemplified by an interesting modification of the ubiquitous Lunar Lander simulation. In most implementations of this program, the computer asks how much fuel the user wants to burn during the following second. The computer prints out the velocity of the lander, its altitude, and the remaining fuel at the end of the time interval, and asks the user to enter his next decision about fuel.

Using the D+7A and INP and OUT commands, we can modify the Lunar Lander so that:

- a. the user enters the fuel to be burned continuously using a potentiometer and INP command,
- b. the computer, using the OUT command, displays the remaining fuel, the velocity, and the altitude on three voltmeters which might be labeled "Fuel Gauge," "Speedometer," and "Altimeter," resp.

This capability will make a whole range of computer simulations much more realistic for the student. Essentially, the combination of the microcomputer and the D+7A board converts the digital computer into a very powerful general-purpose analog computer without the normal problems of the real analog computer. This may prove to be one of the most exciting educational applications of the microcomputer once we fully understand how to use this capability.

Some of this author's students have used an Altair 8800 B microcomputer in an interesting way as a component in several engineering design projects. These projects are:

- a. The design of a device to convert printed text into a set of electrical signals to stimulate the skin of the abdomen and to replace the lost visual sense by another sensory channel. The microcomputer here is supplemented by the Cromemco Cyclops digital camera, which converts visual images into digital signals which are processed by the computer to determine the patterns of excitation of a set of skin electrodes.
- b. The design of a typewriter which can be operated by severely handicapped people who cannot use a conventional typewriter.
- c. The development of a biofeedback system to permit victims of cerebral palsy to control muscle tremors which they cannot control normally.

There are other exciting capabilities which this author hasn't yet had the time to explore, but which seem to offer exciting possibilities for the educator. Among these are the TV dizzler and other graphic devices, speech generators, speech recognizers, and music synthesizers.

What new things are in store for us in this area? It is difficult to predict, but if the past two years is any indication, the possibilities boggle the mind!

CLASSROOM MICROCOMPUTING: HOW ONE SCHOOL DISTRICT LEARNED TO LIVE WITH THE STATE OF THE ART

Peter S. Grimes, Curriculum Supervisor
 San Jose Unified School District
 1605 Park Ave.
 San Jose CA 95126

Introduction

Last year was a memorable year in San Jose Unified. The cost of computers had finally dropped to a price we could afford. After many years of dashed hopes we were able to demonstrate that an amount of money which did not seem unreasonable (about \$25,000) would purchase an amount of student computing power that was significant in terms of student contact and curriculum impact. This investment, together with an older 8-user PDP 8/E time-share system, would place two computer terminals in each of our six comprehensive high schools and one computer terminal in each of our seven junior high schools.

The paper that follows is the story of how we achieved this goal and coped with the attending problems. Ordinarily this would seem to offer nothing more than a droll recitation of how a school district purchased eleven microprocessors for use in its mathematics classrooms. I was assured, however, that we were at the threshold of a new era of classroom computing and that our early experience with the new state of the art would be useful information to other schools and school districts contemplating the introduction of the microcomputer into their curriculum. We have run into some unexpected difficulties which you can avoid repeating. We have given much thought to the role of the microcomputer in education. We have gotten involved with teacher inservice which is important to the operation and use of a microcomputer as contrasted to the operation and use of a time-share terminal. We are very familiar with costs and can give some good advice on what you will need in the way of hardware to get started. We also have opinions about obsolescence, product support, maintenance, and assembling your own system which should help you form a reasoned opinion about microcomputers and education.

We are excited about what we have done this year! We have formed a deep commitment to the future of classroom microcomputing. The micro-minituarization of digital electronics continues to progress rapidly. The day of inexpensive computing has arrived. Its effects on the curriculum will be profound. We truly believe that we are in the midst of a revolution of unusual significance to our culture. Since we view what is happening as positive and beneficial, we are convinced that the appropriate policy for our schools is to advance in concert with what people are doing. The tide of change is sweeping in rapidly. We should start to prepare for the future now rather than be engulfed later.

Classroom Computing in San Jose Unified

Back in 1972 we purchased a PDP 8/E 8-user time-share system. Starting with several terminals in only two schools, we gradually expanded the system to its full 8-user capacity. By 1974 we had one terminal in each high school and several junior high schools. Programs to utilize the terminals rapidly proliferated. With the additional help of programmable pocket and desk top calculators, our high schools introduced programming into the mathematics elective curriculum, and junior high schools introduced computer literacy as a component of their required mathematics program.

Although this computer facility was very limited, it did provide the incentive for teachers to become acquainted with BASIC and it did lay the foundation for further development as hardware prices dropped to within an affordable range. By the 1975-76 school year it became apparent that with the advent of microcomputers, a significant price breakthrough had occurred. With encouragement from top level administration, we purchased an IMSAI 8080 machine in March 1976 and evaluated it for a major expenditure program during 1976-77. After some tinkering with the teletype paper tape reader, the system ran without fault for eight months (when the paper tape reader failed and we had to add an audio cassette I/O for system loading).

By May 1976 we had concluded that microcomputers were reliable, that excellent BASIC interpreters were available and that enough money could be made available to significantly upgrade our computer resources. Price, reliability, and software were the major ingredients of this determination. But just as important to our considerations was the fact that the physical presence of a complete computer system, as contrasted to a time-share terminal, opened up vast new areas of instruction which were previously out of reach (system operation, assembly programming, and computer science, for example). At a cost of about \$3,000 for each system (we were still committed to teletype terminals last year) we could afford to place a computer in each junior and senior high school.

The decision, then, was to enter into a two year program which would provide one computer terminal in each of our seven junior high schools, two terminals in each of our six senior high schools, and a roving terminal for our elementary MGM program. Moving toward this goal, we have purchased, as of March 1977, a total of ten IMSAI 8080's and one Polymorphic 88 (the Polymorphic was a less expensive machine which we wished to evaluate). Most of the machines were configured with 12K of RAM and an MIO to interface a teletype terminal and audio cassette for system program storage. We chose the ASR 33 teletype terminal because students could save programs on paper, a point which we thought to be psychologically significant for the neophyte programmer. However, we did configure one IMSAI and the Polymorphic system with a vidio interface and TV monitor, and two other IMSAI's with Lear Siegler serial CRT terminals. This was done partly to save money and partly to evaluate the suitability of systems which do not produce hard copy. As for software, all of the installations are using 8K BASIC interpreters (which most math teachers consider minimal) with one exception. We are using 4K ROM BASIC for the elementary roving system. Although ROM is expensive, it does avoid the hassle of loading the BASIC interpreter after each power down--and 4K BASIC is perfectly adequate as an introductory language for elementary students. To date, our elementary system has operated flawlessly for six months with no operating difficulties on the part of the elementary teacher or student.

During the 1977-78 school year we will need to acquire several more systems to meet our two year goal. The advent of microcomputer time-sharing, however, may enable us to add more terminals than originally planned. (Cromenco, a local manufacturer, is introducing an 8-user time-share microcomputer for the school market.)

The Microcomputer Has Put Us in a State of Altered Consciousness

When we purchased our PDP 8/E system four years ago, we weren't familiar with the rapid progress taking place in the field of large scale integration. As a result, we were still thinking in terms of expanding our time-sharing system in the fall of 1975. Our goals were rather modest, consisting of the development of computer literacy in the junior high school mathematics program and continued expansion of our support of computer programming in the senior high school curriculum. When we became aware of the microcomputer, our thinking underwent rapid and radical change.

With single terminal BASIC speaking systems available in the price range of \$2,000 to \$3,000, it was quite obvious that time-sharing systems were no longer cost effective. We recognized that time-sharing, in situations where large data bases had to be maintained, was still valid, but we also recognized that our envisioned use of computers for instruction did not require data base access. It was clear to us that even tutorial and drill and practice CAI, both of which have fairly large memory requirements, could easily be accommodated by the relatively inexpensive (\$1,000 - \$2,000 range with prices still going down) diskette mass storage systems. Since our only reason for instructional time-sharing in the first place was cost reduction, and since microcomputer systems were each less expensive than the addition of a single time-sharing terminal, we completely reformulated our rationale for the use of the computer in the classroom. After a few weeks study and interaction with local institu-

tions like the Peoples' Computer Company and the Lawrence Hall of Science we came to these conclusions:

- Computers and computer concepts (D/P, CAI, Information Retrieval, Problem Solving, Computer Games) should be introduced to elementary pupils.
- Computer literacy should be forthrightly introduced into the junior high school mathematics curriculum.
- Computer programming and computer operation experiences should become a regular feature of the senior high school elective mathematics curriculum.
- A new course, Introduction to Computer Science, should be added to the senior high school mathematics curriculum.
- The use of computers as an instructional tool should be extended into other curricular areas as rapidly as resources permit (science, business, social studies, for instance). Such uses would include simulation, drill and practice, and problem solving.
- Broaden the definition of computer to include programmable pocket and desk top calculators.
- Modify the existing mathematics curriculum so as to include the obvious effects that easy access to computers will have: i.e., the phasing out of logarithms and trig tables and the phasing in of pocket calculators as the source for these functions; phasing out of the slide rule; phasing in of iterative and recursive mathematics techniques.

The logic which compelled us to make the above recommendations were based on these very probable future developments:

- Continuing micro-minituarization and the associated drastic drop in costs will make computers and related devices commonplace at work and in the home (BASIC speaking pocket computers, home computers, computer T.V. games).
- Microprocessors will become common as control devices in consumer products (microwave ovens, home washer/dryers, electric typewriters, television, automobiles, clocks, watches, cameras).
- Digital electronics will become a very large new career area spanning all degrees of sophistication and preparation in research, design, assembly, and service.
- Present industrial and commercial uses of computers will expand dramatically (data processing, information retrieval, word processing, and process control).

The above points indicate that vast numbers of people will need to become familiar with computer concepts of varying degrees of sophistication. Millions of people will work in the area of digital electronics as operators, service technicians, programmers, engineers, and research scientists. Our entire population will become consumers of digital electronic devices. Indeed, as computers invade the home and become as readily available at work as the typewriter and calculator, various degrees of computer competency and knowledge will become survivor skills in a society characterized by the commonplace application of digital electronics to everyday life. It is this conviction that leads us to the conclusion that computer literacy must become part of the core curriculum of public education, that the effect of cheap computing on the existing mathematics curriculum must be accommodated, that computers and related devices should become instructional tools, and that computer career electives be included in the course of study at all deliberate speed. Our students must be prepared for the new world which is going to be so dramatically molded by digital electronics.

To Assemble or Not Assemble--That is the Question

Two years ago ALTAIR (MITS) brought forth the 100 pin bus and the 5" x 10" "plug in" printed circuit board. Following explicit instructions, it was envisioned that a novice could assemble a complex CPU, 4K RAM memory, or serial I/O interface. After all, LSI had reduced enormously complex circuitry to small dual-in-line packages of N number of pins which could be soldered into precisely matching plated through holes in a PCB. True, there were still discrete components such as resistors, capacitors, diodes and power regulators, but they were easily handled. What was important was that

complex logic circuitry had been reduced to functional sealed units which were easily assembled as long as due respect was given to the possibility of cold solder joints, bent pins, solder bridges and the like. Thus, with some dexterous skill, even a 7th grade junior high school student should be able to assemble a 4K or 8K RAM memory, plug it in and have it work!

But wait! Is the real world really like that? We have had numerous students and teachers assemble with results mostly negative. The product more often than not does not work. Sometimes the cause of malfunction is obvious. Indeed, look for solder bridges and improperly inserted pins. Just as often, however, the problem resides in a faulty DIP. I am informed, for example, that a standard DIP such as a 2102 memory chip has a 5% failure rate in commercial kits. This projects a probable failure rate of about $P = 0.9625$ for an 8K memory. The problem, then, is to have the expertise to locate the failure. If one does not have the expertise, then one must find it and then pay for it. Our assembling experience has been that while debugging is commonplace and expensive (the going rate is over \$20 per hour), we usually end up paying less in total (kit plus cost of debugging) than the cost of an assembled unit. Yet the satisfaction of a successful assembly is missing, and this can be a potentially devastating experience for a young student. Of course, one can pay more for kits with hi-rel components, but competition being what it is, "high quality" is not presently a common characteristic of the kit market.

Now, multiply the chance of a board assembly failure times the number of assemblies in a microcomputer and then add the complexities of the necessary input-output devices and one begins to get a picture of the problems encountered in assembling and starting up one's own microcomputer system. Even when a microcomputer is purchased from a reliable vendor, troubles abound. One of our pre-assembled installations, for example, had the following chronology of problems:

1. Teletype printer errors
2. Memory failures during "burn-in"
3. Wrong microprocessor--a NEC 8080 was substituted for an INTEL 8080. We found the BASIC interpreter would not run with the NEC 8080.
4. Faulty teletype paper tape reader which characteristically dropped several bits during the course of reading in an 8K byte tape.
5. Failure of a major microcomputer hardware manufacturer to release a bootstrap loader for its cassette I/O device (needed to circumvent the faulty teletype paper tape reader which we were never able to adjust).

Altogether it took us three months to get the system up and running because of the time it took to obtain reliable technical help and the delays involved in trouble shooting and getting spare parts. Of course we discovered some of the problems ourselves. For example, after repeated telephone calls to the manufacturer we were told: "Oh, you must have one of those NEC processors! Our BASIC doesn't run with the NEC 8080."

If you choose to assemble for its educational value, however, help is available. Vendors and manufacturers are beginning to provide debugging and repair services of a quality and cost which makes assembling your own system very worthwhile. You will learn a great deal and your cost will still probably be less than a fully assembled system. But there will almost certainly be frustrations. Your system in all probability will not work on completion of assembly. You will need help. There will probably be lengthy delays and you will not be able to determine the cost in advance for a burned-in successfully running system.

These problems can be extremely vexing and, because of the delays, costly to morale. You will have to make your decision whether to assemble or not to assemble based on such factors as:

- The educational value to be gained
- Availability of repair money
- Your ability to trouble shoot and debug (which very few educators can do)
- Your tolerance for delay and other frustrations.

If we had this year to do over, we would do things differently. We would not have assembled as many systems to save money; we

would not depend on teletype paper tape readers for system program input; and, we would have purchased more functioning systems complete with terminals and audio cassette input/output.

Recommendations for Making Your Purchase of Microcomputers a Pleasant Experience

In purchasing your own microcomputer systems, I would recommend the following:

1. Do not rely on teletype paper tape readers (the ASR 33) to read in system programs such as a BASIC interpreter. While some TTY's work, others do not. They are quite satisfactory for saving short student generated programs, but lengthy tapes simply multiply the chance of failure to the point of near certainty and a down system.
2. We have found CRT devices to be entirely satisfactory as student terminals provided the student can save programs on cassette tape. We have used commercial CRT's (such as the Lear Seigler), television monitors with vidio boards and modified home television sets. They all work with high reliability compared to the teletype.
3. If you are buying an assembled system, I would suggest you buy a complete turn-key installation with a 90 day or longer warranty and a fixed delivery date. Don't do as we did by purchasing terminal and microcomputer separately. That simple error caused us months of delay in establishing our systems because one vendor was not responsible. It is a simple fact that technicians representing different vendors have a very difficult time getting together. Even when computer and terminal are purchased from one vendor, but separately, you may encounter extra costs and delays if the purchase does not include installation and successful interfacing.

Costs and Configurations

A minimal performance secondary classroom microcomputer will need to consist of the following components:

<u>Component</u>	<u>Approximate Cost</u>	
	<u>Kit</u>	<u>Assembled</u>
Central processor	\$ 200	\$ 300
12K of memory	450	650
I/O interface (for cassette recorder)	200	300
Cabinet and power supply	200	300
Vidio board	200	300
T.V. Monitor	150	150
Audio cassette recorder	100	100
Keyboard	100	150
Miscellaneous cables and hardware	50	50
Operating system	100	100
8K BASIC	0	0
	<hr/> \$1750	<hr/> \$2400

The operating system is a software substitute for a front panel. It enables the operator to manipulate the computer on the T.V. monitor instead of flipping switches and observing display lights on a front panel. A hardware front panel, instead of an operating system, will cost from \$100 to \$200 extra in kit, or from \$200 to \$300 extra assembled. The BASIC interpreter and an assembler are either included free with purchase or listed separately at a low price, generally well under \$100.

If the preceding is minimal, what would be considered better? Two items: hard copy and mass storage. If a teletype were substituted for the T.V. monitor and vidio interface, the additional assembled cost would be about \$700 (reconditioned teletype cost of \$900 less \$450 for the vidio interface and T.V. monitor plus \$250 for a front panel (a hard copy system must have a front panel for computer control)). There are much better hard copy printers on the market than teletypes--they print faster (30 characters per second vs. 10 characters per second) and are somewhat less expensive to maintain, but they are more expensive at around \$2,000 and do not have a

paper tape reader and punch.

Microcomputer mass storage is provided by the new diskette storage systems. They have large file capacities (100K to 250K character storage) for data and program save. Assembled diskette storage devices presently cost in the \$1,000 and \$2,000 range including interfacing and software file management, but these prices are still subject to fairly substantial reduction over the next year or two as technology and production gear up for the potential market.

The best microcomputer system, of course, would be a combination of hard copy and mass storage, but the cost of such systems would be in the \$4,000 to \$5,000 range at which point we are back to examining the cost effectiveness of time-sharing.

Before making a decision, you should compare microcomputing costs to the rental for a time-share terminal. A typical time-share terminal will be linked to a computer supporting up to 32 teletypes. With SUPER BASIC and very ample file storage, the average charge is \$300 a month including all communications costs and terminal maintenance. This monthly rental computes to \$3,600 a year, a charge which makes micro-computer systems very attractive.

Adding additional fuel to the fire, an exciting announcement is being made this very month (April 1977). Cromemco, a local microcomputer manufacturer, will market an 8-user time-share microcomputer. Starting at about \$3,000 for a basic single-user system, additional user terminals (hard-wired teletypes or teletype simulating CRT terminals) can be added for about \$1,200 each, including all interfacing. For an 8-user system, fully assembled and installed, the cost per terminal would be about \$1,400. That's not bad for a teletype system!

What about maintenance costs? At this time our experience is rather limited. We set aside \$3,000 for this year's support of our 11 microcomputers and terminals (including teletype maintenance). It appears that this figure is within the ball park. I think that \$300 per year per single user system will be about right for next year's maintenance budget. You should add about \$100 to this amount for hard copy systems using teletypes or other printers because they require periodic attention for cleaning and lubrication.

What about future costs? I hear talk of major manufacturers coming out with \$500 "sealed units" which would drive T.V. monitor terminals (or perhaps use home T.V. sets with antenna input). Software would be provided by plug-in ROM's. This figure seems not unreasonable to me. I would expect a good BASIC with all functions intact, but no user program storage. Such systems would be ideal for general problem solving, programming instruction and computer gaming and would make microcomputing in the classroom a forgone conclusion.

Lack of Standardization

They say every rose has its thorn. This is certainly true of microcomputing. Although microcomputing will have extraordinarily desirable effects upon the public school curriculum, those very desirable effects may be significantly delayed by the lack of standardization.

By being first on the market with a microcomputer, Altair in effect established a defacto standard with their 100 pin bus. All well and good. Different manufacturers could market devices which were hardware compatible with each other and reap the obvious benefits of an expanded market created by such standardization. But things have not turned out to be that simple. We find that Altair BASIC will not operate with an IMSAI I/O unless hardware alterations are made. Another example of incompatibility is finding a BASIC interpreter that will work with a Polymorphic vidio board in an IMSAI machine. IMSAI basic won't; Polymorphic BASIC won't; but Altair BASIC will (with modifications). What we are finding is that contrary to claims of the manufacturers true hardware compatibility does not exist. While this may not be a large problem to computer hobbyists who have the technical skills to make hardware and software modifications to accommodate the incompatibilities, there are large potential markets for general computing in which lack of standardization will be very harmful. The public school market is one such example. Lack of standardization has certainly caused us problems by

creating unexpected delays, added expenses, and reduced selection of software options.

The problem, of course, lies in the added expense of custom fitting a desired component into a system of different manufacture. Perhaps of even more significance is finding someone to do the work for you. A perfect illustration of this situation is the addition of audio cassette I/O. Seemingly simple, it turns out to be quite complex. Not just any cassette recorder can be used. The cassette interface board must be "tuned" to match the recorder. Tapes dumped from one system cannot be reliably loaded into another, thus severely inhibiting program interchange. Software to load and dump is not readily available (we have had to pay for the development of special software for our IMSAI's). Of even more importance is the fact that very few BASIC interpreters allow user developed BASIC programs to be saved. The save routines are generally completely separate from the BASIC interpreter, are awkward to use, and usually require the user to exactly specify that portion of memory he wishes to save. Microcomputer users desperately need cassette file commands to become a standard feature of BASIC interpreters. But this cannot happen, of course, until cassette I/O technology becomes standardized.

Perhaps of even more importance to the standardization dilemma is the proliferation of BASIC interpreters. BASIC has become the language of choice for large numbers of general purpose computer users who program in the interactive mode. It is certainly the lingua franca of public school classroom computing. And its popularity continues to increase as many of its disadvantages, as compared to other languages, disappear with steady improvement. BASIC has now evolved into SUPER BASIC with added commands and functions, better format control, more string manipulation, file capabilities, etc. But its continued development has been accompanied by loss of uniformity.

Some systems use "\~" for line "crunching", others use ":". One system will use "?" as a symbol for print, another the "!"'. The conventions for the randomize function tend to be quite different from one system to another. The list of differences continues to grow as each microcomputer manufacturer releases his own version of BASIC. While this competitive development encourages the continued improvement of BASIC, it inhibits the exchange of programs in BASIC. This is a significant problem in education. The Huntington Project simulations, for example, were written in DEC BASIC. But DEC BASIC is not compatible with any of the microcomputer BASIC's. Thus, a very significant potential educational use of microcomputers, SIMULATION, is, at least for the time being, stopped dead in its tracks! It is almost impossible for the end user to make the conversion. The supplier, lacking the resources to translate into a dozen or more versions of BASIC, does nothing!

Please bear in mind that I am not talking about "special features" such as memory allocation, line length, and peek and poke. Such special features can be added to BASIC interpreters while still maintaining downward compatibility. The problem lies in the lack of standardization in basic BASIC. Aside from the fact that various user groups have their own compelling reasons for more standardization, the microcomputer industry itself must recognize that its failure to establish a basic level of standardization will markedly inhibit information exchange which will in turn just as markedly inhibit market expansion. If such a standard can be established, it is a forgone conclusion that users will cooperate. The Huntington Project, for example, will then be able to translate their much desired simulations into a version of BASIC which all users can use.

Product Support

People have expressed the fear that some microcomputer manufacturers will not be around in a few years to support their products. Some will have failed, some will be purchased by larger firms, others will have dropped out of the market in order to concentrate on other products. All of these things will surely happen. Many of today's manufacturers will not be here in five years to back their products. But this really is not much of a problem because these manufacturers are not providing very much support for their products now.

The low cost of microcomputer products is partly due to the

deliberate absence of extensive product support. How can such a state of affairs exist, you may ask. Quite simply, the number of people engaged in digital electronics has now reached the point where there is a sufficient surplus to service the new markets created by micro electronics. Hardware and software services are now being provided by local computer stores. While not as common as radio and T.V. repair shops, they are growing in proportion to the business available. In addition, a large number of engineers and technicians are moonlighting for fun and profit. So.....if anyone advises you to postpone your involvement with micro-computing due to lack of product support, be assured that they are setting up a straw man. There is a great deal of product support, and it is growing in pace with the size of the market. As with many consumer products, the support services are provided by the vendor and small entrepreneur. Small computers are rapidly becoming consumer items and will be serviced accordingly.

What About Obsolescence?

I don't think there is any question that a microcomputer purchased today will be technically obsolete in five years. But that does not mean that you cannot continue to use it for years after. Our PDP 8/E time-share system is now technically obsolete. But it is still less expensive to maintain than to replace it. Even though it has ferrite core memory and lacks the LSI devices that make microcomputers so relatively inexpensive, all of this is transparent to the user. Programming and other computer science concepts have remained essentially the same for years. I would estimate that it will be five or more years before maintenance costs for the 8/E will exceed the replacement cost. And application deficiencies should not be noticeable for an even longer period of time. The same should be true for today's microcomputers. Such long life is not true for many business and industrial applications where other factors such as throughput have extreme importance. The low sophistication requirements of pre-college educational use, however, argues for relative long life.

Teacher Inservice

To date, educational computing has mostly been a time-sharing affair. The central computer was operated by a specialist. The teacher or student did nothing more than turn on the terminal. Microcomputing is somewhat different. There is a computer to operate. How is the BASIC interpreter loaded? What is a bootstrap? How are computer memories organized? How do computers compute? What does the front panel do? Why do we have to learn hexadecimal? These questions must be answered and the skills learned by the teacher so that he or she can be responsible for the operation of the microcomputer.

Because of pre-service experience and because we have had a small time-sharing system for four years, mathematics teachers in San Jose Unified were well acquainted with BASIC and computer programming in general. They also had some expertise with binary and octal arithmetic because these topics were introduced into the mathematics curriculum as part of the "modern math" improvements of the 60's. But operating a computer? Our teachers were completely unprepared! What teacher ever expected to have a computer system for classroom instruction within his or her lifetime?

To rectify this situation we contracted with Don Inman of the People's Computer Company in Menlo Park, California. For nine two-hour sessions every Monday after school last fall, over thirty secondary mathematics and science teachers voluntarily gathered together to learn about microcomputers from Don. Believing hands on learning to be the only way to go, each of our secondary schools (13) purchased a small single board microcomputer of local manufacturer called the Data Handler. Made by Western Data Systems of Santa Clara, California, it cost \$160, used a 6502 microprocessor, had a hex keyboard for input and control and employed LED's to display output and the data and address busses. These microcomputers were purchased as kits, assembled by the teachers and then used to learn about computer architecture and operation, machine level programming, input/output routines, etc. Most of these same teachers are continuing with another series of nine meetings this spring. The Data Handler microcomputers,

by the way, are being used by students when not being used by teachers.

Next year many of our teachers hope to participate in a National Science Foundation funded Microcomputer Teacher In-service Institute to be conducted by the Lawrence Hall of Science (if their institute proposal is approved). This institute, to the best of our knowledge, will be the first ever offered to secondary teachers. Just think, when people ask where it all started, we can say that it started right here in the Bay Area. That's a good feeling and a good thought upon which to end.

THE CONSTRUCTION, OPERATION, AND MAINTENANCE OF A HIGH SCHOOL SYSTEM

Melvin L. Zeddis
1854 Pacific Beach Drive
San Diego CA 92109

ABSTRACT

Developing a computer system from kits was a very stimulating educational experience for high school and junior high school students. There were, however, problem areas: funding, bureaucratic red tape, and a multitude of skeptics. Overcoming these problems lead to the formation of a self developing group of students who were anxious and very able to solve problems in the administration of schools, the instructional programs, career development, while they attempted to create a new computer science instructional program.

Personal computer systems available in kit form provide a foundation upon which to challenge the educational programs in our schools, and human, as well as technological change.

TEXT

The recent development, and marketing of personalized computer system kits is destined to force changes upon educational institutions that have not been remotely anticipated at this time. Students will arrive at the door of the school with an ever increasing store of experience and knowledge of computers. This will create a demand for new courses, updating of existing courses in a radical manner, and will require teachers and administrators to become familiar with the language and thought patterns of the computer age student. We are not only referring to high schools, and college levels, we are including junior high schools, and elementary schools, even the kindergarten child will be coming to school with a computer background.

Those of us who can remember back several years recall the term "generation gap" which was used rather generously to cover many aspects of human behavior. We are now at a point in history where we need to consider a "computer gap." Our generation (those of us over 30) does not have an "easy feeling" about "the computer", the younger generation does. Their entire life span has been coincidental with that of the computer. The younger generation accepts the computer in the same "taken for granted manner" as we did the car and the radio. They see implications and applications that we will not see, they will plunge ahead into areas that we would never even consider venturing into. We do not have a "background" that allows us to include the computer in explorations in a natural and unlaborious manner, as the younger generation does.

What must we do? I would like to describe what we have attempted to do at one high school, in one school district, with one teacher and approximately six students.

In late February 1975 several students and their teacher met to discuss a recent article that had appeared in a "do it yourself" magazine, it was an announcement of the MITS ALTAIR 8800 computer kit. We had all read the article and had agreed to get together to explore the possibilities of obtaining such a kit. It didn't take long for us to agree, it seems we had already made up our minds. All that remained for us to do was to settle upon the characteristics of our system and look for the money to pay for our "dream." A week later we again met and agreed that we needed to make a telephone call or two for some particulars. We made calls, one to MITS and one to SWTPC; both were more informative than we anticipated.

We drew up listings of two systems; the first was the cheapest, and the second was "our

dream system." Our cheap system contained the ALTAIR 8800 with: 8K RAM memory, one RS232 interface, a fan, and 8K BASIC. The I/O device was to be the SWTPC CT-1024 which we would also construct. It was to include an RS 232 serial interface, and a manual cursor control. The actual T.V. for the display was donated by the parents of one of the students.

Our dream system contained: The ALTAIR 8800, 20K RAM, 3 RS232 serial interfaces, fan, vectored intercept, and a cassette tape interface, two CT-1024's, each with a manual cursor control, and a screen read. Also, we hoped to purchase an "old" (reconditioned) typograph, model 3, so that we would have the capacity to produce hard-copy when needed.

We felt either of these two systems would provide a beginning and that from them we could build additional pieces of equipment as both money and items became available. We would also attempt to build a system, whereby we could provide computer services to others on the high school campus; for example the physical education and science departments. Also in our minds was the possibility of freeing ourselves from the timesharing system we were presently using, and paying for at a rate of about \$2200 per year. We realized that we would probably have to do some software development, especially if we wanted to timeshare "our" system. However, we were prepared to launch into that aspect once we had our system in operation.

Our next step was to locate funds and obtain the necessary authorizations to be able to use them for the purchase of our system. No one in the school system seemed to know what we were talking about, at this stage. No one knew who would have to authorize such a purchase because all EDP equipment was purchased by the EDP Department for their use only. Finally, after some time and several discussions we convinced the principal to authorize the purchase with instructional funds, thereby avoiding the EDP Department and producing a new procedure. Now the problem of identifying a particular sum of money for such a purchase was approached.

In California there are two types of categories for the purchase of school items, capital outlay and instructional supplies. Generally speaking capital outlay is used for permanent, non-consumable items, while instructional supply monies are used for the purchase of items that are consumable and/or changed in the instructional program. We convinced the principal that since our system would come in kit form it should be classed as instructional supply, as are other kits, such as radio, calculators, etc., that the industrial arts department purchased of a regular basis. He agreed to the instructional supply argument and we then proceeded to obtain "spending rights" to as much instructional supply funds as possible. We traded with other schools who had over spent in a category where we had underspent, etc. We also traded teacher assistant time for instructional funds, and explored every possible source of funding short of a Federal Funding proposal. The sources of funds we considered included; curriculum consultants, innovation grants, P.T.A., industrial sources, fund raising activities of all types, and trading away some of our next years funding.

To our surprise, we found within a very short time and a few key trading deals, we could purchase our dream system plus a floppy disc. Our troubles were not over though. We had purchased the typograph and it had just been delivered when a directive was issued by the EDP Department stating that all purchases of EDP equipment must be cleared through their office. We took them literally, and since our system was not equipment there was no need to have the cleared.

Our system was ordered in March 1975, and it contained: the ALTAIR 8800, with three RS232 interfaces, 20K RAM, a floppy disc, disc controller, Extended BASIC, a cassette interface, two SWTPC CT-1024's each with screen read and manual cursor control.

The first kits to arrive were those for the

SWTPC CT-1024's. These kits went together without difficulty and worked the first time they were fired up. We now began to use these (CT-1024) for I/Os on the timesharing system, and allowed the students to check them out over night and also over the weekends. The only complaint came from parents who were concerned because the student who had checked the terminal out would arrive home with a group of students who did not leave until the early morning hours. They would keep the phone line in use from the time they arrived until the parents forced a system shutdown, sometimes as late as 1:00-2:00 A.M. Our SWTPC terminals, in spite of being carried around and banged about, continued to function quite well. After two years of operation, we have only had to replace a few of the keyboard switches, everything else remains in operation without modification of any kind.

Several months later our ALTAIR 8800 arrived, and with great fanfare and the taking of pictures, it was unpacked and looked at with great anticipation. We even put the front panel together and propped it in the case so we could see what "it" really looked like. Many students stopped by to see what we were doing and to look at our front panel.

The construction of the ALTAIR went along quite smoothly. The only problems encountered during the construction involved trying to locate the latest field changes and incorporate them into our system. Then there were several transistors soldered in wrong, a few ICs were upside down, and a few misread capacitors. Problems were also caused by solder bridges and flakes of solder, but these were found by using a strong light and small wire brush. Finally, after only three weeks, it was time to apply power to our system. IT didn't quite work they way it was supposed to.

The most difficult task was getting all the bugs out. It took us about a month of work, checking and rechecking everything in each part of the system to get to function properly. This troubleshooting period pointed up the necessity of keeping an operation log, detailed notes on troubles, a skepticism in believing schematics, and double checking everything, taking nothing and no one for granted. Some of the problems we encountered were: several parts were missing, the schematics were wrong for the 4K memory boards, the memory ICs arrived three weeks after the boards and did not work at the proper speeds, the factory field change were not sent to us, and finally, the suggested (by the factory) change in the deposit circuitry didn't work.

The task of getting our system to function properly presented a problem that required the development of a systematic approach, and emphasized the need to be aware of one's assumptions. One of the most common assumptions we made which caused us difficulty was that there was only one deficient part causing the problem. One never knows if this is true until the system is again working well, and the possibility of multiple malfunctions cannot be overlooked.

Troubleshooting also demands the use of test equipment. Our system also included a digital multimeter, an IC tester, and oscilloscope, as well as the needed handtools. The use of these instruments and the interpretation of their information again demanded some knowledge of the theory of the computer system (note also the assumption that they are functioning properly). The students at this point began to obtain a thorough understanding of TTL and how the computer was supposed to function.

When our system finally worked as we thought it should it was a great feeling for all of us. We had accomplished a truly challenging task; had done what many thought we could not do. Those people who had supported us, skeptically, were very impressed. We were all filled with a real sense of accomplishment and pride in our achievement.

As soon as our system was in operation the EDP Division issued another directive, they would not authorize any parts of our computer system to be placed upon the normal district maintenance contract. This was great, because we had the knowledge, experience, and equipment to perform our

own maintenance. The only problem we could envision was developing an ongoing supply of technician/students to insure the well being of the system. This problem was solved by developing a small group of students into certificated maintenance/operators. The training cycle was to include several students from each grade, with the sophomores working with experienced juniors and seniors. These students upon completion of a specific training program were to be licensed as system maintenance/operators. Only licensed personnel were allowed to perform diagnostic work on the system, pull boards, replace components, and checkout the system.

The original authorization for this course was given by the principal of the school, however, the central office personnel were reluctant to authorize it because it did not fit into mathematics, science, industrial arts, and a computer science area (and still does not exist at this time). We even tried to have our course considered as a career education experiment, but that was also turned down. We were finally considered to be interdisciplinary in nature and therefore, the junior administrator was designated to oversee the course.

The official course title, "Computer Technology 1-2", was approved finally. The course included:

1. The development of computers in the historical sense, including the impact upon humankind, with conjectures for the future.
2. The foundations of computer mathematics including basic Boolean Algebra and Logic, stressing TTL.
3. The electrical circuits that are used in hardware including, AND, OR, NAND, NOR, FF, also ICs with LSI, and MSI.
4. Components of a computer system at the block level including basic components such as CPU, memory, etc, also the disc, cassette interface, and other peripherals.
5. The component level of computer systems that includes individual functioning of components in all sections of the computer system. This level is one below the block level of each board or assembly.
6. The use of test equipment includes the VOM, DVM, Oscilloscope, IC tester and handtools. The interpretation of data is also a portion of this section.
7. Troubleshooting the system includes the block level, component level, component replacement, and system checkout.
8. Comprehensive examinations to cover: Theory of computers, computer operation and checkout, and troubleshooting.

Those students satisfactorily completing this course are to be granted a license to operate and maintain the computer system at Morse High School.

The materials used in the preceding course include: Kemeny, J. Man and the Computer, Streeter, Integrated Circuit Logic Elements, and the system manuals for MITS ALTAIR 8800.

At the present time the computer has been almost totally neglected as a useable tool in the instruction program of most schools, only isolated examples exist of the use of computers in the instruction program. The advent of personalized computer systems will force the schools to include it into all areas of the curriculum. Our system provided us with some very important insight into the use of computer systems, these insights are:

1. Young children can understand the internal workings of computer systems.
2. Students at all levels can develop very sophisticated software in a relatively short period of time.
3. Students are very creative in the use of the computer.
4. Students can be very helpful in solving some of the problems at a particular school, especially when using a computer.
5. Adults(teachers and EDP personnel) can become very threatened when students start using EDP equipment in creative ways.

6. The computer may solve some problems, but its use generates many other problems that need to be resolved.
7. Any teacher, administrator, or EDP person that can be replaced by a computer deserves such a fate.
8. Students gain insights and understandings in areas by becoming involved in computer applications in those areas.
9. What happens (almost always good) many times when students and computers interact can not be anticipated.

These points of awareness lead us to develop the following courses of study to be used with our computer system. However, at this time because some teachers and administrators can not understand the need nor the content of the proposed courses, an official authorization has not been given for their implementation.

The proposed courses are:

COMPUTER APPLICATIONS IN NATURAL SCIENCE. This course would provide the student with the opportunity of exploring computer applications in the areas of: biology, physics, chemistry, and any combination of these areas.

COMPUTER APPLICATIONS IN MATHEMATICS. This course would provide the student with the opportunity of exploring computer applications in the area of mathematics, including: numerical analysis, number theory, ordinary and partial differential equations, modern algebra, matrices, geometry, etc.

COMPUTER APPLICATIONS IN THE BEHAVIORAL SCIENCES.

This course would provide the student with the opportunity of exploring computer applications in the areas of psychology, political science, history, economics, sociology, anthropology, as well as combinations of these fields.

COMPUTER APPLICATIONS IN BUSINESS. This course would provide the student with the opportunity of exploring computer applications in the areas of: marketing, finance, management, advertising, or any combination of these areas.

HEURISTIC APPLICATIONS OF COMPUTER SCIENCE. This course would allow the student to explore applications in such areas as: music, art, dance, literature, counseling, etc.

COMPUTER TECHNOLOGY. This sequence of four semester courses would provide the student with the opportunity of learning the theory, operation, and repair of personal computer systems. Also provided is the opportunity to design a small computer system, experimenting with circuits common to the industry. Finally, the opportunity of constructing one's own system is also provided.

INDEPENDENT STUDY IN COMPUTER LANGUAGES. This course would allow the student who desires to study a computer language other than BASIC, FORTRAN, or COBOL, the opportunity of developing a fluency in a language such as, ALGOL, PL/1, APL, RPG, etc.

INDEPENDENT EXPLORATIONS IN COMPUTER SCIENCE.

This course would allow the student who desires to acquire a deep understanding of computers and computer technology, to work at the compiler/translator/interpreter level of programming.

These courses were to be in addition to the regular offerings: Introduction to Computer Science, a course in BASIC programming; FORTRAN IV a course in programming in FORTRAN. These two courses were the only ones available to students within the current curriculum.

The preceding courses were to be phased into existence over a period of several years, and were to be offered during one time block with different students studying different courses within the same class. Credit would be granted upon completion of their course work and demonstrated knowledge of the contents of the course. This demonstrated knowledge would usually take the form of a demonstration for the instructor on the equipment or the presentation of appropriate software items.

Where can your system be used in the school program to enhance the learning of students? We have found there are a variety of applications, many of which demand some form of mass storage and a large core. (Our system had 20K of core.) Our thinking was along the following lines:

1. The physical education department liked to have their statistics kept and we would do it for them.
2. Some diagnostic information relating to the behavior of our teams, and the opposing teams could be analyzed by our system. Our students had developed software to be used in basketball, and baseball which had proven to be helpful.
3. Simulations for the natural and social science classes could be developed and used with our system.
4. A timesharing service could be offered

to the science department. This required some software development however.

5. Development of heuristic applications in all areas.
6. Develop a counseling service that would handle the mundane record keeping for the counselors.
7. EDP attendance, dumping information onto cassettes for easy storage. Maintaining an entire year of attendance records, which would be of immense benefit to the administration of the school.
8. Programming students into classes, also stored upon cassettes for further reference.

No doubt more applications exist and have already been explored, but the fact that students were using their energy, and talents in a positive way to solve some of the school's many problems was very important.

SUMMARY

The schools are in for a tremendous amount of change, forced upon them by the availability of cheap computer systems. The students who attend schools now and in the future will have an ever increasing familiarity with computer systems. At this time it is possible to develop a functional computer system from kits with high school and junior school students constructing, operating, and maintaining such a system.

The use of personal computer systems in the curriculum of the schools will require some additional courses being made available to students. Teachers will need to realize they may never be able to understand the products of many students in the area of computer science.

Personalized computer systems are here, NOW! The battle between the big vs the little system is over. Guess who had won?

BACKGROUND/BIOGRAPHY

Melvin L. Zeddis was first initiated into the computer fraternity in 1957 through the courtesy of the U.S. Army and Fort Bliss, Texas. He has continually "played" with computers since that time, constructing a "crude" computer with junior high school students in 1966. In addition to teaching mathematics in the San Diego schools, he has been involved in many NSF Institutes in mathematics and computer science as a faculty member. Currently on leave from the San Diego schools, he is serving as professor of mathematics and computer science at United States International University, where he has also served as Assistant Dean. He is currently developing the area of personalized computer applications at U.S.I.U. He is married, and he and his wife, Marcia, have two children of the computer era, Marlene (9), and Mark (7). He has an A.B. from San Diego State College, M.A. and Ph.D. from United States International University, also a post doctoral year of study at U.S.I.U.'s Institute for Educational Management. He is presently interested in humanistic mathematics, reading instruction for small boys using computers, and technological extensions of humankind.

BIBLIOGRAPHY

Kemeny, John. Man and the Computer. New York: Charles Scribner's Sons, 1972.

Streater, J. W. Integrated Circuit Logic Elements. New York: Howard W. Sams, Inc., 1974.

Zeddis, M. L., et. al. Individualized Instruction for Gifted Students using Computer Time-Share Systems. San Diego: San Diego City Schools, 1974.

EDUCATING PEOPLE ABOUT PERSONAL COMPUTING: A MAJOR PROGRAM AT THE LAWRENCE HALL OF SCIENCE

Bob Kahn & Lee Berman
Lawrence Hall of Science
University of California
Berkeley CA 94720

*A greatly expanded and more general version of this paper entitled: *Public Access to Personal Computing: A New Role for Science Museums*, is published in COMPUTER Magazine of the Institute for Electrical and Electronic Engineers Computer Society, April, 1977.

Traditionally, one thinks of the museum as a repository for stuffed animals, art, and artifacts from the present and past. The science museum is often a maze of exhibits (frequently automated by push buttons and floor mat switches) illuminating various aspects of science, history, or technology. Likewise, a popular mass media image of the computer depicts a row of magnetic tape drives lining the walls of an air-conditioned, false-floored, glassed-in room where specially trained computer operators and programmers huddle around a central console of blinking lights.

From an educational standpoint, both the museum and early generations of computers, whose image is depicted above, have shared a common characteristic: they have tended to be static and passive. They have offered the public little opportunity for participation or access to technology. A museum visit often involves a lot of walking, looking, and listening, but little touching or participating. And, for those who do not already use computers in business, school, or the university, public access has generally been vicarious, or limited to computer output only.

However, computer technology has advanced considerably beyond what the public has experienced, and some science-technology centers are re-defining their traditional role as a community showcase and repository. This paper discusses ways in which the Lawrence Hall of Science is making modern computers accessible and understandable to the general public.

Slowly, people are becoming aware of the potential power computers have in areas beyond business and data analysis. With good software, the computer can become a personalized tool for extending the mind: a personal laboratory for simulated experimentation, an artistic medium, an information retrieval system, a gaming opponent and personal entertainment center, a teacher, a sophisticated editing facility, and of course, a programmable calculator--all in the same box. This is the essence and the power of personal computing. When people (particularly children) have been exposed to personal computing, they search out and demand access to it.

This demand is evident in the proliferation of homebrew computer clubs, microcomputer kits, hobby shops, and computer hobbyist newsletters. But awareness of personal computing power must spread far beyond the hobbyist movement.

While one no longer needs a crystal ball to foresee a computer in every home, it will still be a number of years before everyone's television becomes a terminal. In those intervening years, it is imperative that adults and children, not just professionals and hobbyists, achieve a reasonable level of computer awareness or computer literacy. Computer literacy has been well defined in a recent report by Roy Amara et al. for the Institute for the Future as *the understanding of basic computer functions in terms of what computers can and cannot do, with particular attention to their potential, as well as their limits, in meeting human needs*. While many recent books and films have attempted to speak to this need, computer literacy can be more quickly and effectively achieved by giving the general public direct, hands-on access to computers in a non-threatening, relaxed learning environment.

One type of institution that is ideally suited to provide public access to computing is the science-technology center--not the traditional science museum described earlier, but an emerging new type of activity-oriented community learning resource devoted to furthering the public's appreciation for the methods, tools, and principles of science and scientific research. Unlike the traditional museum, the hallmark of science-technology centers is *discovery learning or learning science by doing science*.

Instead of rows of glass cases protected by signs and security guards, modern science centers encourage visitors to touch objects, manipulate controls, participate in workshops and classes in an informal, non-school museum

environment. Clearly, interactive computers and computer terminals would be right at home in such places. Indeed, most science-technology centers recently built or still in the planning stages have made provisions for computer-related exhibits and activities.

Of the 167,000 children and adults who visited the Lawrence Hall of Science (LHS) in 1975-76, more than 27,000 (or 16%) of them came specifically for the purpose of gaining access to computers through various workshops, classes, visiting school programs and regularly scheduled public usage hours. In addition, LHS is engaged in providing an educational time-sharing service to local schools. The LHS (NOVA 800-based) time-sharing computer system presently reaches more than 40 schools and educational institutions in Northern California including a Montessori school, elementary, junior and senior high schools (both public and private), the California School for the Deaf, junior colleges, four-year colleges, and some departments of U.C. Berkeley and U.C.L.A.

The Lawrence Hall has put considerable thought and effort into building a computer education facility that would simultaneously meet the needs of public visitors and visiting school groups and extend computing out to local schools and the San Francisco Bay Area community, including a new program providing home access to computing at low cost. On a smaller scale, LHS is following in the footsteps of Dartmouth College, adopting the philosophy of Dartmouth's president, John Kemeny in his book, *Man and the Computer*. Kemeny describes Dartmouth's computing facility as an intellectual resource for the community, like a library, where computing power is freely accessible to the community.

On such a philosophical basis, over the past six years, LHS has defined three major objectives for its computer education program:

- 1) To educate children and adults about the milieu of computers in an enjoyable, intriguing, non-threatening learning environment;
- 2) To offer the public hands-on computing at low cost;
- 3) To develop an exportable educational program of computer activities that will serve as a resource for schools and other learning centers and public educational institutions.

Lawrence Hall has attempted to meet these objectives through a variety of activities and programs. Several terminals are openly accessible to visitors in Lawrence Hall's exhibit areas. These terminals offer a limited selection of games and simulations. Two of these terminals form an integral part of a new exhibit on energy uses and alternative sources.

LHS has an extensive program for visiting school children, and the Hall's two computer terminal rooms (each containing ten teletypes) assure that each individual student receives a maximum amount of hands-on experience during his/her hour in the computer laboratory. The two computer terminal rooms are also used for after-school and in-service classes and for public computer access. LHS offers a complete series of computer classes (to more than 1000 children and adults) throughout the year. The courses are designed to meet the needs and abilities of students at various ages from eight years to eighty. They cover topics from computer literacy for children and adults, to programming and theoretical and applied topics in computer science.

However, exhibits, courses, and workshops provide limited or structured access. LHS also offers unstructured access to computers. People who simply wish to sit down at a terminal and execute any program of their own choosing from the program library or even write their own programs (often exercising skills previously learned in workshops and classes) may do so on the weekends, Thursday evenings, and Friday afternoons. At these times, Lawrence Hall opens one or both of its computer-terminal rooms to the public at a cost of \$1.50 per hour. A person simply buys a ticket which reserves him/her a terminal for an hour (half-hour tickets are available for 75¢). Furthermore, LHS has allocated some ports on its time-sharing system for home use by its members at a monthly rate comparable to its educational rate; a much lower rate is available to members wishing home access restricted to evening and late night hours.

The Lawrence Hall of Science represents an active campaign by a science center to educate the members of its community about the power of personal computing. It provides for increased public awareness of computers in an informal

learning environment; it also offers options for more in-depth, formal learning through classes. Most important, it offers an opportunity for personal exploration of interactive computing through freely accessible, but structured exhibits and through direct, unstructured, self-motivated hands-on access, including access to individuals in the privacy of their own homes. For the mid-1970's, this represents a large step toward making computer power available to the community.

Robert A. Kahn has been the director of the Computer Education Project at UC Berkeley's Lawrence Hall of Science since June, 1972. During this time he has also been studying for his Ph.D. in education. As part of a Master's project, Kahn developed an introductory computer course for 8-10 year old children. He is currently interested in facilitating informal learning in the museum environment.

Lee Berman is coordinator of computer activities at the Lawrence Hall of Science. He is responsible for Lawrence Hall's computer outreach program, and in this capacity, he maintains constant communication with and provides support for more than 40 schools and educational users on the Lawrence Hall time-sharing system.

Being a candidate for a Ph.D. in romantic poetry, Berman has naturally gravitated toward computers. His responsibilities also include editing the LHS computer newsletter, *Educational Computing*, and maintaining the LHS library of computer programs.

CAI ANSWER PROCESSING IN BASIC

Franz J. Frederick
Education Building, Room 112
West Lafayette IN 47907

ABSTRACT

The presentation deals with the implementations of algorithms for keyword answer processing and phonetic answer processing. The algorithms are implemented as subroutines in extended BASIC. Completely commented listings are included.

There are over 30 computer-assisted instruction languages in existence. Of these only one (PLANIT) is "machine independent" to major extent. None are available currently (publicly at least) for use on micro-computers.

The major design intent of these languages was centered around the notion of providing certain basic computational capability for use by teachers. It was assumed that these teachers should be able to use the language without being experienced programmers. It was also assumed that the primary use would be tutorial based upon sets of questions and answers.

The first efforts at answer processing involved exact answer match. After initial experimentation, teachers began to discover that exact answer matching simply was not effective for many types of tutorial lessons. For example, the student's answer to a question might have the same words in a different order and therefore be counted wrong. The exact answer match implicitly required exact order as well.

This dilemma lead to the development of keyword answer processors. In this case, the author specified a keyword and if that word occurred anywhere in the student's response, the response was considered correct. Embellishments quickly became necessary and included such things as (1) multiple keywords and (2) allowing the keyword to be embedded within a larger word. This latter feature allowed the student to respond with the plural form or past tense of the authors' answer and still be considered correct.

After some further experimentation teachers began to find that exact order of keywords was indeed important and desirable for some lessons. Consequently, the next major embellishment of keyword processors allowed stipulation of exact order or no order. These developments in keyword processors were not only very useful in the standard automated lesson but opened the way for experimentation with simulated conversational interaction. The counselor-patient types of simulations became possible providing the teacher could specify reasonable anticipated questions or question sequence which could reasonably occur in real life situations.

The developments were rather useful in computer-assisted instruction applications but basically were available only in CAI languages. The actual implementation of answer processors usually treats the processors as language functions with parameters.

With the advent of low cost micro-processors and in particular the advent of multi-user micro-processor systems, the ideal of low cost CAI in the classroom seems possible. The only generally available higher level language currently on micro systems is of course BASIC in more or less extended versions. BASIC does not have, however, embedded answer processing functions except at the exact answer level. This means that the CAI author must create his own code to do these functions and repeat it extensively through a lesson or create generalized subroutines for these functions.

Recognizing that it may be some time before a micro CAI language is generally available and that extended BASIC with string functions is likely to be a defacto standard, the author designed a generalized keyword subroutine in BASIC. This subroutine was designed to allow specification by the author of up to five keywords and to allow the author to specify how many must be matched in order to be considered correct.

The keyword algorithm is as follows:

KEYWORD ALGORITHM

1. Compare each author keyword with student's response.
2. If a keyword matches, increment the match counter.
3. Compare the match counter with the authors' specified number of matches. If equal, print a correct response message, record indication of correct response and record the student's actual

response. Blank the answer variable. Return to calling routine.

4. If no match in step 3, check to see if all keywords have been compared without success. If so, print an incorrect response message, record indication of incorrect response and record the student's actual response. Blank the answer variable. Return to the calling routine.

CALLING PROGRAM DESIGN

The design of an answer processing routine is dependent to some extent upon the design of the calling program.

Most CAI languages treat lessons as blocks of information to be presented to the student. The block may be composed of (1) information to be displayed, (2) question, (3) specified answers to be compared to a student's response and (4) actions to be performed dependent upon the quality of the students' response.

A suggested block in BASIC would be as follows:

EXAMPLE OF "BLOCK"

```
500 PRINT "NAME A PROPERTY OF COLOR"
505 REM---STUDENT RESPONSE VARIABLE IS A$ AND IS THE SAME
506 REM---IN ALL BLOCKS
510 INPUT A$
520 REM---R IS NUMBER OF BLOCK IN THE PROGRAM; K IS THE
521 REM---NUMBER OF KEYWORDS TO MATCH
530 R=9 : K=1
540 REM---AUTHOR SPECIFIED KEYWORD
550 A$(1)="HUE"
560 A$(2)="VALUE"
570 A$(3)="INTENSITY"
580 REM---CALL KEYWORD SUBROUTINE
590 GOSUB 7000
```

DESIGN OF KEYWORD SUBROUTINE

The keyword subroutine presented in this paper requires the following extended BASIC features.

1. STRINGS (length at least equal to 70 characters)
2. MID\$(A\$, I, J)
3. LEN(A\$)

The keyword subroutine requires the following unique variables.

1. K2 (number of matches)
2. K (number of keywords to match-see BLOCK design)
3. K1 (actual number of author keywords)
4. A\$ (string variable which holds student answer)
5. R\$(I,J) (string matrix used to record student responses and performance record)
6. A(5) (numeric matrix used to hold length of keyword strings)

The keyword routine uses only two temporary variables (I and J). They are used as loop counter variables.

The keyword subroutine allows the following keyword checks:

1. Specification of single keyword match from field of 1 to 5 author specified keywords.
2. The key word is treated as a root keyword consequently it will allow the occurrence of the keyword with prefix and/or suffix in the student response.

KEYWORD PROGRAM LISTING

```
1 REM --- CLEAR VARIABLE SPACE AND ASSIGN 1000 BYTES TO STRINGS
2 CLEAR 1000
9 REM --- DECLARE NUMERIC AND STRING MATRICES
10 DIM A(5),A$(5),R$(10,3)
95 REM --- FIRST "BLOCK" OF INSTRUCTIONAL MATERIAL
96 REM     DISPLAY QUESTION AND REQUEST RESPONSE
97 REM     A$ IS USED AS A GENERAL ANSWER VARIABLE FOR ALL
98 REM     "BLOCKS"
100 PRINT:PRINT "WHAT IS ONE PROPERTY OF COLOR":INPUT A$
105 REM --- THE VARIABLE "R" IS USED TO NUMBER THE BLOCKS AND
106 REM     IS SUBSEQUENTLY USED AS AN INDEX TO THE STUDENT
107 REM     RECORD MATRIX "R$(R,C)"
108 REM     THE VARIABLE "K" IS USED BY THE TEACHER TO SPECIFY
109 REM     THE NUMBER OF KEYWORDS REQUIRED FOR A CORRECT RESPONSE
```

```

110 R=1:K=1
115 REM --- THE SUBSCRIPTED STRING VARIABLE A$(N) IS USED BY
116 REM THE TEACHER TO HOLD THE KEYWORDS CONSIDERED TO BE
117 REM CORRECT ANSWERS
120 A$(1)="VALUE":A$(2)="HUE":A$(3)="INTENSITY"
125 REM --- CALL KEYWORD PROCESSOR SUBROUTINE (THE SUBROUTINE
126 REM JUDGES CORRECTNESS OF RESPONSE AND AUTOMATICALLY
127 REM RECORDS THE INDICATION OF CORRECTNESS IN R$(R,C)
130 GOSUB 7000
195 REM --- ANOTHER INSTRUCTIONAL "BLOCK" BEGINS HERE. THE
196 REM STRUCTURE IS THE SAME AS FOR BLOCK 1 (R=1).
200 PRINT:PRINT "NAME TWO PROPERTIES OF COLOR"
210 INPUT A$
220 R=2:K=2
230 A$(1)="INTENSITY":A$(2)="VALUE":A$(3)="HUE"
240 GOSUB 7000
295 REM --- "BLOCK 3" BEGINS HERE AND HAS THE SAME STRUCTURE
296 REM AS THE PRECEDING BLOCKS.
300 PRINT:PRINT "NAME ALL THREE PROPERTIES OF COLOR"
310 INPUT A$
320 R=3:K=3
330 A$(1)="HUE":A$(2)="INTENSITY":A$(3)="VALUE"
340 GOSUB 7000
495 REM --- THE NEXT STATEMENT CONCLUDES THE BLOCKS SUPPLIED
496 REM TO TEST THE KEYWORD PROCESSOR SUBROUTINE.
500 GOTO 9999
6994 REM
6995 REM *** THE KEYWORD PROCESSOR SUBROUTINE *****
7000 K2=0
7005 REM --- DETERMINE THE LENGTHS OF THE AUTHORS KEYWORDS
7010 A(1)=LEN(A$(1))
7020 A(2)=LEN(A$(2))
7030 A(3)=LEN(A$(3))
7040 A(4)=LEN(A$(4))
7050 A(5)=LEN(A$(5))
7055 REM --- DETERMINE HOW MANY KEYWORDS WERE SUPPLIED FOR
7056 REM TEST BY THE AUTHOR.
7060 FOR I=1 TO 5
7070 IF A(I)=0 THEN 7090
7080 NEXT I
7085 REM --- THE VARIABLE K1 CONTAINS THE ACTUAL NUMBER OF
7086 REM KEYWORDS IN THE AUTHORS LIST
7090 K1=I
7095 REM --- COMPARE THE NUMBER OF KEYWORDS SPECIFIED FOR MATCH
7096 REM WITH THE NUMBER OF KEYWORDS IN THE LIST. IF
7097 REM K IS GREATER THAN K1 THEN ASSUME ALL IN THE
7098 REM LIST MUST MATCH.
7100 IF K>K1 THEN K=K1
7110 I=1
7115 REM --- CHECK TO SEE IF CURRENT KEYWORD BEING TESTED
7116 REM IS EMPTY (IE, HAS NO CHARACTERS). IF SO TERMINATE
7117 REM THE TEST SEQUENCE.
7120 IF A(I)=0 THEN 7200
7125 REM --- COMPARE THE CURRENT KEYWORD WITH ALL SEQUENCES OF
7126 REM CHARACTERS IN THE STUDENTS' ANSWER.
7130 FOR J=1 TO LEN (A$)
7140 IF MID$(A$,J,A(I))<>A$(I) THEN 7150
7141 REM --- IF MATCH, INCREMENT MATCH COUNTER
7142 K2=K2+1
7144 GOTO 7160
7150 NEXT J
7155 REM --- COMPARE MATCH COUNTER WITH THE NUMBER OF KEYWORDS
7156 REM SPECIFIED AS NECESSARY FOR CORRECT RESPONSE. IF
7157 REM EQUAL, PRINT POSITIVE MESSAGE FOR STUDENT. IF NOT,
7158 REM CHECK TO SEE IF ALL KEYWORDS IN LIST HAVE BEEN
7159 REM HAVE BEEN TESTED.
7160 IF K2=K THEN PRINT "RIGHT":GOTO 7230
7165 REM --- IF ALL KEYWORDS IN THE LIST HAVE BEEN TESTED, PRINT
7166 REM NEGATIVE MESSAGE TO THE STUDENT.
7170 IF I=5 THEN 7200
7180 I=I+1
7190 GOTO 7120
7200 PRINT "WRONG!"
7205 REM --- STORE INDICATION OF WRONG RESPONSE IN THE STUDENT RECORD
7206 REM MATRIX.
7210 R$(R,3)=""
7220 GOTO 7240
7225 REM --- STORE INDICATION OF CORRECT RESPONSE IN THE STUDENT RECORD
7226 REM MATRIX.
7230 R$(R,3)=""
7235 REM --- STORE ACTUAL STUDENT RESPONSE IN THE STUDENT RECORD
7236 REM MATRIX
7240 R$(R,2)=A$
7245 REM --- BLANK RESPONSE VARIABLE.
7250 A$=" "
7255 REM --- RETURN TO THE CALLING "BLOCK".
7260 RETURN
9999 END

```

Answer processing using keywords represents a very powerful tool in the development of conversational simulations. Programs can be devised which can carry on a useful conversation in a limited context.

While keyword answer processing opened new horizons in the development of tutorial lessons, teachers began to encounter problems with students who could not spell or who could not type accurately. These problems lead to a concern for some sort of answer processor which could phonetically encode the responses and thus avoid some

basic problems in typographic errors and spelling errors.

A very basic but interesting phonetic answer processor algorithm was developed through a project funded by the National Science Foundation. The project resulted in the development of a CAI language called PLANIT. It was to be "machine-independent". It successfully met that goal with the only restrictions being a 24 bit word size (minimum) and access to a FORTRAN IV compiler. The phonetic algorithm used in PLANIT is quoted verbatim in this paper.

The general algorithm for a phonetic answer processor in BASIC is as follows:

BASIC PHONETIC PROCESSOR PROCEDURE

1. Disassemble the answer or response string into single character strings.
2. Sequentially convert each single character string into its' phonetic equivalent using the PLANIT PHONETIC ALGORITHM.
3. Reassemble phonetic characters equivalents into a single string.
4. Return to calling program for comparison of phonetic equivalent response to a phonetic equivalent answer specified by the lesson author.

DESIGN OF PHONETIC SUBROUTINE

The phonetic answer processor subroutine requires the following extended BASIC features:

1. STRINGS (length at least equal to 70 characters)
2. MID\$(A\$,J,1)
3. LEN(A\$)

PHONETIC ENCODING AND FORMULAS PROCESSING*

PHONETIC ENCODING

The phonetic encoding process is accomplished in four steps:

Step 1 - Letter Equivalent:

All letters are transformed into their letter equivalents. Any remaining characters including blanks are unchanged. The letter in Row 1 is transformed into the letter immediately below in Row 2. PLANIT ignores all other characters.

Row 1 ABCDEFGHIJKLMNOPQRSTUVWXYZ (original letter)
Row 2 ABCDABCHACCLMMABCRCDCABHAC (letter equivalent)

Step 2 - The H Replacement:

Each H in a word is transformed to the preceding letter provided the character is a letter. If not a letter (e.g., a blank), H is unchanged.

Step 3 - Elimination of Successive Identical Consonants:

All but the first element of an uninterrupted sequence of a single consonant is eliminated, (e.g., CC=C, TT=T).

Step 4 - Elimination of As:

All vowels, transformed into A's, are eliminated except if A is the first character of the word to be encoded. The final word contains only consonants and a leading A if there is one.

Examples:

Original Word	Steps			
	1	2	3	4
PHONETIC	BHAMADAC	BBAMADAC	BAMADAC	EMDC
HAZARD	HACARD	HACARD	HACARD	HCRD
ON-LINE	AM-LAMA	AM-LAMA	AM-LAMA	AM-LM
AWHILE	AHHALA	AAAALA	AAAALA	AL

The phonetic subroutine requires the following unique variables:

1. B\$ (phonetic encoded answer)
2. A\$ (string variable to hold student response)
3. R\$(I,J) (string matrix to hold student performance records)

*Bennick, F. D. and C. H. Frye "PLANIT LANGUAGE REFERENCE MANUAL", System Dev. Corporation TM-(L)-4422/002/01, Oct. 1970. (APPENDIX E)

4. B\$(73) (string matrix used to hold individual characters from student response or author answer)

The phonetic subroutine uses only two temporary variables (J and K). They are used as loop counter variables and index pointers for matrices.

The phonetic answer processor subroutine is designed to be called to process both the authors' answer and the students' response (on separate GOSUB calls). The phonetic response processor can handle character strings up to 70 characters length with embedded blanks and punctuation. The only restriction is that the use of a comma in the students' answer may produce a syntax error depending on which BASIC interpreter you use.

The listing of a sample mini-lesson (one BLOCK) and the phonetic subroutine follow.

PHONETIC RESPONSE PROCESSOR LISTING

```

1 REM --- CLEAR VARIABLE SPACE AND ASSIGN 500 BYTES TO STRINGS
2 CLEAR 500
9 REM --- DECLARE STRING MATRICES FOR PHONETIC SUBROUTINE
10 DIM B$(73)
15 REM --- DECLARE STRING MATRIX FOR STUDENT PERFORMANCE RECORDS
16 REM     THE R DIMENSION REFERS TO THE NUMBER OF "BLOCKS"
17 REM     ENCOUNTERED BY THE STUDENT. THE C DIMENSION REFERS
18 REM     TO (1) BLOCK NUMBER, (2) ACTUAL RESPONSE, AND (3)
19 REM     THE RESULT OF COMPARING RESPONSE WITH ANSWER
20 DIM RS(30,3)
70 B$(0)=" "
80 I=0
95 REM --- FIRST "BLOCK" OF INSTRUCTIONAL MATERIAL.
96 REM     PRESENT QUESTION - -
100 PRINT "WHO WAS THE FIRST PRESIDENT OF THE U.S."
105 REM --- THE VARIABLE R REFERS TO THE BLOCK NUMBER.
106 REM     THE VARIABLE I IS USED TO INDICATE THE NUMBER OF
107 REM     ATTEMPTS IN THE LESSON. THE MATRIX RS(R,C)
108 REM     IS USED TO HOLD A RECORD OF THE STUDENTS'
109 REM     PERFORMANCE BLOCK BY BLOCK.
110 R=1:I=1:RS(I,1)=STR$(R)
115 REM --- A$ IS USED TO HOLD THE AUTHOR SPECIFIED ANSWER
120 A$="WASHINGTON"
125 REM --- GOTO PHONETIC SUBROUTINE AND CONVERT ANSWER TO
126 REM     PHONETIC REPRESENTATION. RETURN IT IN B$.
130 GOSUB 8000
135 REM --- MOVE PHONETIC REPRESENTATION FROM TEMPORARY VARIABLE
136 REM     B$ TO P$.
140 P$=B$
145 REM --- REQUEST STUDENT RESPONSE TO THE QUESTION.
146 REM     A$ IS AGAIN USED TO HOLD DATA TO BE PASSED TO
147 REM     PHONETIC SUBROUTINE.
150 INPUT A$
155 REM --- STORE STUDENTS' ACTUAL RESPONSE IN RECORD MATRIX
160 RS(I,2)=A$
165 REM --- GOTO PHONETIC SUBROUTINE AND CONVERT RESPONSE TO
166 REM     REPRESENTATION. RETURN IT IN B$.
170 GOSUB 8000
175 REM --- COMPARE B$ WITH P$.
180 IF P$<>B$ THEN 210
185 REM --- RECORD CORRECT RESPONSE INDICATOR
190 RS(I,3)="1"
195 REM --- PRINT FEEDBACK MESSAGE TO STUDENT
200 PRINT "RIGHT!":GOTO 230
205 REM --- RECORD INCORRECT RESPONSE INDICATOR
210 RS(I,3)="0"
215 REM --- PRINT FEEDBACK MESSAGE TO STUDENT
220 PRINT "WRONG!"
225 REM --- PROCEED TO NEXT FRAME
229 REM *** END OF TEST OF PHONETIC SUBROUTINE *****
230 GOTO 9999
7996 REM ***** PHONETIC SUBROUTINE *****
7998 REM --- SET UP VARIABLES FOR COUNTERS
8000 K=1:J=1
8002 B$=" "
8005 REM --- PLACE EACH LETTER/CHAR IN ANSWER INTO SEPARATE CELL
8010 IF J>LEN(A$) THEN 8200
8020 B$(K)=MID$(A$,J,1)
8022 REM --- TRANSFORM ANSWER LETTERS INTO LETTER EQUIVALENTS. BLANKS
8024 REM     REMAIN UNCHANGED. LETTERS TRANSFORMED AS FOLLOWS:
8026 REM
8028 REM     ORIG. A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
8029 REM     EQUIV. A B C D A B C H A C C L M M A B C R C D A B H C A C
8030 IF B$(K)="E" OR B$(K)="I" OR B$(K)="O" THEN B$(K)="A"
8040 IF B$(K)="U" OR B$(K)="Y" THEN B$(K)="A"
8050 IF B$(K)="F" OR B$(K)="P" OR B$(K)="V" THEN B$(K)="B"
8060 IF B$(K)="G" OR B$(K)="J" OR B$(K)="K" THEN B$(K)="C"
8070 IF B$(K)="Q" OR B$(K)="S" THEN B$(K)="C"
8080 IF B$(K)="X" OR B$(K)="Z" THEN B$(K)="C"
8090 IF B$(K)="W" THEN B$(K)="H"
8100 IF B$(K)="N" THEN B$(K)="M"
8110 IF B$(K)="T" THEN B$(K)="D"
8115 REM --- TRANSFORM H INTO PREVIOUS LETTER UNLESS IT BEGINS WORD
8120 IF B$(K)="H" AND B$(K-1)>" " THEN B$(K)=B$(K-1)
8130 IF B$(K)="A" THEN 8150
8135 REM --- REPEATED CONSONANT DELETED

```

```

8140 IF B$(K)=B$(K-1) THEN B$(K)=" :K=K-1
8145 REM --- REPEATED A DELETED
8150 IF B$(K)="A" AND B$(K-1)="A" THEN B$(K)=" :K=K-1
8155 REM --- DELETE A AFTER CONSONANTS
8160 IF B$(K)="A" AND B$(K-1)>" " THEN B$(K)=" :K=K-1
8165 REM --- A AFTER A BLANK ALLOWED (EG AT START OF A WORD)
8170 J=J+1
8180 K=K+1
8190 GOTO 8010
8200 FOR M=1 TO LEN(A$)
8220 B$=B$+B$(M)
8230 NEXT M
8240 RETURN
9999 PRINT:PRINT "***** END OF PHONETIC SUBROUTINE TEST *****":END

```

Dr. Franz J. Frederick is an associate professor of Media Sciences in the Department of Education at Purdue University. He teaches courses in Computer Assisted Instruction and Information Science. These courses are centered around the use of micro-computers in the schools. His home computer system is based on a SWTPC 6800 board set with 16 K of RAM, a KC standard 300/600 BAUD tape interface, 128 characters per line 16 line video terminal, and a servo controlled plotting robot.

TELEMATH

Lois Noval
Noval, Inc.
8401 Aero Drive
San Diego CA 92123

Instructional games introduce an element of excitement into the learning environment, a welcomed motivational tool. TELEMATH represents a cooperative endeavor between Noval, Incorporated and the San Diego Unified School District to develop and test a computer based educational system. This audio-video graphic system presents drill and practice game activities to complement instruction in basic mathematics computation.

"Way to go, Billy!" A cheer goes up as the drag racer moves closer to the finish line. Enthusiasm radiates from the corner of the room as teammates work together in an effort to bring their car to victory.

The students are playing an exciting, new educational game, TELEMATH. These "action" games, displayed on a television monitor, present challenging mathematics problems to the students. Teams race to enter their responses on calculator-style keyboards. The first correct response is rewarded by exciting game action, moving the team closer to victory.

"Today, games and their pleasurable effects are enjoyed by players of all ages, occupations, social and cultural groups. Whether games are the outgrowth of a natural activity, or contests contrived to accomplish a particular goal or objective, or diversions engaged in for fun and excitement, they continue to motivate, interest, excite, inspire and provoke young and old."

from "What's in a Game?" by Howard G. Ball in The Elementary School Journal Sept. 1976.

From the pinball craze to the successful Othello, games are becoming an increasingly popular form of challenging entertainment. Games bring out the competitive nature in each of us. The origin of games has been attributed to man's desire to amuse and challenge himself. Whether we are playing against another individual, or striving to improve a personal "all time" high score, games help to bring out that little bit of show-off found in everyone.

Game arcades, featuring a variety of coin-operated games, have enjoyed great success in the past few years, coaxing quarters from the competitive public. The tremendous sales of PONG and other home television games are but only another indication of the vast interest in entertaining games. The home electronic video market has expanded from 100,000 games sold in 1975 to sales of over 3 million games in 1976. A conservative estimate projects sales of over 10 million games by 1980. These figures reflect an increased amount of leisure time devoted to competitive game playing.

Games which exploit this competitive spirit can play an important role in the educational process. Instructional games are gaining wide acceptance among the nation's educators, and for a very good reason. Games can provide unique and stimulating learning experiences for students of all abilities.

The instructional game introduces an element of excitement into the learning environment. Highly motivating competition generates enthusiasm from students, who become eager to actively partic-

ipate in the learning process. Games afford students an opportunity to work together. Competing in stimulating game activities, students mature socially as well as academically.

Integrated into the educative process, games can be designed to complement classroom activities. Students can use game activities to work on areas in which they are weak, reinforcing previous instruction. To insure maximum effectiveness, game activities should be selected which make a contribution to classroom lessons.

Formed barely six years ago, Gremlin Industries, Incorporated has grown to become the recognized leader in wall game design and manufacture for the coin games industry. Common sense engineering design has lead to Gremlin's reputation for reliability, known throughout the coin games industry.

Based on a mutual interest in developing computer-based educational games, Gremlin Industries, their sister corporation, Noval, Incorporated, and the San Diego Unified School District joined together in the development, testing and implementation of the TELEMATH system. Following district-written curriculum guides, teachers and programmers cooperated in developing appropriate game activities to complement classroom instruction in mathematics. Noval's commitment to education brings with it years of experience in the electronics industry and a working knowledge of the educational process and game design.

The TELEMATH system is currently being piloted by over 500 students in the San Diego area. The project has been funded under an ESEA Title IV-C grant to the San Diego Unified School District. A self-contained, compact microprocessor system has been developed to deliver district-written curriculum for small group instruction in mathematics computation. The major objective of the project is to improve mathematics computational skills of fourth and fifth grade pupils through use of computer/video graphic instruction.

Eight schools are involved in the TELEMATH project, widely separated by geographic location within the city, by socioeconomic neighborhoods and by the ethnic makeup of their student bodies. Students at the fourth and fifth grade levels have been selected at random to participate in the project. Target students spend one hour each week in a math lab environment, participating in instructional game activities. Four objectives have been established in the experimental design:

- 1) TELEMATH target students will master 80% of the computational objectives in which they receive instruction;
- 2) TELEMATH target students will score significantly higher than the control group on an end-of-year post-test;
- 3) TELEMATH target students in above-average schools will, on the average, score at or above their highest mean score; those in below-average schools will, on the average, reduce the gap between the national norm and their preceding three years by one-third (as measured by the CTBS arithmetic computation section);
- 4) TELEMATH target students will, on the average, score significantly higher than the control group on the end-of-year CTBS arithmetic computation section.

Again, these are only project objectives. The project has just passed the midway point of its first year; as yet, no statistical data has been compiled. We can, however, state that the enthusiasm that students and teachers alike have displayed for the system is overwhelming. In one

project school, sixth grade pupils not involved in the project are permitted to use the equipment after school hours. One slight problem has arisen - the system is so popular, students must sign up a month in advance to reserve the system for their use!

TELEMATH is a creative solution to an important educational problem, the decline of computational proficiency. A computer audio-video-graphic instructional system controlled by a microprocessor, TELEMATH serves as an ideal alternative to the daily classroom drill. This self-contained system displays game activities through a medium which is associated with pleasure - television.

Exciting game activities, designed for small group instruction, are displayed on a television monitor. The system's graphic capabilities go beyond the standard character set (letters and numbers) to allow realistic representations for special effects. A "writeable" character generator permits each program to create figures appropriate to the game design. The system also makes effective use of sound which contributes to game action.

Each activity program is read into the system from a strip of film containing bit patterns which "program" the computer. The use of film as a program storage device is both inexpensive and reliable. The simplicity of film loading and the durability of the film itself permits the students to operate the system themselves.

Dual calculator-style keyboards allow students to enter their answers to problems which are shown on the screen. The digits, 0-9, are represented on the keyboard, as are symbols for each of the operations (+, -, x, ÷), decimal point, comma, equals, greater than, less than, yes and no. Three special function keys are also included: ERASE, which allows the student to delete an incorrect entry; ENTER, which concludes each student response; and "?", which can be pushed at any time during a game to display "help instructions," an explanation of the game activity.

Upon completion of the film loading, the game title appears on the screen. At this time the students have two options. First time students can push "?" on their keyboards for an explanation of the game or they can push "ENTER" and begin the game activity. The teacher also has several options which can be exercised at this point in the program, explained in a section which follows.

At the beginning of a game activity, each player or team of players is instructed from the screen to identify themselves by entering their name initials via a simple numeric code. These initials, which are used to distinguish player scores, serve to personalize the system. Students become more involved in the activity when they are represented by their own initials on the screen.

Students race to enter the answer to problems displayed on the screen. Correct responses are rewarded by exciting game action; if a student answers incorrectly, the correct answer is displayed for drill reinforcement. Students can compete against each other, either individually or in teams, or a single student can compete against the computer.

The versatility of the TELEMATH system allows the teacher to tailor the mathematics drill to meet the needs of individual students. Game activities make use of preset options which the teacher may change as required. For the teacher to exercise one or more of the options, a code sequence known only to the teacher must be entered.

Two options are available to the teacher. Each activity allocates the amount of time that a student has to enter a response to a particular problem. This time may be increased to provide the slower student a longer period in which to respond, or shortened to create a speed drill for the more ad-

vanced student. A random digit option is offered by some activities. The random digit set from which a problem's digits are chosen may be changed to drill students on specific combinations. These options revert to values preset for the activity if the teacher does not desire to change them.

The microprocessor technology used in Gremlin wall games and in our new generation of video games forms the basis for our educational system. This simple and reliable engineering design has been tested and refined over six hardware generations. The primary design goal is to be cost-competitive, vital not only in the game business but in the educational market as well. As a sister corporation to Gremlin, Noval shares in their buying power which yields a lower system cost. Performance has not been sacrificed for this cost reduction. TELEMATH hardware has been field tested for over a year, not only in educational systems, but in thousands of Gremlin games around the world.

Unlike many other systems for computer-assisted instruction (CAI), TELEMATH is a cost-effective system, well within the budget capability of most schools. Unlike a computer terminal, TELEMATH is a stand-alone system, not tied via telephone to a large, expensive time-sharing computer. Although time-sharing systems play an important role in education, most school districts do not have the funds to take advantage of their offerings.

TELEMATH systems are compact and self-contained. Easily moved from room to room on a small cart, the system can be shared by a number of classes within a school. Whereas most CAI terminals can be used by only one student at a time, the TELEMATH system is most effective when used by small groups of students. Students can use the TELEMATH system with minimal supervision. This helps free the teacher for work and activities with other students, an important consideration for today's crowded classroom.

The TELEMATH philosophy is based on providing exciting educational activities to large numbers of students at a price school districts can afford. Our first library of activities deals strictly with mathematics - computational skills objectives in particular. Our aim is to expand the mathematics activities to include libraries in measurement, the metric system and geometry. All new libraries of activities will be compatible with the standard hardware system. There will be no need to upgrade or expand the hardware, the only cost involved being the purchase of the new activity films.

We at Noval are enthusiastic about the future of the TELEMATH system. Reliable engineering, coupled with our experience in the game industry and the assistance of local educators, make TELEMATH an innovative development in the field of educational computers.

Lois Noval, a programmer actively involved in the development of the TELEMATH system, is a graduate of Purdue University with a BS degree in Math Education/Computer Science.

REFERENCES

Ball, Howard G. "What's in a Game?", The Elementary School Journal, Vol. 77, No. 1 (September 1976), p. 42-49.

Davis, Sam. "More Games from National," Electronic Engineering Times, February 21, 1977.

"Factory Report: Gremlin Industries," Replay Magazine, Vol. 11, No. 2 (December 1976), p. 20-22.

Frost and Sullivan, Inc. The Coin Operated and Home Electronic Games Market, June 1976.

Information and Planning Associates. Electronic Toys, Games and Amusements, June 1976.

STUDENT RECORDS SUBROUTINE FOR COMPUTER-ASSISTED INSTRUCTION LESSONS IN EXTENDED BASIC

Franz J. Frederick, Assoc. Professor
 Education Building, Room 112
 Purdue University
 West Lafayette IN 47907

ABSTRACT

The presentation deals with the implementation of an algorithm for creating and reviewing student performance records within a BASIC CAI lesson. It reviews student oriented performance records. A design which implements teacher needs in a student records procedure is outlined. The procedure is implemented as (1) a lesson presentation strategy and (2) a set of two subroutines allowing complete or abbreviated reviews of student performance. Completely commented listings are included.

Computer assisted instruction lessons can be written in almost any computing language. Why then are there specialized computer-assisted instruction languages? The answer is threefold. One, CAI languages typically have a limited set of commands. Two, CAI languages usually have some special answer processing functions. Three, CAI languages usually provide facilities for recording student performance and for providing a copy of those records for the teacher. Virtually all CAI languages are interactive; those which aren't tend to come under the rubric of computer-managed instruction (CMI).

As useful as CAI languages can be for teaching, none exist (publicly at least) for use directly on any of the popular micro-computers. BASIC on the other hand is available in more or less extended versions on the more popular micro-computers. While BASIC is a general purpose computing language and does not have the built-in CAI functions, it is possible to generate these capabilities through subroutines.

This presentation will concentrate on the design of student performance records subroutines in BASIC.

There are three major purposes for maintaining records of student performance. One, the learning process is enhanced when the learner knows immediately of the quality of his performance. It is also of particular value to the learner to know of his performance relative to the total task or lesson. Two, the teacher can more effectively guide the student in learning if the teacher has records which indicate the student's performance with respect to specific items. Three, the teacher can use performance records to assess the effectiveness of the learning materials - i.e., the teacher can identify those areas of the task needing re-design.

The first purpose described above can be accomplished by the use of simple correct and incorrect answer counters of the form:

line # R=R+1 (Right Answer)

or

line # W=W+1 (Wrong Answer)

The appropriate counter is incremented after judging the students' response. At the end of the program, the lesson author would probably use something similar to the following code:

Line # PRINT "YOU GOT ";R;" RIGHT WITH ";W;" WRONG!"

Line # PRINT "YOUR OVERALL PERFORMANCE WAS ";INT((R/R+W)*100); "% CORRECT."

The procedure just described is entirely student oriented, ie, no records are kept for teacher use.

The second and third purpose for maintaining performance records is to make information available for teachers.

STUDENT RECORDS DESIGN

The design of a more pervasive student records procedure may be summarized as follows:

1. Identify the information to be retained.
2. Specify matrix layout required to maintain the information.
3. Identify items to be manipulated or defined for each question/answer block presented.
4. Specify information to be presented in a summary report form.
5. Specify completed record report form.

The design presented here is predicated on a machine with no disk or a machine with a BASIC which permits storage of array/matrix data.

1. Information to be retained.

It is often possible that a learner might encounter the same question/answer item more than once. Therefore, it is useful to know the actual sequence encountered by the student. It is also occasionally very useful to see the students' actual response to an item. Another useful item of information is some identification of which block of information was encountered. Last but by no means least, one needs the result of the response judgement.

2. Matrix layout.

It is probably most convenient to use a two dimensional string matrix for storage of the performance record.

The author has arbitrarily designated the matrix as R\$(R,C).

COLUMNS

	1	2	3
1	1	BASIC	+
2	2	YES	-
3	2	NO	+
ROWS 4			
5			
6			
7			
n		\$QUIT	

EXAMPLE OF RECORDS MATRIX

3. Items to be defined or manipulated in BLOCK.

Most CAI languages treat lessons as discrete "chunks" of information. The "chunks" may be either information to be printed or questions and answers or a combination. We will refer to such an arrangement as a "BLOCK".

The structure of a block would appear as follows:

Line # R=n (Defines number of block)

Line # I=I+1 (The Ith time the student has encountered an item; used as a row index to records matrix)

Line # R\$(I,1)-STR\$(R) (Stores a string representation of the block number in the record matrix)

Line # PRINT "WHAT IS THE MOST AVAILABLE MICRO-COMPUTER LANGUAGE"

```

Line # INPUT A$
Line # R$(I,2)=A$ (Store student answer in matrix)
Line # A$(1)="BASIC"
Line # A$(2)="EXTENDED BASIC"
Line # IF A$>A$(1) AND A$>A$(2) THEN LINE #n
Line # R$(I,3)="-": PRINT "WRONG!"
Line # GOTO LINE #n+
Line #n R$(I,3)="+": PRINT "RIGHT!"
Line #n+1 GOTO NEXT BLOCK

NOTE: After last block, Set I=I+1 THEN R$(I,2)=
      "$QUIT"
      (This signals the end of the entries in
       the matrix)

```

4. Summary report.

The summary report should print a report heading and the columnar headings - "block number" and "judgement". It should also print the total number correct, total number incorrect, and a performance percentage.

5. Complete report.

The complete report should include a column for the actual response as well as all the data for the summary report.

The subroutines are actually only the routines used to print the contents of the records. The records are actually generated through the "BLOCKS". The subroutines are executed after the student completes his lesson by typing GOTO n where n is the starting line number for the desired report.

The following listing shows the subroutines and two example "BLOCKS".

STUDENT RECORDS PROGRAM LISTING

```

10 CLEAR 500
50 DIM A$(5)
60 DIM R$(100,3)
80 PRINT:PRINT:PRINT "      CAI LESSON AND DEMONSTRATION OF STUDENT RECO
RDS "
90 PRINT:PRINT
100 INPUT "WHAT IS YOUR NAME";N$
110 I=0
115 REM ***** FIRST BLOCK IN LESSON *****
120 R=1:I=1:R$(1,1)=STR$(R)
130 PRINT "NAME THE MOST POPULAR LANGUAGE ON MICRO-COMPUTERS CURRENTLY"
140 INPUT A$
150 A$(1)="BASIC"
160 A$(2)="EXTENDED BASIC"
170 A$(3)="MTS BASIC"
180 A$(4)="8K BASIC"
190 R$(1,2)=A$
200 IF A$>A$(1) AND A$>A$(2) AND A$>A$(3) AND A$>A$(4) THEN 250
210 R$(I,3)="+"
220 GOTO 270
250 R$(I,3)="-"
260 REM ***** SECOND BLOCK IN LESSON *****
265 REM --- NOTE THAT THIS BLOCK REQUIRES THE STUDENT TO GET THE ANSWER
266 REM     CORRECT BEFORE HE CAN MOVE AHEAD IN THE LESSON.
270 R=2:I=I+1:R$(1,1)=STR$(R)
280 PRINT "IS IT AVAILABLE IN ALL MICRO COMPUTERS"
290 INPUT A$
300 A$(1)="NO"
310 R$(1,2)=A$
320 IF A$=A$(1) THEN R$(I,3)="+":GOTO 350
330 PRINT "TRY AGAIN!"
340 R$(I,3)="-":GOTO 270
350 GOTO 9995
9000 REM --- STUDENT RECORD PRINT SUBROUTINES
9002 REM     (1) SUMMARY FORM (INCLUDES BLOCK NUMBER AND RESPONSE
9003 REM         INDICATOR)
9004 REM     (2) COMPLETE STUDENT RECORD (BLOCK NUMBER, ACTUAL
9005 REM         STUDENT RESPONSE, AND RESPONSE INDICATOR)
9100 PRINT:PRINT:PRINT "***** STUDENT PERFORMANCE RECORD FOR ";N$;" ***"
***"
9020 PRINT TAB(18);"(SUMMARY FORM)"
9030 PRINT
9040 PRINT "BLOCK NUMBER";TAB(15);"ANSWER JUDGEMENT"
9045 REM --- R IS THE VARIABLE USED TO TALLY THE CORRECT RESPONSES.
9046 REM     W IS THE VARIABLE USED TO TALLY THE INCORRECT RESPONSES.
9047 REM     N IS THE VARIABLE USED TO COUNT THE TOTAL RESPONSES.
9050 PRINT:R=0:W=0:N=0
9055 REM --- MAIN LOOP USED TO EXAMINE STUDENT PERFORMANCE ITEM BY ITEM
9056 REM     INFORMATION EACH ITEM IS PRINTED ON A LINE.
9057 REM     WHEN "$QUIT" IS FOUND IN THE ACTUAL ANSWER AREA FOR THE
9058 REM     LAST ENTRY IN THE RECORDS MATRIX, THE PROGRAM THEN

```

```

9059 REM     PRINTS TOTALS AND PERCENTAGE OF PERFORMANCE.
9060 FOR I=1 TO 100
9070 IF R$(I,2)="$QUIT" THEN 9120
9075 REM --- PRINT ITEM (BLOCK) NUMBER AND RESPONSE INDICATOR.
9080 PRINT TAB (5);R$(I,1);TAB(20);R$(I,3)
9085 REM --- COUNT CORRECT RESPONSE
9090 IF R$(I,3)="+" THEN R=R+1
9095 REM --- COUNT INCORRECT RESPONSE
9100 IF R$(I,3)="-" THEN W=W+1
9110 NEXT I
9115 REM --- SET N COUNTER TO ONE LESS THAN I BECAUSE LAST ITEM IN
9116 REM     STUDENT RECORD IS USED FOR FLAG FOR END OF RECORD.
9120 N=I-1
9130 PRINT
9140 PRINT "TOTAL ANSWER BLOCKS SEEN: ";N
9150 PRINT "TOTAL INCORRECT ANSWERS: ";W
9160 PRINT "TOTAL CORRECT ANSWERS: ";R
9170 PRINT "    PERFORMANCE PERCENTAGE: ";INT(R/N*100);"%"
9180 PRINT "- - - - -"
9190 PRINT "-----"
9200 PRINT:PRINT:STOP
9300 PRINT:PRINT:PRINT "***** STUDENT PERFORMANCE RECORD FOR ";N$;" ***"
***"
9310 PRINT TAB (18);"(COMPLETE RECORD)"
9320 PRINT
9325 REM --- PRINT BLOCK NUMBER, ACTUAL RESPONSE AND ANSWER JUDGEMENT F
9326 REM     EACH ITEM
9330 PRINT "BLOCK NUMBER";TAB(15);"ACTUAL RESPONSE";TAB(50);"JUDGEMENT"
9340 PRINT:R=0:W=0:N=0
9350 FOR I=1 TO 100
9360 IF R$(I,2)="$QUIT" THEN 9410
9370 PRINT TAB (5);R$(I,1);TAB(15);R$(I,2);TAB(55);R$(I,3)
9380 IF R$(I,3)="+" THEN R=R+1
9390 IF R$(I,3)="-" THEN W=W+1
9400 NEXT I
9410 N=I-1
9420 PRINT
9430 PRINT "TOTAL ANSWER BLOCKS SEEN: ";N
9440 PRINT "TOTAL INCORRECT ANSWERS: ";W
9450 PRINT "TOTAL CORRECT ANSWERS: ";R
9460 PRINT "    PERFORMANCE PERCENTAGE: ";INT(R/N*100);"%"
9470 PRINT
9480 PRINT "- - - - -"
9490 PRINT:PRINT:STOP
9995 I=I+1
9996 R$(I,2)="$QUIT"
9999 END

```

Dr. Franz J. Frederick is an associate professor of Media Sciences in the Department of Education at Purdue University. He teaches courses in Computer Assisted Instruction and Information Science. These courses are centered around the use of micro-computers in the schools. His home computer system is based on a SWTPC 6800 board set with 16 K of RAM, a KC standard 300/600 BAUD tape interface, 128 characters per line 16 line video terminal, and a servo controlled plotting robot.

A QUESTION-ANSWERING SYSTEM ON MATHEMATICAL MODELS IN MICROCOMPUTER ENVIRONMENTS

Milos Konopasek & Mike Kazmierczak
School of Textile Engineering
Georgia Institute of Technology
Atlanta GA 30332

Abstract

The QAS language enables the user to communicate with the computer at the level of mathematical relationships rather than sequential programs. Once the information on a mathematical model is stored, the QAS operating system accepts any LM - displaying the list of models available from the relevant question and produces an answer. The first group of operation codes deals with loaded modulus; Successful implementation of a simplified version of RM <model name> - reading the specified model QAS on Compucolor microcomputer widens the prospects of applications of personal computers as knowledge DM - displaying the listing of the assigned model. storing media and problem solving tools.

The paper outlines basic concepts of the QAS. The versatility and flexibility of the Compucolor QAS is illustrated by an extensive sample conversation.

Introduction

The Question Answering System on Mathematical Models (QAS) is a powerful knowledge-oriented interactive language originated at the University of Manchester Institute of Science and Technology in Manchester, England, a few years ago. The language is available at several large computer installations and its further development is going on in Manchester and at Georgia Institute of Technology.

The idea of QAS represents, in some sense, significant upgrading of the human-computer relations. The QAS user does not require writing and inputting the program as a sequence of instructions to be followed by the computer. Instead the QAS user formulates and inputs a set of variables (or "properties" in QAS jargon) and a set of equations linking the properties. He then asks questions by specifying which properties are given and which properties he wants to be evaluated and printed out. The computer automatically finds out the route of solving the problem for a particular subset of given properties (input subset) and comes up with the answer.

Finally, there is a group of operation codes for displaying the information on the current status of the system:

AI <property code> <property value> - assigning a specified property with a specified value to the input subset;
RI <property code> - removing a specified property from the input subset;
AO <property code> - assigning a specified property to the output subset;
RO <property code> - removing a specified property from the output subset.

The next two operation codes EX - initiates solving the problem and printing the results;

ST - terminates the conversation.

There are many programs and program packages which do much the same for highly specialized applications and for limited varieties of input data. The QAS differs from any of them by offering a general framework capable of dealing with models of any kind and purpose in a uniform way.

The QAS provides a lot of problem-solving power at a low expense in terms of required user's skill and computer's resources. It has been realized that this is exactly what a large proportion of the users of microcomputers and personal computers are after. Consequently a simplified version of QAS has been implemented on Compucolor microcomputers built by Intelligent Systems, Inc., Norcross, Georgia 30071.

Several single entries with the same operation code may be combined in one compound unicode entry consisting of an operation code followed by single entry arguments separated by slashes:

<operation code> <arguments1>/<arguments2>/...

Several unicode and single entries separated by semicolons may be combined in one multicode entry.

Besides the answers, the computer's responses include diagnostics of syntactic and semantic errors in user's entries, messages about inconsistencies detected during the resolution of a model, and messages on underdeterminacy or overdeterminacy of the problem.

QAS outline and conversation rules

In the QAS, a model represents a basic unit of information, comparable with a program in BASIC or other high level programming language. The model consists of a list of properties, a list of equations and, optionally, a related data base for relevant empirical constants, notes etc. The models are stored and accessed by the QAS operating system as data. They may be grouped in moduli according to their nature and purpose.

The user-computer conversation controlled by QAS operating system goes through a sequence of user's entries and computer's responses. Each user's conversation entry consists of a two-letter operation code and, are underlined. They were composed of some 1000

Sample conversation

An annotated sample conversation in Appendix exemplifies the main concept of QAS and the practical aspects of its Compucolor version. The conversation covers 10 different topics from elementary math, geometry, physics, chemistry and economics collected in the demonstration modulus.

During the demonstration session the QAS responded to over 40 compound and single user's entries and answered 30 questions, most of them with multiple output, input or both (an equivalent of 100 single answers). The user changed the topic of the conversation 12 times. His entries in the Appendix are underlined. They were composed of some 1000

characters. Our comments are framed.

The asterisks at the property names in the computer's responses indicate elements of input subsets. We were not able to show the enhancement of computer-user communication peculiar for Compucolor QAS: user's entries are echoed in green characters, the answers appear in enlarged yellow characters, and white, purple, red and flashing red characters are used for inconsistency messages, prompts etc.

How does the QAS work?

The demonstrated flexibility and versatility of QAS is based on two interrelated original ideas: 1) representing a mathematical model by elementary equations linking up to three listed or auxiliary properties; 2) an algorithm for automatic resolution of a set of elementary equations by a consecutive substitution procedure.

For example the 19 elementary equations of the "Circle Geometry" model is represented by a 4x19 integer matrix; additional information includes 5 numerical constants, 5 integer parameters of the model, 10 property codes, the model name, and also full property names, original constitutive equations and notes to be displayed when required.

In the current Compucolor QAS the whole operating system is written in BASIC. There are two parallel arrays, one storing all the property values and the other indicating the status of the property (given, evaluated or undefined). During the resolution of the model all the elementary equations are consecutively examined and solved whenever a single undefined property appears.

A more detailed description of the QAS operating system is beyond the scope of this paper. Some more information may be found in [1].

Conclusion

Our first experience with the microcomputer version of QAS amazes everybody including ourselves. It is hard to believe that one gets that level of intelligent response from 8k BASIC + 8k QAS operating system + some space for the model information.

The Compucolor QAS is now ready to "learn" models from various scientific and engineering disciplines, and to be used for investigating and solving problems in a way described in the preceding chapters. Further development of the microcomputer version of QAS will very much depend on the interest and response of the users' community. There are numerous features of the "big" QAS which can be conveniently implemented in the microcomputer environment: entering new models and data bases in free source format, iteration procedures for solving sets of simultaneous nonlinear equations and for optimization, sensitivity analysis using approximate partial derivatives of output properties with respect to input properties, accepting definite integrals and differential equations in the models etc.

On the other hand the development of QAS-like higher level languages for large computers may benefit from the micro-QAS confrontation with the needs and limitations of the users of personal computers.

Authors

Milos Konopasek is teaching at Georgia Tech. He has been involved in computer applications in textiles and elsewhere for 15 years. His current research interests and activities cover interactive systems, computer aided design, pattern packing and covering and nonlinear mechanics. He received his first degree in textile technology from Leningrad and

PhD from Manchester, England.

Mike Kazmierczak studies Textile Chemistry. For the last two years he has been interested and active in computer programming, system analysis and software development.

Reference

- [1] Konopasek, M., "An advanced question answering system on sets of algebraic equations", in 'Proceedings of the European Conference on Interactive Systems', ed. D. Lewin, Online Publications, Uxbridge, England, 1975.

APPENDIX

>>>COMPUCOLOR QUESTION ANSWERING SYSTEM<<<

Our first enquiry, after loading the QAS operating system and the demonstration modulus, is about the list of available models:

QAS MODELS: DEMONSTRATION MODULUS
 TEMP - TEMPERATURE CONVERSION
 UNICO - TRADITIONAL AND METRIC UNIT CONVERSION
 TRIAN - ELEMENTS OF TRIANGLE
 CIRCL - CIRCLE GEOMETRY
 QUADR - QUADRATIC EQUATION
 PROJ - CLASSICAL PROJECTILE PROBLEM
 ELPRM - ELECTRICAL PROPERTIES OF MATTER
 CHEMI - CHEMICAL COMPOUNDS
 LOAN - LOAN REPAYMENT PLAN
 BREPO - BREAK EVEN POINT

We can pick up and ask questions on any of the models. Let's ask, for an easy start, to convert boiling point of water 100°C to °F; then we remove TDC from input subset and ask what is body temperature 98.6°F in °C and °K. After that we let read in and display the projectile model.

? RM TEMP;AI TDC 100;AO TDF;EX

ANSWER: TDF 212

? RI TDC;AI TDF 98.6;RO TDF;AO TDC/TDK;EX

ANSWER: TDC 37.
 TDK 310.15

? RM PROJ;DM

PROJ - CLASSICAL PROJECTILE PROBLEM

1 VEL INITIAL VELOCITY OF PROJECTILE (M/SEC)
 2 H MAX HEIGHT OF PROJECTILE TRAJECTORY (M)
 3 ANG ANGLE OF DEPARTURE (DEG)
 4 RNG RANGE (M)
 5 TIME TIME REQUIRED TO TRAVEL THE RANGE (SEC)

1 H=.05097*(VEL*SIN(PI/180*ANG))**2
 2 TIME=4*SQRT(.05097*H)
 3 RNG=VEL*COS(PI/180*ANG)*TIME

Supposing, the initial velocity is 400 m/sec and desired range is 12,000 m; what should be the angle of departure?

? AI VEL 400/RNG 12000;AO ANG;EX

ANSWER: ANG 23.6843

How does the range change if the angle increases to 25 degrees? Let all the properties be printed:

? RI RNG;AI ANG 25;AO ALL;EX

ANSWER: *VEL 400
H 1456.57
*ANG 25
RNG 12494.5
TIME 34.4654

What is the range in miles? We can use the unit conversion model UNICO:

? RM UNICO;AI M 12494.5;AO MI;EX

ANSWER: MI 7.76372

Let's ask a few questions about triangles:

? RM TRIAN;AI ALPHA 62.8/BETA 75.26;AO GAMMA/C;EX

***INSUFFICIENT DATA FOR C

DEFINE A

OR B

OR REC

? DS GAMMA

GAMMA 41.9398

The two angles are, of course, not enough for evaluation of C and the QAS prompts what properties should be added to the input subset (including the radius of inscribed circle!). The angle GAMMA has obviously been calculated and its value was displayed when asked for.

Let's define the side A and print out all the triangle elements including the triangle area and the radii of inscribed and escribed circles:

? AI A 2.5;AO A/B/ALPHA/BETA/RIC/REC/AREA;EX

ANSWER: *A 2.5
B 2.71833
C 1.87862
*ALPHA 62.8
*BETA 75.26
GAMMA 41.9398
AREA 2.27099
RIC .639991
REC 1.40542

Supposing that instead of ALPHA the C=2 is given and we want to examine the influence of BETA in the interval between 60 and 80 degrees. We can use a multiple assignment mode and get all the results in tabular form. If the a is not removed from the input subset a contradiction is indicated and QAS prompts what to do about it.

? AI C 2/BETA 6 60 64;AO ALL;AO B/GAMMA/AREA;EX

***CONTRADICTORY DATA

REMOVE A

OR C

OR ALPHA

OR BETA

? RI ALPHA;EX

BETA	B	GAMMA	AREA
60	2.29129	49.1068	2.16506
64	2.42204	47.9175	2.24699
68	2.55028	46.6459	2.31796
72	2.67579	45.3052	2.37764
76	2.79835	43.9061	2.42574
80	2.91779	42.4571	2.46202

How do the α , β , γ and area change if we assign b=5? The cosine theorem does not like $(a+c)>b$ and relevant message is printed. We reassigned c and get the solution:

? RI BETA;AO ALPHA/BETA;RO B;AI B 5;EX

***SIN OR COS >1

? AI C 5;EX

ANSWER: ALPHA 28.9552
BETA 75.5227
GAMMA 75.5227
AREA 6.05154

What is the radius, arc and chord in a circular sector with the same area and a central angle equal to the last a? Let's ask the model CIRCL:

? RM CIRCL;DM

CIRCL - CIRCLE GEOMETRY

1	R	CIRCLE RADIUS
2	D	CIRCLE DIAMETER
3	P	CIRCLE PERIMETER
4	A	CIRCLE AREA
5	PHI	CENTRAL ANGLE (DEG)
6	L	ARC LENGTH
7	SRA	SECTOR AREA
8	STA	SEGMENT AREA
9	STH	SEGMENT HEIGHT
10	STC	SEGMENT CHORD

1	D=2*R
2	P=PI*D
3	A=PI*R**2
4	L=PI/180*PHI*R
5	SRA=PHI/360*A
6	SIN(PI/360*PHI)=STC/2/R
7	STH=R-SQRT(R**2-(STC/2)**2)
8	STA=SRA-STC/2*(R-STH)

? AI PHI 28.9552/SRA 6.05154;AO R/L/STC;EX

ANSWER: R 4.8938
L 2.47315
STC 2.44691

Another question: What is the minimum cross-sectional area of an aluminum conductor 40 m long for maximum resistance of 25 ohm? Let's read and display the model first:

? RM ELPRM;DM

ELPRM - ELECTRICAL PROPERTIES OF MATTER

1	I	CURRENT (AMP)
2	R	RESISTANCE (OHM)
3	V	DIFFERENCE OF POTENTIAL (VOLT)
4	L	LENGTH OF CONDUCTOR (M)
5	A	CROSS-SECTIONAL AREA (MM ²)
6	RS	RESISTIVITY (OHM.M)
7	W	ENERGY (JOULE)
8	T	TIME (SEC)
9	P	POWER (WATT)

1	I=V/R
2	R=RS*L/(A/10**6)
3	P=I**2*R
4	P=W/T

NOTE: DATA BASE WITH VALUES OF RS FOR MC=ALUMINUM CARBON COPPER GERMANIUM GOLD IRON LEAD MERCURY PHOSPHORUS PLATINUM SILVER SULFUR

? AI MC ALUMINUM/R 25/L 40;AO A;EX

***MC ALUMINUM NOT LISTED

***INSUFFICIENT DATA FOR A

DEFINE RS

? AI MC ALUMINUM;EX

ANSWER: A .04208

(The word ALUMINUM was misspelled at first and consequently the input subset was short of RS.) What would be the current, power and energy loss per hour if we replace aluminum by copper?

? AI MC COPPER/A .04/V 12/T 3600;AO I/W/P;EX

***CONTRADICTORY DATA

REMOVE R

OR L

OR A

OR RS

We forgot to remove from the input subset the resistance R which, of course, would change.

? RI R;AO R;EX

ANSWER: I .740741
R 16.2
*A .04
W 32000
P 8.88889

? RM LOAN;AI PR 13000/PER 8/IR 6.5/PAYS 1;AO PAY/AMT;EX

ANSWER: PAY 2135.08
AMT 17080.7

In the model CHEM1 the non-numerical properties "Kind of element" E1, E2, are supported by data base with all the values of atomic weights. Let's ask for percentages P1, P2 and P3 of silver, antimony and sulfur in stephanite Ag_5SbS_4 ; we can list the input subset after assigning the elements and see that indeed the atomic weights are there:

? RM CHEM1;AI E1 AG/E2 SB/E3 S;LI
*AW1 107.88
*AW2 121.76
*AW3 32.066

? AO P1/P2/P3;EX

***INSUFFICIENT DATA FOR P1
***INSUFFICIENT DATA FOR P2
***INSUFFICIENT DATA FOR P3

Strange situation: no prompt indicates that the input subset is short of more than one property. Listing of undefined properties shows that we missed the numbers of atoms in a molecule. The following question is about a total weight of stephanite containing 1500 lb of silver and the corresponding weights of antimony and sulfur. Finally, the UNICO model is used for converting the weights in pounds to metric tons.

? LU W1 W2 W3 TW GMW P1 P2 P3 P4 N1 N2 N3
? AI N1 5/N2 1/N3 4;EX

ANSWER: P1 68.3283
P2 15.4239
P3 16.2478

? AI W1 1500;RO ALL;AO TW/W2/W3;EX

ANSWER: W2 338.598
W3 356.685
TW 2195.28

? RM UNICO;AI LB 3 338.6 356.7 2195.3;AO T;EX

*LB T
338.6 .153586
356.7 .161796
2195.3 .995771

The quadratic equation model not only can find the roots of a quadratic equation. Following the basic concepts of QAS, we can ask for the values of any one or two coefficients given one or both roots:

? RM QUADR;AI A -4/B -6/C 18;AO X1/X2;EX

ANSWER: X1 -3
X2 1.5

? RI B;AI X1 3;AO B;EX

ANSWER: B 6
*X1 3
X2 -1.5

? RI A;AI X2 -2.75;AO A;EX

ANSWER: A -2.18182
B .545455
*X1 3
*X2 -2.75

Now something on loan repayment: What is the installment PAY and total amount repaid AMT for principal sum PR of \$13,000 loaned for a period PER=8 years at an interest rate IR=6.5% with PAYS=1 installment per year? And how would the PER and AMT change if we pay \$150 per month?

? AI PAYS 12/PAY 150;RI PER;AO PER;EX

ANSWER: PER 10.0648
*PAY 150
AMT 18116.7

What will the quarterly installment PAY and the total amount repaid AMT be for principal sum of \$4,500 loaned for 2,5,10,15 years at an interest rate of 7, 8 and 9%? The "PAYS" is misspelled and rejected; we list the input subset and find out that all the rest of it is O.K. Then we correct the PAYS and proceed with output specification and execution. Note the modification of the print-out format in multiple assignment mode!

? RI ALL;AI PR 4500/PER 4 2 5 10/PAIS 4/IR 3 7 8

***PROP PAIS UNKNOWN

? LI

*PR 4500
*IR 7 8 9
*PER 2 5 10 15

? AI PAYS 4;RO ALL;AO PAY;EX

	PAY	*IR	7	8	9
*PER					
2	622.228		630.865	639.528	
5	274.377		281.764	289.229	
10	160.175		167.658	175.298	
15	123.519		131.433	139.566	

? RO PAY;AO AMT;EX

	AMT	*IR	7	8	9
*PER					
2	4977.83		5046.92	5116.22	
5	5487.54		5635.27	5784.58	
10	6406.99		6706.33	7011.9	
15	7411.14		7885.99	8373.97	

Four questions on a "Break even point" model: What is the critical volume of production if the fixed cost FAC, cost per unit CPU and sales revenue per unit SRPU are \$14,000, \$340 and \$485 respectively? What profit PRF can we expect if the sales volume increases to SV=120? How should the cost per unit CPU change in order to secure a profit of \$4,000? How, under these conditions, may the price be reduced if the sales volume SV=150, 180, 210, 240?

? RM BREPO;AI FAC 14000/CPU 340/SRPU 485;AO BEP;EX

ANSWER: BEP 96.5517

? AI SV 120;AO PRF;EX

ANSWER: BEP 96.5517
PRF 3400

? AI PRF 4000;RI CPU;AO CPU;EX

ANSWER: CPU 335
BEP 93.3333
*PRF 4000

? AI CPU 335/SV 4 150 180;RI SRPU;AO SRPU;EX

	SV	SRPU	BEP
150	455	116.667	
180	435	140	
210	420.714	163.333	
240	410	186.667	

Finally, we return to CHEM1 model and demonstrate its versatility by asking (a) how many grams of potassium chlorate $KClO_3$ must be decomposed to give 0.96 g of oxygen, and (b) what is the percentage composition of sucrose and glucose. Then we terminate the conversation.

? RM CHEM
? AI E1 K/N1 1/E2 CL/N2 1/E3 O/N3 3/W3 .96;AO TW;EX

ANSWER: TW 2.45114

? RI ALL;RO ALL;AO P1/P2/P3;AI E1 C/E2 H/E3 O
? AI N1 12/N2 22/N3 11;EX;AI N1 6/N2 12/N3 6;EX;ST

ANSWER: P1 42.1059
P2 6.47838
P3 51.4157

ANSWER: P1 40.0007
P2 6.71396
P3 53.2854

READY

.

.

.

USE OF A PERSONAL COMPUTER IN ENGINEERING EDUCATION

Roger Broucke
 Aerospace Engineering
 ENL-414B
 University of Texas
 Austin TX 78712

1. Introduction.

When, in the beginning of 1974, Intel's new 8080 microprocessor was announced, with an impressive cycle time of 500 nanoseconds, it rapidly became clear that we were entering a new era in computing. Indeed, just about one year later, in January 1975, the ALTAIR, a general purpose computer based on the Intel 8080, was announced and made available in kitform.

In November 1975, we decided to evaluate this new type of research tool by actually assembling and testing such a computer. It has now been working since January 1976 and it is being used regularly for small research projects and for teaching purposes in the department of Aerospace Engineering and Engineering Mechanics at the University of Texas in Austin.

The computer has been purchased with our own funds and consists of a CPU, 14K of static RAM, 1K of ROM, a serial I/O port with IBM Selectric, a Keyboard, a Tarnbell Audio Cassette system and a Polymorphics Video system with a standard T.V. set. We believe that this is about the minimum configuration that will allow the solution of most small research projects.

The software we use mostly is an operating system residing in 1K of ROM, a resident assembler and text editor as well as a BASIC interpreter. Besides this we also wrote a Fortran-4 Cross assembler which works on the CDC 6600 computer. It should be said that due to its short wordlength (8 bits) such a computer turns out to be fairly slow in scientific applications. The software floating point operations take an average of 5 milliseconds each, with the standard IBM-360 format (8 + 24 bits).

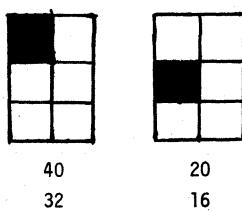
The power of such a system is seriously increased by using the Audio Cassette Recorder as mass storage for Compilers, assemblers and user-type programs. The system uses a tape density of 800 bits per inch and transmits 1500 bits per second. This allows us for instance to load the BASIC interpreter in about 35 seconds. However, the most attractive feature of these inexpensive computers is the flexibility of interfacing with a standard TV set for graphics and numerical displays.

The Video System can display 16 lines of 64 characters on the screen and 128 different characters are available. Besides this, 64 special signs are available for graphics applications with a screen resolution of 48 lines by 128 columns. This feature makes it extremely attractive for classroom applications. We intend to describe here a variety of applications which have been implemented so far in relation with the activities and courses offered in the school of Engineering in Austin. As was said before, we use 1K of ROM-memory for storage of several utility programs such as the software needed to drive the video system. However, besides this machine-language software we also use a flexible graphics subroutine written in BASIC. We have used this subroutine for most of the applications that are described below and for this reason we will first describe the principles of our graphics methods. (Section 2 following.)

As the reader will see, many of our applications require the numerical resolution of systems of ordinary differential equations and we will thus describe one of our most standard subroutines for this purpose in Section 3. The remaining sections will describe some typical applications.



Figure 1



40 20 16 8 4 2

2. Graphics Programming in BASIC

The 64 graphic signs are normally numbered from 0 to 63 (decimal) or from 0 to 77 (octal). For instance, the sign in Figure 1 is normally number 25 (octal) or 21 (decimal). The sign 0 is all white while 63 is black. In using these signs for curve-plotting we will use only six of the 64 available signs. They are the six characters that have only a single black bloc. Their identification numbers are 1, 2, 4, 10, 20, and 40 (octal). Let us note that more complicated graphics signs can be constructed by logically OR-ing the above six signs with each other. This will turn out to be important in our software (Fig. 2).

Before a curve can be plotted for a function such as, for instance, $y = \sin(x)$, it is necessary for the user to define the upper and lower limits for both the abscissa x and the ordinate y . In other words, we assume the four numbers x_{\min} , x_{\max} and y_{\min} , y_{\max} to be given by user. This defines the user's plotting surface and coordinate system, as shown in Figure 3.

It will be convenient to introduce a new system of screen coordinates (x_s, y_s) as shown in Figure 4. As we see, x_s is an integer increasing from 0 to 47 when we move down on the screen, while y_s is an integer increasing from 0 to 63, left to right on the screen. This system is particularly easy to use for plotting with the Video board. We assume here the minimum version with 32 characters on a line. The reader will have no difficulty adapting our formulas to the 64-character line (y_s from 0 to 127).

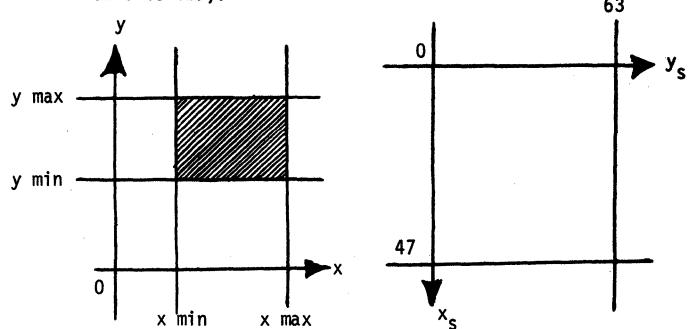


Figure 3

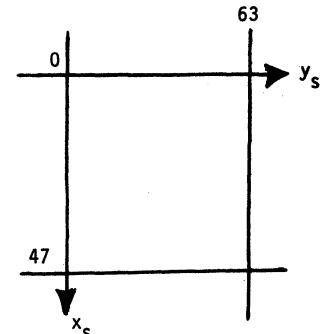


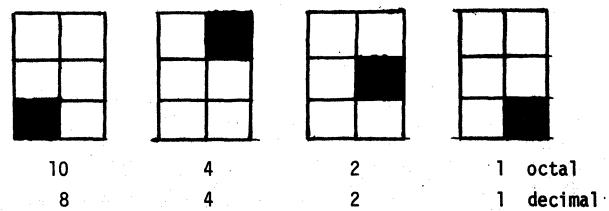
Figure 4

The conversion from the user coordinates (x, y) to the screen coordinates (x_s, y_s) can be easily performed with the formulas

$$x_s = \text{INT} \left[47 \left(\frac{y_{\max} - y}{y_{\max} - y_{\min}} \right) \right];$$

$$y_s = \text{INT} \left[63 \left(\frac{x - x_{\min}}{x_{\max} - x_{\min}} \right) \right].$$

where INT means that we take the integer part of this number. It should be noted that x_s and y_s are the coordinates of a particular black block on the screen, corresponding to a point (x, y) . Next we transform x_s to the line number x'_s and y_s to the character number



1 octal
1 decimal

y_s' by the formulas

$$x_s' = \text{INT} \left(\frac{x_s}{3} \right); y_s' = \text{INT} \left(\frac{y_s}{2} \right),$$

in consequence of the facts that we have three blocs per line and two blocs per character. Finally, we have to determine the address in the screen memory of the byte containing the representation of the point (x_s, y_s) . Let us assume that A_0 is the address of the first location of the video memory. This corresponds to the point $x_s = y_s = 0$ and is in the upper left corner on the screen. In our present system we use $A_0 = 31744$ (decimal) = 76000 (octal). It is known that A_0 should always be a multiple of $2^{10} = 1024$. In our case $A_0 = 31*2^{10}$. This gives us the last one K of memory that can be used with PEEK and POKE in the MITS 8K-BASIC language. For any given point (x_s, y_s) , the memory address can then be computed with the formula

$$A = A_0 + 64x_s' + y_s'.$$

Let us now consider the question of choosing the appropriate bloc, among the six shown in Figure 2, corresponding to a given point (x_s, y_s) . First we will determine if a left bloc (octal 40, 20, or 10) has to be used or a right bloc (octal 4, 2, or 1), according to the ID's given in Figure 2. We can verify that a right or left code α can be computed as follows

$$\alpha = y_s - 2*(y_s/2) = y_s \text{ (MOD 2)}.$$

When $\alpha = 0$ we have a left bloc, while when $\alpha = 1$ we will have a right bloc. Next we have to make the selection between an upper, middle, or lower bloc according to the code

$$\beta = x_s - 3*(x_s/3) = x_s \text{ (MOD 3)}.$$

The values 0, 1, and 2 respectively correspond to the upper, middle, and lower blocs. The ID number 1, 2, 4, 8, 16, or 32 can thus be determined as a function of $\alpha (=AL)$ and $\beta (=BE)$ with the use of three lines of BASIC programming:

```
130 ID=4: IF BE=1 THEN ID=2
140 IF BE=2 THEN ID=1
150 IF AL=0 THEN ID=INT(8*ID + 0.5).
```

The constant 0.5 is added in the last statement in order to force the integer value (INT) of a number such as 7.99999 to be 8 rather than 7. Let us also note that the above three statements can be replaced by a single statement (which may however result in slower execution times) such as:

```
130 ID=INT((8-7*AL)*2+(2-BE)+0.5)
```

or also the following statement

```
130 ID=INT(2+(5-3*AL-BE)+0.5).
```

As was said above this single statement may be slightly slower due to the presence of the exponent. We again add a quantity 0.5 before taking the integer part because of the possibility of a negative round-off error in the exponent subroutine.

To put the appropriate block ID on the screen at the address $A(=AD)$ we could use a statement such as

```
170 POKE AD, ID.
```

However, this destroys the five neighboring blocs and it is thus better to "OR" the new bloc with whatever is already present:

```
170 POKE AD, PEEK(AD) OR ID.
```

To do this, one should initially erase the whole screen. This can be done with a single line of coding, of the following type:

```
100 FOR I=31744 TO 32767: POKE I,0:NEXT I.
```

To finish this section we display below a short program which illustrates all the principles which have been explained. The sample program plots the function $y = \sin(x)$, (statement 117). The statements 102 and 103 give the upper and lower limits of x and y : $X1, X2, Y1$, and $Y2$. The statement 104 determines the number of points to be plotted on the vertical resolution on the screen. All the other statements have

been previously explained. By adding a RETURN statement just after line 150, we obtain a floating subroutine, starting at line 120, having a point x, y as input.

```
100 FOR I=31744 TO 32767
101 POKE I, : NEXT I
102 X1=-6:X2=6
103 Y1=-1.2:Y2=+1.2
104 NP=1
105 DE=(X2-X1)/NP
106 FOR X=X1 TO X2 STEP DE
117 Y=SIN(X)
120 XS=INT(47*(Y2-Y)/(Y2-Y1))
122 YS=INT(63*(X-X1)/(X2-X1))
124 AL=INT(YS-2*INT(YS/2))
126 BE=INT(XS-3*INT(XS/3))
130 ID=INT(2+(5-3*AL-BE)+.5)
140 AD=INT(31744+64*INT(XS/3)+YS/2
150 POKE AD, PEEK(AD) OR ID
160 NEXT X
170 END
```

3. Numerical Integration of Systems of Differential Equations in BASIC

We will show here a subroutine for the numerical integration of systems of ordinary differential equations on a small computer. It uses the standard fourth-order line Runge-Kutta method. The subroutine is written in BASIC and, thanks to the use of multiple statements on a line it is coded in only seven lines. It is a general purpose subroutine for the integration of a system of an arbitrary number of first-order differential equations. It is well known that second-order systems can be reduced to twice the number of first-order differential equations.

The listing of the subroutine is shown below. It starts with the statement number 970 and should thus be called by the statement: GOSUB 970. Each call to the subroutine will perform only one integration step and return to the main program. There are three important Input Parameters that must be defined before entry in the subroutine:

DT is the integration step size.

NEQ is the number of variables or equations.

X0 is an array (with dimension NEQ) containing the initial conditions for each integration step.

The output consists of the array X1 with the values of the variables at the end of each integration step. The dimension of X1 is also NEQ. The main program should transfer the content of X1 to X0 at the end of each integration step, as the output X1 gives the initial condition X0 for the next step. The subroutine uses three other symbols U, F, and D which are arrays used for intermediate storage. The user should declare them with a dimension equal to NEQ in the main program, together with the two other arrays X0, X1.

The Runge-Kutta subroutine calls another subroutine which has to be written by the user for each particular application. This subroutine evaluates the right sides of the differential equations, i.e., the derivatives of all the variables. The derivative subroutine is called by GOSUB 700 and has to start with the statement number 700. The input to the subroutine is the array U (set by the Runge-Kutta subroutine) and the output is the array F.

As an example let us assume that we want to integrate the system of two simultaneous differential equations

$$\dot{x} = x^2 - y^2,$$

$$\dot{y} = 2xy.$$

We then have to write the derivative subroutine:

```
700 F(1) = U(1) * U(1) - U(2) * U(2)
701 F(2) = 2.0 * U(1) * U(2)
702 RETURN
```

We also give below an example of a main program that would integrate the above system with a given step DT and NST steps per solution. This program will print a line at each integration step.

The statement 110 erases the screen first and the statement 165 plots the solution on the screen by calling the video subroutine explained in the pre-

vious section.

```

102 DIM X0(2),X1(2),U(2),F(2),D(2)
103 NEQ=2
105 INPUT NST,DT,X0(1),X0(2)
110 FOR I=31744 TO 32767:POKE I,0:NEXT I
115 T=0.0
160 FOR N=1 TO NST
165 X=X0(1):Y=X0(2):GOSUB 1115
170 GOSUB 970
175 T=T+DT
180 PRINT T,X1(1),X1(2)
190 FOR I=1 TO NEQ:X0(I)=X1(I):NEXT I
200 NEXT N
210 GOTO 105
970 FOR I=1 TO NEQ:U(I)=X0(I):NEXT I:
GOSUB 700
971 FOR I=1 TO NEQ:D(I)=F(I)*DT:U(I)=
X0(I)+.5*D(I):NEXT I:GOSUB 700
972 FOR I=1 TO NEQ:F(I)=F(I)*DT:D(I)=
D(I)+2.*F(I):U(I)=X0(I)+.5*F(I)
973 NEXT I:GOSUB 700
974 FOR I=1 TO NEQ:F(I)=F(I)*DT:D(I)=
D(I)+2.*F(I):U(I)=X0(I)+F(I)
975 NEXT I:GOSUB 700
976 FOR I=1 TO NEQ:X1(I)=X0(I)+(D(I)+
F(I)*DT)/6.0:NEXT I:RETURN
999 END

```

4. Orbital Mechanics Applications

The micro-computer is most useful in Orbital Mechanics because of its capability to display orbits on the T.V. screen in real time. By Real Time we mean that the display is dynamical and that the motion of a particle is visible on the screen as it is integrated in the computer. This gives the student the correct intuitive feeling of the speed. Among the first experiments to be made is the Kepler Problem. In order to have the maximum connection with the real world, the problem is introduced to the student by illustrating the motion of a rocket launched from the surface of the earth. The Earth is assumed circular. The questions of maximum range and minimum initial energy are explained and illustrated first. Next the problem of elliptic trajectories and escape are naturally introduced. Finally the complete discussion of the elliptic, parabolic, and hyperbolic solutions of the Kepler problem is made.

Another orbital problem which is an order of magnitude more difficult is the restricted three-body problem. We introduce this in the undergraduate classes with the example of a spaceship traveling in the Earth-Moon space under the gravity effect of both the Earth and the Moon. The applications to a space colony stationed at a Lagrange Point is impressive to the student. This example allows us to illustrate the concepts of rotating Coordinate Systems, equilibrium points (= Lagrange Points), centrifugal forces and Coriolis forces. With the program that was made, it is possible to let the student visually discover all the equilibrium solutions by placing a satellite with zero velocity on the T.V. screen and observing its motion.

This problem also allows an excellent introduction of the notion of Coriolis Forces. When the orbit is generated on the screen it is very easy to note that the spaceship seems to be influenced by an artificial force which is proportional to the velocity and perpendicular to it. On the T.V. screen this is visible because of the loops in the orbit (the satellite usually appears to deviate or turn to the right!).

The orbits are obtained in the computer by numerically integrating the equations of motion

$$\ddot{x} - 2\dot{y} - x = -(1 - \mu)(x - x_1)/r_1^3 - \mu(x - x_2)/r_2^3 ,$$

$$\ddot{y} + 2\dot{x} - y = -(1 - \mu)y/r_1^3 - \mu y/r_2^3 ,$$

with a classical fourth-order Runge-Kutta method. The parameter μ is the Earth-Moon mass ratio and in the rotating coordinate system which is used, the Earth and the Moon have the coordinates:

$$\text{Earth: } x_1 = -\mu , \quad y_1 = 0 ,$$

$$\text{Moon: } x_2 = 1 - \mu , \quad y_2 = 0 ,$$

so that the origin is the Center of Mass of the system. In BASIC the integration time on the Altair is 2 seconds per step. In the above equations of motion, the dots indicate derivatives with respect to time. Figure 5 displays a typical orbit in the Earth-Moon system.

The use of the computer and T.V. display in the classroom allows us to make orbital mechanics experiments in courses that used to be theoretical and highly mathematical only, without any associated laboratory. These experiments are helpful to get the students acquainted with many abstract concepts (such as equilibrium points and Coriolis Forces).

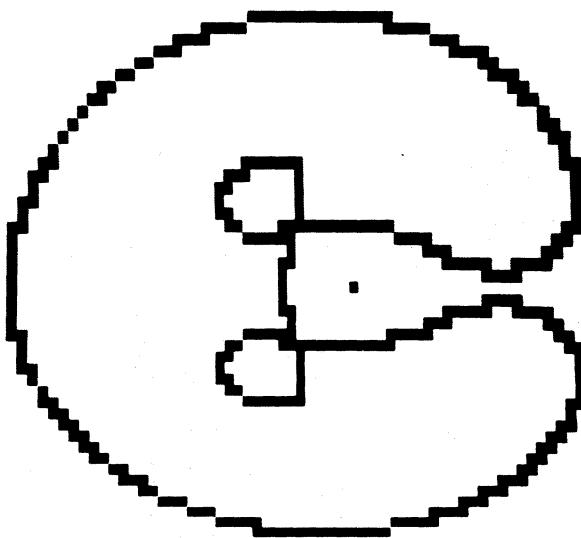


Figure 5

5. Non-Linear Mechanics Applications.

In the applications to Non-Linear Mechanics we find many complex non-integrable systems which have no analytical solutions and which are too complicated to be studied, even on a large digital computer. For this reason, we have studied several simplified non-linear systems which can be integrated on a small computer, while they still have most of the essential properties of the large systems. In other words we have attempted to find the most simple non-linear dynamical systems which are still non-integrable by analytical means or by quadratures. These systems have two degrees of freedom and they are represented by very simple equations of motion. They can thus be integrated numerically on a small computer and the solutions can be displayed on the T.V. screen. These systems have very little physical meaning, except that they are perturbed harmonic oscillators in two dimensions. Up to now we have studied a half a dozen of these systems in detail. The solutions all have the standard property of the large scale non-integrable systems. These properties can be explained very easily to the students by displaying the integrated solutions on the T.V. screen in the classroom.

An example of such a non-integrable conservative system is given by

$$\ddot{x} = -Ax - \alpha(y^2 - \epsilon x^2) ,$$

$$\ddot{y} = -Ay - 2\alpha xy ,$$

where A , α , and ϵ are constants. We have made detailed studies corresponding to $(A=1, \alpha=1, \epsilon=1)$, $(A=1, \alpha=1, \epsilon=0)$, or $(A=0, \alpha=1, \epsilon=1)$ and several other values (Figure 6).

$$x_1 = x \cos \alpha - \beta(y - x^m) \sin \alpha ,$$

$$x_2 = x \sin \alpha + \beta(y - x^m) \cos \alpha .$$

The object here is to start with a given arbitrary point (x, y) and generate the new point (x_1, y_1) , which becomes then the input point in order to generate a new point (x_2, y_2) with the above formulas, and so on. Thousands of successive points may be generated and displayed on the T.V. screen. The interest of the real-time display is that the students get an idea on how certain structures evolve or how some points are generated in a well-defined order. Without the real-time display, the student will see only the final pattern with all its points and he will thus miss much important information.

The above transformation is area-preserving when β is +1 and it reduces to the Hénon-transformation when the integer exponent is +2. The only input for this program is the factor β , the exponent m , the rotation angle α , and the initial point (x, y) . We use as Plotting limits the values -1 to +1 for both coordinates. The student can experiment with different initial points in order to discover such structures as islands, regular curves, irregular (ergodic) patterns, and fixed points. The machine generates about 5 to 10 points per second, which seems to be an appropriate speed to allow the mind to memorize the construction of a pattern. Some typical results are shown in Figures 8 and 9.

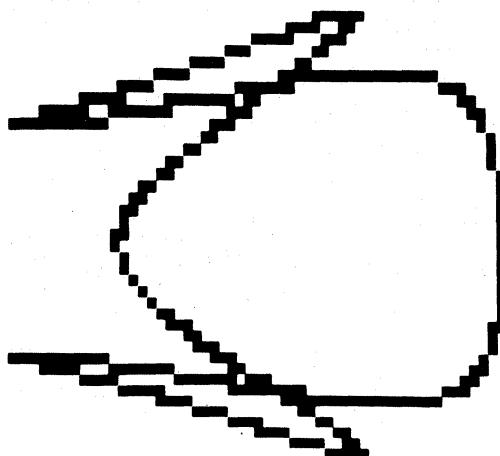


Figure 6

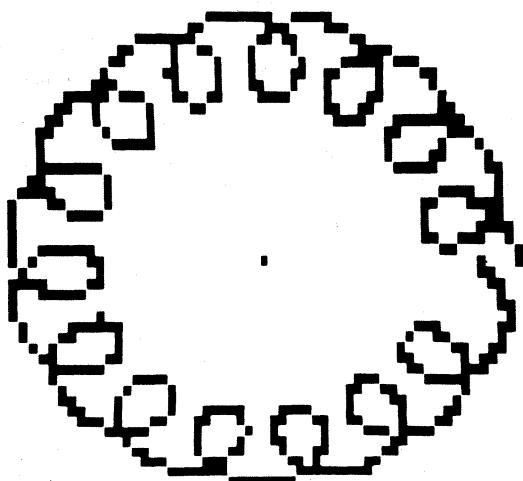


Figure 7



Figure 8



Figure 9

6. Electromagnetic Applications.

A similar application which has been implemented has to do with the motion of a charged particle (electron) in the magnetic field of dipole. This is an important problem in the undergraduate mechanics courses because it illustrates the phenomenon of forces which are perpendicular to the velocity-vector.

The equations of motion in this problem are

$$\ddot{x} = -\alpha \dot{y} / r^3 , \quad \ddot{y} = +\alpha \dot{x} / r^3 .$$

In these equations, α is a constant which is proportional to the magnetic moment of the dipole and the charge of the electron. The solutions usually display a large number of almost circular loops but the velocity is constant (in magnitude). A typical solution is shown in Figure 7.

7. Area Preserving Mappings.

In order to illustrate some important mathematical transformations which occur in the study of non-integrable dynamical systems, we have programmed the following mapping on the Altair Computer:

THE MICROCOMPUTER EDUCATION PROCESS: WHERE WE'VE BEEN AND SOME GUESSES ON WHERE WE'RE GOING

Merl K. Miller
Matrix Publishers
207 Kenyon Road
Champaign IL 61820

As the hobby field moves away from its hardware orientation the need for education grows. Although much can be learned from books, there is a growing interest in formal computer education. The education can be divided into three major categories: the short course, the university course, and the correspondence course. Correspondence courses are not fully developed but we will discuss one of them. However, in this paper we will focus on "university" type courses and short courses.

A short course may be offered by anyone, but they are usually offered by professional short course companies. These companies such as Pro-Log, usually offer sophisticated courses. Pro-Log offers an engineering oriented Microprocessor Architecture course for either the 4040 or the 8080. Topics covered are as follows:

- Design Methods and Techniques
 - Modular Design - Top down structuring, bottom up solving.
 - Integrating program design with hardware design.
 - How to use subroutines.
 - Patching - A simple pencil and paper method of making program design changes.
 - Memory mapping - How to layout programs in memory.
- Documentation
 - How to prepare block diagrams and system flow charts from product specifications.
 - The program assembly form - How to use it in the design process. Symbology, conventions, visual design process. Symbology, conventions, visual aids. Suggested standards.
 - How to redline programs.
 - What the drafting department should do.
- Input/Output
 - Connecting the microprocessors to dedicated I/O - Gates, lights, switches, shift registers, A to D and D to A converters, and other external loads. Includes program techniques for scanning, switch debounce, and noise rejection.
 - Connecting microprocessors to peripheral I/O: TTY, RS 232, printers, terminals.
 - Memory mapped I/O.
 - Interrupt.
- General
 - A comparison of popular microprocessors - 4004/4040, 8080, 6800, and F8.
 - The role of computer aided design - Assemblers, compilers, high level languages.

If you are a computer store owner, you may want to start a course of your own. Before we look at how you might start a short course of your own, let's examine a company that offers both short courses and correspondence courses. Integrated Computer Systems, Inc., was formed in 1974 specifically to teach microprocessor short courses. Two courses should be of particular interest to you even though they assume some scientific or engineering background. Course 102 is a one-day technical introduction and survey. Emphasis of the course is on design and development including: processor selection, I/O and software design, software implementation, development and test equipment, and what pitfalls to avoid. Topics covered are:

- Introduction
 - What is a microprocessor? a microcomputer?
 - Identifying suitable and unsuitable applications
- Fundamental Microcomputer Concepts
 - Terminology
 - Software (SW) - how it works; how it's developed
 - Hardware (HW) - Basic microcomputer configurations
 - The microcomputer design cycle
- The Hardware
 - Microprocessor architectures (4, 8, 16-Bit and slices)
 - Memory systems design - ROM, PROM, RAM, CORE
 - Input/output organization (programmable I/O, interrupts, DMA)
 - Build or buy?
 - Interfacing to the External World
 - I/O port design

- Programmable LSI I/O chips
 - Interfacing to: analog devices, keyboards, displays, cassettes, etc.
- Software Design & Implementation
 - Four implementation methods
 - Editors, assemblers, compilers
 - Assembly vs. high level languages (FORTRAN, BASIC, PL/M)
- Integrating and Testing the HW and SW
 - What really useful tools are available
 - What tools should you build yourself?
 - Isolating and fixing HW and SW bugs
- Technical Survey of Microprocessors and Microcomputers
 - Intel, Fairchild, Motorola, National, Rockwell, Signetics, Texas Instruments, Zilog, and others including the new LSI minicomputers
 - Boardlevel microcomputer systems - PROLOG, PCS, CONTROL LOGIC, WARNER/SWASEY, and others
 - A systematic, application-oriented approach to selecting the right microprocessor family
- Selecting Development and test Equipment
 - Logic analyzers
 - SW simulators
 - Specialized microcomputer debugging equipment
 - Microcomputer development systems
 - Peripherals to buy
- How to Get Started
 - What equipment to buy first
 - Pitfalls to avoid
 - Good information sources
- Course 125 is a two-day course that assumes you have a basic grounding in microcomputers. Each student receives a complete 8080 microcomputer system to use during the course. In this course you learn each new programming concept immediately as it is developed by the instructors. Comparison of the most popular microprocessors are made when appropriate. Topics covered are:
 - Introduction to the ICS Microcomputer Training System
 - Hardware configuration
 - How to use the keyboard/display and built-in commands
 - Exercise: Loading and executing a simple program
 - Exercise: Displaying results
 - Software Fundamentals and Basic Techniques
 - The instruction cycle
 - CPU register instructions - Exercise: Simple arithmetic
 - Conditional testing and loops - Exercise: Time-delay program - Exercise: Testing individual bits
 - Storing/retrieving data in memory - Exercise: Sorting a data table
 - Simple I/O - Exercise: Using a programmable I/O port
 - Subroutines - Exercise: A calculator program.
 - Advanced Software Techniques
 - Interrupt handling - Exercise: A vectored, priority-interrupt response subroutine
 - Real-time programming - Exercise: Organizing the I/O
 - Block I/O - Exercise: Real-time input of a data table
 - DMA I/O - Exercise: Programming a display
 - Multiple precision arithmetic - Exercise: 16-digit addition
 - Program Design
 - Systems analysis
 - Specifying the program
 - Design approaches (top-down, structured programming, modular design)
 - Where High-Order Languages Fit In
 - BASIC
 - FORTRAN
 - PL/M
 - Macro's
 - Utilization of System Development & Test Equipment
 - PROM programmers
 - Logic analyzers
 - Debugging tools
 - Full microcomputer development system with peripherals
 - Final Problem - Implementing a Real-Time Traffic Controller
 - Specifying the problem
 - Partitioning the HW and SW

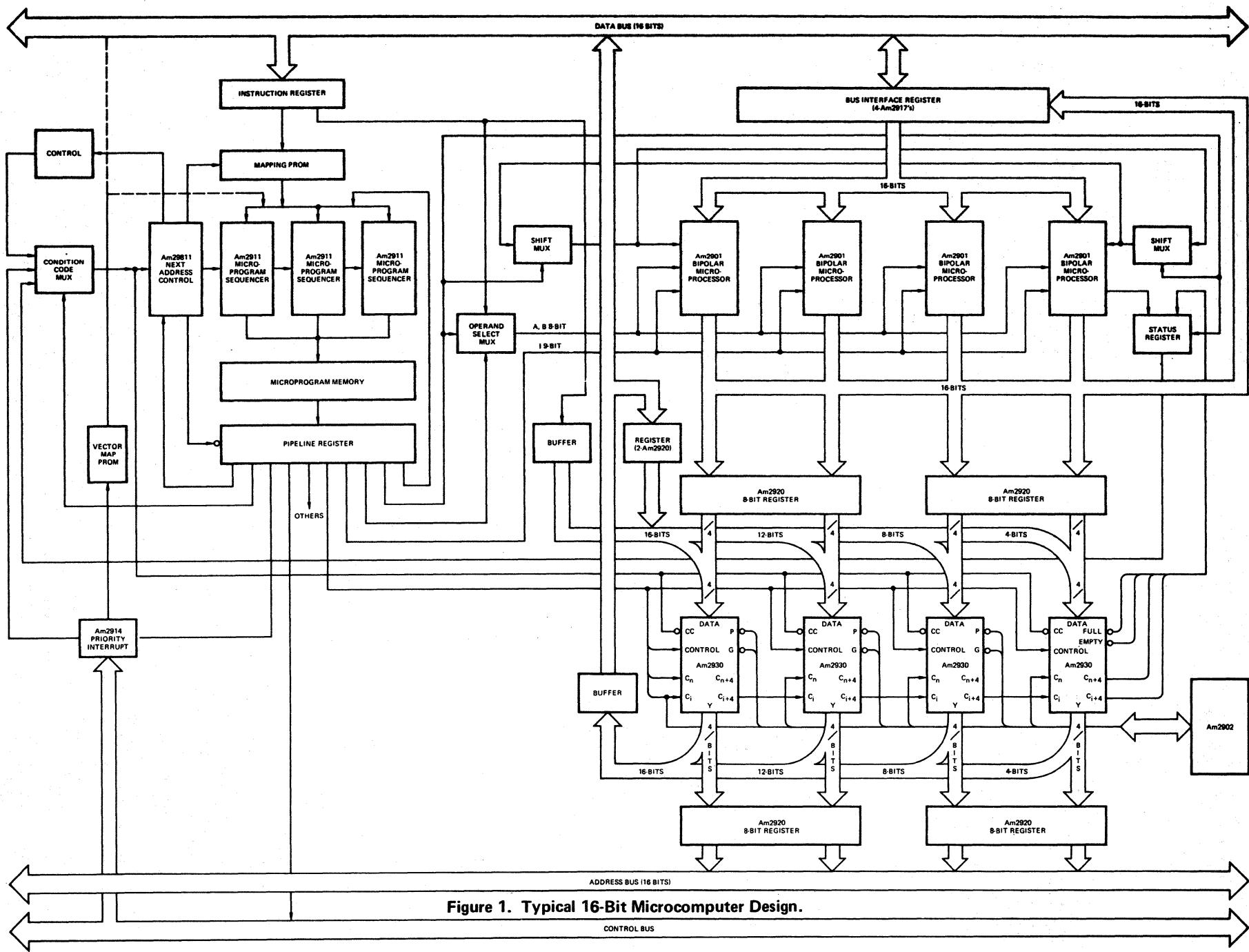


Figure 1. Typical 16-Bit Microcomputer Design.

SW design including flow-charts
 Design critique
 Program implementation (using subroutines developed in class earlier)
 Testing and debugging
 Completing the documentation
 Where to Go From Here

ICS also offers all of their course in a "home-study" mode. For example, their new "ICS Self-Study Microcomputer Training System" includes a complete ready-to-use microcomputer with keyboard and display, educational monitor software, and a self-study text/workbook. The ICS courses are ideal if you want the thorough background necessary for teaching classes in your computer store.

Now, let's turn our attention to how you might offer a short course in your club or store. One easy way would be to buy Digital Equipment's audiovisual course entitled "Introduction to Minicomputers". This audio/filmstrip series teaches computer fundamentals to beginners. It is designed to be used with the Labelle Courier 16T playback unit, and is intended for groups of about 10. Description is as follows:

17 programs
 10 sets of student workbooks
 10 student guides
 10 sets of tests and answer sheets
 1 manager's guide.

This would lay it all out for you but it may be too expensive for your means. An alternative would be to attend a short course or university course yourself and then devise your own course. Such a course could be based on your knowledge of the various equipment available and a group of appropriate books. Following the table of contents of a book is also a good idea if you supplement it with examples appropriate to your equipment. The most important thing about offering any course, though, is to be well versed in your subject. You may want to get someone from a university to help set up the course.

Let's turn our attention to the university type course. Universities, junior colleges, and technical institutes are just now starting to offer courses in microprocessors. Unfortunately most of these courses are offered as senior electives and assume a good background in electrical engineering. A typical course might look like this:

- I Review of Computer Fundamentals
 - a. number systems
 - b. computer organization
 - c. memories
 - d. 2's complement
- II Assembly Language Programming
- III Introduction to the 8080
- IV Hardware Organization
 - a. pin functions
 - b. status information
 - c. machine cycles
- V Interfacing
- VI System Components

This is excellent for someone who has the right background, but it would be more applicable to most people's needs if it could be taught at a less sophisticated level. Here is an example of a less sophisticated course that is taught at Notre Dame:

1. Introduction
2. Basic machine structure
3. Stored programs
4. JUMP instructions
5. Use of memory for data storage
6. System Utility software
7. Uses of terminals for input and output
8. Editors
9. Symbolic assemblers
10. Subroutines
11. Microcomputer architecture
12. Interface devices
13. Interrupts and real-time clocks
14. Peripheral equipment
15. Comparison of various processors
16. Minicomputer vs. microcomputer
17. Resident software vs. cross-assemblers and time-sharing services

If your local college or university doesn't offer a microprocessor course, you may be able to convince them to offer an appropriate one, if you follow these suggestions:

1. Find out who teaches digital electronics in the Electrical Engineering or Computer Science department. He is most likely to be the person who teaches a microprocessor course.
2. Contact the division of continuing education and tell them of your interest in taking such a course.
3. Contact the digital electronics person and ask him where you could take such a course.

This can usually be accomplished by three phone calls.

As the microprocessor continues to change so does the need for more precise information. Few people projected the phenomenal growth of microcomputers in time to react to the changing market. As a consequence, the education function has lagged behind woefully, but there is hope. It seems reasonable to predict that the education "business" will boom in the same way the hobbyist business did.

Initially, education will move a lot faster than the market because it is already far behind the needs. University courses now taught on the senior level will be taught at the freshman or sophomore level within two years, which is an enormous jump for universities. Short course companies will offer a variety of introductory courses aimed at the beginner and the number of computer store courses will probably double within the next six months.

The types of courses offered will also change dramatically. Currently most courses are aimed at microprocessor hardware. New courses will, by necessity, be concerned with computer fundamentals and software. There is already a trend beginning in these two areas. Watch for two new courses in particular: assembly language programming and digital electronics. All in all the next two years should be very exciting for computer education.

MICROCOMPUTERS: A NEW ERA FOR HOME ENERGY MANAGEMENT

Mark Miller
821 Walnut St.
Chico CA 95926

As efficient use of natural resources becomes more important and electronic costs go down, microcomputers will become more common in our daily lives. An area of immediate application is energy conservation and collection. In some climates utility bills for heating and cooling may be eliminated by the use of appropriate building design and low cost microprocessor control systems.

The energy losses in most buildings generally occur in the following order of importance:

1. Excess air infiltration and ventilation.
2. Conduction through the roof and walls.
3. Conduction and radiation through windows.

Most of the energy escaping through the roof and walls may be stopped by high grade insulation. The net energy flow through the windows may be reversed by prudent placement and active control techniques, described later. The energy lost through air change may be reduced by an order of magnitude or turned into a net gain by providing optimum ventilation continuously. For buildings in mild winter climates, the required heat input may come entirely from the active windows and incidental sources like appliances and the metabolic heat of occupants. In areas of severe climatic conditions, fuel requirements may be reduced by over 83 percent, making solar energy systems economical for almost everyone.¹

The target insulation 'R' value of 30 is not difficult to implement in new construction. Existing homes may even be upgraded to this level with plastic insulation.

The 'active windows' consist of double pane glass on the south facing walls. Insulation panels or shutters close automatically when solar heating or night-sky cooling systems are not operating. These panels have a metallic coating that reflects additional solar energy into the building when they are open. This technique is also suitable for retrofit applications.

Most buildings ventilate at about 10 times the recommended rate of 5 c.f.m. (cubic feet per minute) per person.² Reducing ventilation from 250 to 50 c.f.m. at an outside temperature differential of 17 degrees C (30 degrees F) saves 6400 b.t.u. per hour or 45.5 kilowatt hours per day. This equals about \$1.80 per day with electric heating. Energy needed for humidification is not included in the example. Reducing the air change rate further when the building is unoccupied enhances savings.

The microcomputer system functions include:

1. Monitoring human activity and air quality and providing appropriate ventilation control. The ventilation system may also be used to augment the cooling system in the summertime by cooling the building thermal masses with night air.
2. Humidity control. In some climates, indoor humidity control may be necessary. In all climates, relative humidity is an important variable for ventilation control programs.
3. Control of the active windows, solar water heaters or similar devices with data from solar and thermal sensors.
4. Providing a readout on the energy status of the building. Of particular interest are building thermal load, stored energy available and occupancy. This data is also stored for future use.
5. Providing control for any auxiliary functions like wind electric systems, backup heaters, lighting circuits, irrigation systems, intrusion alarms, fire sprinklers or whatever.

In the event of hardware addition or change, the control program may be changed by the supplier or the technical user. The software consists of modular algorithms which can be selected to suit individual applications. The systems use real time interrupt generators to service the subprogram modules. Override switches in the output circuits are provided for manual control.

Some system design considerations are building location, power or fuel availability and building use. Two application examples will be described. The 'Bangor house' is a modest rural dwelling in the Sierra Nevada foothills. It will rely on locally generated electricity. Thermal comfort will be provided by a modified 'drumwall' solar heating system. Several 50 gallon drums positioned adjacent to the active windows will store thermal energy. The rate of energy release will be controlled by a partition between the drums and the living space. This allows the storage mass to attain greater efficiency by using higher differential temperatures. A C-MOS logic system was chosen for its low power consumption and high reliability. This system will use ultrasonic motion sensors to determine occupant activity and ventilation requirements. Humistors will monitor relative and differential humidity. Thermistors will check temperatures of the thermal storage drums and air temperatures. A solar cell will provide data on available solar energy. Magnetic reed switches will indicate the position of the active window panels. On the output side, model aircraft type servos actuate ventilation louvers and control the draft of the backup woodburner. Relays control the window panel drive motors.

The system designed for P.D.M. Associates provides some contrasts to the rural Bangor dwelling. P.D.M. contracted the renovation of two condemned buildings situated in an older urbanized location. Work is nearing completion on the building and it's modified roof pond solar energy system. Proposed use of the structure is a retail alternative energy devices outlet and office space. The Fairchild microcomputer was selected for this application because it has an I/O oriented architecture and is suitable for decimal number handling for the status readouts. The data acquisition section features a unique analog-digital conversion scheme. A resistor type digital to analog converter is driven from a C.P.U. I/O port. This program controlled analog reference signal is fed to several comparators whose outputs are connected to the other C.P.U. I/O port. This port is sampled each time the digital to analog converter is incremented. With all control functions generated by the C.P.U., a simple multichannel medium speed analog to digital converter is realized. The sensors of the Bangor house are duplicated in the P.D.M. building. However the dynamic occupancy characteristics of a commercial building mandate the use of additional sophisticated sensors such as gas transducers to determine air quality.

The prototype systems cost thousands of dollars. Even at these costs, the energy saved can pay for the system in about 6 years. As production and standardization evolve, some systems will sell for a few hundred dollars.

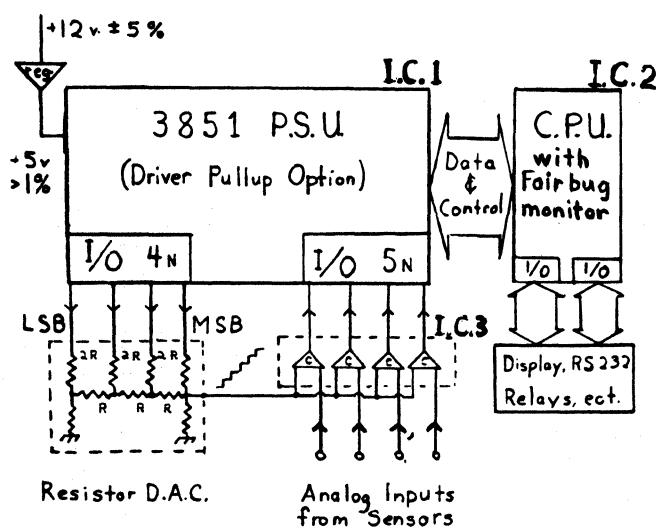
Some problems associated with systems of this kind are:

1. Lack of local maintenance facilities, especially in remote areas. This problem is minimized somewhat by use of a two card system with removable program storage units. These components may be changed and returned through the mail by the user. About 20 minutes each month is required for cleaning of the sensors

- and lubrication of the actuator drive mechanisms.
2. Non standard installations. Since building and climatic situations vary greatly, a standard software design maybe impractical in some cases.
 3. Low end ventilation. Air infiltration rates low enough for periods of low ventilation requirements may be hard to obtain. This is especially true in older buildings and at windy locations. Double door vestibule type entries and vapor barriers in all exterior walls and roof are required. Some manufacturers are claiming good results with sealant and paint type coatings. Special attention must also be given to window seals to reduce infiltration around the edges.

- Representatives." Energy indoors section,
2. Dubin, Fred S., "Guidlines For Saving Energy in Existing Buildings," 1975.

Fairchild F-8



In summary: By structurally integrating the building and the energy collector-storage system, and utilizing intelligent control techniques, great thermal efficiency and economy should be realized. The increase in system efficiency made possible by high technology controls is essential to the simplification of the overall system. This allows substantial reduction of solar hardware. For many buildings in California, recurring fuel costs for heating and cooling can be practically eliminated without a gross investment in hardware.

Energy system control is an exciting application for home microcomputers. Others are doing work in this area. Correspondence is encouraged. I may be reached at:

Box 763, Chico California 95926
Area 916-342-6102/891-1300.

Mark Miller is a member of the International Solar Energy Society and the Association for Computing Machinery. He is currently working on a special major program in energy systems at C.S.U.C. and is an industrial consultant with Microbyte Computer Systems.

References:

1. American Physical Society; from "Hearings Before the Subcommittee on Energy Research, Development and Demonstration of the Committee on Science and Technology, U.S. House of

THE EMPEROR HAS FEW CLOTHES - APPLYING HOBBY COMPUTER SYSTEMS TO SMALL BUSINESS

Michael Levy
 Jethro
 40 Boston Post Road
 Wayland MA 01778

SUMMARY

Present hobby/home computing systems are limited in application to small businesses by problems with hardware, software, and the character of classic programmers and small business entrepreneurs. A suggested method of analytical simulation is described to successfully apply present equipment to this expanding market.

When you open a electronics trade paper now you can usually find an article detailing the tremendous projected growth of small business computer systems. The siren song has sounded and one has visions of an unlimited burgeoning market with growth compounded at 40%, and an estimate for the year of 1.8 billion dollars. By comparison the equivalent hobbyist market is estimated to be only \$200 million, or about 10% as large. The marketing cry of, "all we have to do is get 10% of the market" is heard throughout the industry, and another rash of incompetent hardware salesmen are pushing hardware to people who are neither qualified to use it, nor have the slightest idea of how to get value out of it.

The purpose of this paper is to define the eligible small business in terms of size and purpose, and to discuss prudent limitations imposed by hardware, software, and the nature of prospective programmers and small business entrepreneurs. Finally I would like to discuss a possible successful application of the available equipment which is consistent with the inherent limitations of the present hardware, software, and people.

Reams have been written about the definition of small business and its characteristics. Its purpose is to produce on a timely consistent basis, at a profit, a service or product suitable for the intended use. It is usually managed and owned by a person who is a specialist in some particular aspect of a product or service. He or she is most usually a clever marketer, or a superior technician or expert in his or her chosen field. When presented with the usual situation involving limited data, high risk, and no time, the successful surviving entrepreneur will demonstrate experienced intuitive judgment.

The very small business is a place for mature intuitive judgement, not a place for the simple execution of a proscribed and defined routine. It exists to meet a specialized need which cannot be met by a larger company. Usually it fits into a chink or seam in the market place where size of market is considered not suitable for a larger company, or where entry into a new market would pose an unacceptable threat to an existing profitable large company market. For instance, I.B.M. is slow to introduce new technical developments because to do so prematurely would threaten their existing rental and lease base. Smaller computer companies are always nibbling at the edges of I.B.M.'s market share. In short, these small companies are characterized by great flexibility and adaptability to rapid change.

The preceding description is not a good definition of a programmable algorithm, nor is it a very fitting description of the classic business data processing applications.

Historically these have been the sacred seven: Accounts payable, Accounts receivable, Inventory, Sales analysis, Payroll, P & L and Trial Balance, and General Ledger. Specifically I will try to examine the limitations of present microprocessor based hardware as applied to these and similar data processing applications.

When I.B.M. introduced the system 3 they did so after careful study of their intended Market in the commercial data processing field. They characterized their market as small to medium sized business founded by a single entrepreneur or family group about the time of the depression and now being managed by second or third generation offspring. I.B.M. is very sensitive to the parochial and patriarchal nature of these companies, and has designed a straight forward and simple operating system and language that could be used and applied well by inside company talent. It uses large amounts of core and storage, and has extensive peripheral support. It is targeted at companies ranging in volume from \$1,000,000 to \$10-20 MM, with the smaller System 32 now directed towards companies with volume in the range below \$1,000,000.

I.B.M. has been joined by five to ten major competitors in the lower or System 32 range and essentially many of these companies will try to sell to anyone, no matter how small, that is interested and appears to be able to pay the equipment bill. To supply an economic perspective these companies have a modal

sales volume of 40-100 million as compared to the typical 1-7 million volume of the hobbyist computer company.

We have available cheap CPU power in the so called hobbyist micro field. It is slow, and the peripherals are expensive and not standard. The present real limitation is random access storage. Remember it is a classic data processing environment with which we are dealing. The computation is usually trivial, but the file and record handling is very complicated. Think of the small business system as being composed of three parts. The electronic file cabinet, the file clerk and the typist. The file cabinet is the disk or floppy. The CPU/program is the file clerk, and the typist is the printer. In the hobbyist or home computer system the only element now present in a simple packaged system is the file clerk.

Since our primary DP task will be record and file handling storage becomes a paramount problem. Our experience has been that a very small manufacturing concern can easily eat up 5 megabytes of on-line storage without difficulty. Typical applications involving this level of storage involve cost data, invoicing and receivable records, and production orders.

Floppies in general will not be sufficient, and at the very least 3 floppy drives are needed (a program drive, a "read data" drive, and an update or "write data" drive).

For the hobbyist or home computer market this is the "Catch-22" problem. The low volume, low cost item is the microprocessor. The peripherals and storage media, being electro-mechanical, are still bought by the pound. The commercial systems manufacturers recognized this some time ago, and utilized some of their financial resources to go into the peripheral business. Presently you may go to a commercial system manufacturer and by a packaged dual floppy system including tube and printer for less than the cost of the equivalent system packaged and wholesaled by a hobbyist manufacturer. After all, the commercial system manufacturers also know where to buy Z-80's, 8080's, and 6800's. This situation may be eased in the future when greater proportion of the system is composed of electronics. For instance, when charge coupled devices or bubble memories can be substituted for mass electromechanical storage then there will be a competitive advantage for very small systems.

There are some additional problems concerning the use of hobbyist systems in a small business environment. Service and spare parts availability are a problem. The small businessman's model is probably typewriter or adding machine repair. He just calls and someone comes quickly and fixes his problem. Mailing defective printed circuit boards back to the factory is not his style. Depot spares and service will have to be provided locally for the small businessman. Hardware support for small business is expensive, but it must be provided by the manufacturer or dealer in order to be successful in the small business market.

In the software area the situation is even more drastic. The home computer and hobbyist market is characterized by lack of standardization and a really deplorable lack of systems utilities. Conventional business systems come equipped with utilities such as sorts, file managers, report generators, program cataloging systems and screen menu prompts. These system utility standards save the applications programmer from having to start every job from scratch. You can of course do without these tools just as you can dig a trench, if necessary, with a shovel. Clearly it is more cost effective and faster to use a shovel.

Software can be divided into systems software, which is not directly applicable to the solution of the immediate problem, and applications software which is applicable to the current problem.

The systems software presently available for the home/hobbyist market comes from three major sources: Bundled with the hardware; normal commercial channels involving high prices, restricted access and low volume; low charge, high volume "public domain" software.

Most of the hardware manufacturers have available, for low or no cost, a bundled BASIC interpreter, an assembler/editor, and a video monitor which operates with their particular system. While most of these products are similar, there is in reality no standardization except for that imposed by the fact that most of the manufacturers are using the 8080, Z-80 or 6800 microprocessors. The majority of the disk operating systems are slow, and are now being supplied, bundled, by the disk subcontractors to the CPU manufacturers who then must interface them with their hardware. The package assembled in this manner, instead of being designed as a unit from its inception, is bound to be an operational compromise.

The bundled software which is included with the hardware is largely intended to be a sales aid. There is little or no interest on the part of the manufacturers to make the investment necessary to supply systems utilities such as sorts, file managers etc.

The normal commercial channels which produce commercial software at high cost have largely concentrated on assembler/editors. There have been a number of compilers written for Fortran and other special languages. Very little is available commercially in terms of the classical business application programmers tools. The price is high for most of what's available, and the creators, who are by now aware of some of the past history of proprietary software problems and controversies in the hobbyist market, have carefully restricted access by means of formal licensing programs.

Making software available by means of reasonable price and wide distribution from independent software houses has been proven practical by the sale of various Tiny Basic's, Basic Etc. along with other interpreters. It is my contention that this is the only practical channel for the development of adequate application programmer tools. Non-trivial three array sort and merge routines, video monitor prompting routines, and report generators are critically needed in order to do applications programming on an economical and efficient basis for very small businesses.

There are, in particular, two really important utilities without which the use of hobbyist or home hardware will really be impractical in small business programming. An ISAM (index sequential access method) file handler, and BASIC cross compiler or "cross translator." The ISAM method is a classic "tree" method file handler which allows the rapid retrieval or storage of an individual data record. These file handlers exist for most of the successful mini-based business systems, and there is even one which is written for a proprietary interpreter in extended BASIC. The low cost random access data system that is required by many small business programming applications is easily achieved using an ISAM system. There should be a wide market for this kind of program, and the developer will find it profitable I'm sure.

The second necessary aid is a BASIC cross compiler. This would be a "front end" program for which the input would be some common version of BASIC source language statements. The output of the cross compiler should be then be passed to a cross assembler to produce object code for the user specified microprocessor. Even more exiting would be a further interconnection with a PROM programmer. A business programmer could then enter his code at a time sharing terminal to a host computer, or directly to a mini and walk away with a programmed PROM which he could then plug into a simple microprocessor based system. It would be a very simple and economical way of providing easy-to-operate applications programs to small business. Most important, the programs would not be processor dependent and could be recompiled and applied to new and faster hardware as it became available.

Separate from systems utilities are the standard applications packages which I previously described as the "sacred seven." Unfortunately for very small businesses they are a procrustean bed. Always, in order to be salable to a wide range of customers the package is written to be as general as possible. That creates a form of "package Catch - 22" in that a very small business will, by definition, have use for only a small specific part of the general program. For instance, the selection of the option in a receivables package between an "open item system" or "balance forward" approach may immediately obsolete 50% of the package. This then leads to the paradox - to be salable a package must be general. To be efficiently usable a program must be specific. For very small businesses, which tend to be unique, and owe their very existence to a high degree of specialization, standard packages probably do not have much applicability without a high degree of custom modification. It is disappointing but true that the majority of small businesses are better off with a tried and true manual system, staffed by people who have operated it successfully for years, than they are when they go to a computer system for their standard accounting procedures.

Having neatly, if not adequately, disposed of hardware and software I would now like to comment on what I see as the most serious barrier to the successful use in business applications of this high quality inexpensive hardware. I previously described the small business entrepreneur as a specialist of some sort, either in marketing, or in the particular technical skill represented by the service or product that he sells. He does not control the business by formal analysis, or with the classic financial controls and budgets that you find in a larger company. His "span of control" is established by artful perception, intuition, and detailed experience. He does it by "feel" and judgment and he iterates his experience in an analog rather than a digital manner. Small business people do not tend to think in algorithms, and trying to arrive at a definition of one can be traumatic to an aspiring system analyst.

Business information in a small business is of two types; historical and analytical. Most hardware and software people seem to be concentrating their efforts on the historical accounting data. The entrepreneur knows instinctively, if not intellectually, that his real need is for analytical information to help him decide the future, not record the past. The continuous survival of the company is proof enough of existence of the manual historical records.

Into this volatile situation comes the amateur systems analyst in the form of the computer hobbyist who is looking for some way to justify the existence of a computer system, and to obtain some

revenue in order to buy his next piece of hardware. In his book, Computer Power and Human Reason, Joseph Weizenbaum has written an elegant description of the arch-prototype hobbyist programmer.

He calls them "compulsive programmers." I cannot think of a more unlikely combination than the classic entrepreneur and the compulsive programmer. The hobbyist programmer uses his machine as an "intellectual snowmobile." He's fascinated with the noises and rhythms and he drives it around without much regard for the disturbance that he causes, or for the permanent distortion of someone else's environment. In a strange way, these highly enthusiastic people who say that they are going to free people from the tyranny of the small business routine, are in an arrogant manner, repeating the same people-crunching behavior that the population has come to resent in utility billing and credit card operations. Remember the dictionary definition of hacker, as Dr. Weizenbaum points out, is "one who cuts with repeated irregular or unskillful blows."

Is there then any sensible cost effective way in which present day hobbyist hardware can be effectively utilized in very small business? Probably the best way for the personal computer to be utilized is as an aid to supplying analytical information. The small business manager customarily makes his decisions based on incomplete and marginal information. Usually they involve uncertainty, high risk - a penalty for a bad decision, and a high payoff for a good one. This is a situation in which a clever iterative simulation will do wonders. It is in effect a "super" game, and present day hardware is perfectly suitable for this kind of application.

The value of a simulation may be that it is a substitute for experience. People who have good judgment, acumen, and insight will have their intuitive business decision-making improve as more decisions are made and experience is gained. Simulations can substitute for actual decision-making experience without risk or the cost of error. Perhaps the value of computer simulation is that it allows the rapid evaluation of alternative courses of action and the practice in making the necessary associated decisions.

If the systems analyst is experienced and sympathetic, and can listen intelligently to the businessman's problem, he can usually find an area of the very small business where the entrepreneur realizes he needs some help. This will usually be an area where there is some repetitive task which is taking up a great deal of his time, and where that task is not being done well and he realizes it.

Some eligible functions of small business are quotation models, bidding/product pricing decisions, job cost, marketing product mix, proposals, raw material mixes, forecasting, cash flows, and break even analysis.

An example of a quotation simulation for a small plastics factory follows. The original purpose was to mechanize a laborious but carefully designed manual costing system in which the cost algorithms could be identified and programmed - a major point.

The program was designed to do the following:

1. To make routine a tiresome clerical job.
2. People can prepare a price without having to know the company's current overhead, margins or other confidential information.
3. To establish a real "refusal" price at which the business was not really acceptable under any circumstances. This really is an aid to marketing since they eventually come to agree. Management has then eliminated one of the age old internal arguments between sales and finance.
4. After everyone believes, it can be an ideal marketing device. A responsible salesperson or executive can price on the spot, eliminating the need for the clumsy "elephant ballet" that goes on every time a customer requests a quote i.e. back to the factory etc. Management can allow this because the program is always consistent. As a matter of fact when very small portable BASIC speaking units are available I would not be surprised to see them become as common as calculators in a salesman's kit.
5. People are willing to redo quotations because it is not laborious to do so - an example of the iterative benefit of using a simulation.
6. It provides a rational approach to quantity discounts and to tool amortization since an amortization curve is built into the program.

The program works simply. A pricing format sheet is appropriately filled out (figure 1) and the data is then entered via terminal or screen console. The program then prints the pricing format, and goes on to produce a print out (figure 2) which dis-

plays the main components of the price: material, labor, employee benefits, the total direct costs, and finally a calculation of the per piece direct costs.

The program then calculates six different prices each of which vary in some aspect of volume dependency, or in their percentage of labor and material.

Figure 1

Customer Name	Pricing Format	Quote Log #
Part Description	Date	
(1) Quantity	1000	
(5) Compound Cost \$/lb.	5.2	
(6) Insert #1 \$/pc	3.2	
(7) Insert #2 \$/pc	3	
(8) Other Mat'l \$/pc	1	
(16) # Cavities	4	
(4) Grams Per Shot	1.2	
(17) Molding Crew Size	1	
(18) Molding Minutes	.5	
(19) Reject Rate		
(13) Post Molding #1 Pcs/hr	400	
(14) Post Molding #2 Pcs/hr	800	
(21) Set Up & Start Up Days	1	
(10) Tooling Value	3000	
(16) Main Mold - # of CAVITIES	4	
(17) Main Hold - CREW SIZE	1	
(18) MOLDING MINS	.5	
(21) SET UP AND START - DAYS	5	

Figure 2

(1) QUANTITY QUOTED
 (3) MOST HE'LL PAY
 (4) GRAMS OF COMPOUND
 (5) COMPOUND COST
 (6) INSERT #1
 (7) INSERT #2
 (8) OTHER MAT
 (10) TOTAL TOOLING MARKET VALUE
 (11) ESTIMATED 2 YEAR PART VOLUME
 (13) POST MOLDING #1-PCS/HR
 (14) POST HOLDING #2-PCS/HR
 (16) MAIN HOLD - # OF CAVITIES
 (17) MAIN HOLD - CREW SIZE
 (18) MOLDING MINS
 (21) SET UP AND START - DAYS

MATERIAL	LABOR	EMPL BEN	TOTAL DIRECT	PER PIECE
102.7238	213.6708	42.7341	359.1288	.3591

PIECE PRICE	TOTAL ORDER	CONTRIBUTION	PERCENT
METHOD 1 .8591	859.1995	500.0706	.582
METHOD 2 .7541	754.1765	395.0417	.5238
METHOD 3 1.0561	1050.1484	631.0195	.658
Y1 .7121	712.1777	353.0489	.4957
Y2 .6464	646.4319	287.303	.4444
Y3 .7809	780.9231	421.7943	.5401
AVERAGE K .887839453087	AVERAGE Y .7131776220563	AVERAGE .8005085586825	

VOLUME IS .5 OF EXPECTED ADD \$ 3 TO PIECE FOR TOOL

Figure 3 shows the corresponding cost algorithms actually contained in the program. Algorithms 1), 2), and 3) correspond to "Method 1," "Method 2," and "Method 3." The "price 2 algorithms" correspond to "Y1," "Y2," "Y3" on the print out. The program also averages the two subgroups and finally prints out an average of the prices.

Figure 3

PRICE ALGORITHMS

- 1) 1.4 (2.2 x Labor + Material)

BREAKDOWN:

Labor
 Material
 Emp. Ben. (.2 x Labor)
 Total Direct
 Mfg. O.H. (.10 x Labor)
 Total Mfg.
 G & A Pft. (.4 x Total Mfg.)

Sales Price

- 3) 4.2 x Labor + Material

BREAKDOWN:

Labor
 Material
 Emp. Ben. (.2 x Labor)
 Total Direct
 Mfg. O.H. (.15 x Labor)
 Total Mfg.
 G & A Pft. (1.8 Labor)

Sales Price

- 2) 2 (1.2 x Labor + Material)

BREAKDOWN:

Labor
 Material
 Emp. Ben. (.2 x Labor)
 Total Direct
 x 2

Sales Price

- 4) Est. Sales

Value of Order
 $\frac{500,000}{500,000} \times 205,000 + \text{Total Direct Costs of Order}$

PRICE 2 ALGORITHMS

- 1) 1.3 (1.9 x Labor + Material)
 2) 1.7 (1.2 x Labor + Material)
 3) 3 x Labor + Material

SALES ALGORITHMS

ACTUAL

Sales = 4.9 Direct Labor

(3) = Material + 3.25 D.L. (Act.)

Material + 4.2 D.L. (Est.)

(2) = 2 (Material + D.L.) (Act.)

2 (Material + D.L.) (Est.)

(1) = 1.3 (Material + 1.2 D.L.)+D.L. (Act.) 1.4 (Material + 1.2 D.L.)=D.L. (Est.)

Figure 4 shows an "non-intuitive" iteration in which the capacity of the mold ("16 MAIN MOLD -# of CAVITIES) is changed from 4 to 8, a doubling of the capacity. Intuition would say that if the product can be produced twice as fast the price should decline markedly. It stays almost the same, only declining by about \$3.00 in the "LABOR" category, an unexpected result. On examination the run is short, only a 1000 pieces, and there are five "SET UP AND START-DAYS" which have much more weight in the cost.

Figure 4

(1) QUANTITY QUOTED	1000
(3) MOST HE'LL PAY	2
(4) GRAMS OF COMPOUND	104
(5) COMPOUND COST	3.2
(6) INSERT #1	2
(7) INSERT #2	3
(8) OTHER MAT	1
(10) TOTAL TOOLING MARKET VALUE	3000
(11) ESTIMATED 2 YEAR PART VOLUME	2000
(13) POST MOLDING #1-PCS/HR	60
(14) POST HOLDING #2-PCS/HR	200
(16) MAIN HOLD - # OF CAVITIES	8
(17) MAIN HOLD - CREW SIZE	.5
(18) MOLDING MINS	.5
(21) SET UP AND START - DAYS	5

MATERIAL	LABOR	EMPL BEN	TOTAL DIRECT	PER PIECE
102.7238	210.0614	42.0122	354.7975	.3547

PIECE PRICE	TOTAL ORDER	CONTRIBUTION	PERCENT
METHOD 1 .8472	847.2885	492.4909	.5812
METHOD 2 .745	745.0749	390.2773	.5238
METHOD 3 1.0342	1034.231	679.4334	.6569
Y1 .7025	702.5768	347.7792	.495
Y2 .6386	638.6356	283.838	.4444
Y3 .7695	769.5536	414.756	.5389
AVERAGE K .8755315265587	AVERAGE Y .7035807158063	AVERAGE .7895601211825	

VOLUME IS .5 OF EXPECTED ADD \$ 3 TO PIECE FOR TOOL

DATE 2/23/1976

Since a common customer tactic in this industry is to request large tool capacity in order to reduce part price, the simulation

program has shown company management that it would be unwise to increase capacity in this case.

This simulation proved highly successful as have other more statistically oriented forecasting programs. In fact later we developed some "ball park" estimates by saving the estimates and calculating their probability distribution. This method saved hundreds of hours of tool estimating time.

If the system analyst is a combination of a Bayesian statistician, a cost accountant, and an investment banker then he can successfully apply present inexpensive hobby hardware to small business. The applications, as I have shown, require a sophisticated and careful approach. The limitations imposed by hardware, software, and the typical psychological mix of people found in small business are not trivial and can only be surmounted by careful planning and experienced judgment.

Michael R. Levy is the proprietor of Jethro, a small systems house in Wayland, Massachusetts. He is a graduate of The College of William and Mary, and has done graduate work at Northeastern University and M.I.T. He has been involved in the application of computers to small business since the early days of time-sharing in 1963.

BIBLIOGRAPHY

Alter, Steven L. "How Effective Managers Use Information Systems." Harvard Business Review 54, (November-December 1976): 97-104.

Boston Consulting Group. Perspectives on Corporate Strategy. Boston: The Boston Consulting Group, Inc., 1968.

Brummet, R. Lee. Cost Accounting for Small Manufacturers. Washington, D.C.: Small Business Administration, 1953.

"Cobol Enters Microworld." Computer World, 28 February 1977, p. 1.

Dale, John D. Managerial Accounting in the Small Company. New York: Reinhold Publishing Corp., 1961.

Hunt, Pearson; Williams, Charles M.; and Donaldson, Gordon. Basic Business Finance, 3rd ed. Homewood: Richard D. Irwin, 1966.

Kazmier, Leonard J. Theory and Problems of Business Statistics (Schaum's Outline Series). New York: McGraw-Hill, 1976.

Kulas, Ludwik; Koppenhaver, R.D.; and Clifford, Thomas J. The Development of Management Information for Small Manufacturers. Grand Forks: University of North Dakota under Small Business Management Research Grant Program, [1965].

Krentzman, Harvey C. Managing for Profits. Washington, D.C.: Small Business Administration, 1968.

Morony, M.J. Facts From Figures, 3rd ed. Baltimore: Penguin Books, 1956.

Nickerson, Clarence B. Accounting Handbook for Nonaccountants. Boston: Cahners Publishing Company, Inc., 1975.

Raiffa, Howard. Decision Analysis Introductory Lectures on Choices Under Uncertainty. Reading: Addison-Wesley, 1970.

Rooney, Michael. "Available Microcomputer Software." Paper for Micro-Mini Conference. Boston Systems Office, Waltham, Ma., 1976.

Schlaifer, Robert. Analysis of Decisions Under Uncertainty. New York: McGraw-Hill Book Company, 1969.

"Small Business Computer Shipments Seen Topping \$2.2 B by 1980." Electronic News, 21 March 1977. Insert after p. 48.

Smith, Jon M. Financial Analysis & Business Decisions on the Pocket Calculator. New York: Wiley-Interscience, 1976.

Smith, Jon M. Scientific Analysis on the Pocket Calculator. New York: Wiley-Interscience, 1975.

Weizenbaum, Joseph. Computer Power and Human Reason From Judgment to Calculation. San Francisco: W.H. Freeman and Company, 1976.

Williams, J.D. The Compleat Strategyst. New York: McGraw-Hill, 1954.

Woolsey, Robert E.D., and Swanson, Huntington S. Operations Research for Immediate Application. New York: Harper & Row, 1975.

Yamane, Taro. Statistics: An Introductory Analysis. Second Edition. New York: Harper & Row, 1967.

Zwick, Jack. A Handbook of Small Business Finance. 7th Ed. Washington, D.C.: Small Business Administration, 1965.

THE SOFTWARE DILEMMA

Carl Helmers, Editor-in-Chief
BYTE Publications, Inc.
70 Main Street
Peterborough NH 03458

How is it possible to simultaneously make software widely available (and low priced) yet reward the producers of good software with adequate compensation for their efforts?

Conventional wisdom has it that proprietary software must come at extremely high prices, commensurate with concentrated work on the part of a small number of dedicated and thoughtful programmers. After all, this wisdom has it, we'll only sell a few copies of package X anyway, so why not keep a tight lid on it and charge as much as possible?

This conventional wisdom has worked well in the past, when the typical computer system might cost upwards of \$10,000 or \$100,000. But when the typical computer system comes in at a price on the order of magnitude \$1000, paying prices which are of this same order of magnitude for software packages is not a very likely move on the part of the individual purchaser with his personal budget.

In the personal computing field, we are participating in a market phenomenon characterized by a change from the situation which supports the conventional software wisdom, to a new situation which has its own characteristics. More and more people are getting into the swing of things with computer use and thus more and more people have needs which can and should be filled by specialized software products. Where computers are concerned, when we talk about a 100,000+ person active individual user market as we do today, we are for the first time talking about the potential for mass marketing of software in ways unheard of in the conventional wisdom of computing. Establishing a new "conventional wisdom" is clearly required; as a step toward that goal, this paper provides a survey of the prospects for mass marketing of software, and a solution of the software dilemma posed above.

Let's Draw Some Parallels: Woodworking

Like many individuals, I dabble a bit in the arts of crafting furniture. Suppose, for example, that I want to build a nice, neat contemporary roll top desk for my study. As an individual with limited time available for such leisure crafts activity, I'd probably want to start with an existing design rather than working out all the details myself. In seeking the end product of a roll top desk, I'd be in the same situation (as a wood craftsman) as the owner of a computer system desiring a compiler, assembler, application product, or peripheral. I know in principle that roll top desks exist and that in principle I could design then fabricate one, or use an existing one as a mental model with my own variations. But to save time and possible mistakes I might want to find some source of a "proven" design with detailed information on achieving the goal of a roll top desk. Well, in the world of wood crafting, as in the world of photography, the world of live steam model engines, or the world of backpacking, there are numerous sources of information including ready made designs and techniques. I refer of course to books which are just published products with specific orientation or theme.

Similarly, when I have a computer system and I know of some neat language or software development tool exists, I also know that in principle I could write such a package myself using my own design or general design concepts taken from any number of existing computer science textbooks. But I'd really like to somehow buy the design in a completely documented form so that like the plan of the hypothetical roll top desk I could implement it literally with custom modifications. Tutorial and "how to" plans books for specialized fields such as those mentioned above are widely available already, and at prices well within the range of an individual's budget. They are marketed in large quantities because large numbers

of individuals use the information; outlets range from mail order book services to retail stores. Drawing out of this parallel between individualized computing and individualized "anything else", it should be obvious what the solution of the software dilemma is: publish detailed plans and tutorial information for software, on a scale commensurate with the size of the market. Publishing ideas is an activity which has a long and distinguished history, and yields both personal and financial rewards those who engage in the practice, as well as real benefits to those who purchase the products. Let's turn now to the application of this concept to the software designs of the computing world.

The Ideal Model Of A Published Software Product

When we talk about publishing software at the present state of technology, we are talking about a product which is akin to the detail design of the roll top desk mentioned earlier. It is a product which serves as the starting point for the home software craftsman, not a recipe which will fit without thought into every conceivable system. This will change a bit as the systems in the marketplace become more refined, but the nature of the computer as an intellectual amplifier tends to require a certain level of technical familiarity on the part of its user. (This is the element which distinguishes the general purpose computer from the applications oriented dedicated computer such as a four function calculator or oven controller.)

In order to make a software package which is optimally configured for the customer's standard or customized use, there is a certain minimal level of documentation which is required. This level of documentation is not necessarily needed by all users all the time, but is in many respects akin to the reference books for integrated circuits: when a question needs to be answered, it is good to have the information needed to zero in on the answer. Here is what I consider to be adequate documentation:

- Users manual textual materials concerning the "standard" uses and limitations of the software. Here is where we find such information as standard IO patch points, relocation tables, etc.
- Complete object code, preferably machine readable along with machine readable relocation information.
- Complete source listing of the package including source language and generated object code for each statement.
- Program logic manuals and tutorials on the design of the product are an excellent option.

The idea is to include enough information to allow the user to do routine field alterations, including relocation. The idea of a published software product is to compile all this information together in a comprehensive book form, to be sold at prices characteristic of books as opposed to the past history of software prices for applications and system software packages. The technology of printing covers all the portions of the "complete" package except machine readable code, at least in the minds of most people. However, as we have demonstrated with experiments published in BYTE, printing technology also covers machine readable representations as well.

Varieties of Machine Readable Representations

User convenience demands that a software product be made available in some form of machine readable representation. While it is certainly possible to take an object listing in printed form and type it into a processor by hand, this is a long, tedious and error prone process. To complete the functional definition of "adequate documentation" given above,

we need a form of machine readable object code at minimum, along with machine readable relocation information. Fortunately there exist several technologies which can be employed for this purpose, which I'll review here:

ROM Releases

This is the most expensive medium presently available for reproducing software, however it has utility in the convenience of use provided by built in software. In terms of practical products, however, this form of software will most frequently show up in manufactured products preloaded at the factory, rather than user integrated software. A ROM program is difficult to achieve in a relocatable form, and has a certain permanence which is both its advantage (convenience) and its disadvantage (difficulty or impossibility of patching.) This method has been successfully employed in several desk top calculator packages in the form of ROM software options, and in personal computing products as built in BASIC interpreters and monitors. For end user markets, this form of software can dispense with all but the user oriented manuals in most purchases, since modification is impossible.

Magnetic Digital Media Releases

As more and more floppy disk products, large and small come to market, the use of the magnetic diskettes for software releases is becoming common. Similarly, the digital tape Philips and 3M cassette media with standard digital recording techniques can be considered as vehicles for release of machine readable data. However, the price of the media and the costs of digitally recording and verifying each copy tend to make this method of delivery limited. It is used, quite naturally, as the vehicle for delivery of floppy disk operating systems from the manufacturers of drive interfaces, but there the writing and testing of a diskette full of data falls out of the expected quality assurance tests prior packing and delivery. The same is true for the other forms of hardware which have related operating system software products that can be recorded.

Paper Tape

This venerable medium has been in existence longer than the modern electronic computer. No survey of distribution media would be complete without mention of it. Very reliable means exist for reading paper tapes into computers at quite economical prices, well under \$100 for the peripheral. As a distribution medium, however, paper tape in my opinion suffers from several disadvantages: it is inconvenient to store, bulky, prone to create a messy tangle due to manual handling with inexpensive peripherals. Whether my opinion is supported in the marketplace is another question altogether.

Audio Media Releases

One of the most useful and practical vehicles for the distribution of software is likely to be the use of audio recording media. Here we can identify two principal methods of distribution, recording on tape cassettes or other magnetic tape audio media; and recording on the audio equivalent of a read only memory, the phonograph record.

The technology of recording on tape, or in record form, results in a product with a fairly high unit cost for each copy of the information. To this must be added the cost factors associated with normal printing of the rest of the documentation. Cassettes, as a recording method, are a logical choice for custom software, or small volume situations, but the high degree of manual labor associated with each copy argues against the practicality of large production runs in this form.

The technology of making phonograph records, is on the other hand a well established mass production technique which can be adapted to the software distribution without much variation from standard methods. To illustrate the point and to test the concept, I made a test in the spring of 1976 at the suggestion of David Fylstra, an associate of mine who is also a homebrew record maker. He arranged for the cutting of a test record with the audio format of my personal monitor program circa March 1976, to test out this method using a master record cut on standard recording industry equipment. Depending upon size and quantity of pressing, the costs per record run well under \$1, which is hard to meet with the cassette duplication method.

Machine Readable Printed Media

As a final option, there is the use of machine readable printed formats for object code and relocation information in the release of software products. This is a form which was suggested to us at BYTE by Walter Banks of the University of Waterloo, and with which we have been experimenting at in the pages of BYTE. In this method, an optical reader is used to scan printed materials which have been formatted into a series of bars corresponding to the digital information. Because of constraints in the design of the layout and the method of scanning, it is possible to simplify the scanner designs to the point where a very inexpensive peripheral is used together with some adaptive software which takes care of the speed tolerant input scan. The beauty of this method is that it "comes for free" in so far as actual production costs are concerned. Why is this true? The reason is that the 200 to 300 pages of documentation needed to support a systems software product with perhaps 12K bytes of object code require only an additional 5 to 7 pages of machine readable bar code copy, hardly affecting the economics of the book at all. This 200 to 300 pages of documentation is required by the product concept, whether or not there is any other form of machine readable code made available.

Economics of Publishing

With media established, and a product concept outlined, what about addressing the problem of rewarding the producers of software products? Here, as in any area of publishing, the answer is quite simple. The publishing house judges whether the particular software package is in its view a readily marketable product with a certain minimum press run potential and puts up production capital, where the author puts up intellectual capital in the form of his work. It is a risk situation in which both parties are making a speculation that readers will purchase the product; as in numerous parallel situations throughout industry, authors and publishers work on an agreed upon split of any rewards from success in the market place.

Applied to the software publishing variant of this business, the author's intellectual capital is in the form of the program, its source code, its object code, and his documentation; the publisher's contribution is the marketing organization, the technical editing of the manuscripts, and the technical details of book preparation. Other than the specialized content, the method of operation and the details of the arrangement are not much different from publishing any item. Rewards to authors now become a small amount (in absolute terms) of royalty recovered from orders of magnitude larger sales for successful software book products.

Proprietary Products

The problem of protecting and keeping software proprietary no longer is a major "new" issue when publishing of software is contemplated. How many people extensively copy from books? Very few, and if they attempt to make a regular practice of it they would tend to be prosecuted by publishers under copyright law. In publishing software, an implicit or explicit license to copy the copyrighted materials for personal use and modification is part of the bargain; the price is low enough so that if you want your own user documentation, you buy your own copy of the book. Even if you may have been using object code derived from your neighbor's computer. Since the documentation is a necessary component of use, no sales tend to be lost in the long run due to the fact that object code can be swapped around.

Conclusions

What I have endeavored to show is that there is quite some potential for the sale and distribution of software using conventional publishing techniques with modifications to suit this type of product. By publishing software along with machine readable code whether in printed or magnetic form, we end up with a way to make the products widely available, yet retain the desirability of compensating authors for their efforts in proportion to the success of the product.

TAX ASPECTS OF LEMONADE STAND COMPUTING: WHEN IS A HOBBY NOT A HOBBY?

Kenneth S. Widelitz, Attorney at Law, WA6PPZ
10405 Louisiana, No. 8
Los Angeles CA 90025

The Internal Revenue Code allows benefits to taxpayers who are engaged in an activity for profit as opposed to those who are not engaged in an activity for a profit (a hobby.) For example, by qualifying an activity as one engaged in for profit, the microcomputer entrepreneur will be allowed an investment tax credit equal to 10% of his investment in his microcomputer and other assets used in the activity. Further, such assets can be depreciated. It is also possible to deduct a portion of apartment rental or mortgage payments if you operate a lemonade stand microcomputer business out of your residence.

Internal Revenue Code (IRC) Section 183 is commonly known as the "hobby loss" section and defines when an activity is not engaged in for profit. Actually, there is no reference to "hobby" in the IRC. Section 183 defines an activity not engaged in for profit as one other than an activity that is engaged in for profit. Such a definition is typical IRC language and not much help.

However, Section 183 does provide a presumption that if the gross income derived from an activity in two or more taxable years in a period of five consecutive years exceeds the deductions attributable to such activity, such activity shall be presumed to be an activity engaged in for profit. In other words, if you can make money in two of five years, there is a presumption that you are engaged in an activity for profit.

If you do not turn a profit in two of five consecutive years, your lemonade stand application may still qualify as an activity engaged in for profit. In such a case, there are a number of factors which are considered by the IRS in determining whether or not an activity is engaged in for profit. Among those factors are the following:

1. The taxpayer's entire history of income or losses with respect to the activity;
2. The amount of occasional profits, if any, which are earned;
3. The cause of the losses;
4. The success of the taxpayer in carrying on other similar or dissimilar activities;
5. The financial status of the taxpayer;
6. The time and effort expended by the taxpayer in carrying on the activity;
7. The manner in which the taxpayer carries on the activity;
8. The expectation of profit by the taxpayer;
9. The elements of personal pleasure or recreation.

The above factors are considered on a case by case basis, with no single one determinative.

If your lemonade stand application is determined to be one engaged in for profit, then you are entitled to substantial tax benefits. Of foremost interest to the microcomputer user is the investment tax credit. An investment tax credit equal to 10% of the purchase price of tangible personal property which is used in a trade or business is allowed. For example, if you purchase a microcomputer and peripherals for \$5,000, you are entitled to an investment tax credit of \$500 in the year of purchase. The \$500 is a credit and not a deduction. It is subtracted

from the tax liability after the tax liability is determined.

In order to qualify for a full investment tax credit, the asset on which the investment tax credit is taken must be an asset which has a useful life of seven or more years. A seven year useful life for a microcomputer is reasonable and falls within the asset depreciation range guidelines for useful lives established by the IRS.

A taxpayer, if he elects a seven year life, may depreciate the microcomputer over that seven year life. The amount of such depreciation is deducted from income in order to determine the tax liability (as opposed to the investment tax credit which is subtracted once the tax liability is determined).

There are a number of methods to determine the amount of depreciation in a given year. The simplest is straight line depreciation and for purposes of example, the straight line method will be used. For example, a microcomputer is purchased for \$5,000. It has a useful life of seven years. At the end of seven years it will have a salvage value of \$1,000. Depreciation in the amount of \$571.43 per year (\$4,000 divided by 7) may be taken.

The cost of software, may be deducted currently as an expense. However, if software is purchased as a package with hardware, it must be depreciated over the life of the hardware.

Other lemonade stand expenses which may be deducted include telephone, trade journals (Dr. Dobb's Journal, etc.) postage, stationery, mileage at 15¢ per mile, wages of employees, if any, and perhaps an amount for a portion of your residence.

To deduct as an expense a portion of your residence costs, the portion of the residence to be deducted must be used exclusively and on a regular basis as the principal place of business. If it is an apartment, a pro rata percentage of rent, electricity, gas, etc. may be deducted. If it is a home, the formula is a little bit more complicated because some expenses such as interest on your mortgage and real property taxes are already deducted.

It should be noted that the lemonade entrepreneur may not wish to take an aggressive stance on his deductions, such as deducting a portion of residence or using accelerated depreciation. The reason is that by taking an aggressive position, the taxpayer might bootstrap himself right out of the presumption that he is engaged in a business for profit. That is, his additional deductions may cause deductions to exceed income.

If an activity is deemed to be one not engaged in for profit, there are some deductions allowed. Those deductions are ones which are allowed without regard to whether or not an activity is engaged in for profit (i.e., interest, taxes.) Additional deductions may be taken equivalent to deductions which are allowed for activities engaged for profit, but only to the extent that gross income exceeds the deductions allowed without regard to whether or not an activity is engaged in for profit. If an activity is not engaged in for profit, the investment tax credit is not allowed.

STUDY OF THE EMERGING CONSUMER COMPUTING MARKETPLACE

Walter Smith, Director of Marketing
 Quantum Science Corporation
 1100 Alma
 Menlo Park CA 94025

Abstract

The emerging consumer computing marketplace is being driven by three concurrent technologies - computer, video, and communications. A full understanding of this market requires consideration of developments which are occurring simultaneously in each of these fields, and which will lead over the next 8 years to development of discrete products, then subsystems, and finally a total interrelated consumer entertainment and information system. The potential market is enormous, and is attracting major corporate product development efforts among the largest computer, communications, semiconductor, and consumer products companies. Early examples of discrete products are video games, home video cameras, home computers, and microprocessor equipped microwave ovens. Early subsystems are appearing, and a study currently underway by Quantum Science Corporation will analyze and forecast the evolution of these subsystems, and the timing of new products to form the total eventual system.

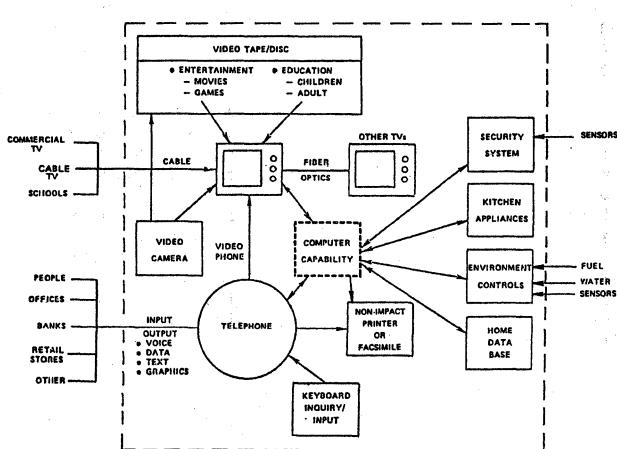
Developments in areas of satellite communications, fiber optics, and non-impact printing will greatly expand the communications capability of consumer systems. This expansion will require an understanding of interfaces, and an appreciation of the total information marketplace. Increasingly, the marketplace will be dependent on systems rather than products, and these systems will provide opportunities for entrepreneurs as well as the largest corporations.

STUDY OF THE EMERGING CONSUMER COMPUTING MARKETPLACE

Quantum Science Corporation, a leading international market research and consulting company is engaged in a major study of "Consumer Entertainment and Information Systems." This study is being funded by leading corporations, and will address the impact of electronic technology on consumer markets through 1985. Impact of digital systems evolving from computer, video, and communications technologies will create a major market opportunity, which is attracting substantial research and development attention among leading edge companies.

The focus of this conference is obviously on computer-driven products. Early examples are the hobbyist computer, the small educational computer, video games, and appliances with microprocessors built in, such as television sets and microwave ovens. Significant developments are also taking place in the video area, with products such as the Betamax Video Recorder from Sony becoming a very successful product and with home video cameras already on the marketplace at less than \$400 each. Currently, in the communications area, networks are being constructed which will carry, in digital form, not only voice but text, graphics, data and video. Ultimately, a system will evolve which appears schematically in Figure 1.

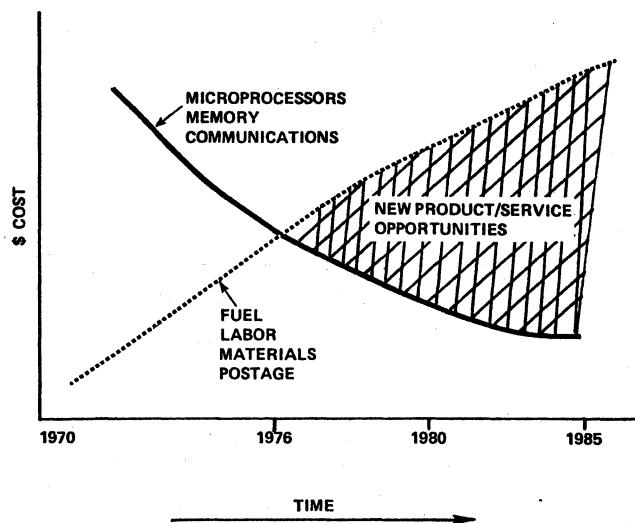
FIGURE 1
 PUT IT ALL TOGETHER TO FORM
 THE TOTAL SYSTEM



This total system will be funded by consumer expenditures for items such as housing, furniture,

electricity, gas, telephone, recreation, private education, etc. Currently, these expenditures exceed \$400 billion per year in the United States. Just how much of this total will be expended for electronic equipment depends on many factors. A strong driving force is the constantly reducing cost of computing, memory, and communications capability, concurrent with the increasing cost of labor, supplies, postage and other elements of the mix. In Figure 2, a cross-over point occurred in approximately 1976 when new product opportunities became economically feasible. Early examples are video games and hand-held calculators. These will be followed by a whole family of products. A key issue relates to the timing, size, price, and competitors in each of these equipment markets.

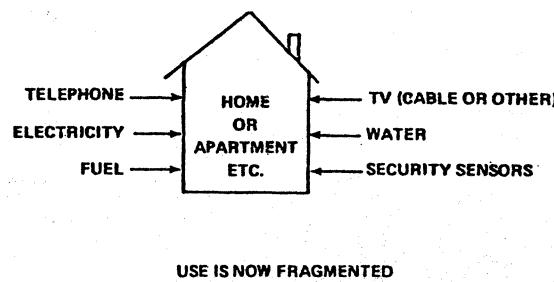
FIGURE 2
 CHANGING COST RELATIONSHIPS CREATE PRODUCT OPPORTUNITIES



As shown in Figure 3, the home currently receives several inputs. These inputs are currently fragmented, and the systems of the future will integrate these inputs, much as a computerized process control system in a petroleum refinery currently integrates similar inputs. This integration can be seen taking place in three subsystems: Entertainment and Education, Computerized Information and Control, and Communications Interface.

FIGURE 3
 NEED FOR THIS STUDY

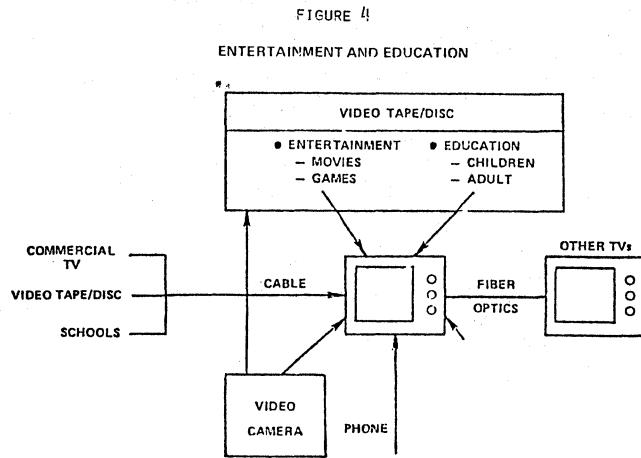
THE HOME CURRENTLY RECEIVES INPUTS



USE IS NOW FRAGMENTED

As shown in Figure 4, the Entertainment and Education subsystem is made up of currently discrete products. Actually, Sony currently markets a major portion of the subsystem, including a camera, a videotape recorder, and a television screen. Fiber optics

and other new techniques for carrying broadband communication will allow linking of several screens. As shown in the diagram, schools become part of this subsystem. Early products are video games and electronic musical instruments.

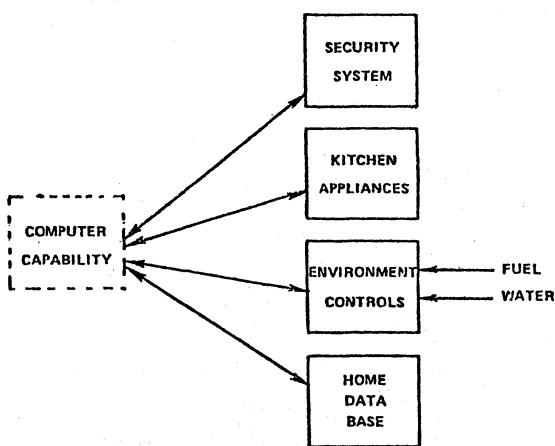


EARLY PRODUCTS:

- GAMES
- ELECTRONIC ORGANS
- SONY VIDEORECORDER
- HIGH SCHOOL EDUCATION VIA VIDEO
- SECURITY SCAN

The computerized information and control is a major subject of products being discussed in this conference. In Figure 5, the "computer capability" is shown as a broken line, reflecting that this capability may either be centralized or distributed, and most probably will be a combination of both. A major economic driving force of great interest in industry is the potential for energy conservation through improved environmental controls. There is also significant activity in electronic security

FIGURE 5
COMPUTERIZED INFORMATION AND CONTROL



EARLY PRODUCTS

- AMANA TOUCHMATIC RADARANGE
- COMPUTER FOR HOBBYISTS
- SMOKE DETECTORS
- COMPUTERIZED TAX SERVICES

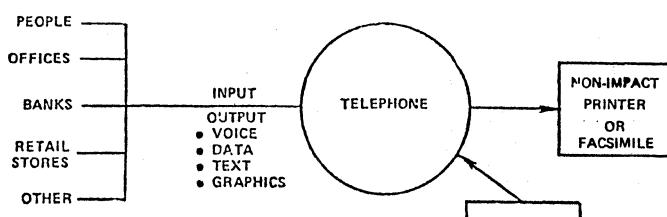
systems which will tie sensors to computer and communications capability to law enforcement agencies. Early products include the hobbyist computer, the Amana Touchmatic Radar Range, and sensors to detect smoke, soil moisture, and the like.

The final subsystem that is emerging is the Communications Interface. As shown in Figure 6, the telephone increasingly functions as a link between the home and the outside world. The enormous quantity of digitized data being developed in electronic funds transfer and point-of-sale systems will form a basis for electronic shopping and banking at the consumer level. Also, as the printing process becomes increasingly electronic, large quantities of text and graphics information will be transmitted over telephone and other communications lines to be outputted in the home over electronic printing (mostly non-impact) techniques. The recently announced IBM Ink Jet Printer is one kind of device which will provide hard copy output in the home. Interesting potential competition is also forming as cable TV companies become alternate information channels to the telephone companies in certain areas. Television, with its capability to carry broadband communications, has high potential in total, integrated information systems.

As the systems discussed previously actually are implemented, they will take many forms. For example, companies may develop an "office in the home" utilizing portions of several of the above subsystems. Also, medical care in the home may develop an entirely different combination of elements, but the basic concept will apply - it will be increased electronic content, the subsystems will interface as part of a total system, and the cost of these new systems will become increasingly attractive with time. Industry experts are currently predicting that, by 1985, single chips will have computing power comparable to the most powerful computers today. Obviously, this enormous capability will be implemented in many forms in industry, and as discussed above, at the consumer level.

In carrying out the current study, Quantum Science Corporation will interview current participants in the consumer computer market, potential participants, government agencies, and leading laboratories. This analysis will be integrated with research on current consumer attitudes toward entertainment, education, information, and computerized control in the home. The results of this analysis will be published for clients in the first half of 1978. One point however, is already clear - the market for consumer entertainment and information systems is larger than most people realize. In a meeting I personally held with one of the leading retailers in this country, the Director of Planning called the market "the

FIGURE 6
COMMUNICATIONS INTERFACE



EARLY PRODUCTS:

- SIMPSON-SEARS ELECTRONIC SHOPPING
- AUTOMATIC TELEPHONE ANSWERING
- VOICE RESPONSE SYSTEMS FOR UPDATING TELEPHONE NUMBERS
- INK JET PRINTER

most significant new consumer market of this century." There is room in this market for large and small companies with the winners being those who can bring the right product to market at the right time, and grow that product as it evolves to become part of the total system.

Walter P. Smith is Director of Marketing for Quantum Science Corporation at their Menlo Park, California location. He currently has project responsibility for the initial phase of the "Consumer Entertainment and Information Systems" multi-client study which is the basis for the above presentation. Prior to joining Quantum Science, he held marketing and executive positions at several companies, including IBM where he was Manager, National Account Marketing. He holds a degree in Engineering from Syracuse University, and a Masters in Business Administration from Harvard University.

SPEECH RECOGNITION SYSTEMS

John Reykjavik & Horace Enes
Heuristics, Inc.
900 N. San Antonio Road-C1
Los Altos CA 94022

Human speech is a natural and desirable method of interfacing human beings with computers. Its advantages are immediately obvious. It uses the most natural and widely used form of human communication and raises the computer's ability to that of the human being rather than reducing the human capability to the mechanical form required by the machine. Computer speech recognition has been an active area of computer science research for at least twenty years and is an active research area today. It is largely an unsolved problem.

As late as 1969 a widely held opinion was that all funds for speech recognition research be cut off, since the problem was seen as unsolvable. Even if the problem could be solved in a multimillion dollar laboratory, it was maintained that it would never be an economically feasible device for real world application.

State of the Art

Technological advances and continued research in speech recognition have proved this opinion wrong. Research institutions and universities are continuing their work in speech recognition. Continuous speech recognition is being attempted, vocabulary size is being expanded, and speaker independent systems are in existence. This research is generally accomplished with an impressive array of computer horsepower and the response time required to solve the more difficult problems is often much greater than real time. Several commercial systems are presently available, with vocabularies of 16 to 40 words with recognition rates in the high 90 percent range, and real time response. These systems operate with isolated words, that is, single words with gaps of silence between words. By taking advantage of current LSI electronic technology the price of these systems has been rather dramatically reduced from the \$40,000 and up range to \$10,000 and up.

The same technological advances that have reduced the price of commercial speech recognition systems have made the personal computer a reality. Today's personal computer owner has as much computing capability as many of the speech recognition researchers of the late 1960s and early 1970s. Now a personal computer speech recognition laboratory is available which achieves the performance of the very expensive research laboratory systems of the late 1960s and early 1970s, and performs as well as presently available commercial units, at a modest fraction of the cost of the commercial system. Personal computer users may expect recognition rates in the mid 90 percent range on isolated words, with a 16 word vocabulary operating in real time. These systems provide the personal computer user the opportunity to delve into a fascinating area of computer science research and to devise his own original solutions to the problems of speech recognition.

Personal Computing Speech Recognition Systems

Today's personal computing speech recognition systems are designed to recognize isolated words, or words with at least 100 milliseconds of silence separating the individual utterances. These systems typically operate in two modes. In the first mode the user "trains" the speech recognition system by supplying an example utterance of each word the system will be expected to recognize. After filing away data extracted from these examples, or "templates" as they are sometimes called, the system is ready to operate in the recognition mode. The personal computing systems take advantage of some hardware pre-processing to extract the information which comprises the template and the information representing the unknown utterance.

Hardware Preprocessing

One of the most fascinating problems in speech recognition research is to determine what features of a speech wave form carry the information of the utterance. Assuming that a 10 KHZ audio spectrum adequately spans the human voice and that 12 bits are adequate for faithful reproduction of the wave form, $2 \times 12 \times 10^4$, or 240,000 bits/sec. of information will adequately describe the acoustic pressure wave generated during human speech. Telephone quality speech has only a 3KHZ spectrum with perhaps 8 bits required per sample,

for a data rate of 48,000 bits/sec. On the other hand, the information contained in the utterance may be represented by about three five-letter words per second. Representing each letter with a six bit code which includes a blank character for the spaces between words would require 108 bits/sec. to characterize the meaning in a typical speech utterance. It is obvious that the information in the acoustic pressure wave form is highly redundant, but it is not so obvious which features of the speech wave form contain the information. However, since the typical personal computer cannot effectively process 240,000 bits/second of information in anything approaching real time, some pre-processing or data extraction is necessary.

One way to preprocess the input speech wave form is to extract the amount of energy in the input wave form contained in each of several frequency bands. This amounts to a crude bandpass frequency analysis of the wave form. The typical personal computing speech system uses three bandpass filters, with bandpassing corresponding the first three resonances (termed formants) of the human vocal tract. Figure 1 is an exam-

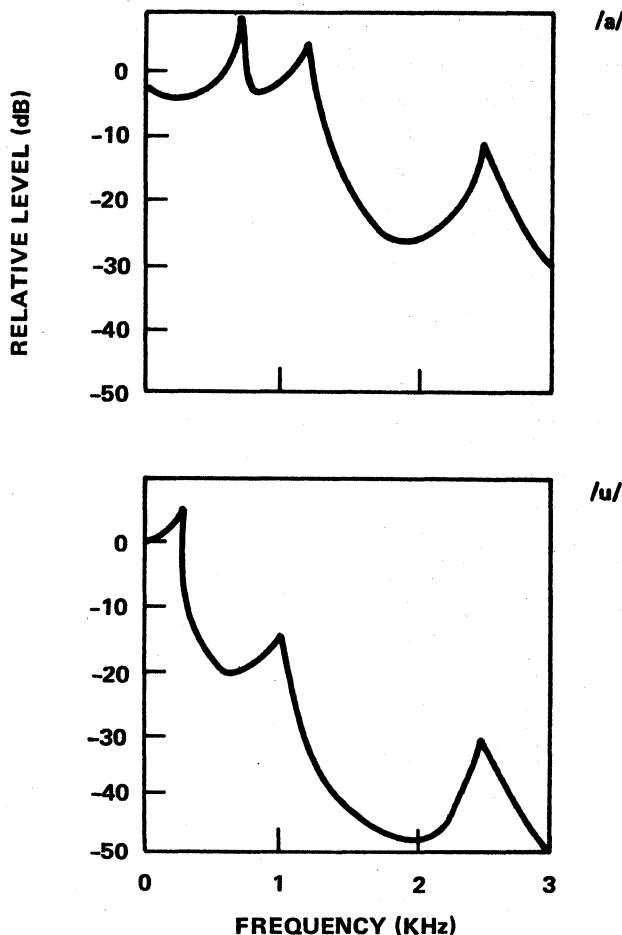


FIGURE 1

ple of the energy versus frequency plots for three vowel sounds, with the formant frequencies indicated by the sharp peaks in the spectrum. These frequency ranges were chosen on the assumption that since the vocal tract resonances are a function of the position of the various elements of the vocal tract, the amount of energy in each of the resonances is a clue to the vocal tract shape and, hopefully, to the actual utterance. Since the vocal apparatus doesn't change shape very quickly, the amounts of energy in each of the resonances

also change relatively slowly, and sample rates of 100/sec. are sufficient to characterize the energy in each of the three bandpass filter outputs as a function of time.

One additional parameter of the speech input is also measured, the dominant frequency of the speech utterance. A time averaging zero crossing counter applied to the input wave form yields the number of times the periodic speech wave form crosses its average level and hence is a measure of the dominant frequency.

Six bits of information are adequate to characterize all of the parameters measured by the bandwave pre-processor. With four quantities being measured at 100 times a second each, the data rate from the bandwave pre-processor to the computer is 2,400 bits/sec., well within the processing range of typical personal computers. This data is input to the computer memory where it will be available for further processing by the software recognition algorithm. Figure 2 shows the block diagram of a typical personal computer speech data bandwave pre-processor.

Computer Processing

After the speech wave form has been transformed to a trajectory in the four dimensional space of bandpass filter energies plus zero crossings, the representation of the speech wave form in this domain must be further processed by the personal computer before speech recognition is feasible. The software speech recognition algorithm must somehow compare the parameters obtained from an unknown utterance with those stored away during the training mode. To do this, the algorithm must determine the beginning point and endpoint in time of the speech utterance, since there will be only silence and background noise preceding and following the utterance, and this data does not contain information about the utterance. To be effective, the software algorithm must compensate for the differences in volume, or amplitude, between utterances of the same word. It must also compensate for faster or slower utterances of the same word. To avoid storing large amounts of data to represent each word, the algorithm may further condense the information which arrives for the pre-processor and may then be required to interpolate between data points in this relatively sparse data domain. Finally, the algorithm must embody an effective means of measuring the "differences" between utterances.

Determining Beginning and Ending of Utterance

The bandpass filter technique greatly simplifies the problem of determining the beginning and ending of an utterance. Summing the output from the three bandpass filters and comparing this result to a threshold greater than the sum generated by background noise is usually sufficient to accurately determine the beginning of a speech utterance. More sophisticated algorithms may start from this point and search backwards in time by examining the data from the zero crossing detector to determine if the utterance started with a low energy, high frequency fricative such as (S).

The determination of the endpoint of the speech utterance uses a similar technique. When the summed energy from the three bandpass filters falls below a threshold, the endpoint of the utterance has been found. Exceptions to this rule are single words which contain a period of silence, or "stop" as it is called. Examples of words with full stops are the digits six and eight. Utterances with stops contain a period of silence usually less than 100 milliseconds in duration. The personal computing speech recognition systems op-

erate on isolated words which, by definition, are separated by a period of silence greater than 100 milliseconds. Therefore, the speech recognition algorithm can determine if a period of silence indicates the end of a word or indicates a stop within a word by examining the length of the silent period.

Time Normalization

Once the beginning and ending of the utterance represented in the speech buffer data memory have been determined, the algorithm may proceed to operate on the actual utterance. One of the first tasks the personal computing speech recognizer does is to normalize the length, or duration, of the utterance. There are many methods of accomplishing this task. One simple but effective method is to divide the utterance into an equal number of parts, typically 16, and sample each of the four bandwave pre-processor outputs at each of the sixteen points. Two speech utterances which are identical in all respects except duration, will thus be represented by the same parameters after this sampling technique is finished. This technique has the added advantage of further reducing the amount of data representing an utterance to 16 X 4 or 64 six bit words per utterance. This reduced representation of the speech utterance is called a template,

Interpolation

The difficulty with the preceding time normalization algorithm is that the desired sampling period, the length of utterance divided by sixteen, is typically not an integral multiple of the sampling period of the bandwave pre-processor. Recall that the bandwave pre-processor sampled the outputs of the bandpass filters and the zero crossing detector 100 times a second, or every 10 milliseconds. This data was placed in memory where it was made available for further processing by the software recognition algorithm. The data is discrete representation of a continuous time wave form. Suppose one utterance of the word X is 320 milliseconds in duration. Each of the sixteen parameters extracted from this wave form will be 320/16 or 20 milliseconds apart. The time normalization algorithm will then simply extract every other sample from the input data buffer between the beginning and end of the word to form the template representation of the word. Suppose, however, that the next utterance of the word X is only 300 milliseconds in duration. The time normalization routine will divide the duration by 16 and discover that it wants to sample the speech utterance every 19 milliseconds. Since the discrete representation of the pre-processed wave form contains data taken every 10 milliseconds, most of the required 19 millisecond interval sample points lie "in-between" the available 10 millisecond data points. At first glance, the programmer may be tempted to round the 19 millisecond interval up to 20 milliseconds and again extract every other data point from the area of the input data buffer containing the utterance. However, the duration error from each sample point accumulates, and samples extracted from the end of the speech utterance may be significantly misplaced relative to the wave form. The obvious solution is to interpolate between data points in the input data buffer, and generate a linear approximation to the pre-processed speech wave form at sample points which lie in between data points. Since the interpolation routine requires a multiplication and must be applied to every speech utterance, the interpolation is used at some computational expense, especially in higher level interpreted languages such as Basic. The improvement in recognition accuracy made possible by this technique warrants its use.

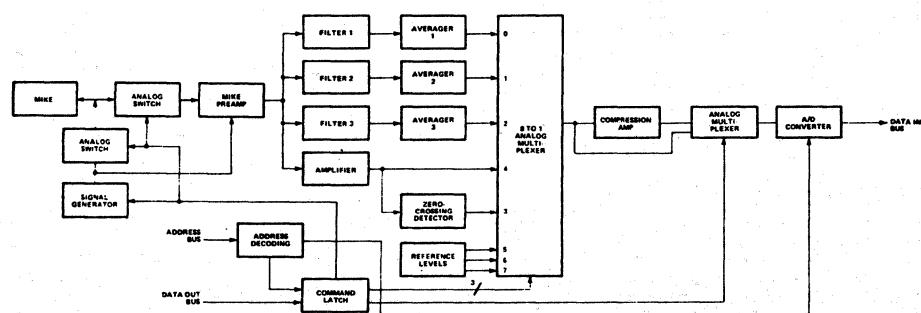


FIGURE 2 - BLOCK DIAGRAM

Amplitude Normalization

The single largest variable in personal computing speech recognition systems is the variation in amplitude, or the volume level, from utterance to utterance. This is due to the natural variation in loudness in human speakers, and to the inconsistent manner of holding the microphone in front of the mouth. Small variations in the distance between the microphone and the mouth produce large variations in the amplitude of the electrical signal from the microphone. Automatic volume control devices have been studied and have produced mixed results. Commercial and research systems largely avoid this problem by using close fitting head mounted microphones similar to those worn by telephone operators which hold the microphone a fixed distance in front of the mouth. The \$70 to \$100 price range for these microphones has been a severe impediment to their acceptance in personal computing systems, however. The personal computing speech recognition system solves this problem by a software amplitude normalization routine.

There are several methods of normalizing the amplitude of the template extracted from the input data buffer by the previous software routines. One successful approach is to determine the average amplitude of the utterance and to add a constant to all values in the data buffer such that the resulting average is zero, or some other convenient constant. An alternative approach is to determine the average amplitude of the utterance and generate a multiplicative scale factor to apply to all elements of the data buffer to again generate a constant reference amplitude. Both schemes work well, but the former is generally employed to save the computational expense of multiplying.

Distance Measures

The previous algorithms are applied to the reference utterances input to the system during the training mode to generate the reference templates. They are again applied to the unknown utterances during the performance mode of operation to generate a template of the unknown utterance. All that is required now is to determine a measure of the similarity or dissimilarity between the unknown template and all the known templates comprising the vocabulary of the system.

The most straightforward method is to subtract the unknown template from the known templates and accumulate the absolute value of the differences. This is done band by band, that is, the data from each of the bandpass filters in the unknown template is subtracted from the corresponding bandpass filter in the vocabulary templates. The vocabulary template which accumulates the smallest sum of absolute values of the differences is deemed the match. This distance measure is called the Chebyshev norm.

A second method is to consider an utterance to be a trajectory in the four dimensional space whose dimensions are the three bandpass filters and the zero crossing detector outputs. The Euclidian distance measure, familiar from geometry, is extended to four dimensions by taking the square root of the sum of the squares of the differences between the four unknown utterance bands and the vocabulary templates. This measure accentuates the sum of the distances since each individual point by point difference is squared.

The distance measures described previously cannot only be used to determine the closest match in the vocabulary but can also be used to predict the confidence that the match is a correct one. When the distance measure between the unknown utterance and the best match in the vocabulary is about three times smaller than the distance measure between the unknown utterance and the second best match in the vocabulary, the algorithm can report with great confidence that it has found a match. When the ratio of distances between the best and second best match is close, the sophisticated algorithm may choose to ask for a second example of the unknown utterance, in hopes that the first one was mumbled or marred by background noise.

The algorithms described here are but one possible way of accomplishing speech recognition. With the advent of the personal computing speech recognition laboratory the number of people actively engaged in speech recognition research will increase by an order of magnitude, and the number of different backgrounds brought to bear on the problem will likewise increase. The home computerist is as well equipped as the university researcher of just a few years ago, and there is nothing to prevent him from making significant contributions to the field of speech recognition research.

SPEECH SYNTHESIS BY A SET OF RULES (OR, CAN A SET OF RULES SPEAK ENGLISH?)

D. Lloyd Rice
Box 1951
Santa Monica CA 90406

ABSTRACT

A description of a speech synthesis by rule system is presented which generates control parameters for a formant frequency synthesizer such as the Computalker model CT-1. The first 3 parts of the system are described in some detail, especially the rules themselves. The rules portion is described completely, with discussion of the phonetic effects of each rule. The fourth part of the system is the actual playback of the parameter data to the synthesizer. The output of the rule system is then compared with a similar set of data recovered from a human pronunciation of the same utterance. Similarities and differences between the two versions are discussed.

INTRODUCTION

Human speech is a complex and variable process. Every time you say something, it comes out a little bit differently than you ever said it before. Most of this variation tells the listener something about you, how you feel, whether you're in a hurry, whether you are tense, tired or thinking about something else. In addition to these kinds of variation there is another kind of variability, determined by the phonetic context. For example, a "p" following an "s" in the same word has little or no aspiration noise following the release of closure by the tongue, while a "p" alone at the beginning of a word may have as much as 60 or 70 msec of aspiration noise.

A computer can be programmed to speak in a cut-off, robot-like manner by fairly simple joining together of fixed sounds. Speech generated in this way is generally quite difficult to understand, even in the best of circumstances. To give the computer a better quality speaking voice, on the other hand, requires a more complex system of rules which express some of the patterns of variation observed in human speech. This paper describes such a set of rules and the linguistic mechanisms in which they operate.

A BASIC SYNTHESIS BY RULE SYSTEM

A top level flowchart of the system is given in figure 1, which shows the four basic steps in the synthesis process. First, the phonetic input string is parsed into a sequence of phonetic symbols, some of which may be marked with stress level

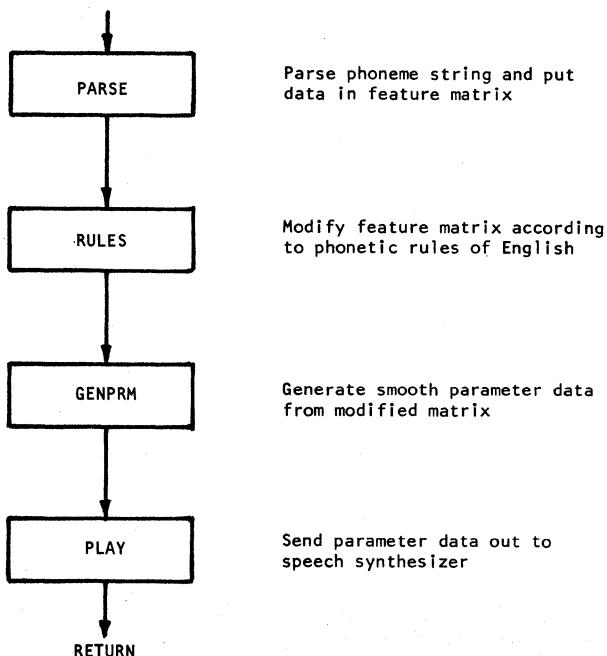


Figure 1 Synthesis by Rule system top level flowchart

values. A phonetic feature matrix is filled in by assigning a column of binary features for each of the phonetic symbols. In the second step, 3 groups of rules operate on the feature matrix, searching for certain patterns of features and modifying individual features in a column or perhaps by inserting or deleting entire columns. Rule group 1 affects only columns as a whole and in effect operates on the symbolic level prior to feature assignment. Its purpose is to revise certain spelling combinations to make the input form correspond more closely to normal spelling practices. Group 2 contains the actual phonetic readjustment rules. Rules in this group do things like deleting the aspiration of a "p" following an "s", softening a "t" or "d" sound when it comes between 2 vowels, and spreading assigned vowel stresses to adjacent consonants, which affects their durations. Group 3 rules are concerned with assignment and modification of durations to each column of the matrix. Examples are the lengthening of the last vowel of a word and readjustments to normalize the total time between stressed vowels. The third basic step in the synthesis process is the construction of smooth synthesis control parameters from the phonetic values in the feature matrix. Target parameter values for each phonetic segment are obtained from a table. These target values are then smoothed into continuous control sequences, governed by the computed durations and affected by a system of priorities assigned to each phonetic segment. The fourth and final step is the realization of the control parameters as speech output by the synthesizer. That step will not be discussed further in this paper. Reference 5 describes the synthesizer hardware and the phonetic nature of the control parameters.

THE PARSING SYSTEM

The alphabet used for the phonetic input to the rule system is ARPABET, a phonetic alphabet adapted to the ASCII character set. A complete listing of ARPABET with examples of each of the phonetic symbols is given in reference 4. The symbols are coded with either single or double ASCII characters in such a way that an unambiguous parse of a correct string can be done with a fairly simple state transition parser. Most input syntax errors can be detected.

In addition to separating the input string into a sequence of phonetic units, the subroutine PARSE builds up a matrix of phonetic information about the sequence. Each column of this phonetic matrix contains a packet of information about each of the phonetic units in the input. The rows represent 4 different items of information: First is the 6-bit code for the phonetic symbol itself; second is a cluster of 14 bits of feature information obtained from a table of feature values for each phonetic symbol; third is a location for any stress value that was specified in the input; and fourth is a location, unspecified at this time, for the duration value which is to be computed for each unit. The matrix, then, contains 5 bytes of information about each input unit. Since the primary purpose of this matrix is to contain the feature data about the phrase being synthesized, it is often called the feature matrix. At this point, it would be helpful to look more closely at these features.

Table 1 presents a listing of the set of feature values corresponding to each of the phonetic symbols used in the system. These features represent a classification of each symbol according to linguistic or phonetic categories which are important to the pronunciation of that symbol. Identifying the pronunciation categories in this way simplifies the rules needed in the system, since most of the rules operate according to the categories and would have to be restated many times with different units if they operated directly on the symbolic units. Occasionally, tho, it will be more convenient for a rule to test the unit code directly rather than the feature values. For this reason, both are retained in the matrix.

THE RULES

The only complication to be considered in the rule control system is the order of application of the groups and the rules within a group. As the matrix is scanned to test for the conditions satisfying a rule, these conditions may be met or not depending on whether certain other rules have already applied and modified the matrix. In order to guarantee unambiguous application of all rules, a particular order of application must be defined, and that order kept in mind when the rules are devised. The order of application defined here is as follows:

Within a group of rules, the matrix will be scanned once from left to right. As the scan comes to each column of

the matrix, all rules in the group are tested in the order they are stated. The right hand side of each rule is tested by aligning the first "/" mark in that rule just to the left of the matrix column currently being scanned. Whenever a rule condition is met, the matrix is immediately modified. When the application of any rule involves changing the phonetic symbol at any matrix column, then all features in the column are set as defined for the new phonetic unit. Any stress value already assigned is left unchanged.

Since there are 3 separate groups of rules, the matrix will be scanned from left to right 3 times. In many cases, when a rule condition is met at a particular column of the matrix, and that rule has been executed, it is known that certain subsequent rules in the same group cannot apply. In those cases, some execution time can be saved by judicious branching to the best point from which to continue testing.

	vowel	front	diphthong	consonant	stop	voiced	plosive	plosive aspirate	fricative	liquid	nasal	dental	boundary	ignore
space	1													
.					1	1								
,						1								
?						1								
+							1							
#								1						
:									1					
IY	1	1												
IH	1	1												
EH	1	1												
AE	1	1												
AA	1	1												
AH	1	1												
AO	1													
OW	1	1												
UH	1													
UW	1	1												
AX	1													
IX														
ER	1													
UX	1													
OH	1													
AW	1	1												
AY	1	1	1											
OY	1	1	1											
EY	1	1	1											
RX	1													
EL														
EM														
EN														
Y	1	1												
YX	1	1												

	vowel	front	diphthong	consonant	stop	voiced	plosive	plosive aspirate	fricative	liquid	nasal	dental	boundary	ignore
W	1	1												
WX	1													
W1		1												
R			1											
L			1	1										
LX	1													
WH					1									
M					1									
N					1									
NX					1									
P						1								
T						1								
K						1								
KX						1								
B						1								
D						1								
G						1								
GX						1								
DX						1								
F							1							
TH							1							
S							1							
SH							1							
V							1							
DH							1							
Z							1							
ZH							1							
CH								1						
JH								1						
HH									1					
Q									1					

Table 1 Phonetic Feature values assigned to each phonetic symbol

RULE GROUP 1

Of the three groups of rules in the system, the first group has the least dependence on the feature categories. Several of the rules in this group could in fact operate on the level of units before the features have been assigned. These are spelling convention rules. Six "dummy" symbols are defined as valid input units, which are changed into the correct phonetic sequences by these rules.

R1A: - ← /,/
R1B: AX,L ← /EL/
R1C: AX,M ← /EM/
R1D: AX,N ← /EN/
R1E: T,SH ← /CH/
R1F: D,ZH ← /JH/

In reading these rules, A ← /B/ indicates that if unit B is found in the unit sequence it is replaced by A. Note that in 5 of the 6 rules just listed, this involves the insertion of a new column into the matrix. When a new matrix column is inserted, the new feature values must of course be looked up and entered into the matrix as well as the unit itself. The "/" marks are used to bound the portion of the matrix currently open for modification.

The next rule in this group specifies that a glottal stop is to be inserted before a stressed vowel at the beginning of a word if the previous word also ends with a stressed vowel.

R1G: Q ← vowel stress>0,wordbound//vowel stress>0

In all rule specifications, capital letters indicate phonetic segments, while lower case letters indicate features or stress specifications. Commas or "/" marks separate references to different matrix columns. If the situation described by this rule is found in the matrix, a glottal stop and its features are inserted as a new column at the position indicated by the adjacent "/" marks in the rule. It is of interest to consider the large number of rules that would be needed to state this modification if they were expressed in terms of phonetic symbols rather than the feature "vowel".

The next 2 rules change the pronunciation of R and L when they occur after a vowel.

R1H: RX ← vowel/R/
R1I: LX ← vowel/L/

The final pair of rules in this group adds an end-glide to any diphthong depending on whether the diphthong is marked by the feature "front".

R1J: YX ← diphth front//
R1K: WX ← diphth -front//

These rules contain examples of combinations of features to be located in a matrix column. This can be interpreted as a bit masking problem, observing the bit polarities of the various features.

RULE GROUP 2

The rules in group 2 perform the major phonetic modifications which are necessary for the sound structure of English. The first 5 rules in this group set consonant stresses for certain consonant cluster patterns as a means of controlling the durations of these consonants. A scheme of marking negative stress values is used to communicate additional information to the duration rules in the next group.

R2A: stress=1 ← /cons/vowel stress>0
R2B: stress=-1,-1 ← /S,plos -voice stress>0/
R2C: stress=-1,-1 ← /plos or (fric -voice),liquid or nasal /vowel stress>0/
R2D: stress=-1,-1,-1 ← /S,plos -voice,liquid/vowel stress>0/
R2E: stress=-1,-1 ← /T or D,SH or ZH/vowel stress>0

Several of these rules contain multiple units within the slash marks, indicating that some modification is to occur at each of the enclosed columns providing all conditions are met. The parentheses here indicate logical operator priorities rather than optional units as in linguistic notational conventions.

Four more rules in this group modify the pronunciation of certain phonemes depending on the consonant or vowel environment. A T or D occurring between two vowels is softened to the phonetic unit known as a flap. This is a single, quick tap of the tongue indicated by the symbol DX. The following vowel must be unstressed unless it is at the beginning of the next word. The vowel UW is raised to UX when it follows a consonant in which the tongue is moved up to touch just behind the teeth. This has the effect of lowering formant 1 and raising formant 2 corresponding to a higher tongue position during the vowel. K and G are modified to alternate forms which have lower formant 2 and formant 3 when they are followed by a back vowel where the tongue must move to the back of the mouth.

R2F: DX ← vowel/T or D/(optional wordbound,vowel) or vowel stress=0
R2G: UX ← dental/UW/
R2H: KX ← /K/vowel -front
R2I: GX ← /G/vowel -front

The next 2 rules in group 2 account for the fact that voiceless aspirated stop consonants lose their aspiration either when preceded by an S or when they come at the end of a word. This is accomplished by changing them to their voiced counterparts, which do not have aspiration by definition.

R2J: add 4 to phonetic unit code ← S/plos -voice/
R2K: add 4 to phonetic unit code ← /plos -voice/wordbound

The phonetic unit codes are assigned so that the voiced un aspirated stops are located just after the voiceless stops which are detected by these rules. As with all rules, the features are changed along with the phonetic code.

The final cluster of 3 rules in group 2 has a similar

effect to the previous pair, except that here only the plosion and aspiration features are changed instead of changing the phonetic unit as a whole. This is the first case encountered where a rule only changes a few features and not the whole unit. Later on, when the features are being interpreted to generate the synthesizer control parameter data, lack of the plosive feature will result in a shorter time being allocated to the aspiration portion of the consonant, while the aspiration feature will determine how much hiss sound is produced during that interval. These rules have the following effect: first, deleting both plosion and aspiration when the stop is followed by a second stop, whether or not the following stop is at the beginning of the next word; second, deleting the aspiration when the stop is followed by WH or HH, whether or not the WH or HH is at the beginning of the next word; and third, deleting the aspiration when the stop is not at the beginning of a word and is followed by an unstressed vowel.

R2L: -plos -plosa \leftarrow /plos/optional wordbound,stop cons
 R2M: -plosa \leftarrow /plosa/optional wordbound,aspirant
 R2N: -plosa \leftarrow -(wordbound or silence)/plosa/vowel -strs

RULE GROUP 3

The rules in group 3 are entirely concerned with the computation of sentence timing structure. Concerning intelligibility of the output, this is without doubt the most important part of the entire system, altho the pitch control parameter (F_0), generated in the next step, also has an important role in the perceived quality. The initial (or target) timing structure is determined by a dual table of duration values. Each phonetic symbol has associated both a long and short duration depending on whether that unit is stressed or unstressed. One of these duration values is inserted into each matrix column depending on the stress value currently stored at that column. This initial duration assignment is considered to be a part of group 3 altho it is not a rule in the usual sense.

The first rule of group 3 is concerned with the tendency in English pronunciation to meter out the timing of unstressed syllables so that stressed syllables occur at roughly equal intervals. In other words the durations are adjusted so that the major temporal pattern moves from stress point to stress point.

R3A: between wordbound or . or silence change the duration of each stressed vowel by $(2*NV+3)/(5*NV)$ where NV is the number of vowels in between

The final vowel of a word and all units from there to the end of the word are lengthened.

R3B: dur*1.5 \leftarrow /last vowel of a word, ... /wordbound

Vowel duration is changed if followed by an unvoiced plosive, a voiced fricative, or by RX or LX followed by a consonant.

R3C: dur*.6 \leftarrow /vowel/plos -voice
 R3D: dur*1.25 \leftarrow /vowel/fric voice
 R3E: dur*.5 \leftarrow /vowel/RX or LX,cons

The duration of W, R or L, with a negative stress value and followed by a vowel is lengthened by 20 msec if it is preceded by an unvoiced plosive and the length is set to 90 msec if it is preceded by an S.

R3F: dur+20msec \leftarrow plos -voice/(W or R or L) stress < 0/
 vowel
 R3G: dur=90 msec \leftarrow S/(W or R or L) stress < 0/vowel

All consonants with negative stress values are shortened.

R3H: dur*.8 \leftarrow /cons stress < 0/

The durations of certain stop,fricative combinations are fixed.

R3I: dur=70,60 msec \leftarrow /T stress > 0,SH stress > 0/
 R3J: dur=70,50 msec \leftarrow /D stress > 0,ZH stress > 0/
 R3K: dur=60,40 msec \leftarrow /T stress=0,SH stress=0/
 R3L: dur=40,30 msec \leftarrow /D stress=0,ZH stress=0/

The durations of certain nasal,plosive combinations are fixed.

R3M: dur=30,30 msec \leftarrow /(M,P or B) or (N,T or D) or
 (NX,K or KX or G or GX)/

The durations of adjacent plosives are cut in half, even if a word boundary comes between them.

R3N: dur of plosives*.5 \leftarrow /plos/optional wordbound/plos/

It is difficult (no doubt impossible) to state a complex rule without an equally complex rule description language. In order to present the rules here without having to define an elaborate notational system, I have taken the liberty of expressing some things in English. In most of the rules which are presented here as clusters, there is some degree of commonality which should be used to advantage in writing code to realize these rules. I leave it as a problem for the interested reader to express these sets of rules most efficiently in flowchart form.

Up to this point in processing the speech sequence, the entire string had to be stored and processed as a whole because of the nature of the rule scan. These steps of building up, testing, and modifying the matrix need not occupy much memory space and should run fairly quickly. By the time this paper appears in print, the actual space and memory requirements should be available.

GENERATING THE SYNTHESIZER CONTROLS

We will now discuss in somewhat less detail the mechanisms used to generate the synthesizer control parameters. The first control function to be produced is the pitch parameter (F_0). This is the only parameter which requires any scanning back and forth in the matrix; all the rest can be generated strictly from left to right. For this reason, the F_0 parameter is handled separately from the rest, and is computed first. Once that parameter is completed and stored in the output buffer, and the remaining parameters have been computed as far as the first phonetic unit, it is at least theoretically possible to begin playing the speech data out to a hardware synthesizer using an interrupt-controlled update rate. Whether or not this is possible in reality depends on the processing speed at which the remaining parameters are being computed.

GENERATING THE F_0 PARAMETER

The plan followed in this model of the F_0 function is based on a few simple rules. The underlying principle is that each time the amplitude of voicing parameter (AV) is switched on by a voiced segment of speech, the F_0 level starts off at a mid-range value of 120 Hz. For as long as the AV control remains on, the pitch drifts slowly downward in an exponential decay fashion toward 100 Hz. Superimposed on this basic downdrift pattern are "intrusions" which raise the F_0 level in smooth upward arcs during all stressed vowels. The degree of pitch raising depends on the stress value up to maximum of 160 Hz at the peak of the arc for a vowel with stress=1. In addition, near the end of a phrase, a final rising, level or falling contour is added depending on the punctuation mark used to end the sequence, whether question mark, comma or period, respectively. A new end-point level is determined by adding 40 Hz for a question mark or subtracting 40 Hz for a period. A linear ramp is then constructed from the previously computed F_0 level at a point 300 msec before the end of voicing to the newly determined end-point level. This linear ramp is then added to the original F_0 curve. This is shown in figure 2.

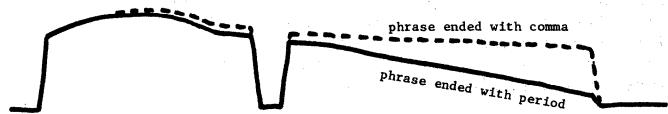


Figure 2 End-of-phrase pitch parameter (F_0) contours

It seems worthwhile to consider at this point the effect on the listener of different models for F_0 . In human speech, the pitch contour is used to communicate a very wide variety of emotional and physical facts in addition to certain linguistic information about the stress structure of the utterance. Since the software system has no access to any of these esoteric, external factors (even if we did know how to model the effects) the speech output must necessarily be rather bland in that respect. The tendency seems to be for linguistics researchers to overestimate the influence of the linguistically related variables such as stress patterns in trying to account for the wide range of variations caused by external, non-linguistic facts. In a synthesis by rule system, the influence of stress values on the F_0 parameter can easily be modified by changing

the frequencies mentioned in the last paragraph. Reducing the stress peak frequencies would give the output less of a sing-song quality. Alternatively, if so desired, the pitch output could be made completely flat by skipping this F_0 calculation entirely and putting a constant value into the F_0 parameter.

GENERATING THE OTHER PARAMETERS

The most essential of the remaining synthesizer controls are, by far, the three formant frequency controls (F_1 , F_2 and F_3). These must be relatively smooth curves during the entire phrase, at least with no abrupt jumps while the amplitude of voicing (AV) is turned on. These controls also bear the main load of determining the correct vowel quality at each instant during the utterance when there is voicing.

In order to achieve efficient coding within 8 bits and for minimum hardware complexity in the Computalker model CT-1 Synthesizer, the formant information on parameters F_1 , F_2 and F_3 is coded not as frequencies in hz, but as a reciprocal log function of frequency. Then, to eliminate complications of extra conversions, all formant information throughout the synthesis by rule system is coded using this reciprocal relationship. Keep in mind when reading graphs such as figures 3 and 4 that the formant scale is inverted from the usual formant frequency plots. Also, all three parameters are plotted overlaid on the same scale to save space. This convention requires some re-learning for those of us accustomed to reading plots of formant frequency vs. time, but it seems advisable to plot the parameters just as they are actually used in the system. In spite of this, I still think of formants as frequencies and referred to frequency scales in the preceding discussions of rule behavior.

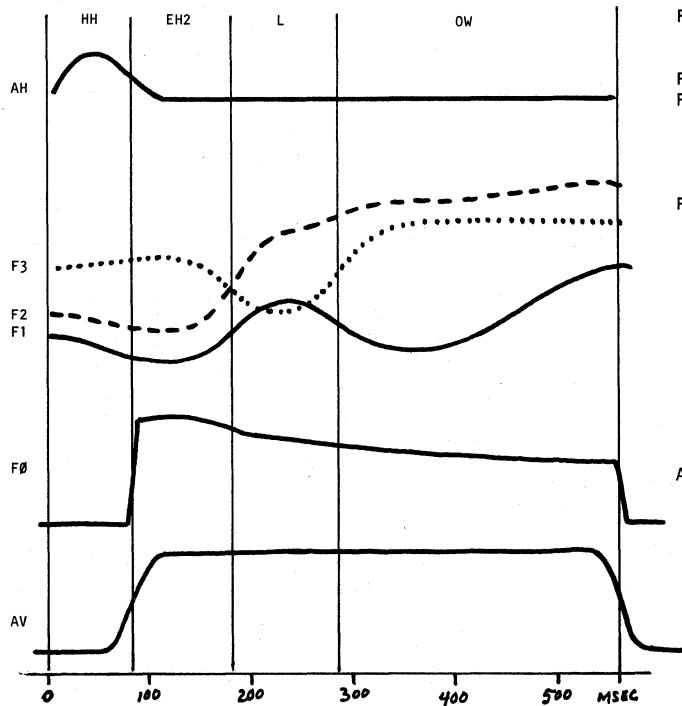


Figure 3 Synthesis parameters generated by Rule

The basic pattern for the formant frequency parameters is determined by a table of formant settings for each of the phonetic units in the system. Values from this table are taken as targets to be approached during the duration of each phonetic segment. This sequence of target values is then smoothed across each segment boundary depending on a system of priorities assigned to each segment. Each segment is assigned an initial time constant and a terminal time constant, which, together with the relative priority of each, determine how it interacts with its neighboring segments. The neighbor with the greater priority imposes its time constants and boundary crossing points on the transition. In most cases, the higher ranking units actually have the weakest conditions, thus guaranteeing that they will be "overtaken" when they occur next to a lower priority segment. As an example of this, HH has a high priority, but no control over the boundary crossing value. As a result, when HH occurs next to any vowel, which would have a

lower priority, the HH determines that the vowel will have full control over the transitions. This effect can be seen in the graphs in figure 3.

Once the formant parameters are computed for the duration of a segment, it remains only to fill in the various amplitude control parameters (AV, AH, AF and AN) and the fricative frequency setting (FF). All of these parameters are controlled more or less directly by bits set in the feature matrix. For example, when a consonant is encountered with the plosive feature set, a pulse is set up on the AH control. The time of release of the plosive burst is computed depending on the plosive aspiration feature. Provided the following consonant is not also a stop, this pulse is then stored at the computed release time. Voicing and nasal amplitudes simply move thru smooth arcs to predefined levels if the corresponding feature bits are set.

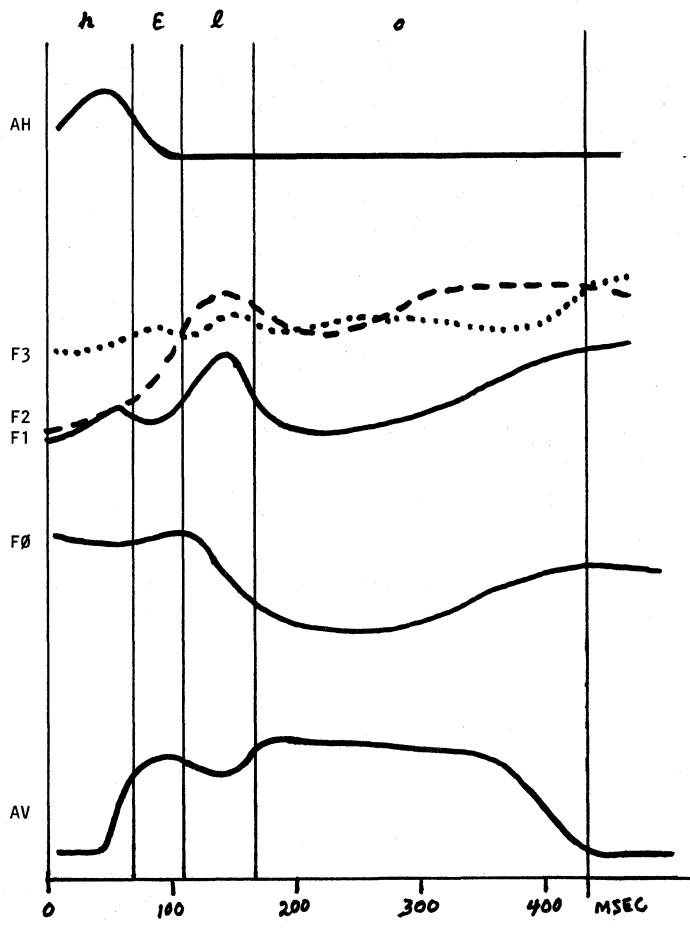


Figure 4 Parameters extracted from human speech "hello"

HOW GOOD IS IT?

As an illustration of the parameter output of a rule system of the sort described, figures 3 and 4 present sets of parameters generated by the program and recovered from actual human speech, respectively. The examples compare the two versions of the word "hello" as produced by these two methods. In the synthesized version in figure 3, the phonetic segments are marked with ASCII characters just as used to generate the word. The input to the program was the string "HHEH2LOW," which was followed by another phrase. In the human version in figure 4, the segments are marked using the International Phonetic Alphabet (IPA) symbols as well as the normal English spelling. The human version was also spoken as a part of a longer phrase.

While there is quite a lot of difference in detail, the overall shapes are reasonably similar. The major difference one sees immediately is in the timing structure. While the overall length of the human version is 130 msec shorter, the length of the OW vowel is the same within 10 msec. The first 3 segments make up this difference, particularly the E, which is barely more than $\frac{1}{3}$ the length of the computer generated EH. The primary reason for this is the 2 stress specified on the EH

vowel. This was necessary to get even the small rise in pitch seen during that vowel. The generated F ϕ curve during the remainder of the word is disappointing to say the least, and shows nothing like the expressive contour seen in the human version. The differences in F3 during the L are probably not significant, altho it's interesting to note that the synthetic version accurately follows the "textbook" pattern for an L, while the human version shows the L quality only in F1 and F2. Of greater interest is the dip in voicing amplitude during the human L. No doubt the perception of an "L" would be improved if the rule system generated such a dip (rise in frequency) during an L sound.

BIOGRAPHY

Mr. Rice holds a B.A. in Linguistics from UCLA (1969). He has been involved in research in acoustic phonetics for the past nine years, which included the opportunity to work in Sweden and Finland during part of 1974. He is currently a partner of Computalker Consultants in Santa Monica, CA, involved with the design of speech synthesis circuitry and related software. His interests include speech production, digital signal processing, microcomputers, and occasionally such pursuits as sailing and SCUBA diving.

REFERENCES

1. R. Carlson and B. Granstrom, "A phonetically-oriented programming language for rule description of speech", Proc. Speech Communication Seminar, Stockholm, Sweden, August 1-3, 1974
2. D.H.Klatt, "The linguistic uses of segmental duration in English: Acoustic and perceptual evidence", J. Acoust. Soc. Amer., vol 59(May 1976)pp. 1208-1221
3. D.H.Klatt, "Structure of a Phonological Component for a Synthesis-by-Rule Program", IEEE Trans on ASSP, vol ASSP-24(Oct 76)pp. 391-398
4. D.L. Rice, "Hardware and Software for Speech Synthesis", Dr. Dobbs J. of Computer Calisthenics & Orthodontia, vol 1, no 4 (April 76)
5. D.L. Rice, "Friends, Humans and Countryrobots: Lend me your Ears", BYTE no. 12 (Aug 76)

TOP-DOWN ANALYSIS OF LANGUAGE RHYTHMS IN SPEECH SYNTHESIS

Alice Wyland Grundt, Ph.D.
Linguistics Department
California State University at Fresno
Fresno CA 93740

Does your speech synthesizer have an outlandish accent? In fact, does it take a new acquaintance a fair amount of time to understand a word your machine is saying? Recent literature (Atmar, 1976; Rice, 1976a, 1976b) suggests that difficulty in understanding computer talk is accepted as one of the more unsatisfactory aspects of current speech synthesis work. It has occurred to me that perhaps a new approach to phoneme synthesis might be helpful.

According to Atmar, the machine is given specific instructions, i.e., each phoneme or distinctive speech sound is assigned a specific set of formant frequencies and a specific duration in addition to the fundamental frequency so that the machine pronounces every phoneme in the same fashion each time it occurs. The transitions between one phoneme and another are very important as both Atmar and Rice point out since for stop consonants like /p/, /t/, /k/, for example, it is the stage and timing of the formant transitions to the vowel plus the timing of the onset of voicing which makes it possible for the hearer to identify which consonant was pronounced and whether or not it was unvoiced /p/, /t/, or /k/ or voiced /b/, /d/, or /g/. Atmar remarks that although such transition data is known, satisfactory rules have not been worked out and machines which do incorporate the known rules quickly become 'elaborate and expensive (in the tens of thousands of dollars)' (Atmar, p. 30). For General American English there are 38 phonemes, which means that, even with the number of prohibited combinations taken into account, there are a great number of transition combinations which must be stored, retrieved and combined for every phoneme.

With simplified rules a much smaller machine can be constructed but intelligibility is hampered and the listener must 'learn' the machine's accent. The difficulty of programming the generation of phonemes offers problems, especially since there seem to be no clear rules for designing the circuitry. Atmar remarks that 'the actual

Copyright: Alice Wyland Grundt, 1977 ©

phoneme perceived is determined by the duration of the excitation and the selected formant filters.' (Atmar, p. 28). Atmar also quotes M. D. McIlroy of Bell Laboratories who says: 'Being ephemeral, sounds must be understood at first hearing. As it turns out, long speeches [on the speech synthesizer] ARE hard to understand, as are extremely short utterances.' (Atmar, p. 30). McIlroy's remarks seem rather puzzling and the reason for lack of intelligibility is not clear.

The difficulty may be in the way the machine is designed to generate the phonemes. I am a linguist and not a computer engineer so I cannot offer any ideas on programming or circuitry design. However, I think that the work being done at The Ohio State University in speech timing under the direction of Professor Ilse Lehiste of the Linguistics Department offers an overall conception of the temporal organization of speech which may be helpful in setting up the problem and its solution.

So far as I can determine - and I will welcome correction if I am in error - Professor Lehiste's work has not been taken into consideration in the theories governing speech synthesis. Some of the problems in generating recognizable phonemes are probably due to the fact that they are assigned a specific set of formant frequencies and a specific duration. From a linguistic point of view, phonemes are negatively defined: phonemes do not have to be composed of any specific combination of formant frequencies or duration; they merely have to be different from each other in some acoustic dimension. Therefore, it would be more correct to say that a certain range of formant frequencies and a certain range of duration values will be identified by the hearer as a particular phoneme. This range will vary according to the context such as the preceding and the following phoneme. The transitions may also be affected by whether or not a word boundary intervenes. This aspect of phoneme reality has evidently been incorporated into current speech synthesis work. However, I have the impression that the method of building up syllables and words out of phonemes amounts to stringing the phonemes together, one by one, with the appropriate transitions included and with pauses at word boundaries.

Since the syntax of English and other languages is hierarchically structured, it would be advantageous if the sound production could also be shown to be hierarchically structured. If so, such structuring ought to be helpful in dealing with the timing aspect of synthesizing speech. It seems to me that Lehiste's work can be very useful in this regard.

Lehiste has been interested since at least 1960 in determining phonological and phonetic units larger than the syllable, based on the timing of syllables and their duration ratios. She began her work with Eastern European languages: Estonian, Finnish, Czech, and Serbo-Croatian. Perhaps for this reason her work has not gained wide attention among those interested in English. Her results, however, are theoretically very important.

She has found that Finnish words 'appear to be constructed of units larger than one syllable: all words consisting of more than three syllables seem to be built of disyllabic or trisyllabic components, whose quantity patterns are similar to those of disyllabic or trisyllabic words.' (Lehiste, 1963, p. 179). In Estonian, the syllables of the initial disyllabic unit of a word are related by duration ratios - the duration of the vowel of the second syllable varies with the quantity or duration of the first syllable inversely, regardless of the number or kind of consonants in the first syllable. In other words, the disyllabic unit is a temporally organized unit with internal structure that maintains the overall temporal value by compensatory adaptation. In addition, Lehiste found that syllabic quantity or duration is NOT the sum of the segmental quantities of its components. (Lehiste, 1970, pp. 156ff). This means that the Estonian temporal units such as disyllabic sequences cannot be built up by stringing together phonemes of constant duration. There must

be a larger temporal context into which the phonemes must fit by mutually adapting their individual durations in specific ways.

Lehiste has also shown that phonetic units and linguistic units do not necessarily coincide. In Finnish 'the quantity relationships between successive syllables serve not only to establish word patterns but also to subdivide words into phonological components that are intermediate between a syllable and a word. No immediate connection between these intermediate units and the morphological structure of the words could be established.' (Lehiste, 1963, p. 180). Popperwell has shown that phonological phrases characterized by an initial stressed syllable and followed by a string of unstressed syllables up to the next stressed syllable need not coincide with word boundaries but may bisect words in Standard Norwegian. (Popperwell, 1963).

Although hierarchies of phonological units have been included in the linguistic theories of Pike and Halliday, according to Lehiste, (Lehiste, 1970, p. 156), she points out that 'higher-level phonological units have been largely ignored within generative-transformational linguistic theory.' She also remarks that the objective establishment of such phonological units is provided by evidence from studies of suprasegmentals: pitch, duration, and intensity of the speech signals. If speech synthesis is based on grammatical word and phrase boundaries, it may be difficult to establish phonetic temporal units that can function as rhythmic units since the phonemic length of words and phrases is unpredictable. In the case of 'sponerisms' such as 'Let me sew you to your sheet' for 'Let me show you to your seat' or 'Is the bean dizzy?' for 'Is the dean busy?' it appears that quite long stretches of speech must be preplanned; otherwise, it is difficult to account for such sequencing mistakes. The problem is to discover how such large temporal units are actually organized internally.

Lehiste found that not only Finnish and Estonian organize their words in terms of disyllabic units but so do Czech, Serbo-Croatian, the language of the Lapps (related to Finnish), Norwegian, and Swedish. Since our concern is with English, however, let us take a look at her work with English.

In testing monosyllabic and disyllabic words in English, Lehiste made the fundamental assumption that the word is stored and processed in articulation as a whole by the brain and that significant relationships exist among all segments that constitute a word, although not all segments participate in segmental quantity oppositions. (Lehiste, 1971, p. 929). The suprasegmentals mentioned earlier - pitch, duration, and intensity - are necessarily patterns in time: any contrastive arrangement of fundamental frequency or intensity or duration is crucially dependent on the time dimension. Lehiste then assumed that 'the arrangement of articulatory events along the time dimension may likewise have suprasegmental function, and may serve to establish higher-level phonological units.' (Lehiste, 1971, p. 159).

Her starting point was the observation that two Russian phoneticians, Kozhevnikov and Chistovich (1965), had shown that when a speaker repeats the same utterance many times, at the same rate of articulation, the durations of adjacent phonemes are quite strongly negatively correlated. That is, if an error is made in the duration of one phoneme, the error is largely compensated for in the following phoneme, which finishes at the originally planned time, despite the fact that it started late. Lehiste says, 'This negative correlation suggests that articulatory events are programmed. . . not in terms of single phonemes, but in terms of higher-level articulatory units. One way to determine the extent of these higher-level units would be to establish the domain over which such temporal compensation takes place, since it seems reasonable to assume that the sequences of sounds which are subject to temporal compensation constitute a single articulatory program.' (Lehiste, 1971, p. 159).

Lehiste worked with monosyllabic closed-syllable words - skid, skit, stay, steed, stead, stayed - and disyllabic words with one medial

consonant - steady, skiddy, and skitty. She established that there was always a negative correlation present between the duration of the vowel and the duration of the following consonant of a monosyllabic word and within the monosyllabic word as a whole as well. However, in the disyllabic words which have an intervocalic flapped /t/ in the Midwestern American dialect of English, Lehiste found no obvious negative correlation favoring one or the other of the flanking vowels. Instead, she discovered that in the words steady, skiddy, and skitty there was temporal compensation, i.e., negative correlation, between the durations of the two vowels such that the duration of the second vowel is adjusted to that of the first and she concluded that this result showed that the sequence of two vowels with their consonant surround constituted a unit of programming at a higher level than the syllable for English.

The discovery of vowel duration ratios in disyllabic words of the skiddy, skitty type means - as Lehiste specifically points out - that the overall actual duration of skiddy is longer than that of skitty, thus providing the necessary acoustic clue that the medial consonant is voiced in English, but as long as the vowel duration ratio is maintained, the vowels will be correctly identified by the hearer. That is, these vowels may be longer or shorter depending on the voicing of the medial consonant or depending on speech tempo but their duration ratio will serve as a necessary acoustic clue to their identification. I might point out that these results were obtained from speakers with considerably different speaking rates: one seemed to the observer to be a slow and careful speaker and the other rather hasty and erratic, yet the establishment of vowel duration ratios was quite clear.

The ratios in question were for one speaker 0.54 for skiddy and 0.55 for skitty while for the other speaker the ratios were 0.89 for skiddy and 0.84 for skitty, out of a total of 110 spoken tokens for each speaker. It is clear from these figures that the duration ratio itself is idiosyncratic to the speaker - part of what makes the individual speaker recognizable - but once the characteristic vowel duration ratio of any particular speaker is determined, his vowels in such units should always be recognizable.

The discovery of vowel duration ratios in disyllabic words with one medial consonant in English suggests that English may be temporally organized like Finnish or Estonian - that is, in terms of units of one, two, or three syllables internally organized by vowel duration ratios. As far as I can determine, the work that needs to be done to establish duration ratios for the possible combinations of vocalic sequences such as in ready, pole, budv, callow, etc. has barely begun. However, it seems to me that here may be the key to the problem of designing the phoneme-generating aspect of the speech synthesizer.

If, as Atmar says (Atmar, 1976, p. 29), analog simulation of the vocal tract is the only method of true speech synthesis known, then the organization of temporal relations within large phonological units like words and phrases as performed by the brain should be simulated by speech synthesis machinery. But on what basis?

What I have said so far implies that every English word will have its own temporal program. And, in fact, Kozhevnikov and Chistovich have concluded on the basis of their work that the basic storage frame of a word in the memory is in the form of the temporal structure of the syllables. However, if we must store each word with an idiosyncratic temporal structure, we will need a large memory. It is indeed conceivable to give the machine a basic vocabulary of 350 to 500 words, and it is also conceivable that we might store stock phrases as well such as 'thank you', 'what do you mean?', 'all right', etc. However, there may be a way to cut down on the storage problem.

Words can be categorized according to the syllable types and syllable-unit types they contain, the number of syllables, and the

place of accent. (For purposes of this paper, I will assume that accent can be independently set up as a change in pitch and, perhaps, intensity). If the syllables and syllable units can be related by vowel duration ratios, they can be cross-referenced with tables which list the timing relations of such units in terms of the duration ratios holding among their internal members. Words could then be stored in terms of these units. As long as the relevant duration ratios are maintained among all members, the actual duration of the units may be longer or shorter, depending on the needs of speech tempo, without sacrificing recognizability.

Perception-oriented tests reported by Garnes (1975) and performed by her for Modern Icelandic, and similar tests for English by Denes (1955), Lehiste (1972) and Bond (1971), indicate that the perceptual process is dominated by perception of the word as a whole (in these cases, the words were mono- and disyllabic), and that the identification of the segments does not proceed segment by segment but depends on the durational ratios of the segments involved rather than the absolute durations of either vowel or consonant in the unit. Warren H. Fay (1966) concluded from his work on temporal sequence in the perception of speech that 'what appears to be the critical factor in any speech perception experiment is the listener's reference patterns, or the so-called "comparator" of servo models. Man's comparison patterns for speech are language patterns and, regardless of whether storage is predominantly auditory or motor in nature, the essential dynamics of language itself cannot be overlooked.' (Fay, 1966, p. 99).

From these observations, it seems to me that the difficulty of determining what phoneme your speech synthesizer is really saying may be partly due to the apparent use of a speech sound no larger than a phoneme when testing for phoneme identification. One must know all the other phonemes of the machine's phonemic system in order to identify any one member, i.e., the hearer must have the complete reference pattern available for comparison. Without a larger unit consisting of more than one phoneme, the hearer has nothing immediate to compare the sound to. The identification of any speech sound appears to be possible only when there exists a complex of mutually conditioning and conditioned speech sounds, related by duration ratios.

I must emphasize that the establishment of specific duration ratios in syllables, disyllabic units, and trisyllabic units for American English seems to be in its early stages. Part of this relative lack of information may be due to the fact that current linguistic theory has not been able to incorporate the discoveries and advances of acoustic phonetics into its theoretical constructs as Lehiste has noted. The linguistic emphasis on contrastive elements - the phonemes, which are static constructs and independently defined - ignores the phonetic conditions under which phonemes can be recognized and identified and, therefore, the mutually interdependent relations that obtain and which are necessary to the perception of the segmental components of a phonetic unit.

English can be said to be an isochronous language: that is, spoken English tends to be organized in stress groups of approximately the same duration. If such isochronous groups include a varying number of syllables, their duration must vary according to the number of syllables included in the group; the spacing between main stresses tends to remain constant. (Lehiste, 1970, p. 39).

Isochrony is most familiar from poetry where the poet consciously sets up a rhythmic figure, called a metrical foot, and chooses his words to fit the framework. Words may be drawn out or compressed in order to satisfy the rhythm, as in Tennyson's famous lines:

/Break,/	/break,/	/break,/
/On thy cold/	/grey stones,/	/oh, sea!/_

In the case of poetry based on metrical feet, the rhythmic frame

is chosen first and the words fitted to it. The speaker of English can do the same: he can set up a moderately flexible sentence rhythm independently of the grammatical divisions of his sentences and also choose his words to fit the rhythm and to allow the rhythmic stress to land on the right words, that is, the nouns and verbs instead of the prepositions and conjunctions. Of course, the rhythmic figure or metrical foot which the speaker voluntarily or characteristically adopts may be long or short and may be changed in mid-utterance for various kinds of effect. But he will have a stable basis for rhythm analogous to musical measures with relatively constant duration. If we make the assumption that the stressed syllable has a constant durational domain, we have two reference points for timing the utterance: the 'measure' and the principal reference point within the measure. If the rhythmic figure is assumed to be independent of the grammatical categories, the selection of units of meaning, i.e., the words, to match the phonetic rhythm allows the motivated choice of words that fit the rhythm instead of those that don't. Your speech synthesizer could then make what appear to be stylistic choices of words but which are actually choices based on rhythmic requirements.

In this way, a kind of hierarchical structure can be built up: syllables as temporal units with internal negative correlations of segment duration, disyllabic units with internal negative correlations of syllable duration or vowel duration, and so on. To illustrate this idea, I have prepared a handout which analyzes three sentences taken from the March, 1977, issue of SCIENTIFIC AMERICAN. A musical format seems to be the best way of showing the interrelations of duration, stress and pitch - which is, of course, highly simplified and schematized. When the sentences are presented as sequential lines of stress groups, the internal balancing of measure types is more clearly seen. For example, in the first sentence - 'An analysis of a simple two-person game can lead into fascinating corners of number theory' - shows regular pattern recurrence: stress plus one syllable occurs in lines 3 and 5, stress plus two syllables occurs in lines 4, 6, and 8, stress plus three syllables occurs in lines 7 and 9. I am tempted to suggest that such intricate rhythmic and syllabic balancing within a sentence is a mark of good style. Certainly, Mr. Gardiner did not write these sentences especially for me to analyze. In any case, these sentences seem to lend themselves to analysis in terms of duration ratios of larger and smaller units in a hierarchy.

I might point out that a musical format might be a very convenient one for working out the timing relations as well as the pitch and intensity relations: the latest issue of CREATIVE COMPUTING (March-April 1977) has several articles on the relation of music and computers, which might offer useful ideas.

This paper is somewhat speculative, but an overall theory of speech timing seems to be a more fruitful way of dealing with speech synthesis than stringing phonemes together, one by one. So far as I know, the relevant duration ratios I have been discussing are not available in useful detail. Pilot studies made at The Ohio State University all remark that this area of speech timing should be investigated more thoroughly. Therefore, I wish to call your attention to this aspect of research and urge those of you who have a phonetics laboratory at your disposal to consider the determination of duration ratios in speech as an important and useful field of inquiry. It seems to me that the problems of both 'accent' and speech recognition might be brought closer to solution if more sophisticated approaches to the problem of speech timing and the reference patterns of the entire phonological system are taken into account. Once the timing has been stabilized, the integration of pitch and intensity parameters should be easier.

TABLE I

Comparison of average durations (in milliseconds) of segments in three monomorphemic and three bimorphemic words, produced by speaker DS.

Word	C ₁	C ₂	V ₁	C ₃	V ₂	Total
stead	133	123	307	149		712
steady	94	98	133	20	173	518
skid	148	104	217	151		620
skiddy	128	97	90	28	166	509
skit	156	104	185	115		560
skitty	110	87	83	23	151	454

Taken from: Lehiste, 1971, p. 167.

TABLE II

Difference between the relative variances of successive segments taken individually and considered as a co-articulated sequence, calculated on the basis of monosyllabic words produced by DS. [negative values = negative correlation].

Word	C ₁ C ₂	C ₂ V ₁	C ₁ C ₂ V ₁	V ₁ C ₃	C ₁ C ₂ V ₁ C ₃
steed	-0.73	-0.38	-0.58	-0.66	-0.79
staid	+0.07	+0.26	+0.13	-0.48	-0.05
stayed	+0.48	+0.14	+0.26	-1.07	-0.79
stead	-0.40	-1.29	-0.36	-4.17	-2.80
skid	-0.06	-0.33	-0.32	-0.27	-1.14
skit	+0.01	-0.20	-0.20	-1.26	-0.53
stay	-0.55	-0.12	-0.08		

Taken from: Lehiste, 1971, p. 163.

TABLE III

Difference between the relative variances of successive segments taken individually and considered as a co-articulated sequence, calculated on the basis of disyllabic words produced by DS. [negative values = negative correlation].

Word	C ₁ C ₂	C ₂ V ₁	C ₁ C ₂ V ₁	V ₁ C ₃	C ₃ V ₂	V ₁ V ₂	C ₁ C ₂ V ₁ C ₃ V ₂
steady	-0.55	+0.03	-0.31	-0.13	+0.18	-0.53	-0.70
skiddy	-0.04	-0.09	-0.22	-0.32	-0.61	-0.38	-0.92
skitty	+0.33	-0.03	+0.35	+0.01	-0.37	-0.73	-0.86

Taken from: Lehiste, 1971, p. 165.

References:

- Atmar, Wirt. 1976. The time has come to talk. BYTE #12 (August). 26-33.
- Bond, Zinny S. 1971. Units in speech perception. The Ohio State University unpublished Ph.D. dissertation.
- Bush, Clara N. 1972. Temporal ratios of sound segments and the perception of English dialect differences. Proceedings of the 7th International Congress of Phonetic Sciences. Mouton. 666-673.
- Denes, Peter. 1955. Effect of duration on the perception of voicing. Journal of the Acoustical Society of America. 27.761-764.
- Garnes, Sara. 1975. Perception of suprasegmental and segmental features. Semiasia: Beiträge zur germanisch-romanischen Sprachforschung. Band 2. 61-73.
- Grundt, Alice Wyland. 1976. Compensation in Phonology: Open Syllable Lengthening. Indiana University Linguistics Club. 1-87.
- Kozhevnikov, V. A. and L. A. Chistovich. 1965. Speech: Articulation and Perception. Translated by J.P.R.S., Washington, D.C., No. JPRS 30,543. Moscow-Leningrad.
- Lehiste, Ilse. 1963. (See bibliography for Lehiste, 1972).
- Lehiste, Ilse. 1964. Juncture. Proceedings of the 5th International Congress of Phonetic Sciences. The Hague: Mouton. 172-200.
- Lehiste, Ilse. 1970. Suprasegments. The MIT Press.
- Lehiste, Ilse. 1971. Temporal organization of spoken language. In: Form og Substance, Phonetic and Linguistic Papers Presented to Eli Fischer-Jørgensen (Hammerich, Jakobson, and Zwirner, Eds.) Odense: Akademisk Forlag. 159-169.
- Lehiste, Ilse. 1972. Temporal compensation in a quantity language. Proceedings of the 7th International Congress of Phonetic Sciences. The Hague: Mouton. 929-939.
- Lehiste, Ilse. 1972. Manner of articulation, parallel processing, and the perception of duration. University of Essex Occasional Papers No. 13.1-24.
- Popperwell, R. G. 1963. The pronunciation of Norwegian. Cambridge University Press.
- Rice, D. Lloyd. 1976. Hardware and software for speech synthesis. Dr. Dobb's Journal of Computer Calisthenics and Orthodontics, Vol.

1.4.6-8. (April, 1976).
 Rice, D. Lloyd. 1976. Friends, humans, and countryrobots: lend me
 your ears. BYTE #12 (August). 16-24.

HANDOUT: Top-Down Analysis of Language Rhythms in Speech Synthesis.
 By: Alice Wyland Grundt
 California State University, Fresno
 Copyright 1977 (C)

Text Analysis into Temporal Units

Definitions: Temporal unit: stress-timed isochronous units, made up of syllables.
 Unit begins with stressed syllable and continues until next stressed syllable.

Let stressed syllable = -constant minimum duration*

Symbols: ' = accented syllable
 x = unaccented syllable

Text:

An analysis of a simple two-person game can lead into fascinating corners of number theory. We begin this month with a charming little-known game played on a chessboard with a single queen. Before we are through, we shall have examined a remarkable pair of number sequences that are intimately connected with the golden ratio and generalized Fibonacci sequences. (Gardiner, Martin. 1977. Mathematical Games, SCIENTIFIC AMERICAN (March). pp. 134ff).

Sentence No. 1

Stress Group		Text (words/syllables)
Number	Pattern	
1.	' x	an a-
2.	' x x x x	nalysis of a
3.	' x	simple
4.	' x x	two-person
5.	' x	game can
6.	' x x	lead into
7.	' x x x	fascinating
8.	' x x	corners of
9.	' x x x	number theory

Pattern Recurrence

' x x	1
' x	3, 5
' x x	4, 6, 8
' x x x	7, 9
' x x x x	2

*stress domain: may include disyllabic words CVCV (e.g., 'steady') as well as closed or long monosyllables (CVV, CVC, CVCC, etc)

CVCV = CVCC, etc, see Grundt, A.W. 1976. Compensation in Phonology: Open Syllable Lengthening. Indiana University Linguistics Club.

Sentence No. 2

Stress Group		Text (words/syllables)
Number	Pattern	
1.	' x x	We be-
2.	' x	gin this
3.	' x x	month with a
4.	' x	charming
5.	' x x	little-known
6.	' -	game
7.	' x x	played on a
8.	' x x x	chessboard with a
9.	' x	single
10.	' -	queen

Pattern Recurrence

' x x	1
' -	6, 10
' x	2, 4, 9
' x x	3, 5, 7
' x x x	8

Sentence No. 3

Stress Group		Text (words/syllables)
Number	Pattern	
1.	' x	Be-
2.	' x x	-fore we are
3.	' x x x x	through we shall have ex-
4.	' x x x	-amined a re-
5.	' x x	-markable
6.	' x	pair of
7.	' x	number
8.	' x x x x	sequences that are
9.	' x x x x	intimately con-
10.	' x x x	nected with the
11.	' x	golden
12.	' x x	ratio and
13.	' x x x x	gen'eralized Fibo-
14.	' x	-nacci
15.	' x x	séquences

Pattern Recurrence:

' x	1
' x	6, 7, 11, 14
' x x	2, 5, 12, 15
' x x x	4, 10
' x x x x	3, 8, 9, 13

MUSICAL SCHEMA - for first two sentences of text.

Let, stressed domain = half-note minimum

Remarks: Tempo, time and key signature, dynamics, pitch changes, etc. chosen arbitrarily for this example.

Andante

an a- nal - y sis of a sim - ple two per - son
game can lead in - to fas - cin - at - ing
cor - ners of num - ber the - ory. We be-
gin this month with a char - ming
lit - le known game played on a
chess - board with a sing - le queen.

HOME TEXT EDITING

Larry Tesler
 Xerox Palo Alto Research Center
 3333 Coyote Hill Road
 Palo Alto CA 94304

ABSTRACT

Your computer's text editor may be the program you use most. If you don't have an editor you like, it is best to copy the design of an existing system. If you decide to design your own, some guidelines are offered both for the design of the commands and for the design of the program. They focus primarily on "two-dimensional" editors that make full use of a display, but they pertain indirectly to "one-dimensional" editors such as those that can be programmed for teletypewriter style terminals.

1. Text editors don't really edit.

Almost every computer with a terminal also has a program that lets people type and alter text. Usually, that program is called an "editor". However, in English, an "editor" is a person who reads a document and improves its style, clarifies points, verifies arguments, and corrects grammar and spelling. Editing is a complex intellectual task best performed by humans.

A computer can be a powerful tool to assist a human editor. Changes can be made to a text without the person having to completely retype it. But the machine does not improve style, does not clarify points, does not verify arguments, does not correct grammar, and usually does not even correct spelling. So a "text editor" is not really an "editor" at all -- it's just a fancy pencil.

The term "editor" is a misleading name for such a program, not only because of what the program *can't* do, but also because of what it *can* do, which goes well beyond the improvement of someone else's prose. It can help a person to compose one's own prose -- or poetry -- by offering a neatly typed version of each new draft on command. It can do a certain amount of typography, including indentation, spacing, and sometimes justification of margins. It also can help a person to fill out a form, to provide data to a computation, to converse with a dialog system, and neither last nor least, to compose and alter a computer program.

A better name than "text editor" might have been "interactive text processor". Unfortunately, the terms "editor" and "editing" have come into common misuse, so I will reluctantly employ them myself in the rest of this tutorial. Henceforth, whenever I use the term "editor", I will not mean a person, but rather a computer program that lets one type and alter text. The human user of such a program will be called the "operator".

2. Designing a Text Editor.

Many computer users find that the program they use most is the text editor. If you don't like your computer's text editor, or if you want to take on a challenging programming task, it is often possible to concoct your own editor. You can design one from scratch, or you can imitate a design you've seen and liked on a friend's machine. The latter is a far safer course. Designing an editor is difficult, and you may put in a lot of work only to end up with a tool that is frustrating to use.

For those who are thinking of designing an editor, the remainder of this tutorial offers some advice to help in your adventure. Not every aspect of the problem is covered; it is assumed that you have used at least one editor and are familiar with the general concepts of editing.

There are two aspects to the design of a text editor: What will be the command structure? How will the program work? I will call these two aspects "command structure design" and "program design". Command structure design is part of a larger problem known as "user interface design", a problem too large to address here.

Although the design of the command structure and of the program are not completely independent tasks, we'll take them up one at a time. Mainly, we'll talk about command structure design. Most people with sufficient interest can manage the program design for an editor on their own.

3. Command Structure Design.

3.1 Rube Goldberg never sold anything but comic strips.

There are several ways to approach the design of commands for a text editor. If you're designing one just for your own use, then you can dream up any that you like. After all, if you like catsup on your cottage cheese, who's to object?

On the other hand, if you would like friends and family to be additional users

of your editor, then it may help to consider the following. Each of us has unconscious idiosyncrasies that make our lives and the lives of our friends more interesting. But when one's idiosyncrasies show up in the design of a program, other people who try to use the program often find it confusing. Even if you do your design by brainstorming with a friend, it's likely that the friend is a Whatsisology major just like you, and that your technical background has prepared you both to think that a Whatsis command is just what the world needs in a text editor.

Therefore, I strongly suggest that you try to explain your command design to someone who majored in Southern Rumanian Poetry or to someone who was lucky enough to get out of school after the eighth grade. If the editor is to be special purpose, explain it to someone who would want to use it for that purpose. When the listener can not understand what you are talking about, please do not think that he or she is stupid. What has happened is that your design, like almost all others that people think up, is the computer equivalent of an Edsel. Learn from Ford's mistake; go back to the drawing boards. And reread the tips that follow.

3.2 You can't bake a cake on a hot plate.

The first constraint on the design of your editor is the physical hardware that your computer has. If there is a display screen, even a one-liner, then use it; it is chock full of advantages. If there is no display, a decent job can be done with your typeout terminal. If you have both a display and a printer, then the display should be used for editing and the printer just for typing neat drafts.

There are two ways to use a display: "one-dimensionally", in which it is just like a fast typeout terminal, but only shows you the last few lines of dialog, or "two-dimensionally", in which many lines of a page are displayed as they would appear on paper, without commands interspersed. From the operator's viewpoint, two-dimensional editors are superior in many ways to one-dimensional ones. However, they are harder to implement; how hard depends on your machine characteristics.

If you choose a one-dimensional approach, or if you have no display at all, then I recommend reading reference [1], which describes the commands and lists the program for a very well implemented one-dimensional editor. The suggestions that follow pertain mainly to a two-dimensional approach, because the problems are less obvious. However, if you are sticking to one dimension, you may still get ideas from the discussion; just translate the concepts from one medium to the other. Sometimes, I do this for you by including [notes in brackets for one-dimensional editors].

The most important part of a text editor is the means the operator must use to specify the part of the text that is to be manipulated; this task is often called "entity selection". In most two-dimensional editors, the operator can select any character or line in the text; the selected entity is often marked by a "cursor". [In most one-dimensional editors, the operator can select any line and by various subterfuges can get at characters within the line.]

It is often useful to be able to select words, sentences, form fields, or other entities of text, or to select the interstices between characters instead of the characters themselves. However, the benefit of such special selections is marginal for many applications. By designing them in, you may simply clutter the editor with unnecessary mechanisms and fill up scarce memory space with the software to implement them.

If your display has a pointing device to go with it (a tablet, joystick, mouse, light pen, etc.), then by all means use it to make selections. If you have an A/D board, you may be able to add a joystick for a few dollars.

Many people think they can control an editor faster from the keyboard than with a pointing device, but some objective studies have found the opposite to be the case.

Even ignoring the issue of speed, pointing devices have advantages over key control. It has been shown that, given a choice of selection methods, operators will usually gravitate to a single general-purpose method and ignore all others, even in cases where they may be faster. A pointing device provides the most general selection method, because every selection -- no matter how far it is from the current selection -- can be made by the same means.

Arrow keys, control keys, space bars, and so forth are less general; with them, the operator ends up devising strategies to get the selection made in an acceptable time. Imagine a car whose steering wheel was replaced by left turn and right turn keys for various angles. It may seem like fun to be devising strategies to get the best use out of a program. However, the fun of toying with the editor wears off when you are trying to get that term paper finished in time or when you are trying to write that neat program or an important love letter.

Another advantage of a pointing device is that it reduces the number of keys needed to run the editor. Sparseness is a virtue in interactive computer programs; the fewer commands and the fewer keys you need, the easier it is to operate the system, even if an occasional command requires extra keystrokes as a result.

Of course, if your computer has no pointing device, then make the best use of the keys. Tens of thousands of people do it every day. If the terminal has cursor control keys, use them if possible. Otherwise, use other keys, but preferably not typing keys. It is very confusing to people when the same key sometimes types the letter W and sometimes selects the next Word in the document.

[If your terminal has no display, then there are many ways to specify a line to select. Each line can have a number semi-permanently attached to it (number them 100, 200, 300, and so on). Insertions can use numbers in between the ones already used. There should be a renumbering command to use just before printing out a draft. It should be possible to print a draft with or without line numbers.]

[Other ways to identify lines are by current ordinal number (constantly changing during the session -- very confusing), by distance from the currently selected line (".+6", ".-10"), by special name ("*" for final line, "↑" for preceding line, etc.), or by sample content ("the stove"). In my experience, most people prefer the last three methods over the others.]

There are other hardware attributes that can constrain your editor design. You may wish "control-M" to mean something special, but on an ascii terminal, that stroke often produces the same character code as a "return", so you can't tell them apart. The agility of the human hand is also a factor; it is easier to strike "control-A" or "control-L" than to strike "control-F" on a standard keyboard layout. It takes a long time to strike a key that is far from the home typing keys, even for a hunt and pecker. So think about which commands happen most often, and be sure to consider any physical disabilities that the potential operators may have.

3.3 Some cuisines.

There are many possible command structures for a text editor. I will try to give a sample to stimulate thought, and suggest some criteria to consider in choosing among them.

If you have used several pocket calculators, you know that there are "algebraic infix" models where you say "2 + 3 =" and "Polish postfix" models where you say "2 enter 3 enter +". It is even possible to have "prefix" calculators, where you would say "+ 2 enter 3 enter". A similar classification exists among editors.

A "prefix" command structure has the operator specify an operation first and then the selection or selections on which to operate. For example, "Delete (select character) (return)" [or "Delete lines 2 to 10 (return)" in a one-dimensional editor]. Some people like prefix commands because they sound like English sentences and because they require confirmation of the command, which provides an opportunity to change one's mind.

A "postfix" command structure has the operator specify the selection(s) first and then the operation. For example, "(select character) Delete". Postfix commands are usually easier to use because the absence of confirmation requires fewer keystrokes, and because machines to which people are accustomed work that way. On a typewriter, you first move the carriage and then type. On a vending machine, you first make your selection and then pull the handle (and then kick it). On a clothes dryer, you first dial the time and then push start.

The trouble with postfix is that confirmation is not required. If you try requiring it, operators may complain about the superfluous keystrokes. To compensate for the lack of confirmation, each operation should have an obvious converse (e.g., "Insert" for "Delete", or "Undo" for any prior command).

There could in theory be "infix" commands in an editor. To move a paragraph from one place to another, you would say "(select source) MoveTo (select destination)". The trouble with this form is that the command ends with neither a confirmation nor an operation, and people feel uneasy: in postfix systems, they are used to an operation stroke at the end of each command, and in infix systems, they are used to a confirmation at the end. It is better in prefix systems to avoid infix altogether. In postfix systems, it can be approximated by a pair of commands: "(select source) Delete; (select destination) Insert".

In prefix systems, it is useful to allow the same command to be applied to a number of selections without having to repeat the command: "Delete (select something) (return) (select another) (return) (select another) (return)". In postfix systems, it is useful for each command to leave some logical entity selected when it is done (like the character after the one deleted), and for a command to omit a selection specification when that entity is the one desired, e.g., "(select character) Delete Delete Delete" could delete three consecutive characters. The latter is especially helpful if Delete is a repeating key on the keyboard.

If there is a display on the machine, and especially when there is a pointing device for it, there is another way to issue commands, namely, through a "menu". The menu displays alternatives and the operator picks one by selecting it. Menus are very helpful for commands that are not performed often, like "Print pages i through j" or "Find a text string".

An entry in a menu can have space for parameters to be typed in. In a prefix system, the logical thing to do would be to select the command first, then type in the arguments (if any have changed from last time) and then confirm the command. In a postfix system, type in the arguments first (if any have changed) and then select the command.

[The one-dimensional analogy to the menu is prompting. For example, after typing "Find (return)" the program can prompt with "key=".]

3.4 Some recipes.

What commands should you have?

The only command that almost every editor offers is "Delete". It is a good idea for the program to save the most recently deleted passage in a special place, and to offer an "Insert" command to put that passage back in the text before the specified selection. Such an Insert command not only can be used to undo blunders but also provides a way to move things around in a postfix system.

A problem with "Insert" is that one may hit "Delete" twice in a row by accident and thereby loses the ability to undo the first Delete. This can be solved by storing the last two deletions and allowing two inserts in a row to bring both back. More than two are of marginal utility (even two are a luxury).

Another useful command is "Copy", especially if the editor is to be used for writing programs. In a postfix editor, Copy should be like Delete in that it puts a copy of the selection in storage after which Insert can put it somewhere else.

For typing in text in a postfix system, it is best if *no* command need be given. After making a selection, the operator should simply be able to type in the normal manner, and the typing should be inserted at that spot. Requiring a command is prone to error; since people are used to typewriters, they may forget to give the command.

Speaking of typewriters, most people find an interactive program easier to use if it works a lot like a familiar machine. Thus, to retype the preceding character, the operator should be expected to type "backspace" (assuming there is a key that can be so labelled), not some arbitrarily chosen key.

Unfortunately, it is not always possible to design an editor so that typing can be done without a command. With a terminal lacking control keys [or with a one-dimensional editor], it may be necessary to use the typing keys for specifying commands. In that case, a command will be needed to enter "insert mode" and some stroke will be needed to leave the mode. Many editing errors result from striking keys without first getting into or out of insert mode; so avoid having modes if you can.

A major problem with typein is what to do when the line fills up. Some editors act like a typewriter; a bell rings and characters soon get discarded. Others allow a line to be quite long, in which case there is generally a way to see the whole thing a little at a time. Still others automatically move the last word of the line to a program-created line below. In some editors of the last kind, a subsequent Delete command may cause words from created lines to move back up. Some people like this arrangement (called "paragraph editing"), but many are confused by it, even with a display. Furthermore, it is hard to implement, so I recommend skipping it on your home computer.

If you don't implement paragraph editing, you may still want a way to specify that a group of lines is a paragraph. This can be done every time a command is to be performed on it, or it can be done at any time, whether by a "Start paragraph here" command, or by establishing conventions, such as a blank line between paragraphs. It depends on what kind of text you will be working with and what you will be doing to it.

If you are editing long texts, rather than just short inputs to a dialog program, then other commands you may want are:
 Find next occurrence of a text phrase
 Change all occurrences of one text phrase to another ("substitute")
 Indent/Unindent paragraph's first line/non-first-lines/all lines
 Retype paragraph with as many words on each line as can fit
 Retype and insert extra spaces to justify each line but last to right margin
 Retype and insert extra spaces to center each line between margins
 Paginate every n inches
 Print pages i through j, or all pages

There should be a way to abort a command, at least if it is time-consuming, such as Substitute or Print. The keystroke chosen for this purpose should be uniform regardless of the command or the mode.

You will probably want a way to move text between files. A sample set of commands for this purpose are Include and Extract. Each takes a file name or number as an argument. The Include command inserts the whole text of the named file at the position selected in the current file. The Extract command moves the text selected from the current file to the named file. Although faster commands can be designed, these are both easy to implement and easy to use.

A fancier way to deal with multiple files is by dividing the screen into "windows", left and right or above one another, and displaying a different file in each window. The "current" window ought to be clearly distinguished to avoid confusion. In a postfix editor, there should be only one selection on the whole screen, not one in each window, otherwise, many people get confused.

You will want a way to browse through the text. The easiest way to implement browsing is to provide page turning, but it is nicer for the operator to scroll a line at a time. Fancier systems allow browsing by headings or through a table of contents. If you do work by pages, then there is a problem when a page fills up that is the same as that mentioned earlier in connection with lines filling up. The possible solutions are analogous.

There must be a way to move text from the floppy or cassette to the editor and vice versa. There are two philosophies about when to write an updated file back on a floppy disk. One possibility is to work it as you would with a cassette, namely, require the operator to issue a "store" command, either specifying a file name or address, or defaulting to the same file or sector as before, or defaulting to a higher numbered version of the same file. Another possibility is to update "continuously", whenever there is a pause in operation, or whenever a certain amount of work has been done, or whenever that page of the text has been scrolled off the display. This technique provides safety against machine failures or later operator errors. However, it is a good idea for the program first to back up the original version of the text on a temporary file.

If the text you will be editing will often be computer programs, it is nice (if the operating system supports it) to have an "assemble" or "compile" command that writes the text on a file and causes the system monitor to start the assembler or the compiler. Of course, any good interpreter should have a text editor as its front end to enhance the direct execution feature.

3.5 Feedback

When using an editor with either modes or prefix commands, operators often get confused about what they just did and what they can do next. In a two-dimensional editor, you can reserve a "feedback area" in which you announce these things in canned English. If you don't have time to display them continuously, it is just as good to display them only when the operator pauses for at least a second, and almost as good to display them only on command. [In a one-dimensional editor, commands can be abbreviated and the program can expand them to supply feedback, but careful: operators get confused when the terminal types in and out at the same time.]

One of the characteristics of a computer editor is that invisible characters like (space) and (return) are usually represented by character codes just like visible characters. This is nice for the program, but when a naive operator is faced with joining two words together, he or she may not understand the concept of "deleting the space in between". It makes no sense, really, to delete something that is not there! Even experts often have trouble telling an invisible character apart from the absence of text.

There are two solutions to this problem, which can be used separately or together. One is to give feedback. Either always, or upon a special command, invisible characters can be replaced by special visible ones to let the operator tell them apart. Depending on your display controller, you may be able to show the difference in other ways (gray background, for example). An easier solution is to let selections be made in empty space on the screen. When typing is done there, have the program surreptitiously insert sufficient spaces before (and sufficient returns above) the selection to give the operator the impression that they were there all the time.

4. Program Design.

4.1 General Considerations.

The amount of "primary" (fast) memory in your machine limits both the program size and the amount of text you can handle quickly. Even if there is not much room for text in the primary memory, you can keep the text on a floppy disk or even on a cassette, except for the page or even for the few lines that the operator is currently dealing with. It is also possible to save program space by putting special features of your editor onto the floppy and bringing them into primary memory only when needed.

The operating system or program library usually provides services useful to a text editor. For example, there should be a buffered keyboard input utility so that the operator can "type ahead" while the system is performing a time-consuming operation.

If the secondary storage is organized into named files, use of the system is simplified. Otherwise, files can be referenced by number or by starting address. Another way to let operators specify files is through a menu. In that case, each file can be identified by a string of text that describes its content rather than by a "name". Even if two files have the same description, they are distinguished by having separate entries in the menu. This alternative is especially attractive if no file system is available and you must provide access facilities yourself.

4.2 Maps.

Editors really aren't that hard to program, so I won't give you much advice. The hardest aspects to deal with are the variable number of characters in lines, the variable number of lines in files, and (in a two-dimensional editor) the continuous update of the display to reflect accurately the current state of the text.

Most solutions to all these problems rest on the concept of a *map*. A map in the abstract is a list of pairs which sets up a correspondence between elements of a *domain* and elements of a *range*. For example, there could be a map from each line number to the memory location of the first character of the corresponding line; a map from line number to screen y-coordinate; a map

from paragraph number to starting line number; a map from command name to subroutine location; and so forth.

In concrete terms, most maps have special properties that let them be implemented more efficiently than by a list of pairs. The simplest implementation is a pair of arrays of equal length. If the domain is simply the consecutive integers 1...N, then only the range array is needed. If the range is only defined for domain values from M...N, then no space is needed for the undefined values, as long as the values M and N are available to help access the map correctly.

If the domain is ordered, you can use a "binary search" to search the table for a given value in $\log_2 n$ steps. If the domain values are unique, you can use a "hash table". These techniques can be learned in any good text book on programming, e.g., [2]. However, a straight linear search is preferable on a small machine unless you can demonstrate that it is causing a performance problem.

A map can be inverted so that each line becomes a record whose fields specify the range elements to which certain maps map that line.

The reason to think of all of these diverse data structures as maps is that your program can be simplified if you use a uniform subroutine calling convention to look up in maps and modify maps, regardless of the way they were implemented. One advantage is that if you improve the implementation of a map, the code that uses it won't have to change. A more subtle advantage is that the program will be easier to construct, understand, and improve because of the uniform application of a general abstract concept.

4.5 Text Representation and Display Update

On a system with 8KB of RAM, the program code of a typical editor written in assembly language occupies 2K to 4K. Of course, to achieve such a small program, you must be sparing of features and careful in coding.

There are two basic approaches to the use of the 5K or so of leftover space. The simplest one is to store the current page of text right in the RAM, and to keep the data structures small enough so the largest page you'll ever need will fit. The more complicated approach is to cache only a small amount of text in the RAM, and to leave the rest on secondary storage. Some of the extra RAM space will now be needed to support the caching mechanism, but some may be gained for programming additional features.

The caching approach has the advantage of allowing "pages" of great length to be handled. However, it will either be sluggish or complicated (or both), depending on whether the secondary storage is disk or tape, and whether the operator processes the text sequentially or jumps around a lot.

On a home machine, it is generally better to restrict pages to about the size that can be displayed in one screenful, i.e., 1K or 2K bytes. This eliminates the need for scrolling within a page, as long as commands are supplied for page merging, dividing, and flipping. Furthermore, the text can be stored as a single contiguous string to which any edit can be performed faster than the operator can type.

With this approach, the map for display updates maps line numbers on the screen to starting character positions in the text string. Every edit must update the map as well as redisplay those lines that are affected. Note: Some displays can not redisplay a line without clearing all the lines below it. If this is the case with your system, defer screen update during typein until the keyboard input buffer is empty.

4.4 Reliability

It's never pleasant when a program fails, but when an editor fails, the operator (e.g., you) often attains new levels of agony. It seems as if the last three hours of work were just lost, and irreproducible work at that. You may find the CRT reduced to fragments of glass on the floor.

Consider providing means of backing up files, checkpointing the program, recovering from crashes, and so forth. In critical parts of the program, put validity tests for data structures and for important values, like disk or tape addresses. You can't fit too much of these aids in a small computer, but a little work in this area may save you a lot of distress later.

Careful command design can help, too. Make it more difficult to delete a whole file than to delete a paragraph, and more difficult to delete a paragraph than to delete a character. Avoid situations where if the operator is looking away from the screen, a couple of stray keystrokes can cause disasters.

When using your editor, don't go too many minutes without writing the text on a file, or too many days without getting printouts and backing up files. If the editor is new and still has lots of bugs, exercise these precautions more frequently.

5. Conclusions.

Designing a program is like writing a story: the possibilities are endless. It is helpful to establish guiding principles in the design to constrain it in a

manner appropriate to one's goals.

In the design of an editor, I recommend that you consider hardware limitations, operator habits, how much awareness the operator can devote to the system as opposed to the task at hand, and requirements for speed and reliability.

The Author

The author was turned on to computers in high school in 1960. He helped finance undergraduate work at Stanford by programming in such areas as text processing, simulation, statistics, graphics, and compilers; ran a freelance software company in Palo Alto for five years; then was a research assistant at the Stanford Artificial Intelligence Lab doing cognitive computing, natural language understanding, higher level languages, and text formatting. Somewhere in there he took off a year to live in rural Oregon and develop other interests such as folk music, cooperatives, and extra-sensory perception. For the past four years, he has been a member of the research staff of Xerox Corporation, specializing in text processing, user interfaces, and programming systems. He likes programming as well now as in 1960. He has never learned to operate a soldering iron.

References

- [1] F. J. Gribble, "A Classy 8080 Text Editor", in *Dr. Dobb's Journal*, Vol. 1, No. 6, June/July, 1976.
- [2] D. Knuth, *The Art of Computer Programming* (especially Volume 3), Addison-Wesley, 1973.

LEARNING TO PROGRAM MICROCOMPUTERS? HERE'S HOW!

R.W. Ulrickson, President
 Logical Services Incorporated
 711 Sierlin Road
 Mountain View CA 94043

ABSTRACT

There is plenty of microcomputer hardware, but not nearly enough software or programmers. Learning programming is easy if you use a systematic and consistent approach. Top down design and structured programming can help you learn to design software in any language and on any machine.

SOFTWARE IS WHERE IT IS AT

There is so much microcomputer hardware available now that the mind boggles deciding what to buy or build next. At this show alone there are a dozen or more CPU's to choose from, several microcomputers for each CPU, and hundreds of memory devices. The list of peripheral and I/O devices is growing by leaps and bounds. The paper that describes all this hardware literally outweighs the hardware itself. What's missing? The SOFTWARE! It may not be completely missing, but it sure is minimal. In spite of recent advances, software for microcomputers is still in its infancy. It takes a lot of time and money to develop good software, and right now there are not enough qualified programmers to keep all the microcomputers busy or even begin to pursue all the potential applications and uses for microprocessors. You can't get much use out of a microcomputer unless you can program it.

Even if you never intend to get a job as a programmer or to use software in your work, learning software design can bring you a wealth of pleasure, convenience, security, and savings in real dollars. If you can program a microcomputer, you can buy some inexpensive hardware and use it in your home to amuse yourself and the kids with computer games; do your checkbook and income tax; keep track of your recipe files; control your heating and energy usage; protect your home from potential fire or theft; answer your telephone; keep track of your appointments; control your appliances; and educate the whole family. The list of potential software applications goes on and on. But, it's going to be a long time before standard libraries of fully documented programs are available to anyone who wants them. Even then there is some doubt that they will be compatible with your system without some tricky modifications. The answer is to learn software design. But how?

Not only are there not enough programmers today, but there is a shortage of decent training and educational courses for all the hardware people who want to add software design to their list of skills. That's the software gap. What can be done about it?

WHERE ARE YOU?

Last year at the Personal Computing Show in Atlantic City, then again at the Mini/Micro Systems Show in San Francisco, we took an informal survey. We asked everyone who came by our booth, "Are you a programmer?" There were a wide variety of answers starting with "knowing glances" and comments about having three degrees in computer sciences and 20 years programming experience. At the other end of the spectrum was one guy who said he was not interested in software, only the hardware. We wondered what he was going to do with the Altair he just bought. In between, most people said they were programmers because they took a course in FORTRAN 10 years ago, or read a book on BASIC recently. The point is, everybody is in a different place as far as computer knowledge and experience are concerned. The guys with all the degrees are the professionals. The others were converted to programming from some other discipline or decided to take up personal computing as a hobby. The pro's had the benefit of years of study, the converts have to find some other way to learn programming. One of the reasons for the wide variation in software knowledge is that programming has been considered a black art involving some kind of mysterious incantation that the high priests never revealed to the uninitiated. Until recently, hardly anybody was willing to tell you the secrets, consequently everybody re-invented the wheel when he wanted to write software. This is a real shame, because software design is remarkably similar to hardware design -- if you learn it right in the first place. The secret is simple: Be systematic and consistent in the way you design software. That's what the pro's do, and their successful techniques are even more appropriate for the hobbyist.

THE RIGHT WAY TO DESIGN SOFTWARE

Software design has two parts, strategy and tactics. The strategy part is what used to be called the scientific method, or the systems approach. It consists of defining the problem carefully, analyzing it thoroughly, then breaking it down into manageable functional blocks, and developing step by step algorithms for each of the resulting blocks. The strategy part of software design works for any language and any computer.

The tactical part of software design consists of implementing the algorithms for each block using a specific language and a specific computer, then testing and debugging each block before putting the blocks back together.

TOP DOWN DESIGN-STRATEGY

Top down design is a systematic strategy for doing software design. You can use this fundamental strategy to solve almost any problem, and you will never have to worry about being stuck with one particular language or one particular computer.

The first step is defining the problem. You should spend a lot of time on this step. You are not finished until you have such a complete specification of the problem that a skilled programmer could take your spec and solve the problem with no questions asked.

The next step is partitioning the problem into its major functional blocks. These may consist of input and output operations, data files and tables, control structures, and various processing packages. Each major functional block can then be partitioned into finer and finer blocks. The level of detail for your finest blocks depends on your experience. The beginner should not hesitate to block down to a level where the process performed by each block is almost trivial. You will often find that detailed blocking makes the algorithm for solving the problem obvious. This is not always the case; some algorithms require a great deal of creativity and ingenuity. When partitioning and algorithm development are complete, you can reassemble the blocks in the form of a diagram or flow chart of each major functional block. By this time the solution to the original problem should be clear and you will have documented much of your work with your diagrams. The fact that you have not yet written a single line of code in any computer language is crucial to good systematic design. Some of the greatest disasters in software design have occurred because the programmer became bogged down in coding problems before the real problem was thoroughly understood and solved.

BOTTOM UP IMPLEMENTATION - TACTICS

Even the most systematic design procedure will not insure a bug-free program. Bugs seem to creep into all programs, even those that have been running for years. The cost of program maintenance usually decreases as the program matures, but later on the maintenance cost can be observed to go up again. This happens when the fixes to one set of bugs generates a whole new set of bugs. The problem becomes acute when "creeping featureism" sets in. Famous last words: "As long as we're fixing this problem, we might as well add this groovy new feature". How can the high cost of debugging be reduced? Consistent tactics for implementing program code can go a long way toward eliminating bugs.

STRUCTURED PROGRAMMING

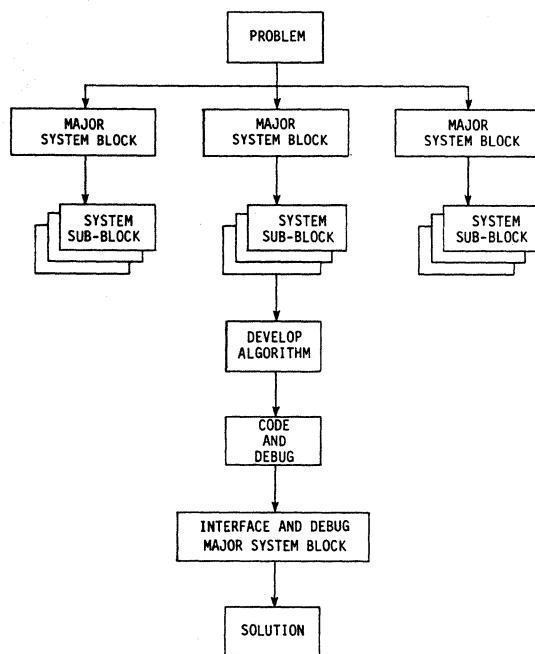
Structured programming means different things to different people. To me it is just a consistent way of implementing programs using a few fundamental structures that have proven to be useful, convenient, and reliable. Some higher level languages like ALGOL and PL1 were based on the concept of structured programming, but the same structures can be used in assembly language programming as well. The cardinal rule in structured programming is that control enters a structure at exactly one point and leaves at exactly one point. This avoids complex transfers of control that can lead to hairy problems in debug.

A few simple structures having this fundamental characteristic can be combined to form programs of any complexity, and your programming will be more consistent, easier to understand, easier to document, easier to change and maintain, and easier to debug. The result is faster and less expensive program development. As usual, there is no such thing as a free lunch. Restricting yourself to a few basic structures can result in slightly longer programs than tricky techniques that minimize code. The savings from a consistent design approach almost always outweigh the cost of a few extra bytes of code.

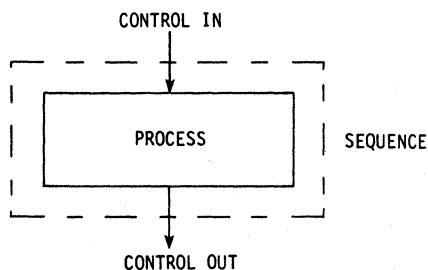
STRUCTURES

The seven basic structures shown in the figures fall into two classes: open structures and closed structures. The open structures provide no means to transfer control back to the entry point of the same structure, while closed structures provide such a path so that a sequence of operations can be performed more than once. In a closed structure or "loop" control remains in the structure until a terminal condition is met. The four open structures are called SEQUENCE, IF-THEN/ELSE, IF-THEN, and SELECT-OPERATION. The SEQUENCE structure simply accepts control, performs a process, and transfers control to the next structure. IF-THEN/ELSE implies a decision element that transfers control to one process or another based on some condition. IF-THEN tests a condition and performs a process if the condition is met or transfers control without any action if the condition is not met. The SELECT-OPERATION structure allows you to select one of N different operations based on evaluating some expression, instead of a simple true/false condition. It is often called the N-way branch, although control is always returned to the common exit point.

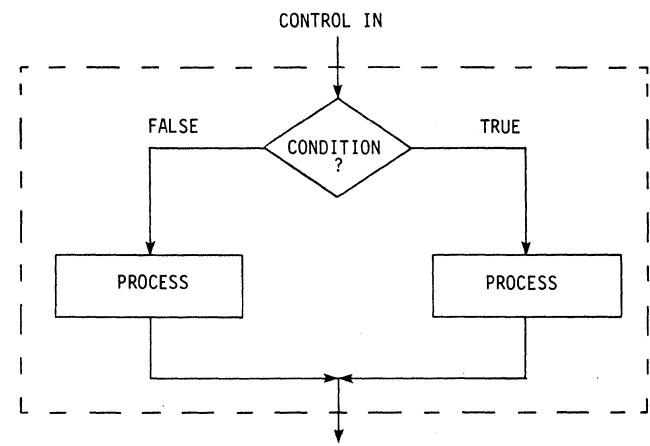
The closed structures, DO-WHILE and REPEAT-UNTIL are loops that differ only in whether the condition is tested before or after the first process is performed. REPEAT-UNTIL executes the process at least once. PROCESS-WHILE is a powerful combination of DO-WHILE and REPEAT-UNTIL where two processes are performed in the loop.



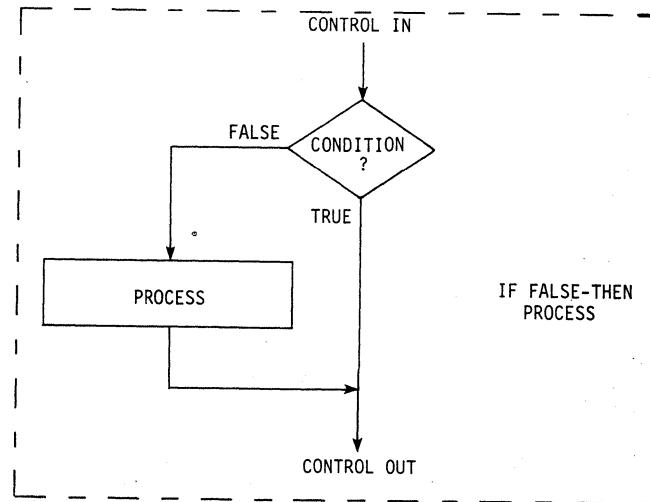
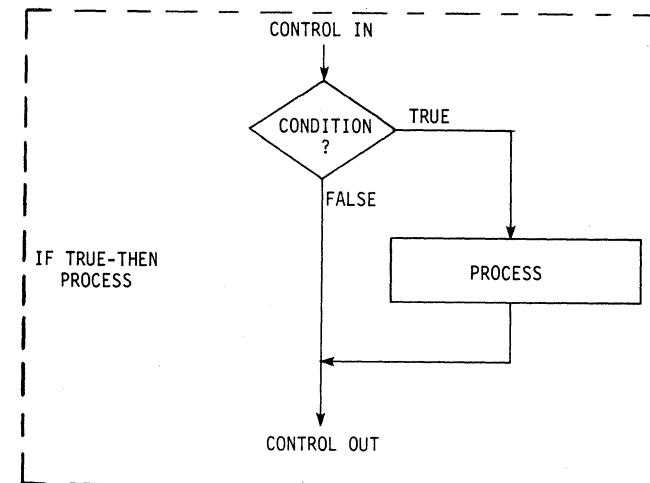
TOP DOWN DESIGN



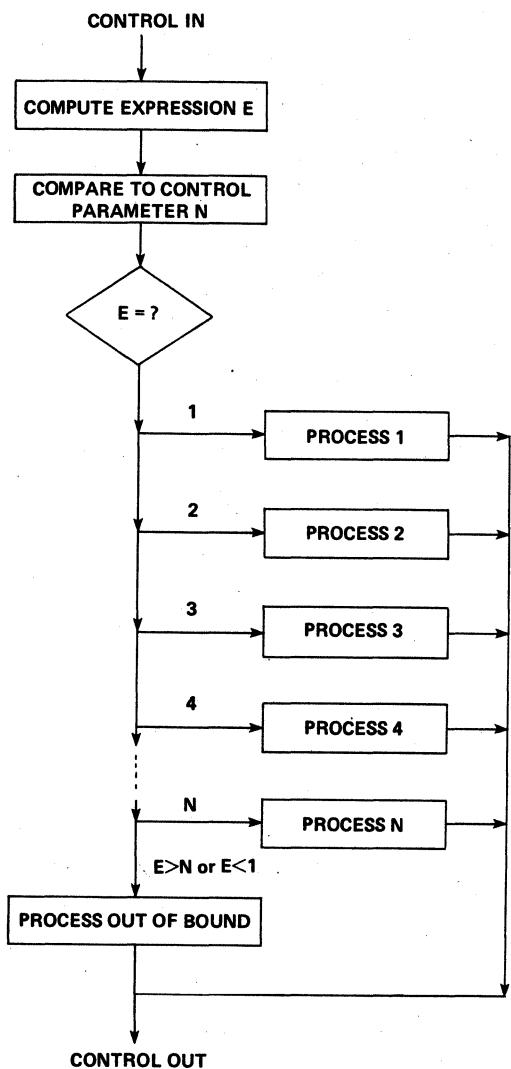
SEQUENCE STRUCTURE



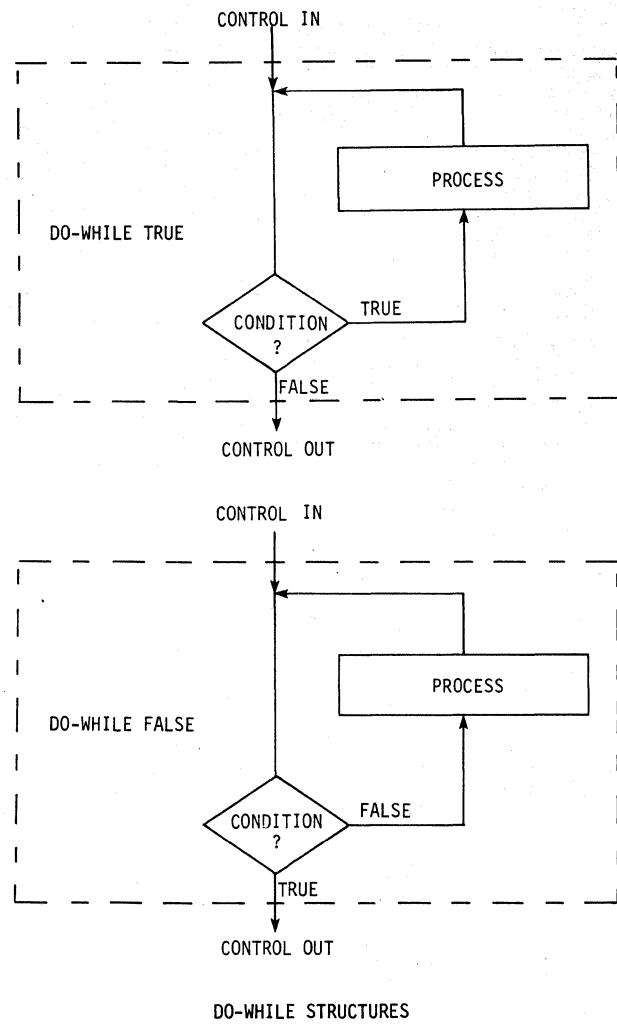
IF-THEN/ELSE STRUCTURE



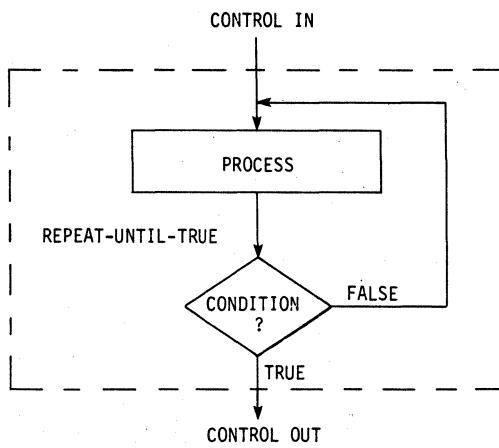
IF-THEN STRUCTURES



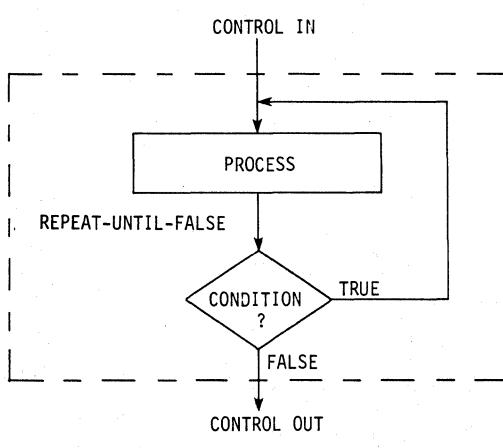
SELECT-OPERATION STRUCTURE

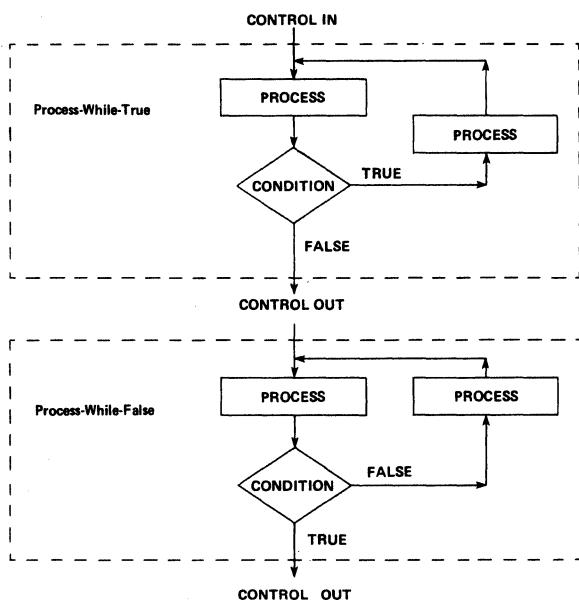


DO-WHILE STRUCTURES



REPEAT-UNTIL STRUCTURES





PROCESS-WHILE STRUCTURES

INTEGRATION, TEST, AND DEBUG

The tactics of programming continue with integration of all major blocks into a complete program that runs in your microcomputer. Testing and debugging can be formidable tasks especially if your system must operate with real time I/O and interrupts. There is a lot more to learn about good software design. Such topics as Input/Output operations, arithmetic operations, debugging techniques, documentation methods, and higher level languages require knowledge and skills that you will develop while learning the basics and practicing with your own application programs.

JUST GETTING INTO SOFTWARE?

If you are just getting into software or trying to decide whether to take the plunge, you have several options for learning how to design software. You can go back to school or take college courses in computer science. You can take one of the intensive short courses offered by the semiconductor manufacturers or the professional training organizations. You can read a lot of books, or take a self-study course. Or you can just hang around the local computer store and hope for the best. No matter what method you choose to educate yourself for the microcomputer revolution, and whether you are into software for fun or profit, think about the advantages of systematic top down design. Take the time to learn consistent and reliable methods like structured programming for implementing your designs. And be sure to practice your new skills on actual microcomputers.

Then as you learn more languages and become familiar with different machines, you will have a firm foundation in systematic and consistent design techniques that will give you better software and greater satisfaction.

RESUME

Robert W. Ulrickson has been President of Logical Services Incorporated in Mountain View, California for four years. Logical specializes in microcomputer hardware and software design and offers a self study course in software design called Modu-Learn™. Prior to founding Logical, Bob held management positions at Fairchild Semiconductor in Digital Marketing, Applications Engineering, Custom MOS Engineering, Computer Aided Design, Test Engineering and New Product Design.

References: Modu-Learn™ microcomputer programming course,
© 1976 Logical Services Incorporated, Mt. View, CA 94043

STRUCTURED PROGRAMMING FOR THE COMPUTER HOBBYIST

Ed Keith
655 E. Orangecrest Road
Glendora CA 91740

In this paper I shall try to provide a basic understanding of what structured programming is and describe a planning technique called a hierarchy diagram. I will also demonstrate the five basic logic constructs.

WHAT IS PROGRAMMING?

Before I can talk about structured programming for anything, I need to find out what programming is. Glancing through the encyclopedia between Profit Sharing and Propaganda (an interesting juxtaposition by the way) I find a void for this term. Oh well, as the ancient commercial says: "Ask the man who owns one!" So I proceed to ask myself "What is programming?" My first and easiest answer is "Writing code for my computer." (I like to think of it as a computer, not a microprocessor.) But then I stop and feel the need to amplify the words "writing code", perhaps I should have said "writing good code", for this seems to be my goal whenever I program (certainly no one wants to write bad code!). This answer doesn't totally satisfy me either since it requires definition of the term "good code". What is good code? Some people will say "working code", if it works it's bound to be good. But this may not be totally true. Just because it works doesn't mean it works well. It might work very slowly or require 20K storage. There are a lot of constraints to the definition of good code. To me, good code is two things above all else: Flexible and clear.

Flexible in that no program is ever truly finished. When it was first written a program might have used a serial interface for a teletype but now you have a parallel interface to a new line printer. If the program can be easily modified for the change in I/O devices and communication interface, it was probably written well. Inference: Plan for change. Capt. Grace Hopper feels that 90% of all computer programs that print a date will malfunction on January 1, 2000. The original design probably did not encompass a century change.

Clear in that to be able to modify a program you must be able to read it to find out what it is doing and when and where it does it. Imagine the excitement as thousands of programmers madly search thousands of programs trying to discover where the 19 comes from. Inference: Plan your programs to be read. Perhaps the best clue to what structured programming is can be found in the word plan. Programming a computer is not simply writing the code. It is thinking about it first.

HIERARCHY DIAGRAMS

In planning a computer program one of the better tools to use is a hierarchy diagram. A hierarchy diagram resembles a company organization chart in that it shows functional relationships between its elements rather than logical ones. Hierarchy diagrams are an organization tool. They help you to plan the programs structure and should be done before any flowcharting. (Of course most things are done before flowcharting! That usually includes writing the program!!) A hierarchy diagram is usually referred to as a tool in top-down program design, probably since it is drawn top-down. First begin with the name of the program written in a rectangle at the top center of a page as shown in figure 1.



Figure 1

In demonstrating these concepts I'll plan a program to play the newly popular game of Othello, either using the computer as a board for two human players or letting a player enter into combat with the computer itself. I'm going to extend the hierarchy diagram by writing in the major components of the game as it is played by humans. Obviously one must set up the board, play until done, and close play. These three components become the next level in the hierarchy diagram as figure 2 shows.

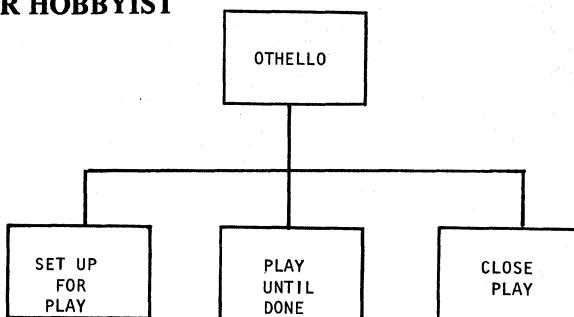


Figure 2

As you sketch out your hierarchy diagram you will be forcing yourself to concentrate upon the program and organize the logical processes involved in making the computer accomplish the task. To develop the next level in the hierarchy diagram select the leftmost box and ask yourself, what are the components of this task. In my thought processes I decided that for an initialization step the computer should print some opening messages, initialize the playing board, discover whether it was to simply keep the board or actively compete and lastly, get the player's names. The second level for the PLAY component involved getting and making a move for the first player, getting a move for the second player, generating and making a computer move and seeing if any further moves are possible. This seems a good place to emphasize that hierarchy diagrams show function not logic. At this level the computer program must be capable of handling all these functions. The logic of the program will determine which functions are executed and in what order but the hierarchy diagrams must show all functions. The final phase of the program will be encountered whenever there is no further move for either player. It involves the determination of the winner, the printing of win and loss messages and a replay query. All the functions developed so far are detailed in figure 3.

The hierarchy diagram is drawn with lower and lower level getting more and more detailed, but it should stop just prior to the level at which the components become actual lines of code in a programming language. After each level is complete, return to that level and ask yourself whether this component can be further subdivided. If so, the subdivisions become the next level in the hierarchy. Figure 4 shows the complete hierarchy that I drew for this programming application. Since hierarchy diagrams expand quite rapidly it is best to start with the long edge of a page as the top. In fact, I tear extra long strips of my teletype paper for most applications.

Once the hierarchy diagram is complete, it should be reviewed for complex logical relationships. If any occur, they should be flowcharted before the code is begun. But, rather than draw one big flowchart it is best to draw several smaller ones. For instance, the first level in the PLAY section could be flowcharted. You must decide how the computer will determine the number of players, whether it is to make a move or not. This will probably mean some sort of flag will have to be set in the 1 or 2 PLAY routine which will be utilized here. From a complete hierarchy diagram you can usually determine these kinds of relationships. In fact you must actively seek them out and make notes about them before you begin to code. Nothing will destroy a programmers concentration more quickly than discovering that a control structure in one area of code depends on some variable from an earlier, already coded section, and not having foreseen the necessity for the variable in the earlier section. Perhaps it is now clear why THINK FIRST should always be the programmers motto. To THINK is human, to THINK FIRST, divine.

PROGRAM LOGIC CONSTRUCTS

To return to the divinity of structured programming requires that we next examine the logical structures that are generally accepted as the standard for use in programming. There are five accepted structures. These are:

1. Sequence
2. IFTHENELSE

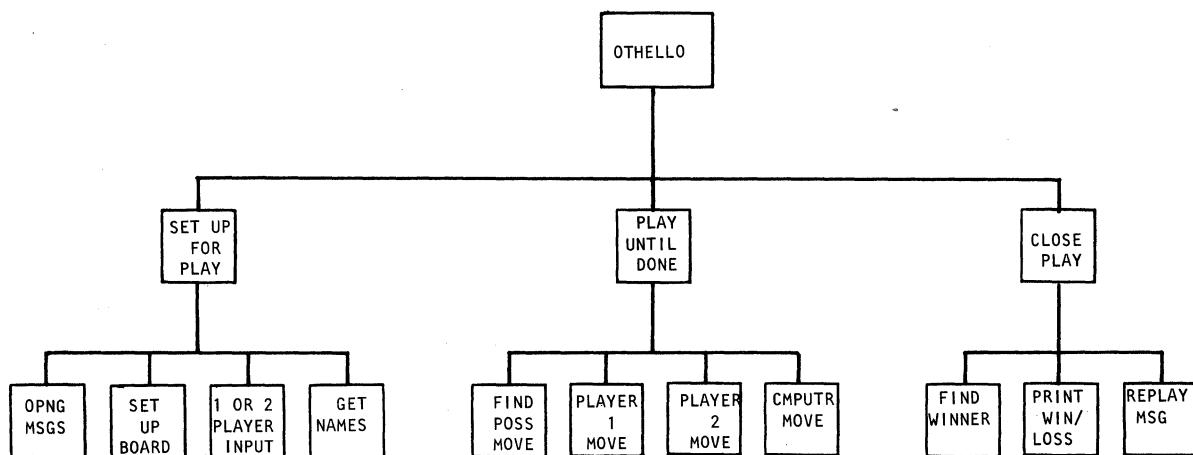


Figure 3

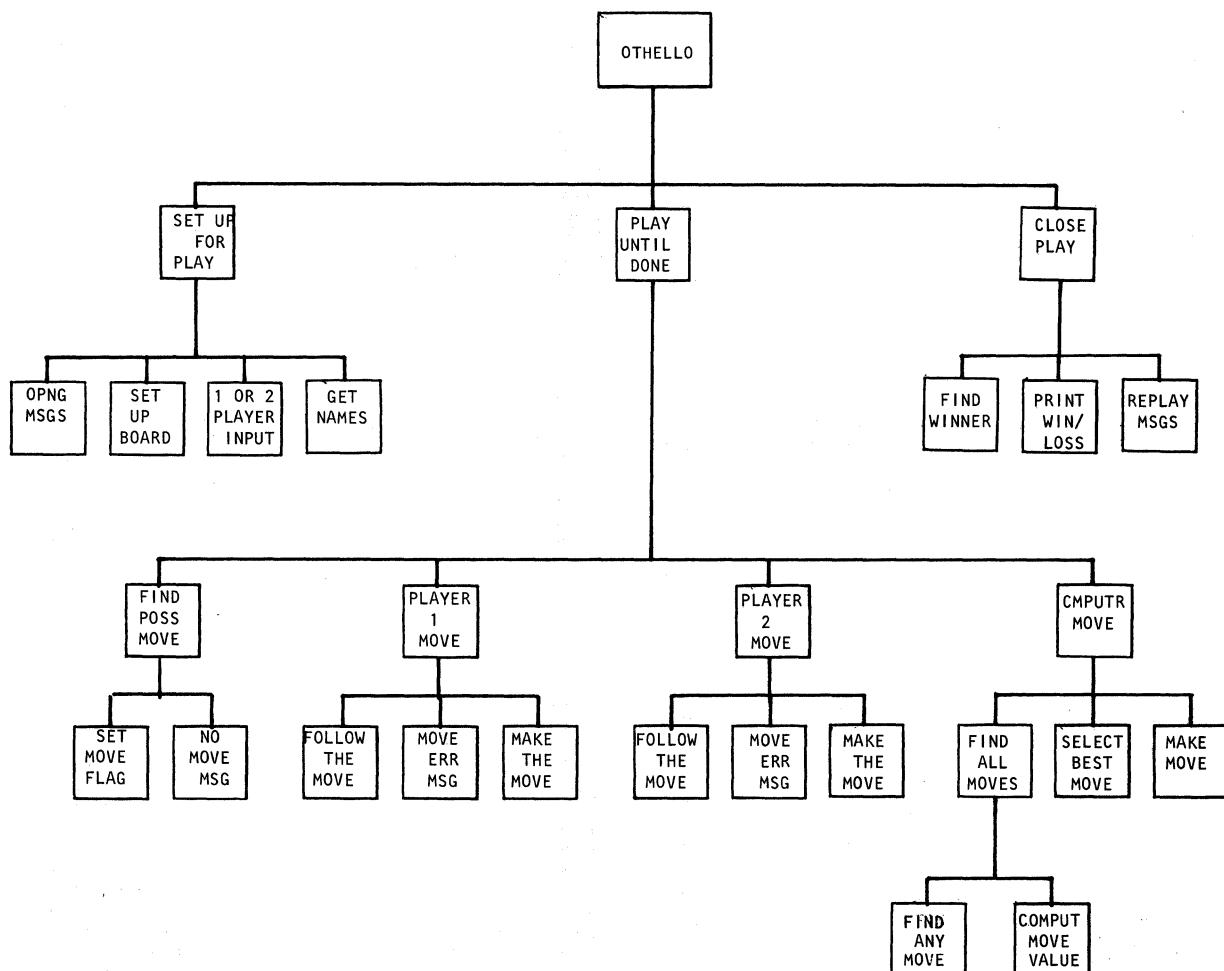


Figure 4

3. DOWHILE
4. DOUNTIL
5. CASE

SEQUENCE

Structure 1 is probably the most important. Sequence means straight line coding as in Figure 5 - no fancy branches to share one line of code between several routines. It will also help if you limit all your routines to no more than fifty lines of code together on a page. It should be further stressed that sequence means that routines should have one entry point and only one exit. This makes them more easily readable.

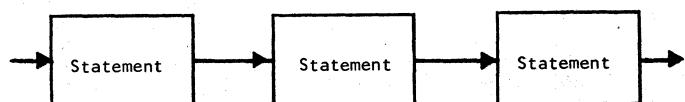


Figure 5

The other four structures are more complex logic structures that name and describe a basic logic case.

IFTHENELSE

The IFTHENELSE structure as shown in figure 6 tests a single condition to determine which of two statement blocks will be executed. Alternatively the ELSE section may be omitted. For example in Othello, IF it is the first player's turn THEN the first player moves ELSE the second player moves.

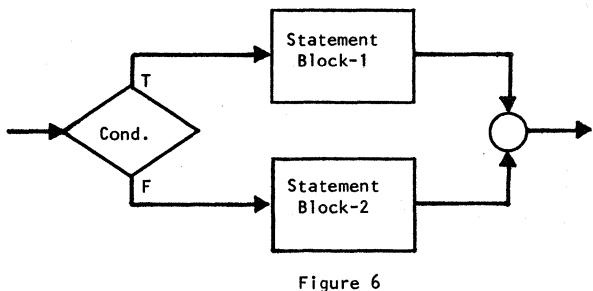


Figure 6

The IFTHEN as shown in figure 7 is simpler and has only one statement block that is bypassed when the condition is false.

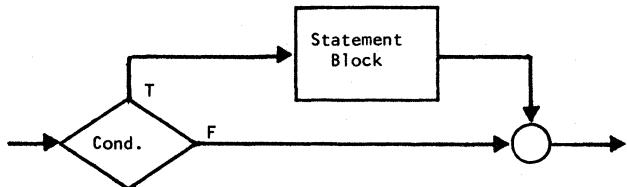


Figure 7

DOWHILE

The DOWHILE structure, shown in figure 8, provides the basic loop capability. A series of statements are repeated while some condition is true. The condition is always tested before each execution. In our game we wish to DO the play routine WHILE there is a valid move left.

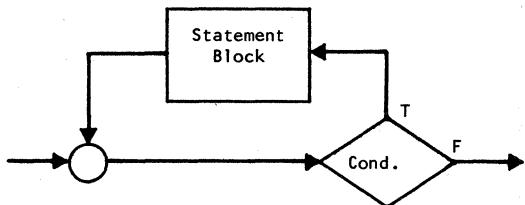


Figure 8

DOUNTIL

The DOUNTIL structure shown in figure 9 differs from the DOWHILE in that the test for continued execution of a block of statements occurs after the statements have been done. In the program called Othello the computer will DO the search for a move UNTIL all moves have been found.

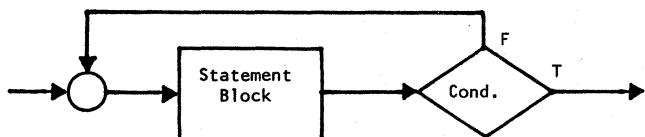


Figure 9

The last basic structure is called the CASE structure and is used when it is required to select from a series of alternatives depending on the value of a variable. In a more complex game the player might enter a numeric code i having a value from 1 to N where this indicates a choice of N alternative. Figure 10 shows the CASE structure for the preceding problem.

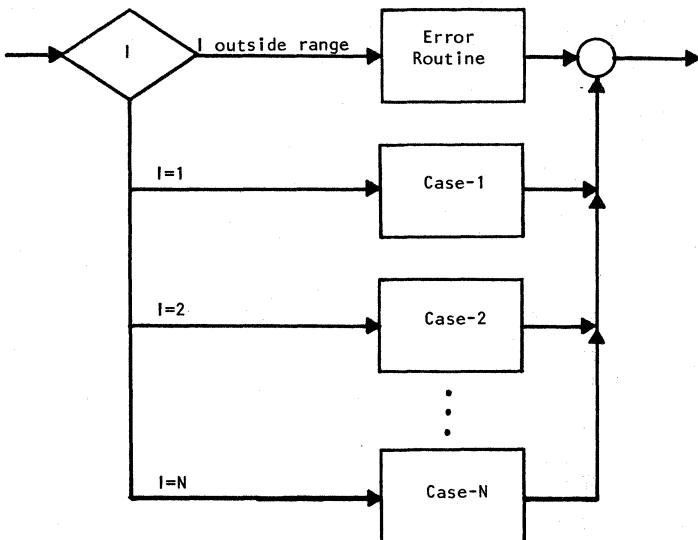


Figure 10

Note that my flowcharts are drawn left to right, top to bottom, as one would read. Sometimes this makes flowcharts easier to understand than the usual top to bottom format.

IMPLICATION FOR PROGRAMMING

Most of the languages currently available to the computer hobbyist do not contain the logic constructs other than SEQUENCE. Although this is not a happy circumstance it is one that we must exist with for a while. We can take positive steps to overcome this problem. For instance, in ASSEMBLY language coding we can keep our routines simple and annotate each line of code. In BASIC we can simulate the basic constructs and use indentation to show logic and we can include remarks to clarify our blocks of code. But, perhaps more importantly, we can structure our experience with a program by remembering to THINK FIRST.

AN INTERPRETIVE APPROACH TO PROGRAMMING LANGUAGE IMPLEMENTATION

Dennis R. Allison
Independent Consultant
169 Spruce Avenue
Menlo Park, CA 94025
(415) 325-2962

Abstract

An interpretive approach to the development of high level language compilers for microcomputers is described in the context of compiling Tiny BASIC. Using this approach, an interested hobbyist can experiment with language design and build useful tools.

0. INTRODUCTION

Few compilers have appeared for hobby use, and those that have are usually expensive by hobby standards. The economics of the market is such that entrepreneurial development of substantial software systems cannot be supported. The hobbyist who wants to have a compiler for his system must build it for himself.

The interpretive approach described here is specific to Tiny BASIC but can easily be modified to handle other languages. It provides for one-pass compilation of source code to unoptimized object code. The parsing scheme is the traditional top-down recursive descent technique used in many production compilers.

In this paper we develop the structure and some of the components needed for compilation of Tiny BASIC. Many of the choices made here are dictated by simplicity of exposition. Compilers, even simple compilers, are very complex programs and are difficult to explain.

We expect to complete and test the compiler described here and publish it in Dr. Dobb's Journal sometime later this year.

One final cavaet: this is a paper system and has not yet been implemented. While there is not reason to expect significant errors, the implementor should be wary.

1. TINY BASIC

Tiny BASIC has been described in detail in Dr. Dobb's Journal, Volume 1. It is a very minimal dialect of BASIC with 16-bit integer variables and a very few statement types. Interpretive implementations usually require about 2K bytes of storage.

A quick summary of Tiny BASIC features and syntax is given below.

```

>> Integer arithmetic, 16-bit
>> 26 scalar variables: A, B, ..., Z
>> Eight Statement Types
    LET var = expr
    PRINT expr-list
    INPUT var-list
    IF expr relop expr THEN statement
    GOTO line-number
    GOSUP line-number
    RETURN
    END
>> Parenthesized expressions
>> Standard operators: +, -, *, /
>> Print list components may be strings or
                           empty

```

It is assumed that all program lines have the format line-number statement and that line-numbers are in increasing order.

This language is actually smaller than the languages many people have implemented as Tiny BASIC, but is more than adequate for our purposes. It is left as

an exercise for the reader to extend the language to something useful.

2. THE TARGET MACHINE

The target machine for our compiler is the Intel 8080. It was chosen not so much for its capability or architecture as for its market position.

We could have equally well chosen any other machine as a target. In particular, we could have chosen to generate code for a hypothetical machine which is particularly good for executing Tiny BASIC. We would then have to write an interpreter for this hypothetical machine's order set and pay for some additional overhead at execution time, but the overall job would be easier. In addition, the system would be much more portable.

3. THE INTERPRETIVE COMPILING MACHINE

The compiler itself is implemented as an interpreter. The interpreter consists of a set of data structures and operations on them. These are encoded in some fashion and a program written to interpret them. The capabilities of the compiler are thus described by the semantics of the interpretive op-code set and the data they can reference.

Data Structures

The compiling machine maintains a number of high level data structures to accomplish its task of compilation. These are described below.

The Line Number Table. This table is used to associate line-numbers with addresses. It consists of two 16-bit integers per entry: the line number and its associated address. The high order bit of the line number entry is used to flag definition (hence the restriction that line numbers be in the range 0 to 32767).

The Symbol Table. This table is used to associate lexical representations with their attributes and addresses (if relevant). All names appear here including the both reserved names (e.g., IF and LET) and special symbols (e.g., + and *). Entries are of fixed length (though they need not be) and consist of a six byte name (NUL filled), a one byte type, and a two byte integer value. Interpretation of the various fields depends upon value stored in the type.

Control Stack. The control stack, CSTK, is used to maintain information about the state of the parse. Effectively it contains return addresses for the various recursively invoked recognizers. It is a 16-bit wide stack under the assumption that the compiler takes more than 256 bytes of interpretive code.

Local Variables. Parallel to the control stack are three parallel stacks: T0, T1, and T2. These stacks provide local storage for the each recognizer and may be used to save values, references, or addresses.

Code Memory. All compiled code is directed to an array, CODE, which corresponds to the memory image which would exist at run-time. In a real compiler with limited memory space, one could not dedicate such an array. Other techniques would have to be used to provide the equivalent of its function.

Global Scalars. A small number of global values need to be available for manipulation by the com-

piler writer. Amongst the global objects needed for Tiny BASIC are

PC	the address of the next program byte
CX	the index of the next available location in CODE
FX	an index for generating fixups inside the code
VALUE	the value associated with current token
TOKEN	the current token
CLASS	the current token class
FLAG	the result of the last TST operation

As the compiler is extended, additional global values will be required.

Expression Stack. Expression evaluation is needed for the construction of instructions, allocation of data, and the like. This is done by providing a 16-bit wide expression evaluation stack, AESTK. As the expressions to be evaluated need not be particularly complicated, this stack need not be very deep; 16 elements should be adequate. Note that all expressions must be programmed in reverse-Polish notation.

Lexical Window. Input to the compiler is via a specialized operation (see below) which maintains the lexical window. The window consists of an one byte TOKEN, a one byte CLASS, and a two byte VALUE. These are global quantities and are referenced as such. In the Tiny BASIC compiler only the TOKEN and VALUE fields are used. More complex languages may utilize the CLASS field.

Op-Code Set

The choice of op-code set in an interpretive machine of this sort is mostly a matter of convenience. They divide rather neatly into several different classes: those concerned with the flow of control, those concerned with the parse, those concerned with expression evalution, , and those concerned with code output.

There is nothing particularly special about the particular set presented here. Additions, modifications, and deletions are to be expected during the design of any particular compiler. In fact, many needed features such as listing have been totally ignored.

We have avoided all issues of encoding and have shown all operations and addresses symbolically. Of course, encoding is an important implementation issue and cannot be totally ignored. One would expect for an 8080 to use an 8-bit op-code and 8-bit and 16-bit address/data fields. Since fewer than 128 globals are required, an 8-bit address would be adequate. Since there are only three locals, a special op-code could be used for each one, etc.

Flow of Control. A procedure call and return facility is provided. Unconditional transfer of control is provided. Conditional transfer of control are based upon the value of a global variable, FLAG.

CALL	procedure call
RTN	return from procedure
JMP	jump unconditionally
JT	jump if FLAG true
JF	jump if FLAG false

Parse Instructions. These instructions manipulate the input data stream through the lexical scanner.

NLINE	input a new line to be processed
SCAN	move the next token into the window
TST	set FLAG true if arg matches token otherwise set it false
DFLN	define line number
LLA	load line address to top of AESTK by referencing line number table.
UDLN	test for undefined line numbers and report error

DONE	exit compiler
ERROR	report error

Expression Evaluation. Expression evaluation utilizes a RPN set of instructions and the AESTK. All binary operations pop the top two elements of the stack and combine them and push the result. Unary operations operate on the top of stack only. Load and Store move data to and from the stack.

LDI	Load immediate value
LIG	Load global value
SIG	Store global value
IDL	Load local value (T0, T1, or T2)
STL	Store local value
ADD	Add
SUB	Subtract
AND	And
OR	Or
XOR	Exclusive Or
SHL	Shift-left, zero fill
SHR	Shift-right, zero fill
MUL	Multiply
DIV	Divide
NEG	Negate
NOT	Complement
GT	Test greater than, set FLAG
GE	Test greater than or equal
EQ	Test equality
LT	Test less than
LE	Test less than or equal

Code Generation. The code generation provides a mechanism for moving data and constants into the CODE array. As a side effect, the PC and CX global variables are incremented appropriately.

Parse Control Instructions.

PP	pop AESTK and put low order byte into CODE[CX]; increment CX
PA	pop AESTK and store as an address into CODE[CX] and CODE[CX+1]. adjust CX.
PCB	put constant byte. source is interpreter code not stack.
PCA	put constant address.
PFB	put fixup byte. Use global FX, not CX. Increment as usual.
PFA	put fixup address.

4. LEXICAL PROCESSING

Input to the compiler is processed character by character by a filter routine called the scanner. The algorithm used identifies atomic elements--identifiers, reserved names, reserved symbols, numbers, and so forth--in the input stream.

At the beginning of the scan the old lexeme is discarded and the next lexeme copied to the current lexeme.

Characters are read until the first non-blank character is found.

If the character is alphabetic then it is either an identifier or a reserved word. The input stream is marked and characters scanned as long as the input is either alphabetic or numeric. The delimited string is then looked up in the symbol table. The type and value fields of the symbol table are transferred to the next lexeme window. (In Tiny BASIC all tokens are predeclared. In a more complicated language, symbols which have not been encountered before would be entered into the symbol table at this point and given some sort of default attributes. If the language required declarations the incoming symbol might be tagged as NEW-ID and given a zero value field with the expectation that the compiler might verify that it is a new variable and allocate storage for it. In a language with default

declaration, for example, FORTRAN, the variable might be given attributes based upon its leading character and allocated appropriate storage.)

If the leading character is a digit, then the object is a number. The scan continues collecting the value of the number in the value field of the next lexeme by multiplying by the base (10) and adding a suitably adjusted incoming digit character until a non-digit character is found. The token type is set to indicate a number has been discovered.

If the character is neither a digit, a space, or alphabetic, then it must be a special character. It may be part of a two component special symbol (e.g., <, =, or >=), the leading character of a string (indicated by a quote), or simply stand for itself. In the first case, the next character is inspected. If it is one of the allowable successors, the two character string is locked up in the symbol table; if it is not, the single character string is looked up. If it is a quote, a special global flag is set. All succeeding calls on the scanner produce just the character found except for " and carriage return. The leading quote reports the start of a string, the trailing quote or carriage return the end of string.

For Tiny BASIC an appropriate initialization of the symbol table might be:

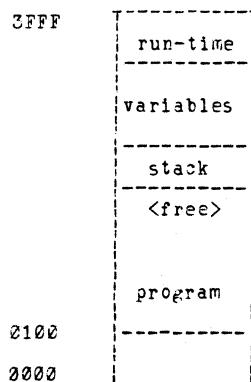
NAME	TYPE	VALUE
let	LET	--
print	PRINT	--
input	INPUT	--
if	IF	--
then	THEN	--
goto	GOTO	--
gosub	GOSUB	--
return	RFTN	--
end	END	--
=	EQ	--
+	PLUS	--
-	MINUS	--
*	TIMES	--
/	DIV	--
>	GT	--
>=	GE	--
<>	NF	--
<=	LF	--
<	LT	--
"	QT	--
<cr>	CR	--
A	VAR	{address of A}
B	VAP	{address of B}
.	.	.
.....	VAR	{address of Z}

Each of the various types mentioned symbolically above are encoded as a small integer. In addition to these types there are a small number of others which are generated by the scanner:

ERR	error (bad character or name)
CHAR	character in a string
EOF	end of file in input

5. RUN-TIME REPRESENTATION

The memory layout for compiled programs might look like



The program origin is at 0100H and grows towards higher addresses. The stack begins at the highest possible point and grows downwards. Variables are preallocated since the maximum number in Tiny BASIC is 26 (52 bytes).

In the compiled code it is presumed that adequate stack space is available for the program to execute; no checks are made for stack underflow or overflow. (Adding such checks is a good exercise for the reader.)

The naked 8080 system is inadequate for supporting Tiny BASIC. At least some operations are best accomplished by making calls on systems subroutines. A listing of those assumed by the compiler is given below.

INNUM	input a 16-bit number
PRNUM	print a 16-bit number
PRSTR	print a string (zero terminated)
NULIN	print cr and lf
MULT	multiply two 16-bit numbers
DIVD	divide two 16-bit numbers

Each of these routines must conform to the particular register allocation conventions used by the compiler. In this case, single argument functions find their argument in BC; two argument functions find their arguments in FC and DE; results are always left in HL. The PRSTR routine is a special case. The only strings in Tiny BASIC are in PRINT statements. The PRSTR routine expects to find a string beginning at its return point and continuing up until the first zero byte. The NULIN routine prints a carriage return and line feed and takes no parameters.

E. CODE GENERATION

For the most part, code generation can be tied to the various statement types. These are examined one by one below together with the more general problems of forward references and expression evaluation. Remember some of the descriptions here refer to the Tiny BASIC run-time environment and some to the compile environment. It is important not to confuse one with the other.

Forward References. A forward reference is generated whenever a GOTO or a GOSUB references a statement which has not yet been encountered. For a GOTO the compiler generates a

JMP line-address

and for a GOSUB it generates

CALL line-address

To acquire the line address, a LLAD (Load Line Address) operation is used to move the line address to the AESTK. This routine pops the line number from the top of the AESTK and searches the line number table for a match; if no match is found it is entered with a line address of zero. If it is found, the high order bit of the tabulated line number is examined; if it is on it is a forward reference. The contents of the line address cell is returned as its address and the current PC value is placed in the line address. This allows a list of various forward references to be chained together within the code output.

The DFLN, define line number, operator must check to see if the new line number appears in the table. If it appears it is either a forward reference or an error. If it is a forward reference, the chain of references must be followed back and each replaced by the current PC value. To do this the location referenced by the line address entry of the table is loaded into a temporary. The current PC is stored in its place. If the temporary is zero we are done. If not, the location given by the temporary is used and the process repeated. When the chain has been resolved, the current PC is placed in the line address part of the table and the line number marked as defined.

The final wrap-up operations must scan the line number table to insure that no lines were referenced without having appeared and give an error message if there are.

Expression Evaluation. Generating good code for expression evaluation on a machine like the 8080 is particularly difficult. In this compiler we really beg the question and generate very simple (and rather poor) code.

The 8080 is unsuited to doing much 16-bit arithmetic. Were we to use subroutine calls we would probably save space since even a simple negation requires more bytes than a subroutine call.

In this compiler we plan to generate subroutine calls for multiplication and division. Addition, subtraction, and negation are to be done in-line.

The HL register is used as an accumulator. The CD register is used as a secondary accumulator and the hardware stack used for temporaries as needed. For example, the assignment statement

```
LET A = (B+C)/(D+E)
```

might generate (writing VA for A, etc. to avoid confusion)

```
LHLD H,VB
LHLD D,VC
DAD D
PUSH H
LHLD H,VD
LHLD D,VE
DAD D
POP D
CALL DIVE ; call system proc
SHLD VA
```

Code generation is straightforward for Tiny BASIC since there are no computed addresses and LHLD and SHLD can be used all the time. Other languages can be much more complex.

Let Statement.

The address of the target is saved in one of the temporary variables local to the recognizer. The expression is evaluated and a store instruction generated.

Print Statement.

The flow is complicated by the possibility of several different types of print elements. Strings generate the string itself terminated by a single zero byte. Expressions are evaluated leaving their result in HI. Following evaluation a call is made on PRNUM, the system number print routine. The end of a (possibly empty) print list generates a call on the system procedure NULIN to upspace the printer.

Input Statement.

The address of the variable to be input to is saved in a local temporary. The system routine INNUM is called to input the number and leave the result in HL. A store of HL is then generated.

If Statement.

The two expressions are evaluated and compared by subtraction using the 8080's usually positive integers only arithmetic. The appropriate jumps are then generated; local variables are used to save the fix-up addresses. Then, the statement recognizer is invoked recursively. Upon return, the jump addresses are fixed up to jump around the code generated by the statement.

Goto Statement.

A jump instruction is generated to the appropriate address. A forward reference is handled as described above.

Return Statement.

A RET instruction is generated.

Gosub Statement.

A CALL instruction is generated to the appropriate address. A forward reference is handled as described above.

End Statement.

The end statement indicates program termination. In keeping with this interpretation the compiler generated a halt instruction after enabling interrupts.

7. AN EXAMPLE RECOGNIZER

It is not our intent to list the entire compiler in this paper; however, to show some of the techniques, a portion of the compiler which processes statement lists and print statements is given below.

```
; process the program a line at a time
lire: NLINE      ; get next line
      SCAN       ; init scanner
      TST        NUM   ; line number?
      JT         lin.0 ; yes, jump -
      TST        EOF   ; end of file
      JT         lin.1 ; yes, finish
      ERROR     100   ; line number expected
;
lin.0: LTG        VALUE   ; load number
      DFLN      ; define line num
      SCAN       ; scan over it
      CALL      stmt   ; process statement
      TST        CR    ; end of line
      JT         line   ; yes, get next
      ERROR     101   ; CR expected
;
lin.1: UBLN      ; undefined line num?
      DONE      ; finished
;
***** process a statement --
;
stmt: TST        PRINT   ; is it a print stmt?
      JF         inpt   ; no, try input
;
pelem: TST        QOT    ; process a print element
      JF         pe.5   ; quote => string
      PCB       0CDH   ; no, look for expression
      PCA       PRSTR  ; send CALL PRSTR to code
;
pe.0: SCAN      ; send string char by char
      TST        CHAR   ; to code
      JF         pe.1   ; quote
      LTG        VALUE   ; get character
      PB        ; put it into code
      JMP        pe.0   ; here at string end
;
pe.1: TST        QOT    ; here at string end
      JT         pe.2   ; quote expected
      ERROR     103   ; quote expected
      PCB       0        ; terminating zero
;
nxtpe: SCAN      ; next print element
      TST        COMMA  ; more in list
      JT         pe.4   ; more to do
      PCB       0CDH   ; put CALL NULIN in code
      PCA       NULIN  ; upspace
      RTN
;
pe.4: SCAN      ; scan over comma
      JMP        pelem
;
inpt:
```

E. CONCLUSIONS

A reasonably simple approach to compiler implementation for hobbyists has been described. The system is intended for language experimentation and is not suitable for the implementation of other than "toy" compilers. However, it is a reasonable learning vehicle for the interested hobbyist.

NUMERICAL CALCULATIONS ON MICROPROCESSORS

Roy Rankin
2273 St. Francis Drive
Palo Alto CA 94303

Abstract

This paper introduces the basic concepts used in numerical computations on microprocessors. Different representations of integers are presented followed by algorithms for the addition, subtraction, multiplication and division of integers with examples of the algorithms. The basic form of floating point numbers is also presented along with algorithms for addition, subtraction, multiplication, division, float, entier, and fix.

1.0 Introduction

All large scale computers have either built in or library functions to do basic arithmetic operations. As yet such items as hardware number processors or libraries are not widely available for microcomputers. The purpose of this paper is to present the basic theory needed to either understand or write the basic routines. Although algorithms are presented no attempt has been made to write them in code since such an effort would make the paper very processor dependent. For persons actually interested in public domain numerical codes references 1 and 2 are recommended for 6502 users and reference 3 for 8080 users. It is hoped that those interested in hardware processors such as the new National Semiconductor "number cruncher unit" 57109 (reference 4) will gain some appreciation of the devices.

2.0 INTEGERS

Numbers are naturally represented on computers simply by assigning each bit in a word a value. If the bits in a word are assigned the following values the number is a binary number.

bit number n ...	7	6	5	4	3	2	1	0
value	$2^{*(n-1)}$	128	64	32	16	8	4	2

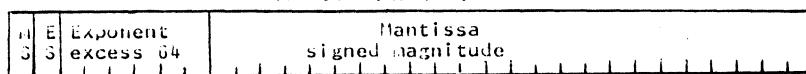
In binary representation an 8-bit word would have the range 0-255. Using just one byte for counting would give us a very limited range. However if two 8 bit words are used, the range becomes 0-65,535. Two words or 16 bits is the standard word length for a single precision integer. If still higher numbers are required, more bits can be used. At this point it should be noted that binary numbers can be grouped into either sets of 3 or 4 digits for ease of handling. If grouped into threes the notation is called octal and each digit has the range 0-7. If however the digits are grouped into fours the notation is referred to as hexadecimal or hex. Hex has the range of 0 to 15 with the numbers above 9 represented by the letters A through F.

So far only positive numbers can be represented and thus if negative numbers are of interest the representation will need to be modified. There are four commonly used integer number representations. They are signed-magnitude, one's complement, two's complement, and excess $2^{*(n-1)}$.

2.0.1 Signed-magnitude

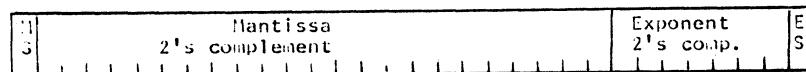
Signed magnitude representation uses one bit, usually the most significant bit, as a sign bit. If the sign bit is clear the number is positive,

IBI 370 (Ref. 6)



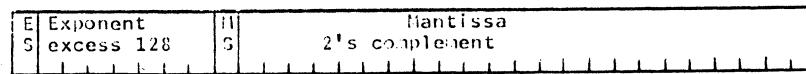
Exponent base = 16 Range = $10^{***}/-75$ Precision 6-7 digits

HP 2100 (Ref. 6)



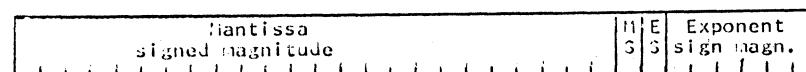
Exponent base = 2 Range = $10^{***}/-38$ Precision 6.9 digits

6502 Format (Ref. 1,2)



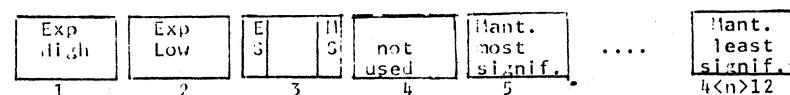
Exponent base = 2 Range = $10^{***}/-38$ Precision 6.9 digits

LLL 8080 BASIC (Ref. 3)



Exponent base = 2 Range = $10^{***}/-10$ Precision 7 digits

57109 (Ref. 4)



$4 < n > 12$

Exponent base = 10 Range = $10^{***}/-99$ Precision select 1-8

Figure 1 Some Floating Point Representations

but if the sign bit is set the number is negative. Thus for an 8 bit number (7 bits plus sign) 0000 0110 in binary would represent 6 while 1000 0110 would be -6. One might note that it is signed magnitude representation that we use in everyday life.

2.0.2 One's complement

In ones complement as in all the representations a bit is set aside for the sign. To change a negative number from a positive one, all the bits are inverted, i.e. 1's are changed to 0's and 0's to 1's. 1111 1001 is an example of -6 in one's complement format. Since the addition of one's complement numbers requires the checking of carries into and out of the sign bit, a process not done by most microcomputers, one's complement representation is not seen on micros.

2.0.3 Two's complement

Most microcomputers are set up to do two's complement addition and subtraction. To generate a two's complement number the number is first one's complemented and then incremented. Thus to generate -6 in two's complement form the one's complement (1111 1001) is incremented to give 1111 1010. A second method can be used to generate two's complemented numbers. Take 2^{n+1} , which is one bit larger than the words being used, and subtract the number to be made negative from it. The result is the two's complement of the number. A nice thing about this method is that it also works in octal or hexadecimal representation as seen here.

binary 1 0000 0000 - 0000 0110 = 1111 1010
octal 400-006 = 372 = 11 1111 010
hex 100-06 = FA = 1111 1010

Two's complement is the most popular representation because the sum of any two two's complement numbers gives a result in two's complement without further adjustment and because unlike signed-magnitude and one's complement zero has a unique representation. Integer numbers represented in two's complement form have the range -2^{n-1} to $2^{n-1} - 1$. Thus for a 16 bit number the range is -32,768 to 32,767.

2.0.4 Excess 2^{n-1}

The final representation is excess 2^{n-1} . This representation is very closely related to two's complement representation in that if the sign bit of a two's complement number is inverted, the number becomes excess 2^{n-1} . It can also be viewed as assigning the number 2^{n-1} , which is just the most significant bit set, as zero and counting up for positive numbers and down for negative numbers. This representation has the advantage that numbers are ranked naturally by the notation from lowest (-32,768= 0000 0000) to highest (32,767= 1111 1111). Other examples of excess 2^{n-1} representation is 6= 1000 0110 while -6= 0111 1010.

2.1 Binary Coded Decimal (BCD)

There is another important integer representation which is a blend of binary and decimal numbers and is called binary coded decimal or BCD. In BCD 4 bits are used to represent the decimal digits 0-9.

The digits are represented using binary numbers which leaves 6 unused bit combinations which are illegal in BCD. Two four bit digits can be packed per 8 bit word. The major advantage of BCD is that it is easy to interface with a base ten world. The disadvantage is poorer bit utilization as seen by the fact that a 16 bit unsigned integer has the range 0 to 65,535 while 16 bits worth of BCD can only represent 0 to 9999. Also to represent signed BCD numbers an extra word must be used to contain the sign bit. Negative BCD numbers are usually represented as signed magnitude as in the new National Semiconductor

57109 "number cruncher" or in 10's complement, which is also called excess 100 for 8 bit words.

2.2 Integer Math Operations

Before discussing floating point calculations which is the goal of this paper it would be well to examine algorithms for the four basic mathematical functions for integer numbers which are addition, subtraction, multiplication, and division. The algorithms presented for integers will only have to be modified a slight amount to be used in floating point computation.

2.2.1 Addition and subtraction

The addition of two 8 bit numbers is trivial on all the popular microprocessors, but it is also of marginal interest. Of more interest is the addition of numbers longer than 8 bits. This is still fairly easily done on microprocessors. The idea is to use the carry to bridge the operation between bytes. The following is an algorithm for adding two multibyte integers.

Algorithm A multibyte addition

- 1) clear carry
- 2) load accumulator with n'th byte of first number
- 3) add, with carry, accumulator to n'th byte of the second number
- 4) store the n'th byte of the result
- 5) loop to 2 until all bytes added.
- 6) check for overflow by testing overflow flag

Subtraction is equally easy.

Either a number can be two's complemented to change its sign and then added with the preceding algorithm or the following subtraction algorithm can be used in processors with a subtraction instruction.

Algorithm B multibyte subtraction

- 1) set carry
- 2) load accumulator with n'th byte of first number
- 3) subtract, with carry, from accumulator the n'th byte of the second number
- 4) store result
- 5) loop to 2 until done
- 6) check overflow flag for overflow

2.2.2 Multiplication

Since most microprocessors do not multiply, the design and implementation of a multiplication algorithm is more challenging, and there is no unique algorithm. For the algorithm presented here two n bit integers will be multiplied together allowing for a product of length $2n$. Of course if the product actually exceeds n bits overflow has occurred, but allowing for a $2n$ product simplifies the algorithm. The method used is based on the pencil and paper method of multiplication which consists of multiplying one number by the j'th digit of the second number shifting the result j-1 positions to the left and summing as i is incremented from 1 to m where m is the number of digits in the second number. In binary the multiplication of a single bit and a number results in either the number or zero. Thus this method in binary is reduced to shifts and selective additions which are operations a microprocessor can do. One trick is used which is to shift the sum right after each partial product is added rather than shifting each partial product left j-1 times. This allows both more efficient coding and allows the multiplier to be initially placed in the lower n bits of the $2n$ product register. The algorithm is as follows.

Algorithm C Multiplication of integers

- 1) make multiplicand and multiplier positive as required and set sign flag if only one was negative
- 2) clear upper half of 2n bit product register and place multiplier in lower half
- 3) clear carry
- 4) rotate product register 1 bit right (i.e. carry shifted into MSB, each bit shifted one bit right and LSB shifted into carry)
- 5) if carry is set add multiplicand to upper half of product register (after clearing carry)
- 6) go back to 4 if n+1 passes not complete
- 7) if sign flag is set make result minus

To help make this algorithm clear an example of the product of the four bit numbers 5 (0101) and 7 (0111) will be given.

step	product reg.	carry	comments
3	0000	0101	0 start of operation
4	0000	0010	1 roll right
5	0111	0010	0 add, carry set
4	0011	1001	0 roll right
5			skip carry clear
4	0001	1100	1 roll right
5	1000	1100	0 add, carry set
4	0100	0110	0 roll right
5			skip, carry clear
4	0010	0111	0 roll right
5			skip carry clear
7	0010	0011	0 final answer

$$\text{answer} = 0010 \ 0011 = 35$$

note that we got the correct solution and that for four bit integers overflow has occurred since the upper word is not zero.

2.2.3 division of integers

Division like multiplication could be done by any of several algorithms. The algorithm presented here is based on the pencil and paper method. This algorithm will divide an n bit number, the dividend, by an n bit number, the divisor.

Algorithm D division of integers

- 1) check divisor, if zero then error
- 2) Make dividend and divisor positive as required and set sign flag if only one was negative
- 3) shift divisor left and count number of shifts until highest nonzero bit is in sign bit
 - 4) shift divisor right one bit and quotient left 1 bit
 - 5) subtract from dividend the shifted divisor
 - 6) if result of subtraction is positive increment quotient and replace dividend with answer
 - 7) go back to 4 until loop is executed same number of times as divisor was shifted in 3, i.e. divisor is in same position as before 3
 - 8) if sign flag set make quotient and dividend(which is now the remainder) negative

To demonstrate this algorithm the 5 bit numbers 13(01101) will be divided by 2(00010). In the example the registers are shown after the completion of the step.

step	dividend	divisor	quotient	shift	count	sub.
2	01101	00010	00000	0	0	--
3		10000		3		
4		01000	00000			
5						00101
6	00101		00001			

4	00100	00010	2
5			00001
6	00001	00011	
4	00010	00110	1
5			11111
6	00001	00110	

routine complete quotient = 6 (00110) and remainder 1 (000010)

3.0 Floating point representation

Integers are very good at representing numbers in a limited range, but if the numbers get very large or very small integer representation is not practical. To represent numbers less than one the integer representation can be modified by assuming a fixed decimal point in the interior of the word. This would require minor modifications to the multiplication and division algorithms. Fixed point notation is in fact used for money calculations, but fixed decimal notation lacks the range needed in many calculations. Floating point notation was developed to overcome this limitation, but a price is paid. The price is that the number is no longer represented exactly, but is rather an approximation to the number good to a certain number of digits. The number of digits approximated accurately is known as the number of significant digits. What has become known as a single precision floating point number has 6 to 7 significant digits. The general form of floating point numbers is $m * b^{e}$ where m is the mantissa, b the exponent base, and e the exponent. The length of the mantissa determines the number of significant digits. If the mantissa is represented in binary the number of significant decimal digits is given approximately by .3 times n where n is the number of binary bits in the mantissa excluding the sign bit. Thus single precision numbers with n=23 have about 6.9 significant decimal digits. The range of the number is determined by both the exponent base b which is implied in a given floating point representation, and the number of bits in e, the exponent.

Scientific notation is a form of floating point notation. The number 125. in scientific notation becomes $1.25E2$. Thus the mantissa is 1.25, the exponent base is 10 and the exponent is 2 and the number has three significant digits. Scientific notation will be used to demonstrate the rules needed to perform floating point arithmetic operations. First it should be noted that scientific notation is usually written in normalized form. Normalized form is where the decimal point is located in a specified position relative to the first (most significant) digit in the mantissa. In scientific notation the decimal point is to the right of the most significant digit. In floating point representation the location of the decimal point is fixed in the mantissa. Thus normalization insures the maximum number of significant digits.

Although floating point numbers have the general form discussed previously, the detailed representation of floating point numbers in computers is highly varied. Figure 1 shows 5 different floating point representations, four of which are binary with the fifth being a BCD representation. This figure is not an exhaustive list of the different representations used. The main point of the figure is to show that when working with floating point numbers the representation cannot be assumed.

3.1 Floating point addition and subtraction

The addition and subtraction of floating point numbers require three major operations. First the exponents are aligned, secondly the mantissas are added or subtracted, and lastly the result is normalized. In binary representation a number is normalized if the sign bit and the most significant bit of the mantissa are different.

Algorithm E, floating point addition and subtraction

- 1) align exponents
 - a) determine which number has smaller exponent
 - b) make exponents equal by increasing the exponent while shifting the mantissa right
- 2) add mantissas using algorithm A or B
- 3) normalize mantissa
 - a) if overflow is set in step 2 carry has sign of mantissa do a shift right with carry and increment exponent
 - b) else check if mantissa is normalized, if not, shift left mantissa while decremneting exponent until normalized

To demonstrate the algorithm the scientific notation numbers 1.25E2 and 3.00E1 will be added.

step	operation
1	3.00E1 becomes 0.30E2
2	1.25E2+0.30E2=1.55E2
3	1.55E2 is normalized

If instead the operation was to subtract 30 from 125 the result of step two would have been 0.95E2 which is not normalized, thus from rule 3b the answer becomes 9.50E1. The following example shows the weakness of floating point numbers. Calculate the sum of 1.25E3 and 3.00E0. In step 1, 3.00E0 becomes 0.00E3 so that the result of the addition is to yield 1.25E3. If this addition was occurring in a loop which was repeated 100 times the final answer would be 1.25E3 rather than 1.55E3 which is the correct answer. For single precision numbers (24 bit mantissa) all precision is lost if the numbers differ by more than seven orders of magnitude i.e. seven decades.

3.2 floating point multiplication

The multiplication of floating point numbers requires three major operations. First the exponents are added together. If overflow occurs during the addition process than the floating point number has either overflowed or underflowed depending on the sign of the exponents before the addition. If they were plus, overflow occurred. Note that overflow or underflow cannot occur if the signs of the exponents are different. Secondly the mantissas are multiplied together. For the last step the resulting floating point number is normalized.

For the multiplication of the mantissas, algorithm C can be used with minor modifications. The need for the first modification depends on the location of the assumed decimal point in the mantissa. If it is to the left of the most significant bit, then the first modification is not needed. If however the assumed decimal point is to the right of the most significant bit so that the range of the mantissa is 1.0 to 2.0, then the inner loop of algorithm C should only be executed n times rather than n+1 times. The result wanted from the algorithm in either case is the upper n bits rather than the lower n bits as would be used for integer multiplication. The second modification involves checking the sign bit of the multiplication before if the sign bit is set the mantissa is shifted right 1 bit and the new exponent is incremented.

Algorithm F- floating point multiplication

- 1) add exponents and check for under or overflow
- 2) multiply mantissa using algorithm C as modified in text
- 3) normalize result

This algorithm will be demonstrated by the product of 1.25E2 and 3.00E1.

step	operation
1	2+1=3

2	1.25 * 3.00 =3.7500
3	3.75E3 is result

3.3 Floating point division

Floating point division is similar to floating point multiplication. The first step is to subtract the exponent of the divisor from the exponent of the dividend. If the sign of the exponent for the divisor is negative and the sign of the dividend is positive overflow is possible. Likewise if the sign of the divisor exponent is positive and the sign of the dividend exponent is negative underflow is possible. The second step is to divide the mantissa of the dividend by the mantissa of the divisor. This step can be performed using algorithm D with one modification. Omit step 3 where the divisor is shifted left since it is not needed as the divisor should already be in normalized format. The inner loop (steps 4-7) should be executed n-1 times. The last step is to normalize the result.

Algorithm G- floating point division

- 1) subtract exponent of divisor from exponent of dividend and check for over or underflow
- 2) divide mantissa of dividend by mantissa of divisor using modified algorithm D
- 3) normalize result

The number 3.00E1 will be divided by 1.25E2 to demonstrate the algorithm.

step	
1	1-2=-1
2	3.00/1.25=2.40
3	2.40E-1 is answer

3.4 Float, Entier, and Fix

The final routines to be discussed are three functions for the conversion of integers and floating point numbers. Float converts a integer to floating point. Fix converts a floating point number to an integer. Entier does the same thing as fix except that entier returns the integer which is smaller than or equal to the floating point number. Thus the fix of -12.5 is -12 while the entier of -12.5 is -13. Entier is useful in certain numerical algorithms like sin and cos and is more straight forward to generate than fix. The following algorithms are for an m bit integer(including sign) and a n bit floating point mantissa (including sign). There is an important constant used in these routines which depends on the location of the assumed decimal point. If the assumed decimal point is to the left of the most significant bit the constant is equal to m-1, but if the assumed decimal point is to the right of the most significant bit, then the constant is equal to m-2.

Algorithm H- float

- 1) Store integer in upper m bits of mantissa and clear lower n-m bits of mantissa
- 2) load exponent with n-1 (or m-2)
- 3) normalize result

For an example, 15 will be floated. Integers will be 8 bits long and floating point mantissa 12 bits. So that m-1 is 7.

step	mantissa	exponent
1	0000 1111 0000	0000
2	0000 1111 0000	0111
3	0001 1110 0000	0110
3	0011 1100 0000	0101
3	0111 1000 0000	0100

Thus the result is .0375 *2**16 which is 15.

Algorithm J Entier

```

1)if exponent equals m-1 go to 4
2)else do right arithmetic shift of
   mantissa and increment exponent
3)check exponent for overflow, if none
   loop back to 1
4) answer in upper m bits of mantissa

```

To demonstrate this algorithm it will be applied to -15.5.

step	mantissa	exponent
1	1000 0100 0000	0100
2	1100 0010 0000	0101
2	1110 0001 0000	0110
2	1111 0000 1000	0111
4	answer 1111 0000	

The result is in binary 1111 0000 which is equal to -16.

Algorithm K- fix

```

1)do entier
2) if integer is positive go to 4
3)else if lower n-m bits of are not
   zero then increment result
4)done

```

This algorithm will also be applied to -15.5

step	mantissa	exponent
0	1000 0100 0000	0100
1	1111 0000 1000	0111
3	1111 0001 1000	----
4	answer is 1111 0001	

Thus the result is -15 .

4.0 Conclusion

This paper has presented the basic integer and floating point algorithms to do numerical calculations on microcomputers. These algorithms are just the basics, but most further numerical algorithms will use these as building blocks. A good source for numerical algorithms is reference 5.

5.0 Biography

I am a Ph'D student in Mechanical Engineering at Stanford University. My interest in computers dates back to high school when I taught myself FORTRAN and ran programs on IBM 1620s. Working with system software, engineering programs, and hardware on my labs hp2100 for four years has given me an intimate experience with computers. I have a 6502 JOLT based microcomputer for which I have been developing both hardware and software. It was my interest in number crunching that got my interested in writing floating point routines for the 6502. Steve Wozniak and myself got together through the Home Brew Computer Club to write the routines which appear in references 1 and 2.

6.0 References

- 1) Rankin,R., S. Wozniak, "Floating Point Routines for 6502", Dr. Dobb's Journal of Computer Calisthenics and Orthodontia, Vol 1, No. 7, Aug. 1977.
- 2) Rankin, Roy, Steve Wozniak, "Floating Point Routines for 6502", Interface Age, Vol 1, No. 12, Nov 1976.
- 3) Dickenson, John, Jerry Barber, John Teeter, Royce Eckard, Eugene Fisher, "Lawerence Livermore Lab's 8080 BASIC", Dr. Dobb's Journal, Vol 2 No. 1, Jan 1977.
- 4) Weissberger, Alan J., Ted Toal, "Tough Mathematical Tasks are Child's Play for Number Cruncher", Electronics Vol 50, No. 4,

Feb 17,1977.

- 5) Hart, John F. COMPUTER APPROXIMATIONS, John Wiley & Sons, New York, 1968.
- 6) Stone, Harold S., INTRODUCTION TO COMPUTER ORGANIZATION AND DATA STRUCTURES, McGraw-Hill, New York, 1972.

MODULAR RELOCATABLE CODE

Dennis Burke
4042 W. 139th Street No.7
Hawthorne CA 90250

Relocatable code is a form of object code that allows a program to be loaded into any location in memory and run. It is not pure object code because it contains the extra information required to alter the necessary locations to allow the program to execute correctly at its new address. Relocatable code eliminates the necessity of re-assembly of a source code each time it is desired to change the execution location of a program.

Considering the bulk of a source program and the space necessary for the assembler, it is obvious that relocatable programs afford a decided advantage in both memory requirements and time.

However, it should be noted that relocatable code can not stand alone. It must have a relocating loader to be utilized. If loaded directly into memory relocatable code will not run. It requires the assistance of the relocating loader to look for the "flags" imbedded in the assembled object code and to adjust the appropriate addresses for the new location prior to placing the code in memory.

These flags are called type codes. The assembler or compiler places them in the object code as they are produced. A type code is an instruction that tells the loader what to do next with the data that follows it. In one or more bytes, depending on the nature of the type code, it will specify both the operation and the number of bytes upon which the operation should be performed. At the end of the operational sequence, the loader expects to find another type code for more instructions.

There are four basic type codes:

1. Program: The program type code consists of one byte that acts as a descriptor of the whole record. i.e.
 - A. Source code
 - B. Object code
 - C. Data
 In addition, the program type code can specify the processor (type of machine) for which the record is intended. i.e. 8080, LSI 11, 6800, etc.
2. Origin: Origin type codes specify the location or the offset from the program to the present working point. Namely, where the next byte goes into memory. (Note: Origin type code is a proper name for a process. It should not be confused with the origin of the program which is always a location in memory.)

Origin type codes can perform the following functions:

- A. It can set the origin of the program.
- B. It can also be used to reserve storage in a program by causing the storage pointer to skip ahead a specified number of bytes.
- C. In all cases, it affects the pointer to the next location to use in memory.

3. Data: Data type codes identify the class of operation to be performed on the data to relocate it in memory.

There are two basic distinctions:

- A. Data which require no modification, i.e., instructions or data that are not references to memory.
- B. Data which must be modified. i.e., the operand to an instruction that references memory (within the program being loaded).

Data type codes may be multi-byte to include a count of how many bytes are acted upon.

- 4.. End: End type codes occur at the end of the last record. They inform the loader that the end has been reached. In addition, the end code can specify an execution address. This should not be confused with the origin type code which specifies the location to start placing code in memory. The execution address might be some point in the program other than the "origin". For instance, the popular "ESP-1" assembler loads at 6D00H and executes from 7000H. (1)

Description of the format: The code is divided into records of

512 bytes. Each record starts with a record mark, followed by a byte count for that record and ending with a checksum or a CRC. A byte count of zero is considered to be the end of file marker.

Within each record the first byte is always a program type code. This allows the operating system, or relocating loader, to get upset before progressing very far should an operation, in error, fetch this record.

The following byte in a record could be either an origin or data type code. Each type code specifies how many bytes are to be included in the operation to be performed. At the end of that operation the loader will look for a new type code and perform its attendant operation until the byte count for the record is exhausted. If the loader reaches the end of the current record, without encountering an end type code, it will fetch another record, and repeat the process until it finds an end code.

It may be desirable to separate the program, data and temporary storage while relocating a program. This is possible by the proper combinations of origin and data type codes where a field in each type code conveys the reference to the nature of the data to be relocated.

Currently, there are few relocating loaders available to the microcomputer end user, though this state of affairs will certainly change in the near future. Intel has had a relocating loader and assembler available for some time (2) that runs on the 8080 and T.D.L. has published an assembler and relocating loader for the Z-80 chip (3). Both use different schemes than the one discussed here, but inspection of their methodology would prove informative to anyone planning to write their own assembler and loader.

The T.D.L. assembler produces a modified intel hex format tape as its object code file. Each location that is to be altered is preceded by a non-hex character. These act as flags to the relocating loader which adds the appropriate offset to the object code. These non-hex characters act as flags only and convey no further instructions.

Hence, this schema can only handle linear relocations. The program can only be relocated in memory as a single string of object code and complex assignments of data, program, and temporary storage are not possible.

Another popular method of relocation is the "pointer table" system. This, as its name suggests, creates a table - either before or after the actual object code - which points to each location in the record that must be relocated. It is also an inherently linear scheme and adds an equal offset to each location to be relocated.

It might be well at this point to touch briefly on how relocatable code is generated by the assembler. Again, we will refer principally to modular relocatable code, but it will be readily appreciated that this discussion can also apply where relevant to the linear schemes.

Mnemonics are given to the assembler to be translated into code and on the first pass the assembler compiles a symbol table. All labels are extracted and tabulated.

For instance, here are a few lines of code:

0010	START	LXI SP,0200H	;SET STACK.
0020		LXI H,MESSG-1	;GET OUR LOGO.
0030	NEXT	INX H	;POINT TO MESSAGE.
0040		MOV A,M	;GET BYTE IN THE ACC.
0050	OUT1	CALL TTYO	;SEND IT TO THE TTY.
0060	POP	CALL GONG	;DO SOMETHING.
0070		JMP NEXT	;JUMP TO NEXT ROUTINE.
0080	TTYO	EQU 7810H	;DEFINE TTYO ROUTINE.
0090	GONG	EQU TTYO+44H	;DEFINE GONG'S LOCATION.
0100	MESSG	ASC "HERE WE ARE"	;STRING TO PRINT.
0110		DB ODH	;STRING TERMINATOR.

START	0000	NEXT	0006
OUT1	0008	POP	000B
TTYO	7810	GONG	7854
MESSG	0011		

In the foregoing code the assembler would have to assign a flag to each label in the symbol table so that the correct type code could be generated. A flag of either 00 or 01 is used, the value of 00 indicating that the address in question is absolute and should not be relocated; the value of 01 indicates that this is a relocatable quantity. The location of "TTYO" is for example fixed and does not vary with the relocation of the program. Hence, it would get a flag of 00.

The other labels, since they refer to relative quantities will get a flag of 01 indicating that the correct type code for their relocation should be issued. In the second pass, the assembler issues these codes and places them in the object code for the loader to act on. At this point the assembly is complete and a

purely binary object tape has been generated with large savings of time over the generation and loading of a hex tape. The pointer scheme also requires more memory to produce and to load than does the modular method. And this at the cost of less flexibility.

SUMMARY:

In the outlines form of relocatable code, there are four special type codes:

"Program" Type code - To identify the record to the loader.

"Data" Type code - Which bears specific instructions to the loader for operations on data to be placed in memory.

"Origin" Type code - Which sets the address relative to the (actual) origin where the object code will be placed.

"End" type code - Which supplies a final record terminator and can indicate the execution address for the program.

A loader, acting upon a record with these elements is capable of placing object code anywhere in memory for execution without the necessity of reassembly of a source file for each new location.

The relocating loader takes information entered at "load time" by the operator to set the origin for the object code as it loads. The loader utilizes the information derived from the type codes to modify the data as loaded. In the process the type codes are stripped from the object code and only the "new" modified object code is placed into memory.

After the relocating loader has placed the record in memory, the record becomes an absolute object program and is no longer relocatable. All information necessary to adjust the object code has been deleted from the program in the process of loading.

The process of generating the type codes in the assembler is also discussed.

- (1) ESP-1 Shrayer, 930 Bonnie Brae, L.A. 90006
- (2) Intel Corporation, 3065 Bowers Ave., Santa Clara, 95051
- (3) Technical Design Labs, Inc., Research Park, Bldg. H,
1101 State Rd., Princeton, N.J. 08540

I should like to express my gratitude to Jeff Barlow for technical assistance in preparing this and to D. E. MacLean for his assistance in the preparation of the manuscript.

AN IMPLEMENTATION TECHNIQUE FOR MUMPS

David D. Sheretz
Medical Information Sciences
University of California Medical Center
San Francisco CA 94143

Abstract

The MUMPS Language Standard is specified most precisely by a set of "transition diagrams", which give both the syntax rules and semantic actions for the language. The notion of a Transition Diagram Language (TDL) is introduced to allow direct encoding of the MUMPS transition diagrams into machine instructions for a virtual machine. The TDL represents a symbolic assembly language for a hypothetical computer, called the "MUMPS machine". The MUMPS machine is an abstraction which can be realized or emulated in hardware, or simulated in software. The advantages of this formal approach to implementing a MUMPS interpreter are described, and some examples are presented. Additional MUMPS implementation issues are discussed. Finally, more recent uses of the TDL concept are mentioned.

Introduction

MUMPS (Massachusetts General Hospital Utility Multi-Programming System) provides features for the creation of interactive programs which share access to a hierarchical data base. The MUMPS Language Standard provides an effective pattern-matching construct, a powerful set of string manipulation operations and functions, a timing facility for data input operations and resource allocation, and a file management facility which relieves the user of nearly all physical file organization details. In addition, MUMPS provides an interactive programming environment which greatly simplifies the tasks of editing, debugging and managing MUMPS programs.

MUMPS has traditionally been implemented on rather small minicomputers with a modest amount of memory (32K words). Because of the space limitations, and for efficiency reasons, MUMPS implementations until recently have been written almost exclusively in the assembly language of the host computer. This paper presents an implementation strategy which allows the complete syntax of the MUMPS language to be expressed in a machine-independent manner, and provides a well-defined interface for the modules which perform the semantic actions of the language.

The Transition Diagram Language

Included in the specification document for the MUMPS Language Standard (O'Neill, 1975) is a set of "transition diagrams", which specify both the syntactic structures and the semantic actions of the language. A transition diagram is a directed graph whose successful traversal identifies a syntactic construct in the language (for example, WRITE X is a syntactically correct command); traversal of a diagram may also cause an associated semantic action to be performed (for example, writing the current value of the variable X). The MUMPS transition diagrams are based upon a technique developed by M.E. Conway (Conway, 1963). A complete set of such diagrams constitute a precise specification for a programming language. If the set of transition diagrams could be encoded so they could be directly processed, the resulting code would make up a complete translator for the MUMPS language.

The encoding technique used here is the introduction of a symbolic assembly language, called the Transition Diagram Language, or TDL. The TDL is a fixed-field assembly language, which has instructions for testing syntactic constructs, for executing semantic actions, and for communication within TDL procedures and between the TDL and the semantic actions.

The general syntax for the TDL is as follows:

Column 1 10 16

[label] INSTRUCTION operand [comment]

where the bracketed fields are optional. A label is an optional unique name of up to eight alphanumerics; comments can be inserted inline, provided the comment is separated from the operand field by at least one blank. Comment lines can also be added by placing a "*" in column 1. The TDL instructions are summarized in Figure 1.

The TDL utilizes a few common variables for some of its operations, as follows:

TEST - a test flag which is set true (1) or false (0) by the TS, TR and TL instructions depending on the success or failure of the test, and by the RT instruction to indicate if a correct syntactic construct was scanned;
CASE - a flag set by the SV instruction and tested in the JE and JN instructions;
FLAG - a bit string in which particular bits can be set, reset and tested with the SB, RB and TB instructions, respectively.

A syntax stack is also needed for the CL and RT instructions of the TDL. One syntactic construct may be composed of a number of more primitive constructs. This hierarchy is handled with the CL instruction, which stacks the next instruction address as a return and then calls the appropriate construct by branching to the TDL label at which the construct begins. Such calls may be recursive, and can be nested arbitrarily deep. The RT instruction indicates the success or failure of finding the called construct, and also resets the instruction register to the address on top of the stack (and pops the stack).

As an example of using the TDL, consider the transition diagram on the left-hand side of Figure 2. This diagram represents the MUMPS construct lyn (local variable name), and is taken from page II-60 of the MUMPS Language Standard (O'Neill, 1975). The diagram can be directly translated into the TDL code on the right-hand side of Figure 2. The correspondence between the

INST	Operand	Operation
ER	n(char,label)	Calls the error handler with error number n set. The optional information in parentheses gives the character after which to resume syntax scanning, and the TDL label to which to branch after reporting the error.
TS	char	Tests the current character being scanned to see if it matches char. If so, TEST is set true and the syntax scan moves to the next character; otherwise, TEST is set false.
TR	char1-char2	Tests to see if the current character being scanned matches any of the characters of the ASCII collating sequence in the inclusive range between char1 and char2. If so, TEST is set true and the syntax scan moves to the next character; otherwise, TEST is set false.
TL	"string"	Assuming k is the length of string, sees if the next k characters to be scanned match string. If so, TEST is set true and the syntax scan moves to the next character after the matched string; otherwise, TEST is set false and the syntax scan is not moved.
CL	label	Stacks the next instruction address and then branches to the specified label.
RT	n	Sets TEST false if n = 0; otherwise TEST is set true. The return address is popped off the top of the stack and control is returned to that point.
DO	ACn	Performs a semantic action by invoking a call to external subroutine number n.
SB	n	Sets bit n in FLAG to 1.
RB	n	Sets bit n in FLAG to 0.
TB	n	Tests bit n in FLAG. TEST is set true if bit equals 1; otherwise, TEST is set false.
SV	n	Sets the value of CASE equal to n.
J	label	Unconditionally branches to label.
JT	label	Branches to label if TEST is true.
JF	label	Branches to label if TEST is false.
JE	n,label	Branches to label if CASE is equal to n.
JN	n,label	Branches to label if CASE is unequal to n.

(Note - if the conditional jump instructions, JT, JF, JE and JN, do not branch, the next TDL instruction in sequence is executed.)

The following pseudo-op instructions are added to the TDL:

label	DC	Enters label into a table with the relative address of the next TDL instruction, so that jumps may be made into the TDL code from the outside by loading the address associated with label into the instruction register.
	FI	The end statement of a TDL program.

Figure 1 - The TDL instructions

TDL code and the portions of the diagram it implements should be rather obvious.

A few points of Figure 2 may need clarification. Several of the MUMPS transition diagrams (including the one for indnam) have multiple exits (labelled W,X,Y and Z in the diagrams) for different cases. These exits are handled in the TDL code with the SV instruction by setting a value which represents the desired exit (for example, 4 represents the W exit case). The JE and JN instructions are then used to test for the different cases. The TDL code assumes an execution stack which is used to save the local variable name, along with any subscripts, so that action 6 of the transition diagram is unnecessary in the TDL code. Also note that the test for name-level indirection (Nameind is a bit in FLAG) and any repeated executions of action 5 in the diagram are all handled in ACS of the TDL code.

The MUMPS Machine

The TDL instructions of Figure 1 rather obviously suggest a machine instruction set for a virtual machine. An assembler for the TDL would produce machine instructions for this hypothetical computer, which will be called the "MUMPS machine". The MUMPS machine would need machine instructions for each TDL instruction, registers for the TDL variables TTEST, CASE and FLAG, as well as a program counter and an instruction register, and a stack for the TDL syntax stack. Such a machine could easily be built in hardware, or emulated on a microprocessor such as the 8080. Figure 3 shows the TDL

instruction set as it might be implemented on a 16-bit word machine.

Once the MUMPS machine is constructed, either in hardware or software, an assembler for the TDL needs to be written. After this is done, the MUMPS transition diagrams can be coded in the TDL, and the machine code produced from the TDL assembler could be executed by the MUMPS machine. In the present effort, both the TDL assembler and a simulator for the MUMPS machine were written in FORTRAN, in order to achieve a substantial level of machine independence.

The TDL code for the MUMPS transition diagrams can be constructed fairly quickly from the MUMPS transition diagrams. The TDL assembler and MUMPS machine emulator can also be implemented rather rapidly. What remains to complete a single-user MUMPS interpreter is to implement the various semantic actions invoked by the DO instructions in the TDL. A mechanism for reading input lines to be scanned (using the TDL code as a driver) is also needed.

Advantages to the TDL approach

There are a number of distinct advantages to the TDL approach for implementing MUMPS, or any other language. First the syntax of the language is effectively separated from the semantics. A syntax analyzer can be constructed very quickly, and decisions about the details of some of the semantics (for example, symbol table organization) can be postponed. Users can experiment with the syntax of a new language, and problems can be resolved in a proposed syntax prior to implementing much of the semantics. The syntax can also be altered easily, with little or no effect on existing semantic actions.

Secondly, the TDL neatly breaks the semantics into well-defined, generally simple action modules. Thus the actions invoked by the TDL DO instruction are for the most part small subroutines. Another advantage is that because the implementation of the semantics is highly modular, performance data can be obtained easily for identifying heavily-used action modules. Since the interaction among modules is minimal, heavily-used or inefficient modules can be rewritten without adverse effects elsewhere. Note also that any semantic action can be implemented in the language best suited for the purpose.

Thirdly, the TDL concept makes experimentation with different

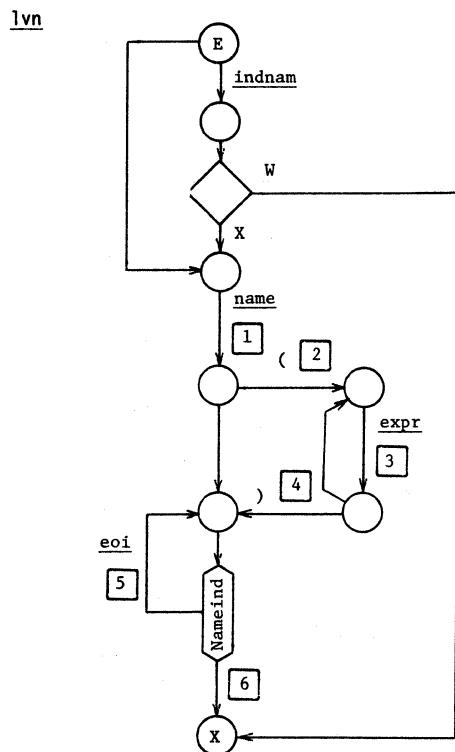


Diagram actions

1. Result → A
2. PUT Nameind on the Stack. False → Nameind
3. Replace the n-tuple in A with the n + 1-tuple (A,Result)
4. GET Nameind from the Stack
5. GET previous level's Nameind and Window position from the Stack. GET the indirect name off the Stack. Link window to retrieved Window position
6. A → Result

LVN	CL INDNAME	Check for local variable indirection
	JE 4, LVNEND	W return; syntax checking only
	CL NAME	Check for variable name
	JT L1	Continue if name is found
	RT Ø	No name; not a local variable
L1	* AC1	saves the name and sets the subscript counter to zero
	DO AC1	Save name and set subscript counter
	TS (Test for left parenthesis
	JF L3	Name has no subscripts after it
	DO AC2	Stack Nameind flag and set false
L2	CL EXPR	Check for next subscript expression
	DO AC3	Stack subscript and increment counter
	TS ,	Test for comma (more subscripts)
	JT L2	Branch to get next subscript
	TS)	Test for right parenthesis
	JT L3	End of subscript list
	ER 12	Error - missing closing parenthesis
L3	DO AC4	Restore Nameind flag from the stack
	DO AC5	Check for end of all name-level indirection
LVNEND	RT 1	The lvn diagram was scanned successfully

Figure 2 - Transition diagram and TDL code for lvn

implementation techniques feasible. Because the syntax is all performed within the TDL, the implementation strategy for the various semantic actions has no effect on the syntactic analysis. Thus, for example, the MUMPS local symbol table could be changed from variable-length to fixed-length entries, could be hashed or linear, or could share space among several users without changing the TDL code. Evaluation of different approaches is encouraged by the TDL, and changes made in any portion of the interpreter are greatly simplified.

	Bits	
INST	01 45 78	15
ER	0000	n
TS	0001	char
TR	0010	char1 char2
RT	0011	n
DO	0100	n
SV	0101	case
SB	0110	bit
TB	0111	bit
RB	1000	bit
TL	1001	k
	byte1	byte2
	...	bytek
J	1010	adr + 1024
CL	1011	adr + 1024
JT	1100	adr + 1024
JF	1101	adr + 1024
JE	1110	adr + 1024 case
JN	1111	adr + 1024 case

Note: The address in the CL instruction and all the jump instructions is stored as a relative address to the present location, offset by 1024 words. Thus a reference can be made up to plus or minus 1023 words away from the present location.

Figure 3 - Reference guide to the MUMPS machine instructions

Other Implementation Issues

A MUMPS interpreter requires implementation decisions in primarily four areas:

- 1) the local symbol table, which maintains the names and values of all currently active local variables;
- 2) the global variables, which reside on secondary storage and are shared among all users;
- 3) the handling of resource allocation among several users, as well as the input and output mechanisms;
- 4) the maintenance on secondary storage of each user's set of MUMPS routines.

In addition, a multi-user MUMPS system must have an efficient scheduling algorithm for making good use of the CPU resources. A good discussion of the multi-user issues can be found in (Wasserman, Sherertz and Zears, 1975). Implementation techniques for globals are described in detail in (Wasserman, Sherertz and Rogerson, 1975). The other areas of concern in a MUMPS implementation are analyzed in (Wasserman and Sherertz, 1975). The MUMPS community has vast experience in implementing MUMPS on a variety of computers, including the PDP-15, PDP-10, PDP-11, Nova, Artronix-PC, IBM 370 and the MODCOMP IV. Work is continuing on an implementation of MUMPS for an 8080-based microprocessor.

Conclusions

The TDL concept is a useful tool for the design of an interpreter (or a compiler). Nothing inherent in the TDL approach limits its use to MUMPS. An effort is now in progress using the TDL technique for the development of a language called PLAIN (Programming LAnguage for INteraction). PLAIN is similar in structure to PASCAL, but embodies the concepts of pattern-matching, string manipulation and timing from MUMPS, as well as an interface to allow conversational access to a data base. The TDL has been useful in detecting syntactic problems in PLAIN, and has allowed a syntax analyzer for PLAIN to be rapidly constructed directly from the BNF specification.

The TDL approach has thus proven to be an effective technique for the implementation of language translators. It allows much of the translator to be written in a machine-independent manner, which greatly reduces the number of modifications encountered in transferring to a different computer. Although MUMPS has not yet been implemented on a computer affordable to the normal hobbyist, the TDL concept is well-suited to a microprocessor. Experience with the MUMPS language indicates that it would be a useful addition to the repertoire of available languages in the hobbyist market.

About the author

I am presently Senior Systems Analyst in Medical Information Science at the University of California, San Francisco. Currently I am completing a MUMPS interpreter for the MODCOMP IV/25 computer, as well as a number of other software tools for use in medical applications and teaching. Previously I was Project Program Leader for the Medication Systems Group, Massachusetts General Hospital, Boston, where I helped design and implement a real-time medications ordering system which was in daily use on two hospital wards.

References

- Conway, M.E., "Design of a Separable Transition-Diagram Compiler," *Communications of the ACM*, 6:7 (July, 1963), pp. 396-408.
- O'Neill, J.T. (ed.), "MUMPS Language Standard," National Bureau of Standards Handbook 118, Washington, D.C., 1975.
- Wasserman, A.I. and D.D. Sherertz, "Implementation of the MUMPS Language Standard," MDC 2/3, MUMPS Development Committee, 1975.

Wasserman, A.I., D.D. Sherertz and C.L. Rogerson, "MUMPS Globals and their Implementation," MDC 2/1, MUMPS Development Committee, 1975.

Wasserman, A.I., D.D. Sherertz and R.W. Zears, "Design of a Multiprogramming System for the MUMPS Language," MDC 2/2, MUMPS Development Committee, 1975.

NOTE: For information on obtaining MUMPS Development Committee publications, contact Joan Zimmerman, MUMPS Users Group, 700 South Euclid Avenue, St. Louis, Missouri 63110.

COMPUTER LANGUAGES: THE KEY TO PROCESSOR POWER

Tom Pittman
Box 23189
San Jose CA 95153

The invention of the steam engine in the 18th century by James Watt ushered in the industrial revolution because it allowed machines to assist and replace the physical efforts of human beings. The invention of the computer in our era promises to do the same thing all over, because it allows machines to assist and replace the mental efforts of human beings.

The machines of the industrial revolution were controlled by the physical manipulation of mechanical linkages. The machines of our time are to be controlled by the mental manipulation of verbal linkages -- or at least so if the analogy holds. Actually we are only beginning to scratch the surface of these verbal linkages. This paper deals with one small aspect of man-machine communications.

The computer is a Logic Machine, an information processor. Information in general, and logical propositions in particular, are communicated and manipulated by what we call Language. This fact is the major contribution of Ludwig Wittgenstein to modern philosophy.

A language is defined by the form of its statements and the meaning of its vocabulary, by its syntax and its semantics. The English statement,

"We were playing games."

makes sense to us because its form is correct (subject, followed by compound verb, followed by object), and the words are meaningful. If the word order is different, say,

"Were we playing games."

we have a different sentence with a different significance. Or yet a different ordering,

"We playing games were."

does not make sense, because the word order is not correct for the English grammar, though it may be perfectly valid grammatical form for German.

Similarly,

"We wurdren playing games."

does not make sense because one of the words has no meaning in English.

Computer languages are similarly defined by their form and the meanings of the words:

"10110110 00110011 11111101"

is a perfectly good instruction (sentence) to a 6800 computer, but it makes no sense at all to an 8080 or a 6502. Change the word order,

"11111101 00110011 10110110"

and it also makes no sense to the 6800. Notice, however, that neither of these sentences makes any sense to you. Similarly, none of the English sentences have any significance to a computer. This brings us to the subject of languages suitable for man-machine communications.

To be useful, a language must be understood (in some sense of the word) by both the originator and the receiver. For the moment let us concentrate on the single link from the human to the computer as being the difficult one. The subject matter to be communicated is usually a set of commands directing the computer to perform some task. The language in which these commands are communicated is called a Programming language, and the set of commands is a Program.

The machine's natural language is binary, and for reasons that quickly become obvious to anyone trying to use it, binary is not a practical programming language. Although we hear frequent claims to the contrary, English is also not a practical programming language. We are left with the necessity of inventing a new language.

When two people with differing natural languages wish to communicate with each other, some translation is necessary. One of the two is obliged to learn the other person's language, or possibly they both learn some third language (such

as Swahili or Pidgin, which are used as trade languages in their respective parts of the world), or else they hire another person fluent in both languages as an Interpreter or Translator.

Each of these three alternatives has its counterpart in the human-computer interface. There are those few hardy souls who program in binary absolute, toggling their programs in via switches, and reading their answers from the front panel lights. On the other hand, current artificial intelligence research has computer programs understanding a limited amount of natural English. But a large number of people earn their living translating and interpreting communications between people and computers. And then we have an ever growing number of new languages designed to be understandable to both people and machines. Let us again narrow our scope to examine these new languages.

The first and most obvious new language preserves the form or syntax of the machine language, and translates (or transliterates) only the vocabulary into the Roman alphabet. We call this an Assembly language. Every assembly language statement corresponds exactly to one machine language instruction (actually there are some important exceptions which we will not discuss here). A person writing in assembly language must know almost enough about the machine language to write in it directly. The computer needs only a relatively small number of instructions to properly understand a program written in assembly language. The human writing in assembly language, however, needs a great deal of skill and training. Even so the message is often improperly communicated; we say of this situation that "the program has bugs." In most cases the originator of the message still needs to hire another person to translate from the human version of the message to the assembly language; we call these translators "programmers."

Programmers are a cantankerous breed, somewhat like the scouts of the "Wild West" days. They tend to be social misfits, a necessary link between civilized society and the Unknown but often less sympathetic to us than to the Other Side, be it wild Indians or mad computers (my apologies to AIM and/or IBM). They are few in number (compared to the rest of the population), and very expensive. Therefore we hope for a language more like our own so we can avoid that expense. We call these "High Level Languages" or HLLs. As it turns out, HLLs have not rescued us from the clutches of that villainous breed of programmers at all, but only enabled us (or them) to communicate bigger and ever more complicated messages or programs to the machines. Let us focus on this aspect of programming.

It is generally accepted in the computer industry that a typical programmer will produce an average of one line of debugged and documented code (roughly equivalent to "one sentence, with footnotes") per hour. This figure seems rather low to us hobbyists, but then we probably do not document our own code as extensively as they do, nor, for that matter, do we test it as fully. The important thing to notice here is that the speed figure is relatively constant and independent of the language being used. Couple this with the fact that a program written in Assembly language is some three to thirty times longer (more statements or lines) than an equivalent one written in a HLL and we have a piece of data that translates directly into dollars in a commercial environment.

There are also other reasons for favoring HLLs. We mentioned that HLLs are more like our own language and less like the computer's. The more of the burden of translating that we can put on the machine, the less of a burden it is to us. Furthermore, the language that we speak (or write) affects our thought processes. Some HLLs can help us to think in ways that develop into good thinking habits. Finally, by being more distant from the object machine language, a program written in a HLL has a greater chance of running on a different computer, just as a message in Pidgin may be understood by any of several South Pacific tribes,

whose own natural languages may be as different as Spanish and Hebrew. We will discuss these more fully later.

With so many advantages, what can be said against HLLs? For one thing, not all computers accept them. Those that do require big, expensive, cumbersome, and inefficient compilers to translate them to machine language. The translated program is almost never as good as one translated by a human programmer, and usually 30% to 100% bigger (more machine language instructions) and slower (in machine time to perform the programmed task).

A High Level Language Compiler translates the HLL into an equivalent program in machine language (or possibly into assembly language, which an assembler can accept). There is another way for a computer to accept a HLL. It is called an Interpreter, because the computer accepts and interprets each statement in the HLL at the time the action commanded by that statement is to be performed. The lengthy compilation process is eliminated, but now the program can only run as fast as the interpreter is able to understand each new statement. If an operation is to be repeated, each statement must be interpreted afresh each time. For programs which are not used very often, or which are frequently changed, or for which the execution time is less important than other considerations, this is often the best way to go. Where the same calculations are repeated thousands or millions of times, a compiled program is probably better.

What are we looking for in a HLL? A small business will be repeatedly coming up with requirements of the form, "here is the input data on this file, these are the manipulations to be done, and this is what the output should look like." Ideally the businessman would like to tell that to his programmer, who will then retreat into his hole and come out after a while with a program that does the job. And to some extent this is what happens in the large establishments. But the programmer is expensive, so we would like instead to give the same information directly to the computer and have it crank out a program to do the job. Wishful thinking? Not really. This is exactly the concept behind an RPG compiler: you fill out a special form indicating the type of input data, where the values are, what parts to ignore (as in a database with an entire inventory, but you are only looking for items below their reorder levels), what kinds of computations need to be done (if any; much of the programming requirements are just summary reports), and how to format the output with dollar signs, decimal points, commas, identifying labels, column headings, page numbers, and the like. The less we actually need to specify, the better. So we check a box that says "this is a dollar amount; make it look pretty." If this can be done on a 4K byte minicomputer nine years ago, why can't we do it now on a 16K micro?

Now there are some business applications for which RPG is awkward. One that comes to mind is filled with instructions of the form, "Subtract line 19 from line 18 and enter difference (but not less than zero)." We would like to tell the computer something like, "SUBTRACT LINE19 FROM LINE18 GIVING LINE20. IF LINE20 IS LESS THAN ZERO MOVE ZERO TO LINE20." The language is COBOL. COBOL on the small system? Why not?

Everyone talks about business applications, but most of us come in contact with the personal computers in recreational circumstances. We play games like Star Trek, Hammurabi, and Lunar Lander. Perhaps we need a language designed for writing games. Today almost all games are being written in BASIC, because it is available. Would the availability of a better language change that, or are we stuck with BASIC the way the larger computer systems are stuck with FORTRAN?

What is "good" in a computer language? One of the most hotly contested issues is that called

"Structured Programming" which is a special buzzword meaning languages without a GOTO instruction. Structured programming languages are a lot like organic food. Chemically, all food is organic; all programs have structure. But when we say "organic foods" we usually mean only those grown without artificial treatments in the form of chemical fertilizers or pesticides; when we say "structured programs" we usually mean only those written without the GOTO construct in the language. The proponents of organic food attribute to it the power of life, health and strength; the proponents of structured programming attribute to it the power of correctness, readability and efficiency. The opponents in both cases discount any association of these virtues with the alleged cause, and readily point to spry old people who eat junk food, or magnificent programs written with GOTOS. I do not intend to take sides on either issue, but it is important to realize that just as the kind of food we eat affects our health, so the languages we use affect the quality of our programs.

Another important consideration in the selection of a language is the matter of efficiency. Aside from the question of programmer efficiency, we must also look at the development cycle of a program and its runtime throughput. Most of us with personal computers can expect to spend a much larger part of our time writing and debugging new programs or modifying existing programs to do something different, than in running old programs unchanged. Thus we would favor optimizing the development cycle over the runtime environment, and we should select languages which are interpreted rather than compiled. On the other hand, people with business or control applications may find that most of their processor time is spent in running the finished programs and their interests are best served by a compiler. I think the best situation is a combination of these two pioneered at Lawrence Livermore Labs: an interpreter for initial writing and testing, because it makes changes easy and quick, then a compiler to reduce the debugged program to fast-executing machine language. The interpreter can include such esoteric error diagnostic features as bounds checking, trace mode, and so on. The compiler can assume the program is already correct and concentrate on code optimization. I think we will see more of this.

Let us take a look at some of the HLLs available for personal-sized computers today. Notice that some of these are compilers, some are interpreters, and some are both. Some require large amounts of memory to enable the computer to accept a reasonable program, some are very frugal. Some require mass storage such as floppy disk; some do not. Some cost hundreds of dollars, some are very inexpensive, and some are in the public domain. You may notice that many of the HLLs are variations of BASIC. Here are some noteworthy language processors:

ALTAIR BASIC (8080, 6800) \$200. Considered one of the better BASIC interpreters, it was also one of the first generally available for a microprocessor. Its price is excessive for the normal hobbyist, resulting in wide-scale distribution of illicit copies. There are three sizes of the 8080 interpreter, occupying 4K, 8K and 12K bytes with room for a small program. There are versions which support paper tape, cassette, or floppy disc as peripherals.

PROCESSOR TECHNOLOGY BASIC 5 (8080) \$19.50. Recognizing the problem with ALTAIR BASIC, PTC has chosen to make the software more accessible ("free" if you buy their hardware). The interpreter fills 5K bytes; more is required for program space.

Lawrence Livermore BASIC (8080). This interpreter was developed with government funds, and is therefore in the public domain. Source listings are published in several periodicals. It is not as powerful as some of the others. The interpreter fills 6K bytes, program space is extra.

SWTP BASIC (6800) \$20. 4K, 8K; Paper Tape or Cassette.

SCELBAL (8008,8080) \$49. This is a book containing not only the complete listings for a BASIC interpreter, but the commentary to explain why and how it works.

TINY BASIC (8080,6800,6502) \$5. This is probably the smallest viable language for microprocessors, with the interpreter requiring just over 2K bytes. Source listings for 8080 TINY BASIC and for 6800 "Micro BASIC" (extension of TINY) have been published.

NIBL (SC/MP) This is an extension of TINY BASIC, with emphasis on control applications. The source listing for the 4K byte interpreter has been published, so it is in the public domain.

PL/M (8080) \$975. At this writing, this is the only resident compiler for a microprocessor HLL which is generally available. It requires 65K bytes of memory and a dual floppy. It is obviously not intended for the mass market.

FORTRAN (6800) \$600. This compiler is announced but not yet released at this writing. It supports ANSI FORTRAN, and requires 16K bytes and a floppy.

SEQUENTIAL PASCAL (8080) This is an adaptation of Brinch Hansen's Sequential Pascal, which compiles down to an interpretive code. It is not the same as Standard PASCAL as developed by Niklaus Wirth and his associates in Zurich. The interpreter is slow, awkward, and requires 32K bytes and a dual floppy. It is expected to be placed in the public domain.

This list is by no means complete. It is intended to show the more significant language processors for each of the systems in widespread use as personal computers. There are several other language processors coming out for existing and new personal computers. Nearly every new entry into the personal computer market includes some form of BASIC. There is likely to be several good FORTRAN compilers for less than \$100 by the end of the year. There is a lot of activity in PASCAL, a language which is better in many ways than either FORTRAN or BASIC. And the increasing use of personal computers in small businesses will lead inevitably to the availability of some good business languages.

A PILOT INTERPRETER FOR A VARIETY OF 8080-BASED SYSTEMS

John A Starkweather, Director
 Computer Center
 University of California Medical Center
 3rd and Parnassus
 San Francisco CA 94143

INTRODUCTION

PILOT is a programming system for controlling interactive conversation. PILOT stands for Programmed Inquiry, Learning or Teaching. It has most commonly been used as an author language for Computer-Assisted Instruction. It was first developed at the University of California in San Francisco and has been implemented on a variety of large and small computers. This guide provides a description and operating procedures for a version of PILOT that will run on computers using the Intel 8080 microprocessor. It is an interpreter written in assembly language that requires less than 4k bytes of memory for the interpreter code. Total memory requirements depend on the space required for PILOT program text and will often be no more than 8k bytes.

PILOT is designed to be simple in its syntax so that those without prior computer experience can easily learn to control its features. Dialogue programs can be rapidly constructed and tested.

WHY ANOTHER LANGUAGE

It is possible to imitate the operation of PILOT in a general purpose programming language such as BASIC, APL, FORTRAN, or PL/I. The reverse is certainly not true, for PILOT is a specialized language oriented toward dialogs, drills, tests, etc. If a BASIC program is to be made interactive for the handling of free-response dialog, the programmer must expend a large effort in processing input and arranging for comparisons with possible words or portions of words that might be important to recognize.

PILOT makes this kind of processing easy and keeps the program sufficiently readable that it can be reviewed by someone with a primary interest in the logic of how language content is to be presented. PILOT is relatively poor at kinds of computation handled well by general purpose languages. For many instructional uses, that doesn't matter, but some versions of PILOT allow a mix of programming with a concurrently available general language.

PROGRAMMING PILOT

A SAMPLE PILOT PROGRAM

A PILOT program consists of a series of statements which give a step-by-step description of what the computer must do in order to interact with a person sitting at a keyboard. An instruction code, consisting of one or more letters followed by a colon, defines the statement type and determines its function.

Single letter codes define "core" instructions that occur in all versions of PILOT. Of these, three are most important, and can be used to demonstrate the writing of a simple PILOT program:

```
T: DO YOU ALREADY KNOW ABOUT PILOT?  

A:  

M: YES, SURE, YEP, RIGHT, TRUE  

TY: THEN YOU CAN SKIP THIS INTRODUCTION.  

TN: THIS INTRODUCTION IS FOR YOU.
```

The three codes are:

- T: for "type text"
- A: for "accept an answer"
- M: for "match"

The letters Y and N are "conditioners" that when appended to another code cause it to be effective or not depending upon the success of the last attempted match. Thus, the text "THEN YOU CAN SKIP THIS INTRODUCTION." will be typed only if the answer given to the question about PILOT contains the words "YES" or "SURE". The text "THIS INTRODUCTION IS FOR YOU." will be typed only if the match was not successful. A sequence of operation might be as follows:

```
DO YOU ALREADY KNOW ABOUT PILOT?  

I SURE DO.  

THEN YOU CAN SKIP THIS INTRODUCTION.
```

ANALYSIS OF RESPONSES

PILOT is designed to allow the development of conversational programs that allow relatively free response by the user. Recognition of user responses is accomplished, as indicated in the example above, by searching the responses for specific words or word stems. In ordinary conversation, one or more words in a sentence often carry most of the sentence's meaning. The M: statement is used to define those words, portions of words, or word groups that we wish to be recognized in an answer. Program authors can allow conversational replies to their presented questions and develop a list of word elements which are keys to appropriate recognition and handling of the replies.

PILOT STATEMENTS

There are five more types of "core" statements, using the letters J, U, E, C, and R. J stands for "jump" and causes the PILOT program to continue from another part of the sequence of statements marked by a label in the J-statement. U stands for "use" and causes a marked group of statements to be "used" at that point in the program. The group of statements (called a subroutine) begins with a label which is named in the U-statement and ends with an E-statement. The E-statement, meaning "end", indicates the end of a subroutine or the end of the entire program.

C stands for "compute" and a C-statement contains notation calling for computation of numeric constants or variables. The portion of the C-statement to the right of the colon contains a complete statement in the syntax of a general programming language such as BASIC. The present system is limited to assignment statements.

R stands for "remark". An R-statement contains text which will not affect program operation but which can be helpful as information to the program author.

CONDITIONAL EXECUTION

The letters Y and N are conditioners that affect the operation of any statement type when appended to the code. Operation will then depend on the last attempt to match M-statement patterns with the latest A-statement response. Y: by itself is allowed as a shorthand form of TY: In like manner, N: by itself is allowed as a shorthand form of TN:

A PILOT statement can also be made conditional upon the value of an arithmetic or logical expression placed in parentheses after the instruction and before the colon. In the present system this method is limited to a numeric variable name. If the current value of the referenced numeric variable is > 0, then the statement will be executed. Otherwise it will be ignored. C-statements can be used to set the value of the numeric variable. In the present system, this method of parenthetical condition will take precedence over the Y-N condition.

CONTINUATION

A colon (:) at the beginning of a line indicates continued use of a text statement (T, Y, or N). Thus:

```
T:THIS TEXT WILL BE DISPLAYED  

:AND THIS WILL ALSO BE DISPLAYED.
```

PILOT LABELS

Any PILOT statement may include a label prior to the operation code which allows the statement to be referenced by JUMP (J:) or USE (U:) statements. Labels begin with an asterisk (*) and end with a blank. They may have a maximum of ten characters. A label may also be used as the only element in a line, terminating with the end of line carriage return.

In this system, no complaint is made if there are duplicate labels, but only the first in sequence will be found when the label is referenced.

EXAMPLE:

```
-----  
*LABEL  
T:DISPLAY THIS  
....  
J: #LABEL (causes jump to the above)  
-----
```

STRING VARIABLES

Pilot allows the response to an ACCEPT (A:) operation to be saved as a character string and retrieved later in the program. PILOT will save the response if a string variable name is written after the colon in an ACCEPT (A:) statement. A string variable name begins with a blank or with the end of line carriage return.

Later on, if the string variable name appears in any portion of a TYPE (T:), YES (Y:), or NO (N:) statement, then it will be replaced with the latest response given to the A-statement which contained it.

If there has been no prior response saved for a string variable name in a T, Y, or N-statement, then the string variable name itself will be displayed.

A string variable name may have a maximum of ten characters in addition to the dollar sign. The character string stored as a string variable is limited by the length of response to ACCEPT (A:) allowed at that point in the program.

EXAMPLE:

```
-----  
(program) (execution)  
A:$NAME "ROBERT" entered  
T:HELLO, $NAME HELLO, ROBERT  
....  
T:THIS IS $UNKNOWN THIS IS $UNKNOWN  
-----
```

NUMERIC VARIABLES

In the present system, numeric variables are limited to integer values from -99 to +99. Numeric variable names begin with the pound sign character (#) and consist of a single alpha letter (A-Z). Numeric variables may be used in an ACCEPT (A:) statement to allow a numeric value to be entered and in a TYPE (T:), YES (Y:), or NO (N:) statement for retrieval and display. They may be altered either by an A-statement entering a new value or by a C-statement causing assignment of a new value. They are set to zero at the time that PILOT is initialized.

When a numeric variable name appears in a conditional statement (such as T(X):HELLO) or in a C-statement (such as C:X=A+B), the # character is not used.

EXAMPLE:

```
-----  
(program) (execution)  
A:#X "23" entered  
T:THE VALUE IS #X THE VALUE IS 23  
-----
```

IMMEDIATE OPERATION

Preceding a PILOT statement with **ctrl-Z** causes immediate execution of the statement when it is terminated with a carriage return. Thus, whenever PILOT is awaiting response to an ACCEPT (A:) statement, an immediate statement may be used. After the operation is complete, PILOT will return to expect a normal response to the same A-statement. For example, "(**ctrl-Z**) DP;" will display the current program in memory and "(**ctrl-Z**) U:#LABEL" will cause execution of the referenced subroutine.

Ctrl-Z followed immediately by a carriage return will abort the current program operation and return PILOT to its starting point.

SUMMARY OF PILOT STATEMENTS

PILOT-8080 contains the following "core" instructions:

NAME	CODE	EXAMPLE
TYPE	T:	T: HOW ARE YOU, \$NAME? (\$NAME is a string variable)
ACCEPT	A:	A: \$NAME (may contain \$NAME string, or #X number)
MATCH	M:	M: PATTERN, PATTERN,...PATTERN (moving window string match)

JUMP	J:	J:#LABEL (transfer program control to #LABEL)
USE	U:	U:#LABEL (transfer control to #LABEL, return at E:)
END	E:	E: (end of subroutine or program)
COMPUTE	C:	C: X=A+B (assigns computed value to numeric variable)
REMARK	R:	R: THIS IS A REMARK (does not affect program operation)
YES	Y:	Y:GOOD FOR YOU (same as TY:GOOD FOR YOU)
NO	N:	N:INCORRECT (SAME AS TN:INCORRECT)

Note that the core instructions are limited to single letter codes. These instructions are standard in syntax and operation for many different implementations of PILOT. It is therefore advantageous to write as much program material as possible with the use of core instructions if you wish others to benefit from your work. When extensions to the language are made to meet special needs, multi-letter codes are used and termed keywords to distinguish the difference.

PILOT-8080 contains the following keyword extensions:

NC:	MATCH COMMAS EMBEDDED IN INPUT
INMAX:	SET INPUT LINE LENGTH MAXIMUM
NEW\$:	ERASE CURRENTLY STORED STRING VARIABLES
DP:	DISPLAY CURRENT PROGRAM
LOAD:	LOAD NEW PROGRAM
SAVE:	SAVE CURRENT PROGRAM
PRINT:	PRINT CURRENT PROGRAM
EDIT:	EDIT CURRENT PROGRAM
BYE:	EXIT FROM PILOT

THE APPLICATION OF PILOT TO INSTRUCTION

A major use of PILOT is in the preparation of interactive programs to assist learning. When someone is using the computer as a tool for learning, it is important to make the computer helpful and easy to use. The programming language used for creating computer-based learning aids should also be made readable and easy to learn so that program authors can read the material and make program changes easily.

In addition, equipment complexity can be reduced by avoiding communication problems between machines and using stand-alone equipment. Such equipment can be easily carried to a new location and be used immediately. At the University of California in San Francisco we have demonstrated that interactive instruction useful for medical students learning complex subjects can be accomplished on equipment of microprocessor size. In addition to ease of use, such small equipment tends to be more reliable than more complex systems and any breakdown that does occur can affect only an individual unit.

There appears to be a trend of increasing advantage in using stand-alone equipment for many kinds of individual instruction.

DESIGN AND IMPLEMENTATION OF HI

Martin Buchanan
2040 Lord Fairfax Road
Vienna VA 22180

Abstract

HI is a simple procedural programming system that I have designed and am now implementing. Great care has been taken to make HI simple and elegant, while powerful enough for general use. I hope that it will be suitable as a first programming language, especially for children, and as a language for personal computing.

The Problem

Why is programming a computer in a high level language so often an exercise in frustration?

First, advanced FORTRANs, BASICs, COBOLs, etc. are extremely complex languages, in syntax, and in the number of distinct statements and features.

Second, existing languages often do not reflect principles of programming style, of structured programming, and of functional composition that demonstrably improve program reliability and readability, and programmer productivity.

Third, most existing languages, including BASIC, do not realize much of the potential of the interactive computer system, for example, the direct invocation of functions by the user, the definition and deletion of global data bases available to all programs, and the ability to input expressions rather than simple constants.

Style

Great care has been taken to make HI programs very readable and to encourage good style in HI programming.

Names may be any length, beginning with a letter, and may include digits and an "extension character", defined as the colon, ":" , in standard ASCII.

Functions, global variables, and local variables are syntactically distinct. Functions are always followed by a parameter list in parenthesis, which may be empty. Globals are always prefixed by a period, "..".

There are no reserved words, and no implicit declarations of names (found in FORTRAN and PL/I).

Multiple statements in a single line are not allowed. Continuation lines can be entered indefinitely, until the user enters a ; (not within a character string) or a carriage return. A control-N ("next") allows user control of where a statement is broken into lines.

Comments begin with a semicolon, and occupy the rest of a statement. Comments may be continued just as statements may be. Comments and HI statements may share a line.

Line numbers are used only as a guide to location within the program text, and not at all for branching. Line numbers are generated by the system, and the user does not need to enter them.

Program-oriented text editing gives the user facilities for easier indenting of programs and aligning of comments:

R^C "remarks" tabs to comment column (determined from previous statement with a comment) and enters ";".

S^C "start" tabs to the column containing the first non blank character of the previous line.

I^C "indent" same as S^C minus two columns.

O^C "outdent" same as S^C plus two columns.

A G^C ("bell") is generated when a control function cannot be applied.

Disciplined Flow of Control

Computer scientists have shown that any program flowchart may be composed by nesting a few simple flowcharts. Aside from a linear sequence of statements, HI has only two control structures, a conditional structure and an iterative structure, with a total of six different forms. Syntax and flowcharts for the two main forms appear in Figure 1. There is no branching statement. Learning structured programming and how to structure unstructured algorithms will be a necessary part of learning HI. The complete abolition of the GOTO has important consequences. Syntax is simplified, since labels are no longer needed. A stack can be used for generation of code for flow-of-control, rather than tables of branches and labels. The resulting simplicity of control structures allows the compiler to detect undefined variables and to enforce certain rules regarding the scope of variables (See following).

Disciplined Variable Scope

"I propose such a rigid discipline that:

1. the flow analysis involved is trivial;
2. at no place where a variable is referenced can there exist uncertainty as to whether this variable has been initialized."

- E.W. Dijkstra in
A Discipline of Programming
p. 85

Consider this HI program:

```
DEFINE F(X)
F←X3**2
END
```

X3 is clearly undefined. Some language compilers will set X3 to zero; others will return nonsense; a few will detect the error.

A more subtle problem, is the common conditional assignment:

```
DEFINE F(X)
IF X >=0
  F←X**2
END
.
.
.
END
```

If X is less than zero, F is undefined. Admittedly, a variable only conditionally defined may only be used later when the condition is true, but there is always a program structure possible which will isolate within the one conditional clause all calculations dependent on that condition. In HI, a variable which is conditionally defined is treated as local to the conditional block where it is defined, and is deleted on clause completion (See following rules).

A third problem, encountered in programs which call other programs, is tampering with the calling program's data by a subprogram. In HI, a program

may not tamper with its parameters, neither by assignment nor by input, and the only value returned to the calling program is associated with the function name (program name). This is not as restrictive as it seems, for a HI program name may return any valid HI data structure.

distinguished by their parameter list. Programs may create and delete globals, or change their value.

Used to delete global or local variables is the HI deletion statement:

```
TEMP← ; LIMIT THE SCOPE OF A TEMPORARY
; VARIABLE USING THE HI DELETION
; STATEMENT.
```

There is no distinction between system globals (those supplied with HI) and user defined globals, allowing the user to change (or mess up) the HI operating system. HI presumes that the user is running on either a dedicated personal system or a virtual dedicated machine, and can interfere with any feature of HI without interfering with other users.

During execution, nonsense (such as a subscript out of range, or an attempt to input a string rather than a number, etc.) which depends on run-time values for its detection, will be detected and will cause an error.

In summary, several features of HI's treatment of variable scope and definition are important:

1. a program may not change its formal parameters;
2. a variable initialized in an IF or ELSE clause is considered defined outside that clause only if it is initialized in both the IF and ELSE clause;
3. the use of an undefined variable is detected by the compiler;
4. Global variables provide a growing permanent base for each user's programming needs;
5. the deletion operation is available to limit the range of temporary variables;
6. during execution, the range of subscripts, the lengths of vectors in distributed operations, and the types of operands are all monitored for correctness.

Disciplined Composition

A key development in programming languages is the ability to compose programs from other programs. This composition may be via a text copying or "macro" system, or via closed subprograms passed parameters and returning results. HI encourages functional decomposition of problems into subproblems whose solutions can be formulated as a single HI data structure, though global variables may be used for more extensive communication of data between HI programs. The calling program is shielded from "side effects", that change local variables and are caused by invoking a subprogram, by the prohibition of assignment to formal parameters. The global availability of programs, and the direct accessibility of functions to the user, encourages the development of "software tools" by the user, utility programs used in the program development process.

Interaction

What kinds of things does the computer system say to the user or the user to the system? The user may want to execute a certain program with certain parameters, input the value of a certain expression, perform direct calculations, or enter or change program text. The computer system may indicate an error in a program, either at program entry, compilation, or during execution. The computer system may also detect errors in input data, and must output data during program execution. It should be made clear that many of the commands used on present language systems are means, and not ends, for the user. The user may have to say LOAD file, COM file, RUN file, but his real interest is program execution.

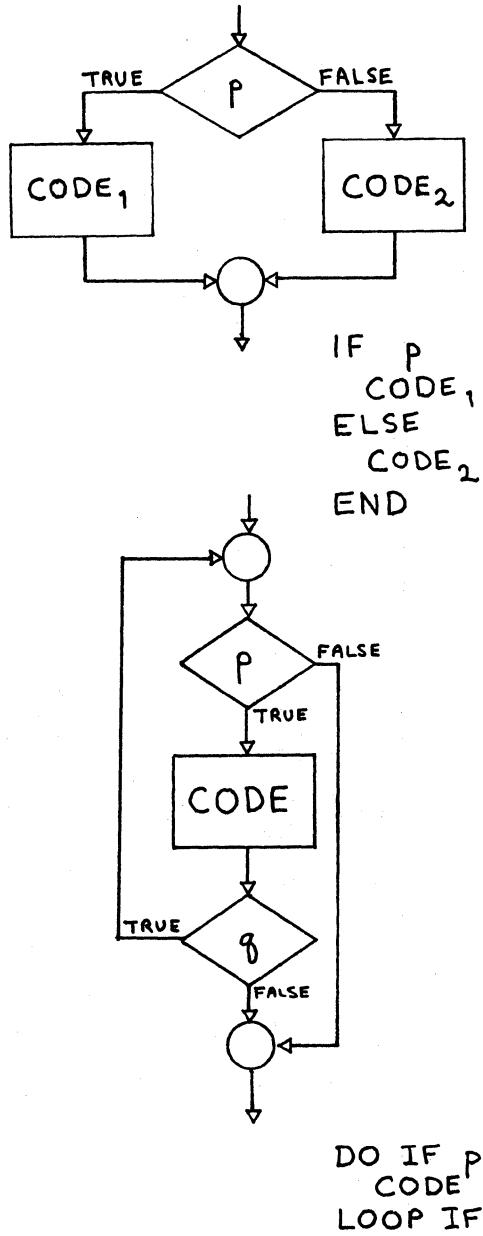
Accessible to all programs is the world of HI globals. A global has universal scope, and endures until explicitly deleted, as contrasted with local variables within programs, which are unknown outside their program, and are deleted on program exit, if not sooner. All HI programs are themselves globals, accessible by the user or any other program. The global concept is extremely powerful, allowing the user to define data used by many programs and directly by the user, such as:

```
.PI←3.141592
.TRUE←1
.MILES←5280*.FT
```

The period prefix is used to denote global data, and is not used for program references, which are

The goals of the HI interactive features are:

FIGURE 1



1. early, clear, and comprehensive error detection, and easy correction;
2. easy execution and testing of programs;
3. extremely simple input and output, allowing formatted output without introducing any additional statements or syntax.

During program entry, a syntax-scanner verifies that each line is a valid HI statement or comment. If an error is detected, the line is not appended to the working file, an error message is typed, and the user re-enters the line. This feature alone will eliminate all syntax errors except those involving the compound structures such as DEFINE-END, IF-ELSE-END, and DO-LOOP.

The compiler imposes a strict discipline with regard to detecting undefined, or possible undefined variables, as previously described. Other errors detected include mistakes in the syntax of compound structures, and type errors for variables whose type is known (attempting to add a string, for example). Execution monitoring of variables is described previously. If at all possible, the compiler will be incremental, allowing incorrect code to be corrected during compilation.

All error messages are in plain English, and during compilation, give line numbers, print offending lines, and name the guilty variables.

To execute a program, simply type its name followed by a parameter list in parentheses (which may be the empty list). If a program has not been compiled, or its source text has been changed since it was last compiled, it will automatically be compiled. Since the user may directly invoke functions he or she writes, the need to write special test programs for such functions is eliminated, an extremely powerful feature. For example, to directly test a sorting function:

```
SORT(5 3 7 9 1)
      1   3   5   7   9
;UNDERLINED ITEMS ARE TYPED BY THE COMPUTER.
```

Input is an operator, the unary question mark, "?", which gets an expression from the user and assigns its value to the variable following the question mark. The prompt "?" is typed on the terminal. The user terminates each input with a carriage return, though a carriage return-linefeed will be sent only if there is no more I/O to be performed on the line. The user may input expressions, which is an extremely powerful feature of HI:

.

.

.

WHAT IS THE HYPOTENUSE OF THE RIGHT TRIANGLE WITH SIDES OF 14.5 AND 7? SQR(14.5**2+7**2)

SLANT RANGE? 57*.MILES

The applications of expression entry to scientific simulation alone (where unit conversions are a significant headache) are major. Computer-aided instruction is a second major application area.

Output in HI is extremely simple - just place the expression or sequence of expressions to be output on the same line. Each element of an integer vector becomes a 6 character string. Formatting functions allow formatted output without special syntax (See language summary following).

```
"SAMPLE OUTPUT" 1 2 3
SAMPLE OUTPUT    1   2   3
SKIP(1)
B(10) L("ALPHA",3)
      ALP
```

Formatting functions in HI include:

SKIP(n) returns n CR-LF

B(n)	returns n blanks
L(string,n)	returns string left justified in n length string
R(string,n)	returns string right justified in n length string

Input and output may coexist on one line, for example:

```
"FLIR APERTURE" ?APER
FLIR APERTURE? 108*.INCHES
R((?A+?B),10)
      325      8
```

Calling a program, input, and output, are all examples of the HI evaluation statement, a sequence of concatenated expressions. Each expression in turn is evaluated and printed if anything other than the null string results. If an expression is simply the inputting of a variable, then input and printing are coincident.

Interpreter

The HI interpreter executes three of the six HI statements or structures, evaluation, assignment, and deletion. Each interpreter session corresponds conceptually to execution of a program, in that local variables may be created which are automatically deleted when a session is over. A session begins on power-on or when a previous session ends, and the system will say:

HI

A session ends when the user types END, just as a program is ended.

Data Types, Structures, and Operations

HI allows character, real, integer, and logical data types. Conversion of types in one direction is automatic, but in the other direction, may result in an error:

every logical value is an integer value;

every integer value is a real value;

every real value is a sequence of characters.

The basic HI data structure is the indexed set. An indexed set of atomic data elements is called a vector. This basic structure can be generalized to include arrays, trees, and lists. In the smallest version of HI, TINY HI, there are no real numbers or arrays, but the syntax and semantics for these features have been defined.

As in APL, operations can be distributed over vectors, and vector subscripts are allowed.

Character and numeric/logical constants are distinct even on input, as characters must always be enclosed in quotes.

Basic arithmetic, relational, and string operations are provided. The operator "number", "#", returns the number of elements in a vector. The infix operator "ellipsis", "...", generates an integer vector from its first scalar operand to its second and is often used to designate substrings.

Implementation

I began to try to design a better general purpose programming system in January 1976, while a student of Computer Science at George Washington University. The HI system has gone through 6 distinct versions already, and I am still struggling with certain problems involving the internal representation of real numbers, declarations, handling trees, and an editor for program entry. I have begun to code basic operating system functions for dynamic storage allocation, and for accessing globals, on the Intel 8080. I and one other person

are working on Intel 8080 implementations. A comprehensive language standard for the full HI language (including real numbers, data declarations, operator definition, arrays, trees), and for the TINY subset may not be out for several months. I expect that my personal implementation of HI may take me two more years of part time labor.

HI is in the public domain, and persons or companies interested in implementing HI please write to me at the following address, and I will send information. If I get too large a response, I will institute a postage and copying charge - till then, its free:

Martin Buchanan
2530 N. 18th Street
Arlington, Virginia 22201

(703) 527-1909 (home)
(202) 333-6800 (work)

The dynamic character of HI data will cause HI programs to be only semicompiled - the resulting code will call many HI supporting routines to perform run-time checks, allocate and free storage, link with global variables and subprograms, evaluate expression, etc.

It is my hope that a basic HI system will run on a personal computer with 16K bytes main memory and some form of random access secondary storage, whether floppy disk or bubbles, CCD, or videodisc. Even if HI is more expensive, it is worthwhile to plan it now, given the exponential decline in computing costs.

I will do all I can to standardize all features of differing implementations and act as a clearing house for HI information.

I believe that HI is practical for personal computing, and that HI is a good language for talking to computers. Even if you find fault with some specific idea or feature of the language, I hope that you can appreciate the need that motivates it, and perhaps take an interest in improving it or implementing it.

Though HI is in the public domain, implementations are not. I hope that the manufacturers of personal computing systems will consider a proprietary HI as a good investment in systems software.

Conclusion

Perhaps ten million Americans will learn computing programming within the next ten years. A language is needed for personal computing that combines extreme simplicity with general scope and power, and that encourages style, structure, and good composition in programming.

TINY HI SUMMARY

Vocabulary: DEFINE END IF ELSE DO LOOP
 Comments: ;
 Arithmetic operators: + - * / ** unary -
 Relational operators: = > < <= >= <>
 Logical operators: & | ~
 Length operator: #
 Ellipsis (sub vector operator): ...
 Concatenation operator: blank or other lexical separator
 Subscripting: []
 Nesting: ()
 Input: ?
 Global name: .<name>
 Function reference: <name>(<expression list>)
 Assignment statement: <variable>← <expression>
 Deletion statement: <variable>←
 Evaluation statement: <expression>
 Control Structures: IF p DO [IF p]
 code1 code
 [ELSE
 code2] LOOP [IF q]
 END

Data types: character, integer, logical

Data structures: the vector

Data conversion: logical to integer, and integer to six character string.

Biography

Martin Buchanan is a Computer Science student, scientific programmer, and occasional writer. For the past year he has been obsessed with how we talk to computers, and with getting his computer to say more than */?@l5A!.

Bibliography

Dijkstra, E.W. A Discipline of Programming, 1976

FORTRAN FOR YOUR 8080

Kenneth B. Welles II, Ph.D.
2823 Fenwick Road
University Heights OH 44118.

High level languages are a tremendous aid to both hobbyist and professional programmers. The oldest high level language, FORTRAN, has now become available for use with the 8080 processor in the form of the FORT//80 compiler from Unified Technologies Incorporated (distributed by Realistic Controls Corporation). It is easiest to give the user an idea of the characteristics of FORT//80 by comparing it to the most prevalent 8080 high level language: BASIC.

There are several popular versions of BASIC available in different sizes with different features and different execution speeds. All of these versions share one common factor, they are all interpreters. With interpretive programming, the user loads the interpreter, loads the source program, and executes his program.

FORTRAN's major difference from BASIC is that FORTRAN is a compiler. To compile a FORT//80 program, the user loads the compiler, loads the source program twice, and receives a compiled object code program which may then be loaded and run. The time penalty incurred in having to make multiple passes with FORTRAN is offset by the fact that the compiled program executes much faster than the corresponding BASIC program.

A comparison table follows which briefly lists several major differences between BASIC and FORT//80. No single version of BASIC is used in the comparison, but the most popular features of the most frequently used BASIC's are listed. Also no attempt was made to show all of the inherent structural differences such as GOSUB versus CALL or DO 10 I=1,200 versus FOR I=1 TO 200.

<u>BASIC</u>	<u>FORTRAN</u>
Interpreter	Compiler
Slow execution	Fast execution
Fast program modification	Slow program modification
Run time debugging available	Symbol table available for debugging
4K, 8K, 12K memory required to run	16K memory required to compile
No punch device needed	Some punch device required
Supports only one console device directly	Supports Reader, Punch, List device and multiple ASCII terminals
Handles one or no interrupts	Handles 8 interrupts and reset line, allows interrupt handlers to be in FORTRAN
Usually allows only one entry to user programs, and passes only one or two variables	Allows any number of entries to user programs, with any number of parameters.
Runs only in low memory	Compiles only in low memory, but runs final program anywhere in memory
Some interpreters can be stored in ROM	Any part of the compiled program can be stored in RAM or ROM anywhere in memory
Only one user program in memory at any time	Multiple programs can reside and be run simultaneously
Multiple statements per line	One statement per line
No continuation lines allowed	Any number of continuation lines are allowed
PRINT USING	FORMAT
PEEK, POKE	A single dimension statement makes the entire memory accessible as an array
String Manipulation	Strings in arrays, or I/O
SIN, COS, EXP	To be added
Some integer and floating point arithmetic	1,2 byte integers, 4,8 byte floating point (up to 16 digits of accuracy).
Decimal and ASCII values allowed	Binary, octal, decimal, hexadecimal and ASCII values are valid anywhere
Only one or two letters allowed to uniquely define a variable	Up to 31 characters allowed in each variable name

From the preceding list, which is definitely not an exhaustive description, you can see that there is no clear cut "winner" between BASIC and FORT//80. Different applications with different requirements will dictate whether BASIC or FORTRAN is the preferred language. The definite winner is the user, who now has a wider variety of powerful programming tools at his disposal.

Kenneth B. Welles II received his PhD from Cornell University in Applied Physics in the area of computer image processing, with a minor in Computer Science. He has designed and constructed microprocessor interfaces for TV camera input, vector graphics output, IBM Selectric output, and a floppy disk drive controller. He has written a complete floppy disk operating system for the 8080 in conjunction with this controller. He is currently employed by the Lighting Research Division of General Electric as a research physicist where he designs and develops microprocessor based process control equipment.

EMUL-8: AN EXTENSIBLE MICROCOMPUTER USER'S LANGUAGE

Bob Wallace
Box 5415
Seattle WA 98105

ABSTRACT

EMUL-8 is a "paper design" for an interpreter language based on the Intel 8080. The language has a basic similarity to BASIC or PASCAL. There are eight main data types: integer, pointer, short real, long real, string, class, label, and entry. The syntax is very free-form; there is no distinction between main programs, subroutines, function calls, commands, and operators. Structured control expressions (IF, CASE, and LOOP) are provided. The interpreter is very modular and not difficult to understand. EMUL-8 works closely with EMOS-8, an operating system presented in another paper.

Introduction

This paper describes a "paper design" for an interpreter language, based on the Intel 8080 or other microprocessor. The overall design goals include:

Modularity - The interpreter consists of a relatively small top-level driver, and a number of functionally separate command routines, using a standard set of calling conventions.

Extensibility - The user can add new command routines (written in assembly language or EMUL-8 itself) to add new operations on existing data types.

Conviviality - The overall design is clean and simple, and will be well documented down to the source code level. Advanced users can understand the interpreter and modify it easily.

Before I describe the language, here is a short example which calculates the sum of the squares of the elements of a vector. This function is invoked in a program with:

... size SUMSQ vector ...

where "size" is an integer giving the number of elements to sum, and "vector" is the vector of integer elements. The routine:

```
>>SUMSQ
#TOTAL := 0
#1 FORLOOP INC #1 UNTIL > A.CV
    #TOTAL := (TOTAL +(A.1 [I] ** 2))
    POOL
RET #TOTAL
```

Except for some number signs and added parenthesis, EMUL-8 programs look much like BASIC or PASCAL programs.

Data Types

An EMUL-8 program is made up of names (variables and commands), and constants, separated by delimiters such as blank and parenthesis. There are eight kinds of names (data types): integer, pointer, short real, long real, string, class, entry, and label. The first four are called numeric types, and the first five, variable types; the last two are command types.

Integers and pointers each have a 16 bit value; integers range from -32767 to 32767, and pointers from 0 to 65535. Bytes and single characters are kept in integer variables, in the low half of the word. Relational expressions (such as A = B, or NAME = "SMITH") produce an integer value, 1 if the relation is true and 0 if the relation is false. Integer and pointer constants can be decimal numbers (0, 126, -12345), hex numbers (\0, \FFFF, \AO1), or characters ('A', 'BC', '\\$'). Octal constants could be added, I suppose.

Dyadic operators (commands) on integers and pointers are:

+ addition	&+ logical OR	= test for equal
- subtraction	&* logical AND	<> test for not equal
* multiplication	/& logical XOR	> test for greater
/ division	:- store word	>= greater or equal
// remainder (MOD)	% store byte	< test for less than
** raise to power	:= assignment	<= less than or equal

Monadic operators on integers and pointers are:

-- negative	/2 right shift	% swapped bytes of
% logical NOT	%, left byte of	@ word at address
*2 left shift	.% right byte of	% byte at address

For example, to rotate right the word whose address is in P:
P := (@ P & * 1 * \8000 + & /2 @ P)

A format for short and long real numbers has not been finalized, but will probably be a 4 and 8 byte floating point representation. Real constants can be numbers with a decimal point (0., 14.71, -.71828), or can be in scientific notation (0.E+0, 3.14E+1, -3140.E-3). Sorry, the scientific format does not allow embedded blanks (as in 3.14 E 1). Integers and pointers may occur in real number expressions, and are converted automatically; a real expression can be assigned to an integer (or pointer), and is truncated (add .5 at the end to round).

Dyadic operators on real numbers:

+ addition	** raise to power	:= assignment
- subtraction	/ division	(all relational)
* multiplication	// remainder (MOD)	operators)

Monadic operators on real numbers:

LOG	log base 10	SIN	trig sine	INT	integer of real
LN	log base e	COS	trig cosine	SR.	short real of int
EXP	e to power	TAN	trig tangent	LR.	long real of int
SQRT	square root	ATAN	arctangent	ABS	absolute value
-- negative		RND	random number	SIGN	sign (+1 or -1)

For example, to calculate a harmonic series:

#VAL := (SCALE * SIN (SPEED * THETA + PHASE))

A string is an arbitrary block of bytes (frequently characters), of any length. String constants can be text strings ("Alpha", "THAT'S IT", "Doublequote \""), or more complicated block constants ([LEFTPTR, RIGHTPTR, \F1, "text", \AO * 32]). The format of a block constant allows strings to be constructed using:

- byte values and integer values (8 or 16 bits)
- pointers to other strings or data elements
- results of programs embedded in the block constant
- multiples and arrays of the above

This facility is real handy for creating control blocks and list structures; the full format specs are beyond the scope of this paper.

Operators on strings ("n" is integer value, "s" is string):

n HEAD s	make a string from the first n characters of s
n TAIL s	make a string from the last n characters of s
s JOIN s	concatenate two strings (result is a string)
s FIND s	find first string in second string, result is integer index to first character matched
s COPY nn	make a string from first index through second
s INS ns	insert second string into first, at index n
s DEL nn	delete bytes from first index through second
LEN s	result is integer length of string
EXEC s	execute string as EMUL-8 program
ATOM s	convert string as if EMUL-8 atom
@ s	get pointer to start of string
s := s	string assignment
s rel s	all relational operators (=, <, etc) allowed

For example, to change a name from "firstname lastname" format to "lastname, firstname" format:

```
#NF := (NAMEFL FIND " " - 1)
#NL := (LEN NAMEFL - NF - 1)
#NAMEFL := (NL TAIL NAMEFL JOIN ", " JOIN(NF HEAD NAMEFL))
```

Classes are not fully defined yet; they will be a subset

of SIMULA classes. The class concept allows new data types to be added (such as PASCAL records) with associated operators, simulation of concurrent processes with RETACH and RESUME commands, and other interesting applications. I'll just briefly mention that the data associated with a class variable includes a pointer to the class name entry, a pointer to a local name list, and a pointer to the current execution character.

Entry names and label names are pointers to programs in machine language and EMUL-8 code, respectively. All the operators listed for the other data types are just function calls; each is either a label or an entry name. There is no distinction between operators, commands, functions, subroutines, and main routines. If the user wants a new operator, say "++", to do something, s/he just writes an EMUL-8 program with the label name "++". This will be covered more later.

The variable data types (integer, pointer, short real, long real, and string) can be made into arrays, of any size and any number of dimensions. There is no "dimension statement"; an array is created as a block constant and assigned to a new variable, as in:

```
#ARRAY := [10 BY 12 BY SREAL]
```

Square brackets are used to index arrays; indexing starts at 1:

```
#ARRAY [1, 12] := (2 * VECTOR[12])
```

The Language in General

First, EMUL-8 is an expression language, like ALGOL or BLISS. Every piece of code has a value (and type):

```
17      has the value (integer, 17)
2 * 3.6    has the value (short real, 7.2)
#I := 56     has the value (integer, 56)
IF (I = j) 8 ELSE -8 FI has the value (integer, 8 or -8)
SUBROUTINE ARGUMENT    has the value returned by SUBROUTINE
"text" DEL 2 3        has the value (string, "tt")
```

As EMUL-8 executes, it uses the current value (or CV), much the way a computer uses the accumulator. For variable data types, the occurrence of the name loads the value into the CV. For commands, the occurrence of the name causes the command to be executed, which can use and modify the CV. Commands can also use arguments following. For example:

```
17      put (integer, 17) into the CV
+ 18    add the argument (integer, 18) to the CV
<35    if the CV is less than the argument, make CV 1 else 0
#T      put (variable name, pointer) into the CV
:= "XYZ" assign the argument to the variable name in the CV
```

Most interpreters do much the same thing, except they have operator precedence. For example, they do all multiplications before any additions. EMUL-8 does everything strictly from left to right. (APL, by the way, does everything strictly from right to left). For example:

```
BASIC LET VAR = 7 + 2 * 8  VAR becomes 23
EMUL-8 #VAR := (7 + 2 * 8)  VAR becomes 72
```

Parenthesis can change the order of evaluation:

```
BASIC LET VAR = (7 + 2) * 8  VAR becomes 72
EMUL-8 #VAR := (7 + (2 * 8))  VAR becomes 23
```

As mentioned, EMUL-8 consists of names, constants, and delimiters. Names (both variables and commands) may have 1 to 31 characters, and may use any ASCII-128 characters except blank, comma, semicolon, number-sign, the parenthesis, and the square brackets. Also, a name can't look like a constant.

Legal names: A ALPHA beta GAMMA +-*/\$ IT'S \$.100
Illegal names: A,B 12#7)(HI 123 'to' ;;;

A delimiter must appear between two names, two constants, or a name and a constant, and cannot be embedded within a name or a constant. The delimiters are:

- blank, comma (a comma is equivalent to a blank)
- a line boundary or record boundary
- a comment (a semicolon through a line or record boundary)
- the parenthesis (used to group expressions)
- the square brackets (used for block constants and arrays)

Variable names are global or local. Global variables are accessible to any program, and exist forever once defined. Local variables are only accessible in the program defining them, and are newly created each time the program executes and released when the program returns. By the way, EMUL-8 allows completely recursive programs; a command can call itself.

In most languages, the type of a variable is set in a declaration statement or implied by a naming convention. In EMUL-8, the type of a variable is set the first time a value is assigned to it. This first assignment also creates the variable. Using a variable before it is created, or assigning the wrong type of value to an existing variable, are error conditions (except for automatic conversions among numeric types).

Names may have a prefix to create them or assign a value to them. The "#" prefix creates a local variable (if necessary) and allows its value to be modified. Examples are:

```
#X := 5    assign (integer, 5) to X
INC #I    add one to the value of I
#A FOOBAR  allow the command FOOBAR to change A's value
```

The ">" prefix operates the same way for global variable names:

```
>#MAXMEM := 16383
```

The ">>" and ">>>" prefix define command names and class names, respectively. Class names are always global; command names are global unless they occur in a class program.

Lack of operator precedence, required blanks between things, and clumsy assignment syntax are the main restrictions in the language. However, these factors greatly add to the simplicity and modularity of the interpreter.

EMUL-8 Syntax

I'll describe the syntax in terms of a program file, made up of programs, made up of sections, made up of expressions, made up of atoms.

An atom is a variable name (perhaps with a prefix), a constant, a command with its arguments (which are also atoms), or an expression in parenthesis.

An expression is any string of atoms and control expressions. There are three kinds of control expressions: IF-expressions, CASE-expressions, and LOOP-expressions. I'll get to these in a moment.

A section is a named expression. The section is headed with a label definition, such as ">>LABEL". This section is now called whenever the label is encountered in another program. If a return is not executed in a section, the flow of control simply falls through to the next section (i.e. a program may have several entry points).

A program is one or more sections, terminated with an END command. If the END command is encountered, an automatic return is executed. Programs can also be named, with a class name. Class programs are headed with a class name definition, such as ">>>PICTURE".

A program file is an operating system file containing one or more programs.

Control Expressions

An IF-expression takes one of two forms:

```
IF atom expression FI
IF atom expression ELSE expression FI
```

If the atom's value is not zero (or null for strings), the first expression is executed, else the second is executed. Examples:

```
IF (I > J) #MAX := I ELSE #MAX := J FI
#MAX := (IF (I > J) I ELSE J FI)
IF BOOL, IF NAME RET I FI, ELSE INC #I FI
```

A CASE-expression executes exactly one of a series of expressions, depending on the value of an atom:

CASE atom

```
IS atom1 expression1 IS atom2 expression2 ...
ELSE expression-final
ESAC
```

If the first atom's value is equal to that of atom1, expression1 is executed; if it's equal to atom2, expression2 is executed, and so on. If none of the atoms are equal, the ELSE expression is executed. At most one of the expressions is executed; the ELSE part is optional. Example:

```
CASE OPCODE
IS 'AD' #R := (A1 + A2)
IS 'SU' #R := (A1 - A2)
ELSE #ERROR := 17, RET Ø
ESAC
```

A LOOP-expression, in its simplest form, just loops forever, as in:

```
LOOP PRINT "I'm trapped", CR ! POOL
```

Naturally, there are several loop exit commands. WHILE exits a loop if its argument is false (zero or null); UNTIL exits if its argument is true (not zero or null). Example:

```
LOOP
UNTIL (@ PTR = Ø)
INC #PTR
POOL
```

EXIT directly leaves any number of levels of looping. It takes one argument, the number of levels to exit from. Example:

```
#I := 1 LOOP
#J := 1 LOOP
IF ARRAY [I, J] EXIT 2
INC #J UNTIL > 10
POOL
INC #I UNTIL > 12
POOL
```

Note that these exit commands can appear anywhere in a loop, allowing a much greater amount of flexibility than (say) the PASCAL WHILE-DO and REPEAT-UNTIL loops. Note also that I haven't included a GOTO command, although one could probably be added.

Often a loop is executed a fixed number of times, as in a FOR loop in BASIC or PASCAL. As mentioned, the INC command adds one to its argument, which must be a variable name with the prefix "#". The DEC command has the same format but subtracts one; the INC2 and DEC2 add or subtract two (since word pointers change by two). Since these commands leave the incremented or decremented value in the CV, and the UNTIL and WHILE commands do not modify the CV, they can be used together, as in:

```
INC #I UNTIL > SIZE
DEC2 #TABLEPTR WHILE >= BASEPTR
```

There is also a STEP command, allowing a variable to be modified by some value other than 1 or 2. Examples:

```
#TADDR STEP 4
#K STEP -14
#BITS STEP 8 UNTIL = 64
```

The FORLOOP command is exactly like the LOOP command (and is terminated by POOL), except that it zeros the preceding variable name. This curious property is used almost exclusively to make loops such as:

```
#I FORLOOP INC #I UNTIL > 10 expression POOL
```

(This was as close as I could get to the standard "FOR I=1,1,10" statement in other languages, while sticking to the simple EMUL-8 syntax and interpreter driver).

Command Routines

As mentioned, the occurrence of a label name causes the EMUL-8 section with the label to be executed. Upon entry, only one local variable name is defined, "A.CV". It contains the current value from the calling program. Arguments (atoms) following the label in the calling program are accessed with the local names "A.1", "A.2", and so on. These arguments are

passed in true "call by name" fashion; it's as if the atom "A.n" is textually replaced by the n'th argument atom. For arrays, an argument can be an entire array or one element of it. For example, the LOG command expects one real number for its argument; it could be used with a real array element, as in "LOG ARRAY [2, 3]". On the other hand, a SORT command would expect an entire vector as an argument, so it could be invoked by "SORT VECTOR"; inside the SORT routine, elements are accessed with "A.1 [I]". The value of an argument can be modified; for example, "INC #I" is just "#A.1 := (A.1 + 1)". Most commands act like function calls; they return a new value in the CV. The RET command returns from a routine, and RET's argument is the CV to return. For example, "INC" returns the new value of its incremented argument:

```
INC #A.1 := (A.1 + 1), RET A.1
```

When a command returns, all the arguments it accessed are skipped in the calling program. A return also releases all local variable names, and all data associated with these names. This is very important; it allows recursive operation, by creating a fresh set of local variables each time a routine is invoked. By the way, class programs keep their local name list from call to call, making recursive operation more complex.

EMUL-8 Bubble Sort

```
;The SORT command is invoked by "size SORT vector", where
; "size" is the number of elements to sort in "array". SORT
; returns a sorted copy of the array.
;► SORT
;first create new array, copy arg array to it
#VECTOR := [A.CV BY A.1 [I]]
#I FORLOOP INC #I UNTIL > A.CV
#VECTOR [I] := A.1 [I]
POOL
;outer loop repeats until no elements switch
LOOP
#FLAG := Ø ;1 if switches
#I FORLOOP INC #I UNTIL >= A.CV
IF (VECTOR [I] > VECTOR [I+1])
#TEMP := VECTOR [I]
#VECTOR [I] := VECTOR [I+1] ;switch elements
#VECTOR [I+1] := TEMP ;if not in
#FLAG := 1 ;sort order
FI
POOL
WHILE FLAG ;repeat if switch
POOL
RET VECTOR ;return sorted
```

Interpreter Implementation

Most of the implementation algorithm is designed; however, I'm still working on details regarding argument passing, arrays, and class concepts. No code has been written yet. In this section, I'll first define some data structures: the name list, allocated blocks, and the type code. The middle part discusses temporaries, arrays, and assignment. Finally, I'll describe the interpreter driver itself, and its three routines: GETATOM, GETARG, and EMULATE.

Name Lists

Every name created in an EMUL-8 program has a name list entry somewhere in memory. This name list entry contains the ASCII name itself, a one byte type code, and a two byte value:

NAME	T	VAL	→ to next name list entry
------	---	-----	---------------------------

All the ASCII characters of the name have their high-order bit on, and the type code (at least in name list entries) has the high-order bit off, so the type code also delimits the end of the name.

The global name list is accessed by first creating an 8 bit hash code, consisting of the lower 4 bits of the first two characters. This hash code indexes a 512 byte table of pointers to a list of name entries having the same hash code. Since there are many global names defined at any time, I think the cost of a 512 byte table is justified by the increase in access time.

The local name list is just a linked list of the local name entries. Since there are few local names in any given

local name list, and there are many such lists (one for each program nesting level, plus one for each class variable), a 512 byte hash table for each is not worth the memory used.

The name lists are only accessed by calling standard system routines, for security reasons. These routines are GETLOCAL, GETGLOBAL, PUTLOCAL, PUTGLOBAL, and FREELOCAL. These are explained later.

Blocks

Pieces of data that need more than 16 bits (real numbers, strings, programs, etc.) are kept in blocks, allocated from a memory area called the heap. The heap builds from the bottom of memory upward, and the stack builds from the top of memory downward. Naturally, if these two areas meet, memory capacity has been exceeded and the system aborts (unless a dynamic array or program can be moved to the disk to free up space). The format of a block is a two byte length, a one byte type code, and the data bytes; the high order bit of the type code (in blocks) is 1 if the block is in use, and 0 if the block is free. The heap is always accessed with the two standard routines GETHEAP and FREEHEAP. GETHEAP is passed a type code and the length of the block needed, allocates memory from the heap, and returns a pointer to the block allocated. FREEHEAP is passed a pointer to the block, and frees the block for later use.

Type Code Byte

The type code byte is found in three places: in a name list entry, in the current value (CV), and in an allocated block. The format is:

Bit 0 - in a name list entry, 0 to delimit the name
- in allocated block, 1 if in use, 0 if free
- in the CV, temporary data if 1 (see below)
Bit 1 - BLOCK: value is pointer to allocated block
Bit 2 - NAME: value is pointer to name list entry (# form)
Bit 3 - ARRAY: block in array format
Bits 4-7 indicate a particular data type:
CODE TYPE VALUE IS
0000 integer 16 bit integer
0001 pointer 16 bit pointer
0010 short real pointer to data block
0011 long real pointer to data block
0100 string pointer to data block
0101 argument pointer to arg in calling program
0110 entry pointer to machine code routine
0111 label pointer to EMUL-8 code routine
1000 unassigned zero (newly created name)
1001 dynamic pointer to block with disk address
1010 class name pointer to local label name list
1011 class var pointer to control block
(1100 to 1111 are free for expansion)

Temporaries

When a computation is in progress, intermediate results must be saved in temporary memory locations, and these locations must be released when no longer needed. Consider the following real number calculation:

(PETER + MARY) * (14.7 - BOTH)

The + command gets two arguments (PETER and MARY), and neither is a temporary, since the contents of each must be preserved during the calculation. A new block for a real number must be allocated (using GETHEAP) for the sum, a pointer to this block put into the CV, and the CV type becomes "block, temporary, short real" (11000010). The - command gets one temporary argument (14.7) and one non-temporary (BOTH). The computed difference can be put into the temporary, wiping out the 14.7 which is no longer needed. The * command receives two temporary arguments. One can hold the result; the other must be released (using FREEHEAP), or it will float around the heap forever.

Arrays

Arrays are kept in an allocated block. The first data byte is the number of dimensions; it is followed by two byte fields containing the size of each dimension, and then by the actual data. An array internally contains the number of bytes per element as an added dimension; i.e., a 2 x 2 array of long reals looks internally like a 2 x 2 x 8 array of bytes. An array of strings is just an array of pointers to string blocks. Arrays of class variables are legal, but not defined yet.

Note that operations involving arrays may use one element

of the array or the whole thing. If array brackets follow the array name, a single element is used; if the brackets are omitted, the whole array is used. One array can be assigned to another as a whole; other array operations (such as in APL) may be added later. The type code distinguishes between the whole array and a single element (in a way I haven't defined yet).

Assignment

The assignment operator ":=" is somewhat complicated, due to all the type codes involved. Conversions among the numeric types occurs automatically. A string can be assigned to a pointer variable, setting it as a pointer to the string. If the left part is a newly created name, the type must be assigned as well. If the left part is a block of some kind, FREEHEAP must be called to release the block, since after the assignment there will be no way to access it. If the right part is a non-temporary block, a new block must be allocated by calling GETHEAP, and the data copied to it. Assignment involving arrays adds its own complexities; an assignment with an array element on the left implies the data is copied, and then the right side must be released if it is a temporary. Note that for all assignments, the NAME bit must be set in the left part's type code.

The Interpreter - Preliminaries

I need to introduce another concept: the current context. The current context is much like the program counter in a computer; it contains a pointer to the next byte of an EMUL-8 program to execute. In addition, it contains a pointer to the local name list, and a pointer to the previous context saved on the stack. Later, I'll explain how the current context is used during command calls. The pointer to the current program character is kept in the local variable PROGPTR.

As mentioned, the name list entries are accessed using standard system routines. GETLOCAL looks for a name starting at PROGPTR, searching the current local name list. If no match is found, PROGPTR is unchanged and the CV type code is set to a special flag value, \FF. If a match is found, PROGPTR points at the first delimiter following the name, the CV value becomes a pointer to the value field in the name list entry, and the CV type code becomes that of the entry. GETGLOBAL works in the same way, using the global name list.

PUTGLOBAL and PUTLOCAL just add a name to one of the name lists. The name itself is found at PROGPTR; the type and value are in the CV. FREELOCAL scans the local name list and releases all list entries, as well as all data blocks associated.

The interpreter also calls GETCONS, to look for a constant at PROGPTR. As in GETNAME, if a constant is found, GETCONS advances PROGPTR to the first delimiter after the constant, and sets the CV to the constant's value. The "temporary" bit is set in the CV type code. If a constant is not found, PROGPTR is unchanged, and the CV type code is set to \FF.

The Interpreter - Operation

The actual interpreter routine, EMULATE, just starts executing a program at the current context. As we'll see, it is very simple; most of the work is done in GETARG. GETARG is really the heart of the interpreter; it is called both by EMULATE and by assembly routines which need to get the argument following an entry name. GETARG first calls GETATOM, so I'll start there.

GETATOM does the dirty work for GETARG; it returns the next program element in the CV, like the fetch cycle in a computer. GETATOM first skips blanks, commas, end-of-line, and comments (semicolon through end-of-line). It looks at the character(s) at PROGPTR, and does one of the following:

If "(" or ")" is at PROGPTR, the CV type is set to the special flag value, and the CV value becomes 128 or 129, respectively.

If "#" is at PROGPTR, skip it and call GETGLOBAL. If the name is in the global name list, an error has occurred; the user is creating a local name which conflicts with a global name. If a global name isn't found, call GETLOCAL. If a local name is found, and it is an array, check for a square bracket following, and call GETCONS to pick up the array indices. The array element address is calculated and put into the CV. If a local name is not found, call PUTLOCAL to create it.

If ">#" is at PROGPTR, the above occurs for global names. Again, a global name cannot be created which conflicts with an existing local name.

If ">>" is at PROGPTR, the label definition is skipped, and the next atom accessed. If ">>>" is at PROGPTR, an error has occurred, since it should have been preceded by END.

If none of the above apply, try calling GETGLOBAL, then GETLOCAL, then GETCONS to recognize the atom. If none of these find anything, an "unrecognized symbol" error has occurred. If a name is found, load the name's value into the CV; for arrays, this involves calculating the array element address (for reals and strings), or array element value (for integers and pointers).

The GETARG routine first saves the CV on the stack, and calls GETATOM. Based on the return, it does one of the following:

If a "(", call EMULATE (recursively); when it returns, just return.

If a ")" , simply return to EMULATE, which will cause it to return as well.

If the command RET is found, GETARG is called (recursively) to pick up the CV value to return; if the value is not a temporary, it is copied to a temporary. FREELOCAL is called to free the local name list and it's data blocks.

If an argument name is found (A.1, A.2, etc.), and it has not been encountered before, a pointer to the atom text for the argument in the calling program must be located, based on the previous context's PROGPTR. This PROGPTR must also be updated to skip over arguments used. Once a pointer to the argument is found, the current context is pushed on the stack, and a new context (a pointer to the argument and the calling program's local name list) established. GETARG is called (recursively) to load the argument into the CV.

A name of type entry causes the CV to be restored and the assembly routine whose address was in the CV to be called. When the assembly routine returns, GETARG returns. I hope to be able to pass all registers between assembly routines, and not just the CV. By the way, in the 8080 the CV type code is the C register, and the CV value is the DE register pair.

A name of type label causes the current context to be pushed on the stack, and a new context created. The address in the CV becomes the new PROGPTR, and the local name list is initialized to the name A.CV containing the saved current value. EMULATE is called (recursively); upon return, the current context is popped off the stack, and GETARG returns.

For dynamic data types, the actual data or routine is not currently residing in memory, so it must be located and loaded. This feature isn't completely defined yet.

For all other data types, GETARG just returns, since all the work has been done in GETATOM.

The EMULATE routine itself is just a short loop, executed "forever". In the loop, GETARG is called. If a ")" or "RET" is round, EMULATE returns, else it continues to loop.

Final Thoughts

Although the language and interpreter design have gone through several iterations, it is still not complete. Class variables have only recently been added to the language, and need a lot of conceptual work; arrays and argument passing also need some shaking down. Calling assembly routines (entry names) needs tighter specifications, as well; this involves the operating system linking conventions. An EMUL-8 compiler is another possibility I'm allowing for.

I don't feel proprietary about the EMUL-8 design; the concepts (and this paper) are free for anyone to use, as long as I'm given credit and receive a copy of anything based on the design. However, if and when I program the EMUL-8 interpreter, I may copyright and sell it. I'd like to receive any feedback based on this design; I hope to start implementation the summer of 1977.

EMUL-8 is designed to work closely (although not exclu-

sively) with EMOS-8, an operating system covered in another paper. Input/output commands have not been covered in this design; they will be a function of an EMUL-8 / EMOS-8 integration yet to come.

About the Author

Bob Wallace wears several microcomputer "hats" in the Seattle area. He is a founder, past newsletter editor, and currently secretary of the Northwest Computer Club. In addition, he is publications manager for The Retail Computer Store; he orders all their books and magazines. Finally, he is a graduate student in computer science at the University of Washington. In his spare time, he consults for New World Computer Services, Inc., does event logistics for ComNet, and tries to get his computer running. He is 27 and single.

(EMUL-8 is being designed under contract for New World Computer Services, Inc.)

EMOS-8: AN EXTENSIBLE MICROCOMPUTER OPERATING SYSTEM

Bob Wallace
Box 5415
Seattle WA 98105

ABSTRACT

EMOS-8 is a "paper design" for an operating system based on the Intel 8080. The system offers a memory allocation mechanism; a method of using user names rather than addresses for subroutine linking and other chores; the ability to do multi-tasking on a time-shared basis; and compatibility with EMUL-8 (described in another paper), CP/M by Digital Research, and PL/M by Intel. The most important facility is the input/output system, which features logical unit names assigned to physical devices; a generic I/O device model allowing device independent I/O; and a file control block. Utilities include a video display module driver and editor; and assembly, load, and debug facilities. The EMOS-8 design is not proprietary.

Introduction

This paper describes a "paper design" for an operating system, based on the Intel 8080 or other microprocessor. The overall design goals are the same as the EMUL-8 language: modularity, extensibility, and conviviality. In addition, EMOS-8 is tailored to the two basic design goals of any operating system:

- to allow a wide variety of programs to run with a wide variety of hardware configurations, and
- to make program development as easy as possible.

Most of EMOS-8 is concerned with I/O operations. All I/O uses a general model of a peripheral, allowing fixed or variable length records, collected into a file. Programs do I/O with logical units, which are assigned to particular devices. Therefore, EMOS-8 is not a floppy disk operating system or a cassette operating system, it is a general use operating system.

In addition to I/O, EMOS-8 also supports timeshared and interrupt-driven multi-tasking; memory block allocation and release; loading and linking of program modules; editing of textual and binary files; and various debugging facilities. Since all routines are modular, there is no fixed memory requirement, except that the first 16 memory bytes (and first two restart / interrupt vectors) are used for various purposes.

EMOS-8 is being designed concurrently with a language called EMUL-8, described in another paper. Naturally, it is expected that the two will be used together. The EMUL-8 design started as a simple command language for EMOS-8; it sort of grew (almost by itself) into a fairly powerful programming language. However, EMOS-8 and EMUL-8 could be run separately.

Modularity and extensibility are supported with well defined parameter and linkage conventions. The user can easily add new capabilities (such as device drivers or user commands). The particular module configuration can be changed for different purposes and hardware configurations; this can even be done dynamically, as subroutine calls and parameter passing can be based on the name of the routine or parameter as well as the address. EMOS-8 is convivial, in that it offers a set of facilities to the user, who is free to use them, modify them, or build in new components. It is intended that source code always be distributed, and be very well documented, both at the program level and with a "principles of operation" manual.

There are two problems with this open approach. First, there is much less protection within EMOS-8 from user errors. Fortunately, most microcomputers are used as single user systems; at least errors affect only one person. However, since microcomputers are inexpensive there tend to be more users, and therefore less experienced users. Internal consistency checking maintains some security. The other problem with supplying source code is the ease of copying it, and the lack of financial rewards for the designer based on copying. However, to make money selling programs, the programs must run on as many machines as possible. If and when I write the main EMOS-8 programs, I intend to put them in the public

domain, and encourage their use. Then many machines will be able to run the application software I'll be selling later.

Although the operating system is not partial to any particular file device (such as audio and digital cassettes, floppy disks, memory blocks, etc.), it is partial to a particular interactive device, the Video Display Module (VDM) and keyboard. A VDM is simply a section of main memory that is also displayed on a video monitor or television set, as a number of ASCII characters. The VDM and a keyboard is more flexible, faster, and cheaper than a traditional terminal. A CRT or hard copy terminal could be used instead, but some EMOS-8 features (particularly editing) won't work as well.

Memory Use

The Intel 8080 (and other microcomputers) has a hardware stack using main memory, building from the high addresses downward. It also vectors interrupts and one-byte instructions (restarts) which act as subroutine calls to locations in the first 64 bytes of memory. EMOS-8 uses memory as follows:

```

0000: jump to reset routine / monitor interrupt / trace / boot
0003: reserved for Intel operating system I/O byte
0004: reserved for CP/M compatibility
0005: jump to direct I/O system call, CP/M format
0008: restart for assembly language call-by-name, EMUL-8 entry
000B: pointer to global name hash table
000D: 2 byte length of first allocated block
000F: 1 byte type code of first allocated block

0010 to 003F: interrupt / restart vectors. These can be managed by EMOS-8 or left alone for the user.

0040: beginning of allocated memory, building upward
XXXX: beginning of stack, building downward
XXXX+1 starts the resident portion of EMOS-8; may be PROM
YYYY: bootstrap PROM location (could also be in low memory)
ZZZZ: VDM memory addresses (could also be allocated at boot)

```

EMOS-8 uses a static memory allocation scheme. The main block of memory between the first 16 bytes and the stack is called the heap. Memory blocks of any length can be allocated and released from the heap, but blocks are not moved, so the heap is never "compacted". Each block starts with a two byte length and a one byte type code. The high order bit of the type code is 1 if the block is in use; the entire type code is zero if the block is free. The allocation algorithm is first fit, starting at the base of the heap. Adjacent free blocks are collapsed during allocation. The total of all lengths of all blocks must equal the memory size, and the block containing the stack must be free, or an error has occurred. Memory is allocated by call GETHEAP, with the type code in register C and the length in register pair DE (standard EMOS-8 / EMUL-8 convention); on return, the DE pair contains a pointer to the block found. The user is informed if there is not enough memory for the allocation. Memory blocks are released by calling FREEHEAP, with a pointer to the block in the DE pair. Both GETHEAP and FREEHEAP check that the heap is in the correct format. Both are also reentrant, so they can be used in a multi-task environment.

Multi-Task Operation and Interrupts

The Intel 8080 provides a primitive hardware interrupt system. EMOS-8 provides a command to vector exactly one occurrence of an interrupt to a particular address; if a second interrupt occurs, an error message is given. Normally an interrupt handler will set itself as the entry point for the interrupt if another is expected, re-enable the interrupt system, and return. If no more interrupts are expected, it normally sets the end-of-record status flag in the file control block associated with the interrupting device, re-enable interrupts, and return. The operating system catches any errant

interrupts following. Interrupts may also be used to invoke "foreground" tasks, such as real-time data acquisition. All interrupt routines must leave the stack and all registers as they were found.

If the hardware includes a real-time clock, a single priority task-switching (and time keeping) clock interrupt routine is available. Any number of tasks (user programs, peripheral spoolers, etc.) may be logically running concurrently; in actuality they are given control of the machine in "round robin" fashion. Each task must have its own stack. Any task, when given control for its time-slice, may pass control to the next task in line; if it is (say) waiting for the end-of-record flag to be set by an interrupt handler, and has nothing better to do, the task might as well let another task get some work done. Tasks may also create new tasks and terminate themselves as tasks; however, there is a fixed maximum number of tasks assigned when the EMOS-8 system is generated.

Please note that this task-switching clock interrupt routine and the task skip, activate, and terminate routines are not in themselves a complete time-sharing system. Such a system usually includes a shareable, reentrant language processor, such as EMUL-8 or BASIC. I'm not sure I can make EMUL-8 reentrant, and I haven't heard of an 8080 reentrant BASIC. Still, the facility is interesting, and may be useful for some applications. I do intend to make the memory and disk space allocation routines reentrant, so that these two important resources can be managed in a multi-task environment.

Named Entity Facility

It is desireable for the user to be able to name various entities in the system: programs, entry points, variables, logical I/O units, file control blocks, and so on. There is a general global name list; each entry contains the ASCII name, a type code byte, and a two byte value (usually a pointer to the entity). There are standard system routines used to access this global name list: GETGLOBAL and PUTGLOBAL. This facility is covered in the EMUL-8 paper, so I won't repeat it here.

Assembly language routines can call each other by name, rather than using a linking loader to fill addresses in the call instructions. The assembly language programmer uses the restart 1 instruction, followed by a pointer to the name of the routine, rather than a call instruction. All registers and flags are passed unchanged to the called routine, and the top of the stack contains the return address, just like a normal call. Arguments may be passed in registers, on the stack, or following the "call"; the process is transparent to the calling and called program. The advantages are the ability to trace subroutine calls during debugging; the option of having some routines non-resident, and only loaded into memory when called; the ability to record the amount of time used by any routine; a simpler assembler and loader; and the ability to write routines in assembly or EMUL-8 as required.

The main disadvantage of this "call to entry name" facility is the time overhead to look up the name. Since many routines are really not called that often, this isn't a bad problem. However, a special option allows the first occurrence of a "call to entry name" to construct a machine language call to the routine and insert it into the code of the calling routine. Most of the advantages listed for the "call to entry name" are lost; however, since the actual linking is done by the operating system, the assembler and loader still do not need to be concerned with linking.

Input/Output - Files

All input/output operations handled through the operating system are channeled through one routine: DOIO. All parameters and intermediate storage used by this and lower level routines are kept in a file control block (FCB). Rather than calling DOIO directly, many user programs will call individual operation routines instead. These routines can be tailored to the I/O requirements of a particular user program, such as a BASIC or EMUL-8 interpreter. Examples of operation routines might be to read an entire record, read the next byte of a record, read the next ASCII format real number from a record, read the next ASCII format number from the last record of the console device; these routines are meant to be as general or as special purpose as required. Some will pass an entire FCB; some may only pass a couple of parameters for inclusion into an FCB; some may only

need to pass a single byte.

Operation routines may call other operation routines, but eventually one will call DOIO to actually transfer I/O data. DOIO is a very short routine; basically all it does is call the actual device driver, whose address is in the FCB. The various device drivers all look the same to DOIO; they process the same operation codes and set the same status bits. This indirect method of doing I/O uses a little more time and memory than is absolutely required; however, it is structured, and easy to comprehend and modify.

All I/O devices must act as closely as possible like the EMOS-8 generic I/O device. A generic input/output device transfers fixed or variable length binary records of 0 to 32767 bytes; records are collected into a file of 0 to 32767 records, which is ended by an end-of-file record. The generic device can also skip records, forward or backward by record, forward to the end of a file, or backward to the beginning of a file. Obviously making all actual devices conform to this standard sometimes leads to clumsy situations. For example, paper tape and audio cassette controllers usually cannot skip backwards, and a floppy disk is capable of skipping many records at once; still, I think it is a reasonable compromise.

User programs do input/output with a logical unit, having a three character name. Using the naming facilities of EMOS-8, the logical unit name is associated with a pointer to a unit assignment block 12 bytes long, containing:

- a 2 character device name, itself associated with a pointer to a device driver entry point (one driver program may have several entry points, each with its own name).
- a 1 character drive number, from 0 to 9 or blank. This allows specifying a particular tape drive, printer, etc.
- an 8 character user file name, left-justified and padded with blanks if the name needs less than 8 characters. Non-file devices, such as printers or terminals, have none.
- a 1 character file type, indicating the kind of information in the file (A for assembly source, S for saved memory image, etc.); implies ASCII or binary data for the editor.

For example, the editor reads a logical unit called EDR and writes to a logical unit called EDW. The user can give a command, such as:

#EDR := "TP1:PROGRAM.S" #EDW := "PT"

and the editor will read file PROGRAM.S from tape unit one, and write to the printer. This command may be entered interactively before starting the edit; it may be given by a program about to call the editor; or it may be given as a response to an error message from DOIO that no logical unit has been assigned.

File Control Block Format

There are eight operation codes an individual driver is expected to handle (the op code is in parenthesis):

- | | |
|------------------|-----------------------------|
| (0) close a file | (4) skip to start (rewind) |
| (1) open a file | (5) skip back one record |
| (2) get a record | (6) skip forward one record |
| (3) put a record | (7) skip to end-of-file |

The "overlap" mode bit in the status byte indicates an I/O operation is just initiated by the call. The operation is not actually complete until the "complete" flag is set in the status byte. This facility allows various forms of overlapped I/O and computation processing. If the "overlap" bit is zero, the driver does not return from a call until the operation is actually complete. For simple I/O systems without interrupts or spooling tasks I/O cannot be overlapped and this bit is ignored.

Four 2-byte fields are used to control the user record buffer. The first is a pointer to the start of the buffer (and the record). The second is a pointer to the "current record byte"; this is used by formatted I/O operation routines to write a byte, read an integer in ASCII format, etc. The third field is the length of the record to read or write. The last field is the length of the buffer itself (since variable length records may be read, both these fields are necessary).

There are several different processing modes; a simple read/write bit is not enough for many applications. On the other hand, for devices (such as paper tape units) which can only read or write, all these modes will be interpreted as

either a simple read, simple write, or can't-handle-it error. These processing modes are:

- (0) read records; read mode for simple devices.
- (1) read and release; this mode is used by memory files (discussed later), when a temporary file is read and each record is released to the heap after reading.
- (2) mixed read and write; allowed for fixed length records only, on devices which can do it (digital cassette, floppy disk, etc.)
- (3) disk sector access; used with the disk only to read and write actual disk sectors or relative file sectors.
- (4) write new file; write mode for simple devices.
- (5) over-write file; allows a named file on a directory device to be written over.
- (6) write with backup; writes a new file, then renames the old file with a Y (ASCII) or Z (binary) type code to indicate a backup file. An existing file with the Y or Z type code is deleted.
- (7) append to file; skips to the end of a file, and writes records on the end of it.

2 byte length of file, in records
 2 byte length of file, in blocks (in bytes for a memory file)
 2 byte allocation factor for file, in blocks (bytes for m,f.)
 2 byte starting block number (memory address for memory file)
 2 byte record length, if fixed (zero or maximum if variable)
 2 bytes unused (for future expansion)

Device Handlers

The individual device handlers are responsible for executing the basic 8 operations, initiating interrupt handlers, buffering records into blocks, and all error conditions. The driver interacts with the user as necessary to deal with errors; error conditions are never passed back up to DOIO and the operation routines, and user programs never have to handle I/O error conditions. There is usually very little a user program can do about an I/O error except tell the operator, anyway. Since many aspects of driver operation are identical (such as record buffering) there is a set of utility routines shared by all drivers. Many of the driver operations below could be better handled by separate spooling tasks; this is beyond the scope of this paper.

The simplest file device I'll call a serial device. Examples are an audio cassette deck (which may use a variety of encoding methods and formats), and paper tape (Teletype (TM) or high speed reader / punch). Various driver entry points may be used for different physical record formats; remember that a driver entry point is associated with a 2 character device name. Actually, each driver entry point can support one ASCII and one binary record format. Most serial devices will have one file per volume, will not have a directory, and will have one variable length physical block per logical record (although more advanced drivers could add these features). Generally, a serial device can be started and stopped, but not rewound or switched from read to write under computer control. However, file names are supported; the user becomes the file directory mechanism. For example, if the user wants to edit to an audio cassette file, at some point the logical unit is set, as in:

```
#EDW := "AC: MYFILE.S"
```

When the open command is received by the driver, it sends a message to the user on the terminal:

```
MOUNT MYFILE.S ON AC, RECORD
```

The user puts the audio cassette containing MYFILE.S into the drive (cassette player), does a fast forward if needed to position the tape to the file, pushes the "forward" and "record" buttons, and hits "return" on the keyboard to signal the driver that the tape is ready. If the driver needs to do a "skip to start" or "skip back one record" operation, the user is instructed to rewind the file; to skip back a record, the file is skipped forward until it is positioned correctly.

The next step up is a digital cassette deck. Since these can be controlled by the hardware (rewound, set to read or write, positioned), they can be used as full directory devices, having an initial directory file and several files residing on a volume. However, only one file per drive may be open at a time. The driver buffers variable length records into fixed length blocks. Blocks in a file are recorded sequentially, and a fixed number of blocks are allocated for each file. Each block has a header, containing the block number, and a displacement to the first record starting in the file (used for back-space). Each record has a header, containing the record number and the length of the record. The digital cassette driver can do all operations by itself, although the back-space operation is usually hard on the tape and should be avoided (as in audio cassettes).

The floppy disk driver supports all I/O operations easily; in addition, multiple files per drive may be open at one time. Variable length records are buffered into fixed length sectors; optionally, disk sectors may be read and written directly. When a file is created, it is given an "allocation factor", in sectors. If the allocation factor is 1, individual sectors are chained together, and the logical sector order for the file has no fixed relation to the position of the sectors on the disk (the driver will try to allocate sectors so as to minimize read time). The allocation factor may include all sectors that

<u>File Control Block</u>		
Bytes	Field	Set By
<u>Basic Information:</u>		
1	operation code: 0:close 2:getrec 4:rewind 6:skip +1 1:open 3:putrec 5:skip -1 7:to end	user
1	status byte: b0: operation complete if 1 b1: overlap mode if 1 b2: file open if 1 b3: ASCII if 1, binary if 0 b4: end-of-file reached if 1 b5: end-of-record flag, set by interrupt routine b6: reserved for EMOS-8 b7: reserved for user	driver
2	pointer to driver entry point	DOIO
1	processing mode: 0: simple read 4: simple write 1: read and release 5: over-write 2: read and write 6: backup write 3: disk sector I/O 7: append write	user
3	logical unit name, in ASCII	user
2	free for user	
<u>Record Buffer Data:</u>		
2	pointer to user record buffer	user
2	pointer to current record byte	varies
2	length of record, set by user/write, driver/read	driver/read
2	length of buffer (max record length)	user
<u>Driver Information:</u>		
1	drive number, binary 0-9	DOIO
1	free; reserved for driver	
2	record counter	driver
2	block counter	driver
2	pointer to block buffer	driver
2	disk sector address: high bit 0: relative sector number high bit 1: track and sector	user/ driver
<u>Directory Entry:</u>		
2	length of file, records	driver
2	length of file, blocks (or bytes)	driver
2	allocation factor	driver
2	starting block number (or address)	driver
2	record length, bytes (or zero or max)	driver
2	reserved for directory information	
<u>Reserved for Driver:</u>		
8	temporary data for reentrant operation	driver
48	bytes total	

More advanced devices (such as digital cassettes and floppy disks) support a file directory facility. The first file on the device (except perhaps a bootstrap) is the directory file; each record contains the entry for one file. This directory can be edited to create a new (empty) file, delete an old file, or rename a file. The directory entry format is also standardized for all devices. The entry format is:

1 byte free; zero for a free directory entry slot
 8 byte file name
 1 byte file type

are expected for the file; in this case the sectors are physically contiguous. Finally, the allocation factor may be a sub-multiple of the file size; say 8 sectors. Then the file is allocated contiguous sectors on one track, in groups of up to 8. The groups themselves may be in any order on the disk (again, attempting to minimize read time). A bit-map of free sectors is maintained by the disk driver for allocating new sectors. However, the bit-map method of listing the sectors used for each file would take too much room in memory, so the sectors in a file are chained forward and backward. A sector header includes the sector (i.e. block) number, a forward sector chain, a backward sector chain, and a displacement to the first record in the sector (8 bytes total).

I hope eventually to add an access-by-name mechanism for floppy disks, as well as a relational data base system with inverted file indicies.

A special driver allows main memory to be used as a file device. Each record is allocated as it is written; in "read and release" processing mode, each record is also released as it is read. This is very useful when a file is edited (or otherwise translated) and memory is limited. If a memory file runs out of room while it is being written, it can be transferred (with the user's help) to an external file device, and processing can continue. The memory file driver supports all I/O operations and multiple files may be open at one time.

Naturally, other file devices, such as a holofile or a parallel / serial CCD or bubble memory could easily be added to the system, transparent to all currently written user programs. That's the great advantage of doing I/O with logical units instead of the physical devices themselves; upgrading to a new and better peripheral, or the breakdown of an I/O device, or transferring to a different machine, can all be done without having to modify user programs.

The VDM and Editing

The video display module and keyboard driver is fairly complicated, since the display is handled as two "windows" and editing is done partially with the driver rather than with a separate editing utility. At the lowest level, the driver reads the characters typed at the keyboard (using interrupts or polling for a "keypressed" condition), and manages the section of memory displayed on the CRT. The display area is usually 16 lines of 64 (or sometimes 32) characters, but the generic VIM driver can use any number of lines or characters per line.

The display area is divided into two windows, called the display window (upper portion) and the command window (lower portion). There are two corresponding driver entry points, and two device names (DS and CM). The command window is used for operating system commands and error messages, and the display window for text being edited; user programs can use either. As text records are written in a window, it expands at the expense of the other window; the other window may be left with a minimum of one line. There may be a line of dashes separating the two windows. Records are scrolled off the top of a window when new records are added to a full window or the other window expands. Records scrolled off the top of the display window are automatically written to the display-write logical unit, DSW; likewise, records scrolled off the top of the command window go to the command-write logical unit, CMW. These logical units may be assigned to the "dummy" driver if they do not need to be collected into a file.

The editor is of the scrolling type; the file being edited is scrolled through the display window, and the user can edit it as it goes by. Lines (records) in the display window can be inserted or deleted; characters can be added or erased, the cursor can be moved around, and so on. The line(s) at the cursor can be written to the display-write unit, allowing the lines above it to be saved and moved to another position in the file. The display-read unit, DSR, is assigned to the file to be edited; lines can be read to the display window, either to the bottom or to the cursor line. A search command allows lines to be continuously read until a search pattern is found. Normal keyboard functions such as rubout and cancel line are also supported.

There are actually two editing modes: text and binary.

As a default, the file type code determines the editing mode, but this can be changed. In text mode the editor operates on ASCII characters. In binary mode, a record is displayed and edited in hexadecimal format as well as character format. A display line consists of a relative address (not editable) on the left, a hexadecimal display in the center, and a character display on the right. Editing either the hexadecimal or the character display automatically edits the other. One special editing mode allows all of main memory to be a "file" allowing main memory to be edited; the starting point for this edit can be an absolute address or a named entity.

Editing and the VIM driver are somewhat complicated by the fact that a line usually does not correspond exactly to a record. If the record is shorter than a line, the convention is that the line is padded with blanks when it is displayed, and stripped of all but the last blank if it is written to a file with variable record lengths. For records longer than a line, a special "continue" character is put at the end of each line to be joined into a record. When a record is read to the display, all the lines forming the record are scrolled in; individual lines scrolled off the top are collected into records before being written.

The command window also has a logical unit for reading lines (command-read, CMR); a special command allows one, a number of, or an entire file of records to be read to the command window, allowing a batch-like operation. Each window has its own cursor, but only one cursor is "active" at one time (indicated by blinking). One keyboard character is used to switch the active window; also, if a program writes to a given window, that window becomes active.

It is possible to use a CRT or hardcopy terminal as the console device; however, a completely different editing style is needed (meaning a different editor); lines written to either window are just output to the console; and there is no logical unit for automatically writing lines (CMW and DSW). The VIM driver allows the cursor to be positioned to a particular line with the "skip" operations; naturally these have little meaning with a hardcopy terminal, although some CRT terminals may support them. A really advanced CRT terminal and driver could even simulate the operation of a VDM device and driver.

Multiple VDM units are allowed, and interrupt-driven keyboard input supported, so (with the task-switching facility) limited time-sharing could be supported. I hope to make all drivers reentrant, so their code can be shared among tasks. However, reentrant programming depends heavily on machine registers, and the 8080 register set (let alone the register set of some other microprocessors) may make this too difficult.

Loading and Assembling

The loader supports relocatable object files, but does not have to link object modules together, since this is done by the "call to entry name" facility. However, the loader must add all entry names defined in an object module to the global name list. Since all assembly language global entry points are named, the user can invoke them for testing purposes and dump them from the console.

The assembler must produce relocatable object code, and also must keep global names in a separate area so the loader can put them into the global name list. The object file consists of three records; each record is further sub-divided into lines in Intel hex format. The first record contains the name list entries for all global names; the format of each entry is the ASCII name of the entry, a one byte type code (hex 46), the relative address of the entry point, and two bytes for eventually linking the entry into the name list. The second record of an object file contains the actual machine code, assembled with a base address of zero. The third record contains the relocation information; simply a list of relative addresses of words in the assembly program to be relocated.

The assembler mnemonics are the standard Intel plus Processor Technology set. Eventually I'd like to do an assembler which also takes another set of mnemonics, such as "HL+1" instead of "INX H", "B=C" instead of "MOV B,C", and so on. A standard absolute assembler could be used to produce relocatable code, if the programmer includes the name information and relocation information as constants in the program.

An EMUL-8 loader is needed as well, to scan the EMUL-8 text for label and calls names and put them into the name list(s). The same loader could be used for both, since EMUL-8 programs always start with ">>" and relocatable object files will not.

The loader and assembler have their own set of logical unit names for input/output, as follows:

ASR - assembler source
 ASW - assembler relocatable object output
 ASL - assembler listing output
 ASE - assembler error message output
 ASI - assembler user input, for error correction
 LDR - loader object input
 LDW - loader memory image output (optional)
 LDE - loader error message output
 LDI - loader user input, for error correction

The loader should be small enough that it can be resident, allowing dynamic loading and unloading of programs under user or program control. This facility will be expanded to be a virtual subroutine facility: if a routine is called, and has a name list entry indicating it is not resident, the file containing it can be loaded dynamically. If memory is needed to load it, another program file, such as the "least recently used", can be swapped out to the disk. This only works if routines are called by entry name, and not if they are actually called with a machine language call instruction. Some studies have indicated most subroutines used in a program are invoked rarely, so this may not add too much time overhead. This allows a 64K program to execute in (say) only 16K bytes of actual memory. This dynamic link facility should also work for data blocks as well as programs.

Debugging Facilities

As mentioned, main memory can be examined and edited, using the editor. A monitor interrupt (either a keyboard key or a separate button) is supported, to location zero. This allows a user to suspend a program and examine the registers, examine memory, set a breakpoint, or initiate a trace. If the monitor interrupt is invoked twice within one second, a bootstrap load is initiated from one of the file devices after user confirmation. The trace can operate in single instruction mode, or subroutine entry mode. It can also operate in "slow motion", executing instructions or subroutines at a much reduced rate and displaying the contents of the registers. The subroutine trace, if a real-time clock is available, can keep track of the time spent in each routine, so inefficient routines which are executed often can be identified and rewritten (perhaps from EMUL-8 to assembly) to execute more efficiently.

The user can dump the current memory image, and reload it at a later time. This memory image file is the same format used by the bootstrap loader; the first record is only the stackpointer (all registers and the current program counter have been pushed on the stack), and the next record contains the entire memory image. Since both EMOS-8 and EMUL-8 are extendable, the user may have spent some time loading the exact configuration wanted and can save this configuration for later use.

Two non-resident routines are available to examine the global name list, and to examine the memory blocks allocated on the heap. Another can be used to examine the task queue in a multi-tasking environment. Data in these areas can also be modified by these routines.

Compatibility

I hope EMOS-8 and EMUL-8 can be as compatible as possible with PL/M programs and the CP/M operating system (made by Digital Research, Pacific Grove, CA). Both CP/M and EMOS-8 use the eight lowest memory bytes in the same way, and programs written to do I/O with the CP/M system will operate as well under EMOS-8. PL/M subroutines can call EMOS-8 or EMUL-8 routines; however, PL/M function calls require that the returned value be in the A,B and HL registers (for the cross and resident versions of PL/M, respectively). Therefore PL/M calls will have to go through a small interface routine. EMUL-8 programs can be written to do I/O with the CP/M operating system. If the loader for PL/M is modified to create global name list entries, EMOS-8 routines could call PL/M subroutines.

Again, calling PL/M functions would require a short interface routine.

I make a plea for standardization to all operating system designers: first, use the eight lowest memory bytes as follows (CP/M and EMOS-8 convention):

- 0 - jump to warm start routine
- 3 - Intel I/O byte
- 4 - free byte (I don't think it's used by CP/M)
- 5 - jump to I/O system; the I/O system entry point is also assumed to be the top of available memory.

When the I/O system is called, the function code is in register C and another argument (usually a pointer) is in register pair DE. For EMOS-8, the function code has the high-order bit set, and the low-order 4 bits determine the operation code. The DE pair contain a pointer to the FCB. For CP/M, the function code has the values 0 through 27.

I haven't looked at Intel's ISIS operating system, or Processor Technology's HELIOS disk operating system, so I don't know if they are compatible. EMOS-8 uses the second 8 bytes in memory, which I believe is the Intel standard, as well. I think it's reasonable to give the first 16 bytes to the operating system, allowing the remaining restart vectors for user restart routines or interrupt vectors.

Conclusion

I have described a "paper design" for an operating system based on the Intel 8080 microcomputer. It features modularity, extensibility, and conviviality; memory allocation; multi-tasking ability; named entity facility; a generic file input/output system with logical unit names and a file control block; a video display module driver and editing facility; loading, assembly, and debugging utilities; and compatibility with the Extensible Microcomputer User's Language, CP/M, and PL/M.

The design is not proprietary; the concepts are free for anyone to use, as long as I am given credit and receive a copy of anything based on EMOS-8. I hope to implement the first pass of the system this summer, and place it in the public domain.

About the Author

Bob Wallace wears several microcomputer "hats" in the Seattle area. He is a founder, past newsletter editor, and currently secretary of the Northwest Computer Club. In addition, he is publications manager for The Retail Computer Store; he orders all their books and magazines. Finally, he is a graduate student in computer science at the University of Washington. In his spare time, he consults for New World Computer Services, Inc., does event logistics for ComNet, and tries to get his computer running. He is 27 and single.

(EMOS-8 is being designed under contract for New World Computer Services, Inc.)

A NEW APPROACH TO TIME-SHARING WITH MICROCOMPUTERS

Joseph G. McCrate
Cromemco, Inc.
2432 Charleston
Mountain View CA 94043

ABSTRACT

A time-sharing system gives a microcomputer increased flexibility and power. Several people can share the use of such a system, or one person can run simultaneous tasks on it. A microcomputer equipped with a time-sharing system can easily be re-configured to handle an additional task. The paper discusses the functioning of time-sharing systems and describes new hardware which facilitates the implementation of such systems on microcomputers.

USES OF TIME-SHARING

Time-sharing systems have long been implemented on large computer installations, and for good reason.

Input and output devices are usually very much slower than the rest of the computer. Consequently, a computer is able to handle a large number of these devices. Rather than wait around for a particular I/O device to respond, a well-trained computer will suspend working on a project with laggard I/O and find another task whose I/O is ready. This capacity of multi-tasking and time-sharing systems can greatly enhance the utilization of a computer.

Time-sharing, therefore, is a way of using computer time that would otherwise be wasted. But it does more than this; it also saves the time of the people who use the computer.

If you want to use a computer without time-sharing, you must get in line behind all the people who get there before you. Only after the computer has completely finished with all of their jobs, will it even start yours.

With time-sharing, however, the computer can work on a bit of one job and then another. When one person dawdles at the terminal, or for that matter, while one terminal is slowly constructing a character received from the computer, line by line, dot by dot, the computer can work on another project. In this way, a number of people can get simultaneous attention from the computer.

A microcomputer can be devoted to a single project at far less cost than can a larger computer. Furthermore, there are plenty of cases where the implementation of a time-sharing system on a microcomputer is cost-effective.

A school system can buy one microcomputer which can handle up to eight student terminals at minimal incremental cost per student. (Details of such a system are described by Dr. Alice Ahlgren elsewhere in these proceedings).

Even the owner of a small computer for home or business can benefit from time-sharing. In this case, the issue is perhaps not the efficient use of every split-second of the computer's time. Rather the issue is the increased flexibility and power that time-sharing can give to a computer.

For example, with time-sharing a microcomputer which is set up to monitor and control the temperature of a building, can also be used for running Basic programs, or a budget control program, or both simultaneously.

It doesn't even require multiple terminals to make good use of time-sharing. With only one terminal, a time-sharing system can run Basic and control several independent processes.

The capacity to run several programs at once is not the whole point of time-sharing, however.

A computer equipped with a time-sharing system can be re-configured to handle a different task load very easily.

Imagine how difficult it would be to convert your program for controlling a heating system into a program which did that plus ran Basic, if you couldn't use time-sharing.

Even to add the facility to monitor a home burglar alarm system would take a significant re-programming effort.

The advantage of time-sharing is that you only have to write a program for the new task; you do not have to re-write the old program to mesh the new one in. It is the job of the time-sharing system to fit the two together for you.

IMPLEMENTATIONS OF TIME-SHARING

A time-sharing system is a program and an arrangement of hardware which performs several functions.

- 1) The time-sharing program transfers control of the computer to a user program or task and, after an appropriate period of time, regains control in order to transfer it to another task. This is usually accomplished by setting an interval timer just before control is transferred. When the timer count-down is finished, an interrupt is generated which gives control back to the (time-sharing) system.

- 2) When the system regains control from a task program, the machine state at the time of the interrupt is saved. The machine state includes the contents of the computer registers including the stack pointer, the contents of the stack and all other memory locations which the task program was using.

When control is again given to the task, the machine state is restored so that the task can continue where it left off.

- 3) In addition, many time-sharing systems are equipped to handle input and output for the task programs.

Cromemco has introduced several new products with features that make it remarkably easy to implement time-sharing systems.

The bank-select facility that appears on all Cromemco memory cards is of prime importance. As many as eight different cards can occupy the same address space. This means that up to eight tasks can run simultaneously even with overlapping storage areas of RAM (stack and temporary storage areas). All the system need do in order to save a machine state is save the registers and switch banks.

The rest of the hardware required for time-sharing is provided on the Cromemco TUART™. This board includes two serial I/O ports, two parallel I/O ports, and ten interval timers. It also provides separate prioritized interrupt vectors for each of the input and output ports and timers.

The port addresses are selected by means of DIP switches. In addition, port address assignments may be reversed by an output from the computer to one of the parallel ports on board. This makes it possible to write a simple time-sharing system which, after initialization, takes only 38 bytes to control the simultaneous execution of virtually any two programs. Furthermore, a program does not have to be modified into a "time-sharing version". If it runs alone, it will run under the system.

A further feature of the TUART™ makes it easy to implement more sophisticated time-sharing systems. Up to eight TUART™ may be daisy-chained with distinct prioritized vectors for all possible interrupts. In particular, each of the serial ports necessary to drive the eight terminals has its own interrupt vector. Since the Z80 instruction set allows input and output instructions with variable port numbers contained in the C-register, it is quite easy to have the system handle interrupt-driven I/O for the eight users.

ABOUT THE AUTHOR

Joe McCrate writes software for Cromemco. He has studied engineering and philosophy at Stanford University and holds an M.S.E.E. degree from the University of Southern California. Mr. McCrate was previously a teacher of philosophy at the University of Wyoming and has been a member of the technical staff at Hughes Aircraft Company.

MICROCOMPUTERS AND MULTI-TASKING: A NEW DIMENSION IN PERSONAL COMPUTING

George Pilipovich
MVT Microcomputer Systems, Inc.
Box 62
Agoura CA 91301

ABSTRACT

A highly sophisticated, state-of-the art multi-tasking software system is described. The system is based on the Intel 8080 microprocessor and incorporates features generally found only in expensive hardware systems such as mainframes and minis. This paper describes the components and theory of operation of this recently developed operating system. A general description is given and covers the supervisor functions, storage methods, I/O blocks, disk interfacing and resource management. Also discussed are the problems encountered in developing multi-tasking software for a microcomputer and how these problems are solved.

INTRODUCTION

Multi-tasking is basically the "apparent" concurrent execution of two or more programs or tasks. Thus, a multi-tasking system must be capable of supporting more than one user and, in some cases, more than one program per user. Multi-tasking is possible because most of the real time devoted to a program (in any environment) is wasted waiting for the completion of I/O such as a keyboard or a printer.

Heretofore programming at central processing centers has been reserved for those with financial resources well beyond those of most personal computer users. Even then these facilities, through TSO, are not truly multi-tasking but rather a "take your turn and wait for machine time" operation. With the availability of a reliable and authentic multi-tasking system for the 8080 processor, truly a new dimension is revealed both to the financial community and to personal computing. To the hobbyist, whose resources are severely limited, a central computer, a microcomputer, can be jointly owned. With his own remote terminal he can interact with the computer in concert with 10-25 users. Each of these users will feel that the computer is dedicated to his own terminal. To the hobbyist sharing a microcomputer and using an efficient multi-tasking system, there would be virtually no competition for the processor. At the same time, the system would provide sophisticated services such as multi-sessioning and system tasks.

Unfortunately, the availability of reliable multi-tasking systems for microcomputers has been virtually non-existent at any price. This condition is owing to the considerable expense in developing the system software required for a reliable multi-tasking system.

The following paper describes a tested, state-of-the-art multi-tasking system for the Intel 8080 processor. The software was developed by MVT Microcomputer Systems, Inc. and is called Bossystem/I.^c The discussion evolves in two parts. One part describes the Bossystem/I^c system. The second part discusses the general theory of its operation.

DISCUSSION

Bossystem/I^c

Currently, the MVT Bossystem/I^c is based on the 8080 and a single or dual floppy disc. An obvious feature of the system is multiple terminals, both in-house and remote. The system is limited mainly by the small main memory addressable by the 8080, but it still can handle 20-30 terminals. The system logs users on and off in a system log, and maintains an accounting for each user and account of CPU time, real time and I/O requests. Among the system tasks is a writer for each printer, which is operated through a FIFO print queue. Password protection is provided for each user and account, and also as an option for any file or program --either system or user defined.

A powerful feature of the system is multi-sessioning. A user can operate as many concurrent tasks as memory will allow (although his keyboard can communicate with only one at

a time). This is very convenient when, for example, a large program is being assembled or compiled, since one can execute another program or edit a program while the assembly or compilation takes place. Another feature of multi-sessioning is the ability to 'escape' from a task via a special key on the keyboard. This is very convenient when a new program runs amuck or in 'panic' situations.

A special program must exist to handle multi-sessioning, and it is called the super-selector. Through it a user can initiate, terminate, suspend, or resume any of his tasks, which he assigns arbitrary names to. He can also monitor the system or his tasks and access system functions.

We are currently designing a re-entrant Basic interpreter and PL/I compiler. A re-entrant, or pure, procedure is one which does not alter itself such that more than one task may execute it concurrently.

At present the system utilizes a dual floppy disc system. It utilizes a sector allocation scheme such that blocks can be dynamically inserted or deleted anywhere in a file. This scheme was chosen not only because it is a more powerful method than the usual track-allocation or fixed-length schemes, but because it effectively increases disc storage by 50-100% or more depending on the type of data used. The main problem with this method is forward write speed and ISAM compatibility because a sector must be read before written and indexing must be accomplished with individual file maps. These are minor problems, however, and pose no undue system restraints.

A complete medium sized disc system of 5-10 megabytes may soon be available at a moderate price. Such a system would utilize a track allocation scheme in our configuration.

The basic hardware requirements include an 8080 system with an interrupt-event timer board and at least 24K of memory.

General Theory of Operation

Task control is maintained via a bi-directionally linked linear TCB list, which is dynamically created by a LIFO mechanism. The initial state consists of a dummy system TCB, which is maintained at the tail, which is also used to monitor CSECT and CSCB allocation. The TCB list doubled as the ready list, a separate ready list was avoided because the overhead in maintaining it was far greater than that of scanning the TCB list--memory limits the TCB list to a very manageable length. Besides pointers to various control lists, which will be discussed later, the TCB contains several bytes of flags. Conditions the flags indicate include 'ready', 'unconditional non-dispatchable', 'delete task', 'unconditional retain', 'uninterruptible', 'conditional wait', and wait and completion event bits for various I/O as well as user event bits. The TCB also contains CPU time and I/O counters. Most I/O is serviced only in the supervisory state--event bits are set by the I/O routine.

Dispatching is done after the TCB's are serviced and the next 'ready' task found, unless no task is ready, in which case I/O events are again serviced. Dispatching is done in four 20 or four 10 millisecond intervals, the length of the interval being user specified. Return to the supervisor is made upon the completion of the final interval, or whenever a wait is requested. A longer dispatching interval was not used because we rarely encountered a task which used up even an 80 millisecond slot before requesting a wait.

Storage management is a problem with microprocessors, basically because of the small amount of addressable memory available and insufficient protection hardware. Any form of partitioning, regioning or paging was ruled out because of size limitations. A single FQE chain is used, and with proper fragmentation collection it works very well.

For purposes of task-deletion, an RB queue is allocated and chained in both directions to the TCB for each storage element allocated by a task.

Among the major I/O control blocks are the keyboard control block, CRT control block, and task file I/O control block, also known as the Task Volume Table of Contents. These are respectively abbreviated KIOCB, CIOCB, and TVTOC. The KIOCB and CIOCB are used in terminal communication and are relatively unimportant to the discussion, with the exception of the following very important note: Only one KIOCB may exist per terminal and is attached to a superselector task, while one or more CIOCB's may exist per task. A TVTOC exists for each file open to a task, and they are chained to the TCB. The TVTOC contains information about file position-

ing and buffer location. Buffering is the problem program's responsibility.

Disk I/O is done through a single FIFO queue, for which, of course, no RB's are generated. Each member contains a TCB pointer, a sector address and buffer pointer, as well as other information--such as TVTOC status, including, for most operations, a pointer to a pointer in the TVTOC. Clearly the TCB must be retained while it has a member in the queue. Optionally, a task may operate asynchronous to its disk I/O.

Aside from a normal I/O routine for reading and writing a file, and expanding or contracting a file, there are also disk routines for allocating or deleting a file. Note that sectors may be added or deleted from any point in the file, not just the beginning or end. A disk compress is never necessary, since the files are dynamic and not necessarily sequential.

Resource management presents the same basic problems as in any system. The printer is handled in a standard manner by routing all output through the disc and a system writer. Other serially reusable resources are not so simple: these include storage management routines and routines which alter system queues or the main disc map. For short routines, such as those for storage management, preventing interrupts within the routine may be sufficient. If such routines are nested no more than two deep then the outer one may disable interrupts at a higher level, such as through the interrupt board, than the inner one. The third level program is a bigger problem, however. A FIFO resource queue is a satisfactory solution, provided the supervisor works with it correctly. Now tasks can be interrupted, so that this is a satisfactory lockout for disc routines. Notice that if no competition exists for a routine the lock-out routine will not allocate a queue, and this is not difficult to program, so that the overhead is incurred only when necessary. A final problem occurs when a user attempts to suspend or delete a task executing a serially reusable resource, or residing in its queue. Clearly, disaster would occur unless this is prevented: thus the supervisor and lock-out programs cooperate for this purpose. Note that only one pair of lock programs need exist, regardless of the number of serially reusable routines.

Control of object programs becomes somewhat complex when multiple tasks access a non-resident re-entrant routine. This is accomplished via a fairly standard CSECT control block(CSCB), which contains a usage counter and various flags. Again we need a LIFO queue for each TCB to ensure that all use counts will be taken care of at task deletion. This, of course, requires the passing of a dummy EPA.

We have not discussed the superselector and task-deletion routines, because their objectives and design are fairly obvious, although the actual coding can get somewhat involved. Nor have we mentioned the assembler, editor and loader: this is because they are standard for any system.

The chief cause of system failure is the lack of memory protection. If a program contains an error and writes on system storage, it can eventually lead to a system crash. A frequent result of this is the rupture of the FQE chain. Also, there is no way to restrict interrupt control to the supervisor. These problems stem from the use of a micro-processor. Clearly a high level language is required and, as has been stated earlier, the evolving re-entrant Basic interpreter should offer sufficient system protection as well as stimulating greater program development.

The re-entrant Basic does limit programming capability and will no doubt lead to a PL/I compiler. The PL/I compiler will be similar to the Cornell PL/C. Our version will be small, simple and with a modest language reduction. Enough language will be retained to derive a mixed strategy--precedence grammar devoid of ε-rules.

GLOSSARY

- CIOCB--CRT screen map I/O control block
- CSCB--CSECT control block
- CSECT--program defined to the system
- EPA--entry point address
- FIFO--first in first out
- KIOCB--keyboard I/O control block
- LIFO--last in first out
- RB--request block, monitors task storage requests
- TCB--task control block
- TVTOC--task volume table of contents

BIOGRAPHICAL SKETCH

George Pilipovich is currently responsible for software development for MVT Microcomputer Systems, Inc. Prior to his current position, Mr. Pilipovich founded GM Custom System Software Development. During this time as an independent contractor microbased system and application software were developed. Mr. Pilipovich has a strong interest in Simulated Intelligence and has initiated research in this direction while studying Computer Science at UCLA.



INTERFACING A SELECTRIC TO YOUR COMPUTER

Carl Townsend
Center for the Study of the Future
4110 N.E. Alameda
Portland OR 97212

INTRODUCTION

The IBM Selectric mechanism has capability for providing a highly reliable I/O terminal for micro-computer usage that permits a wide variety of applications as:

- * word processing
- * mailing labels
- * stencil preparation
- * newsletter composing
- * correspondence
- * proposals
- * information retrieval
- * networking

The Selectric offers the advantages of:

1. Hard copy output of both upper and lower case letters.
2. Interchangeable typing elements.
3. Hundred character (or more) line.
4. High reliability with a minimum of maintenance.
5. Low depreciation.
6. Higher typing speed (15.4 characters per second) than a teletype.

SELECTING AND PURCHASING A SELECTRIC MECHANISM

A user has two alternatives available in selecting and purchasing a Selectric mechanism:

1. Purchase an office typewriter and convert it mechanically with mechanical assemblies as the Tycom. Add your own electronics or purchase an electronic interface kit.
2. Purchase a used terminal with the solenoids already installed. Add the interfacing electronics.

The former has an advantage from a maintenance viewpoint. IBM will support mechanical maintenance on office typewriters with the Tycom adaptor, but if you purchase a used terminal you will need to find your own sources to maintain the mechanism. Since a computer driving the mechanism at the fastest rate represents an unusual mechanical demand on a terminal that has been sitting idle or used at the slower office typing speeds, you can expect some maintenance and adjustments involved at least initially. We do not advise trying to repair a Selectric yourself unless you have a considerable amount of mechanical skill.

Used office typewriters are difficult to find and cost only slightly less than a new typewriter. Delivery is slow on new typewriters and also slow on the mechanical interface assemblies as the Tycom.

Used terminals can be purchased from a variety of sources. Watch microcomputer magazines and shop for the best buy. Also check for foreign sources. The mechanism is of high quality and you should expect very little difference if it is Itel, Dura, a 735 or another. All have identical printing cycles. You should, however, be concerned about a few factors in purchasing:

1. Is it a Level 1 or Level 2 mechanism? Early mechanisms of the Level 1 type are not quite as good as the later Level 2. Use the IBM ADJUSTMENTS PART MANUAL to

learn how to distinguish between the two levels.

2. What voltage are the solenoids? You should shop for a 24 volt solenoid machine as it is easier to drive than machines requiring 48 volts on the solenoids.
3. Is it rebuilt? A rebuilt machine will be cleaned and adjusted with parts replaced as necessary. If not rebuilt, you may need to do some mechanical maintenance work.

Once the terminal is purchased, you may integrate it into your system in one of three ways:

1. Use the keyboard for input and drive the printing mechanism as an output.
2. Replace the keyboard with an electronic keyboard, driving the printing mechanism as an output.
3. Use the Selectric as a printer only using a separate electronic keyboard and video display (as the ADM3) for input and editing.

For several months we have integrated our terminal as the first option. Now we are altering our system to the third option for a variety of reasons:

1. The Selectric has no "control" key which makes in-line editing difficult.
2. The input timing is complex, requiring interlocks and special switching.
3. The electronic keyboard is faster, more reliable, and quieter.
4. The Selectric keyboard prints direct which means the software must often be altered to defeat the echo feature. An electronic keyboard with the echo is easier to interface to most software.

Before attempting any electronic design and development, bring the Selectric mechanism to operational order mechanically as well as doing a little preventive maintenance:

1. Use a vacuum cleaner to remove all loose particles and dust.
2. Use a lacquer thinner to remove accumulated ink from the platen and all pressure rollers.
3. Use Zepreser (Zep, Atlanta, GA) to oil all metal-to-metal moving parts. This is a spray oil and highly penetrative. Oil penetrates better than grease, but will require more frequent application. Be especially careful to keep oil from getting on the outside of the shift cam at the right outside of the mechanism. A nylon brake rides against this wheel, and timing requires the cam to be grease and oil free.
4. Clean the typing element with soap and water, being careful not to get any metal parts wet. Do not use type cleaners as they will dissolve the plastic.

Purchase from IBM any materials you can locate that might be relevant, especially those of a tutorial nature on the mechanism. The Selectric mechanism is periodically updated. All mechanisms are not quite the same.

Some general precautions should be observed in

working with the mechanism:

1. Be careful with cables, tapes, and cards. Special tools are needed to repair these.
2. Keep fingers away from rapidly moving parts. Explore with the motor OFF.
3. Keep hands and fingers away from the "home" position for the returning carrier.
4. Check each spring under each key before reclosing the keyboard. These are easily disengaged.
5. Avoid modifying high precision areas. Make changes in the low precision areas as under the keyboard.
6. Raise both margin stops and the carrier position pointer before removing the cabinet, otherwise these may be damaged.
7. Do not lift the mechanism by the more fragile parts.

With some terminals a few of the functions (as tab and shift) are electronic only, and mechanical linkages have been removed. In this case the function will not work without the appropriate electronics, but the mechanics can still be checked by manually working the solenoids. All functions should be checked and problem areas identified. Some functions are simple and require minor adjustments. Others are complex involving interrelated adjustments and special tools. Use a small intense light to trace mechanical problems and try to understand what adjustment does what before turning screws. Respect the design of the machine--it is well engineered.

SELECTRIC CODING

Most terminals (as teletypes and video displays) use a six to eight bit ASCII code to represent characters. The codes for numbers and letters are sequential, simplifying programs that do numerical conversions, as octal to BCD converters. For octal conversion of numbers from ASCII, it is only necessary to mask the right-most three bits.

The Selectric mechanism does not generate ASCII codes. Indeed, the generated codes are not even sequential. To understand why such unusual codes are used, it is necessary to examine how the Selectric element mechanically operates. The Selectric printing mechanism uses a moveable nickel-plated plastic element to print characters without moving the carriage. The element can be tilted or rotated to a given position using a seven bit code, and then strikes the paper. This means it is not necessary to engage each character with a solenoid, but only to use one solenoid for each element motion control--Rotate 1, Rotate 2, Rotate 2A, Rotate 5, Tilt 1, Tilt 2, and Shift. Each control is binary, as it is either on or off. Special solenoids can also be added for tab, index, space, backspace, and carriage return. This minimizes the number of solenoids required, but necessitates electronic coding to set up the desired character. The keyboard utilizes the same type of coding.

The code conversion can be done either by software at the computer or electronically within the Selectric interface. We advise electronic conversion (see Figure 1).

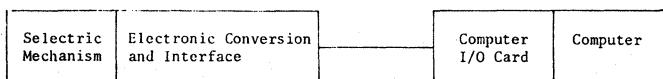


FIGURE 1 INTERFACE BLOCK DIAGRAM

The user will need to identify the various solenoids and switches if he purchases a used terminal. This can be done with the aid of the coding diagram and an ohmmeter (see Chart I and Figure 2). Use the ohmmeter to identify the control solenoids from those that drive the selector magnets. The control solenoids require more current and will measure a lower resistance than the six selector magnets and the cycle clutch solenoid. (Some machines have a seventh selector magnet as a parity check).

The solenoid voltage can be applied to the solenoids to identify the various functions, using the coding diagram. A voltage applied only to the "R1" magnet, for example, will cause the Selectric to continuously type a "y". There is no code for R2 only, and either R2 or R2A will cause the terminal to print a "q". In some cases the magnets will need to be energized together to identify function. If the print cycle clutch only is energized, the Selectric will type a dash. Apply a voltage to each control solenoid and identify the control function you will need as backspace, tab, space, carriage return, and index. Also identify the shift solenoid. Label the wires for each solenoid with color-coded wire or in any other way you can easily identify them later.

We suggest using the Selectric as a printer only. The print cycle is simple and an experimenter can have a good bit of enjoyment and satisfaction designing his own circuit to meet his specific needs. All of the clock speeds are slow and most of the circuit should fit in a multiple test board as the Heath ET-3000. Use plenty of decoupling capacitors, a ground bus strap, and a ground plane under the test boards.

Chart I

PRINTING COMMANDS AND CODING

CHARACTERS	ASCII CODE	SELECTRIC CODE	R2A	R5	T1	T2
A a	301					
B b	302					
C c	303					
D d	304	R1	R2A	R5	T2	
E e	305	R1	R2A		T2	
F f	306	R2	R2A	R5		
G g	307	R1	R2	R2A	R5	
H h	310	R1				
I i	311		R2A		T1	
J j	312	R1	R2	R2A		
K k	313		R2A			
L l	314	R1		R5	T2	
M m	315	R1	R2	R2A	R5	T1
N n	316		R2	R2A		
O o	317	R1		R5	T1	
P p	320	R1		R2A		
Q q	321			R2A		
R r	322	R1		R2A	R5	T1
S s	323	R1			T1	
T t	324	R1	R2	R2A		
U u	325		R2	R2A	R5	T2
V v	326		R2	R2A	R5	T1
W w	327				T1	
X x	330	R1	R2	R2A	R5	T2
Y y	331	R1				
Z z	332	R1	R2	R2A	T1	T2

PRINTING COMMANDS AND CODING

CHARACTERS	ASCII CODE	SELECTRIC CODE					
1	261	R1	R2	R2A	R5	T1	T2
+							
2	262	R2	R2A		T1	T2	
@	300						
3	263	R2	R2A	R5	T1	T2	
#	243						
4	264	R1		R5	T1	T2	
\$	244						
5	265	R1	R2A		T1	T2	
%	245						
6	266		R2A		T1	T2	
£							
7	267	R1	R2A	R5	T1	T2	
&	246						
8	270		R2A	R5	T1	T2	
*	252						
9 (EOA)	271				T1	T2	
(250						
0	260	R1			T1	T2	
)	251						
=	275	R2	R2A				
+	253						
;	273	R1	R2A	R5			
:	272						
/	257	R1		R5			
?	277						
.	256		R2	R2A	T1		
.	256						
'	247	R1	R2A		T1		
"	242						
-	255			(none)			
(none)							
•	(none)	R1	R2	R2A		T1	
!	241						

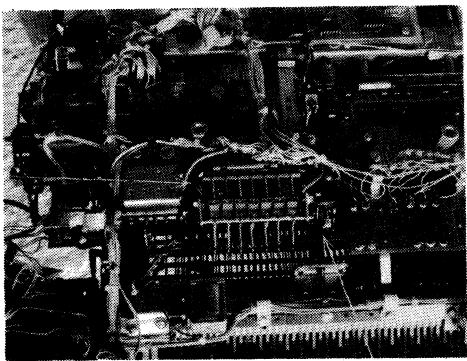


FIGURE 2 SELECTOR MAGNETIC ASSEMBLY

We suggest using a serial I/O with the RS-232-C interface specification with a baud rate of 138 and seven bit coding. If you wish to check parity, you can use eight bit coding at 150 baud. This gives a 15.4 character per second typing rate.

A 2502 UART converts to parallel ASCII, and the ASCII code addresses a 1702 EPROM (Figure 3). The corresponding Selectric code is stored at the EPROM address. Six bits of each EPROM output drives two sets of drivers. One set controls the selector magnets, the other set controls the control solenoids. The eighth EPROM output determines which set of drivers will be active. The seventh EPROM output drives the shift solenoid.

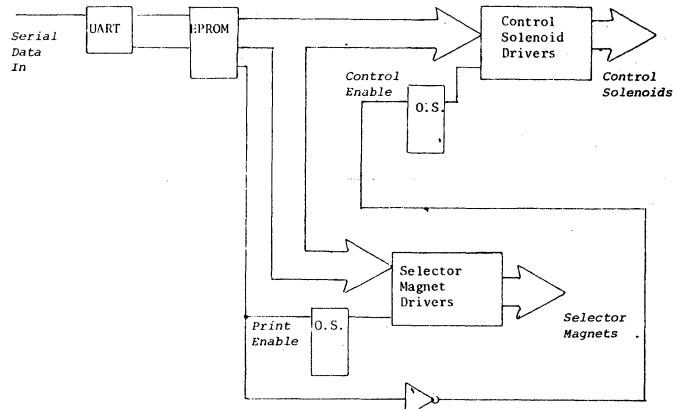


FIGURE 3 SIMPLIFIED CONVERSION AND INTERFACE BLOCK DIAGRAM

In a lower case print cycle, the corresponding selector magnets and the print cycle clutch solenoid are pulsed with a 30 ms. pulse. The cycle is complete before the UART is ready with the next address. If the eighth EPROM output defines the cycle as a control (carriage return, index, etc.), only one of the six EPROM lines is active. This is clocked through to the corresponding control solenoid.

Several of the functions as carriage return, tab, and shift require longer than one character cycle. These special functions can be handled in two ways:

1. Recognize them at the software level and introduce software delays.
2. Recognize them at the hardware level and introduce hardware delays.

The second is the preferred approach. In our system we recognize the characters and use the Request-to-Send line in the RS-232-C to control the ready status on the computer I/O board. During this development be sure to remove the element from the typewriter. The mechanism can be damaged if a shift cycle and print cycle are started together. (A shift requires three character times--shift up, print, shift down). Removing the element protects the mechanism.

CONCLUSION

We hope this introduction to the Selectric has excited you somewhat to begin some experimentation with a used terminal. Our experience with the Selectric indicates it is probably the best of slow speed hard copy devices for a home or small business computer system.

NOTE: This text is typed on a Selectric driven by a MITS Altair 8800a. For more information on our interface kits, schematics, and manuals, send a SASE to the Center for the Study of the Future, 4110 N.E. Alameda, Portland, OR, 97212.

A FLOPPY DISK CONTROLLER FOR UNDER \$50

Kenneth B. Welles II, Ph.D.
2623 Fenwick Road
University Heights OH 44118

A floppy disk drive interfaced to a microcomputer with the proper hardware and software can be the most important addition made to a personal computing system. Frequently it is also the most expensive. Although prices on floppy disk drives are falling and more compact, less expensive versions of the drive are being introduced, the computer to disk drive controller costs and operating system availability remain a problem.

A little ingenuity can substitute for cost and complexity, however. Using only \$25 worth of integrated circuits, an interface can be built which allows complete control of up to eight hard-sectored floppy disk or mini floppy disk drives. The interface, which is described in reference 1, consists of only 17 integrated circuits and plugs directly into the S-100 bus. A brief description of this circuit will include a demonstration of how the addition of 3 integrated circuits allows the control of the Shugart Mini-Floppy T.

In order for such a simple low cost interface to actually operate a disk drive successfully, there are trade-offs that must be made. The compromise does not effect the amount of data that can be stored onto a disk nor the speed with which it can be accessed. Rather, the trade-off requires increasing both the amount of software which resides in memory and the amount of CPU time consumed during a data transfer. Both time and memory were considered scarce and expensive in the days of the old, large computer. Now, however, the amount of time spent by the CPU performing such data transfers is insignificant when compared to the time required by a paper tape or magnetic cassette data storage and retrieval system.

The increased amount of software required to operate this interface incurs some extra expense, but this too can be shown to be nominal. Between 300 and 600 bytes of software are required to make this interface emulate an interface containing one or more of the new single chip floppy disk controllers. At the current prices of S-100 memory, this amounts to \$7.50 to \$15.00 in memory costs.

There are also some "super-intelligent" controllers on the market which contain their own file management system in their own microprocessor at an additional cost. To perform all of these functions on the simple interface presented here requires a 4K byte block of software or about \$100 in extra cost.

A floppy disk operating system was written which operates with this interface on an 8080 system, and resides in a 4K byte block of memory, including all data buffers and stack areas. This FDOS allows a single regular disk drive to store and retrieve or run up to 240 different programs or data files by name. Over 300,000 bytes of storage are available on the diskette, and the filing system is a linked block structure which allows the most flexible use of this storage. The operating system allows the programmer to use relocateable data buffers to transfer data to and from any number of different files simultaneously.

Hopefully, the system described in this presentation serves to demonstrate that a little creativity in an integrated hardware/software design frequently allows a low cost solution to a previously high cost problem.

About the Author

Kenneth B. Welles II received his PhD from Cornell University in Applied Physics in the area of computer image processing, with a minor in Computer Science. He has designed and constructed microprocessor interfaces for TV camera input, vector graphics output, IBM Selectric output, and a floppy disk drive controller. He has written a complete floppy disk operating system for the 8080 in conjunction with this controller. He is currently employed by the Lighting Research Division of General Electric as a research physicist where he designs and develops microprocessor based process control equipment.

- 1) BUILD THIS ECONOMY FLOPPY DISK INTERFACE, K Welles, Byte February, 77, Vol. 2 No. 2 p. 34.

A FAULT ISOLATION TROUBLESHOOTING SYSTEM FOR THE MULTI VENDOR ENVIRONMENT

Robert A. Tuttle, Jr.
Medical Methods Research
3700 Broadway
Oakland CA 94611

Having a computer in the home is becoming more prevalent, whereas the use of computers in industry has been widespread for some time. The home systems are usually dedicated to fun and games, the industrial systems to machine and plant operations. If the home system fails, the only loss is one's pride. If a manufacturer or laboratory, as is our case, has a computer failure, time and money are lost. In our case, the problem is compounded by the fact that a typical system is made up of at least three subgroups of machines--test instrument, microcomputer, and various peripherals such as a card punch and card reader, all handled and supported by different vendors. If the system fails, whose equipment is at fault? Who do you call? The typical answer is to call everyone.

It became apparent that the "call every vendor" approach usually compounds the problem instead of relieving it, and that a vehicle for determining the fault was needed. One approach to solving this problem is the development of a troubleshooting system that can isolate the problem to a single piece of equipment. A Fault Isolation Troubleshooting System called "FITS" was developed to solve this problem.

The basic hardware for FITS is standard S-100 issue. The software was patterned after the system to be tested and a concept of inputting, storing and comparing data. Additional advantages from FITS is system expandability (or the ability to grow with system demands), and versatility.

The hardware in FITS was patterned after a developmental system in that it uses the standard S-100 bus, microprocessor board, random access memory, read only memory, floppy disk, video display terminal, serial and parallel input/output. In addition, an analog-to-digital converter is used for testing power supply voltages found on the S-100 bus under test, and to set logic threshold limits. The main link between the system and FITS is a bus buffer board, which is inserted into the system (S-100) microcomputer bus to be tested. This board contains differential line receivers, and is totally passive in regards to the bus under test, to the extent that it gets its device power from FITS. Differential line receivers are used here to test signal levels, looking for low voltage levels leading to noise problems.

There are three basic signal groups which are handled separately from a hardware standpoint. These are: system clocks (81, 92 and realtime), control signals (MWRITE, PWR, PION, etc.), and data signals (digital input/output, address and interrupts). The clock and control signals must be tested dynamically for detection of error, such as skewing or jitter. Data signals are stored in temporary memory for testing and comparison as specified in the FITS program.

FITS software combines two basic ideas. The first is to duplicate the program or operation of the microcomputer under test. The second is the inputting, storing and analysis by comparison, of significant signals developed by the test system. When a system event occurs (for example, a data transfer from a Coulter Counter Model S blood analyzing machine to the microprocessor of an Abnormal Limit Display), FITS will input and store that data. The Abnormal Limit Display then compares the Coulter data with pre-set normal limits, and if exceeded, notifies the operating technologist via a simple light emitting diode (LED) display. FITS will perform the same comparisons with the same limits, and then compare its own response with that of the system under test. If the test answers differ, FITS will notify the maintenance technician of the problem via a video display terminal.

The areas of the standard bus which fall under the concept of input, store and compare, are the control and data signals. Additional diagnostic programs are set up to test bus voltages and clock signals. A maintenance technician with limited software background can use FITS to localize specific problems by writing simple loop routines, and analyzing FITS front panel display.

As more complicated laboratory instrumentation is developed, FITS will become an even more vital piece of test equipment, because it can be reprogrammed to take into account both hardware and software changes.

A hypothetical example would be if a new piece of laboratory chemistry equipment was purchased. First, a standard bus interface would need to be developed to the new piece of equipment. Once this had been accomplished, FITS would need to be programmed, and the maintenance technician would be able to

troubleshoot. In many instances the interface from instrument to bus is a simple serial input/output.

FITS also provides a method of keeping the maintenance technicians up-to-date. The technician continues to use the same FITS system with which he has become familiar, but the operation and output of FITS will have been reprogrammed to reflect the changes in laboratory instrumentation. Changing the program to test one system vs. another in FITS can be as simple as calling up a disc file location. In the worse case, the technician would have to change a read only memory, taking only two-to-three minutes.

Besides covering system changes, FITS versatility becomes apparent in that it has the same bus structure as the systems under test. This means that FITS can be used as a diagnostic tool for the repair of bus compatible PC boards. Swapping of like PC boards can also assist the technician in the isolation of problems.

FITS can basically be classified as test equipment similar to an oscilloscope or voltmeter, the main difference being that it is designed to operate in an environment where microcomputers using a standard bus have been implemented. Basic hardware and software features which make up the FITS system have been discussed without regard to the negative points. Some disadvantages are that FITS cannot tell the technician the exact component or PC board that failed. It is hoped that in the future, FITS can be expanded to include component and PC board failure isolation, non-bus troubleshooting and quality control.

Troubleshooting multivendor systems with microcomputer, or even multicomputer controllers is no easy task, and one which will require much more effort before it reaches a satisfactory conclusion.

BACKGROUND

Robert A. Tuttle, Jr. received his A.A. (R&D Electronics) from Diablo Valley College and has been a member of the school's faculty and Electronics Advisory Board since his graduation. His primary background is in computer data acquisition and control systems. Currently he is Data Processing Engineering Supervisor for Kaiser-Permanente, Department of Medical Methods Research.

ADDRESS

Robert A. Tuttle, Jr.
Department of Medical Methods Research
The Permanente Medical Group
3700 Broadway
Oakland, California 94611

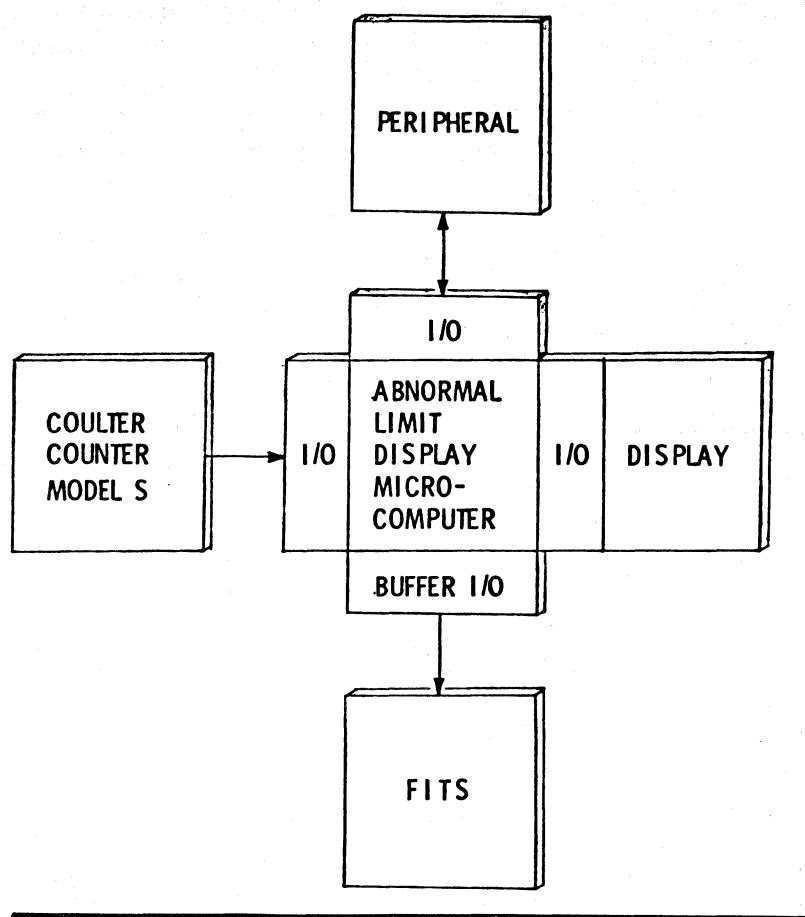
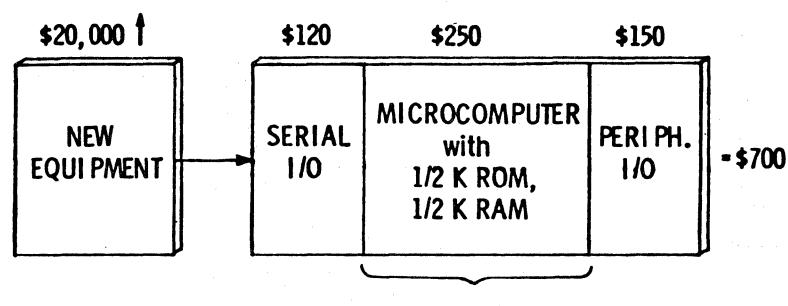
FAULT ISOLATION TROUBLESHOOTING SYSTEM (FITS)

1. HARDWARE

2. SOFTWARE

3. EXPANDABILITY

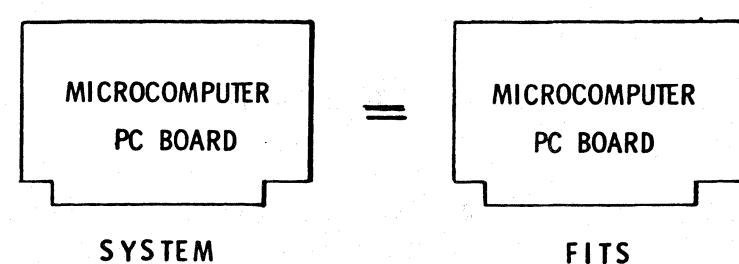
4. VERSATILITY

**EXPANDABILITY**

FITS

SOFTWARE

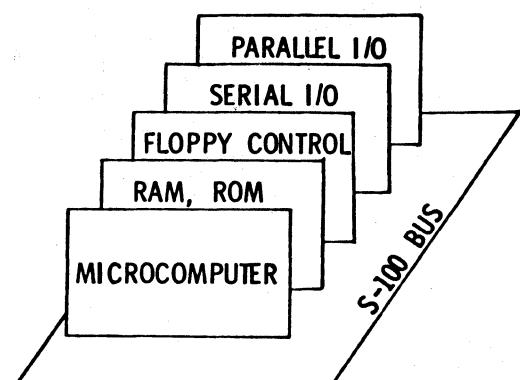
1. PROGRAM IN SYSTEM UNDER TEST



SYSTEM

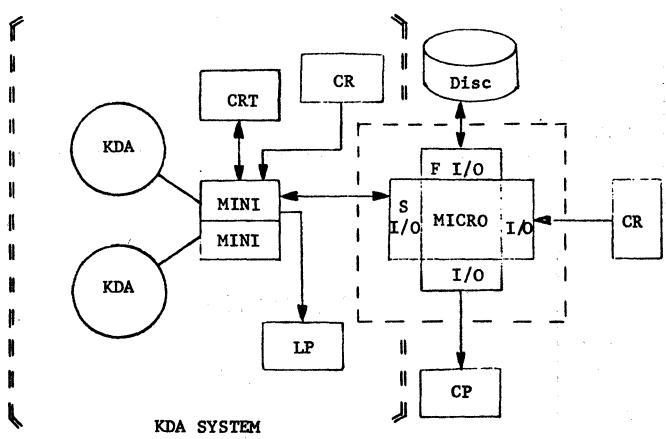
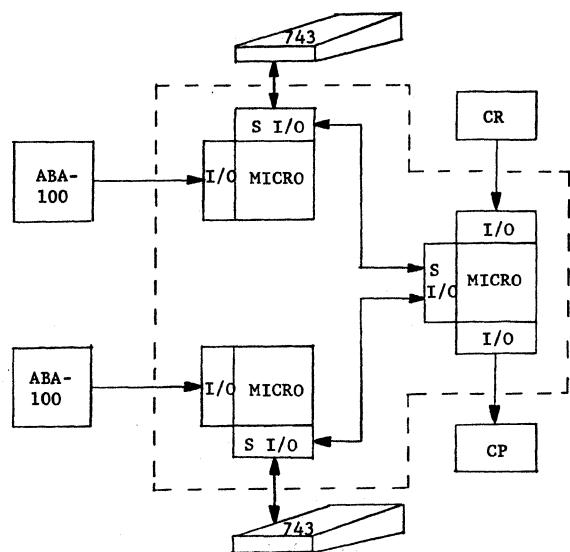
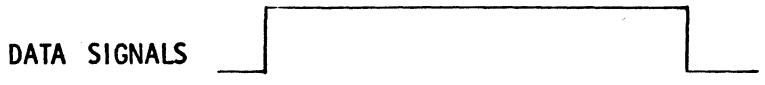
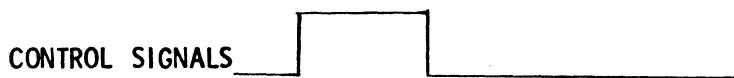
FITS

2. INPUTTING, STORING, AND COMPARING



> \$4,000

SOFTWARE**EXPANDABLE****VERSATILE**



SOLENOIDS PROVIDE SOFTWARE CONTROL OF A HOME CASSETTE RECORDER

William J. Schenker, M.D.
2086 Essenary Ave.
Walnut Creek CA 94596

Described (and demonstrated during the informal presentation) is an add-on to a typical low-cost audio cassette recorder, the J.C. Penny Model #6536. It will provide full software control of rewind, fast forward, play, record, and stop modes.

The device is a prototype. It was developed in the course of evaluating backup systems for data logging in a medical laboratory. It appears attractive to the typical personal computer system because of its low cost. Almost all the parts are surplus at a total cost of about \$15.

The purpose of this presentation is two-fold. One, to describe and demonstrate the system for those who would duplicate it in their home workshops. Two, to encourage some small manufacturer to fabricate a run in terms of 100s to sell in kit form for maybe \$50.

What follows are the design criteria underlying the device and then their prototype implementation.

Design Criteria

1. Precise rewind and FF tape positioning ability.
2. Low price and easy availability of parts (particularly the solenoids).
3. Minimization of internal and external mods to the recorder itself.
4. Selection of the J.C. Penny unit, because it is a de facto personal systems standard and the schematic is available (Sams Photofact).
5. Placement of solenoids to avoid "bulk erase" phenomena due to stray magnetic fields.
6. Reliance on software to provide tape movement precision.
7. Fabrication with non-ferrous metals, again to obviate magnetic field effects.
8. Mechanical linkage design to be as simple as possible.
9. Ability to sense BOT and EOT.
10. Ability to do hi-speed search.

The Implementation

The cassette deck was placed on an aluminum chassis, around the periphery of which were placed a series of pull-type solenoids. The solenoid coils are well below the surface of the deck. The linkage from each solenoid bobbin to cassette button is a one-piece lever of 1/16" aluminum sheet stock. Each solenoid or solenoid pair can pull 2 or 4 lbs respectively, with a 1/4" stroke. They are 24 volt DC intermittent duty solenoids. An integral power supply provides relay and solenoid power. The latter are pulsed for 25MS to actuate the latchable cassette buttons. (The Fast Forward required a mod internal to the recorder to make it latchable.)

5 channels of control are provided, using a George Morrow cassette interface board. (To the 3 Gordo TTL reed relays already on board were added 2 more.) The computer is an IMSAI. The flow of control for each channel is as follows: latched output port bit to a Gordo to a 24-volt relay to a solenoid. The basic software sequence to pulse the solenoid is: MVI A, xx; OUT yy; CALL 25MS DELAY; MVI A, 00; OUT yy

Does The Implementation Match The Criteria?

Generally, quite well. There have been no "bulk erase" problems. Rewind precision is far more than necessary, permitting positioning to the same syllable of an audio message after a rewind of up to 25 seconds, with high repeatability. Cheap solenoids are readily available both surplus and new. High-speed search can be partially-emulated by rewinding

to a point before the block of data, doing a read in "play" mode until the software "locks on" to the ID header of the record block, at which point you're synchronized. With up-to-date cassette interfaces you can then load a block of 256 bytes in a matter of seconds, at which point you do your search on the tape image in RAM memory.

Future Improvements?

BOT and EOT sensing can be done simply, using an op-amp to sense the voltage drop across one of the RF chokes in series with the drive motor. When the motor stalls the current jumps from 110 to 220 MA. The op-amp output can be amplified and then can flip open a latchable motor power relay.

In summary what has been described (and will be demonstrated at the Faire) is an in-expensive upgrade of a home cassette recorder to partially emulate its bigger brother, the digital cassette or 3M systems. Hopefully some manufacturer will pick up the ball and offer a kit--the design is public domain.

William J. Schenker, M.D., is a practising physician and a medical data systems scientist working with microcomputer technology. He also has at home an IMSAI 8080 with CRT, electric typewriter, paper tape, and cassette peripherals.

A MICROPROCESSOR INDEPENDENT BUS

Caesar Castro & Allen Heaberlin
295 Surrey Place
Bonita CA 92002

ABSTRACT: This paper discusses a bus which was developed for three home brew systems. The bus is compatible with both the 6800 and 8080 microprocessors and uses a common 44 pin connector. Two of the home systems use the 6800 and one system uses the 8080. In addition cost of the systems are also included.

INTRODUCTION

Early in the summer of 1976 our group (Caesar Castro, Allen Heaberlin and Kirk Mitchell) started to build home computers. The group was made up of electronic design engineers and we all realized that if we used the same microprocessor, we could share in the design and debug work. Plus we would have common software. However, agreement on one processor proved to be impossible. One person wanted to use the 8080 because it was the most popular processor and there was software available for it. Another person wanted to use the 6800 because it used only one power supply, it didn't need as many support chips and the instruction set seemed a little better. A third person had just won a LSI-11 as a door prize and so wanted a system in which he could use the interface and memory cards that would be designed.

It rapidly became apparent that the group was not going to settle on one microprocessor. Therefore, the next best thing was to standardize on a bus structure and I/O addresses. We decided to use the 44 pin Vector connector because it was a common connector and available at most electronic stores. A thorough discussion of the bus structure will be made in the following section.

After the bus structure had been settled and each person had decided on a microprocessor, the job of developing standardized cards was at hand. The first card to be standardized was the television terminal card, TTV. The next card designed was an 8K RAM card using the 1K 2102 RAM. A cassette interface card also emerged but each person modified it to his own liking.

Cards that are in the planning stages are a 16K RAM card which will use a 4K, 16 pin dynamic RAM, a TV graphics card and a paper tape reader interface card. In the future we hope to design a floppy disk controller card and an arithmetic processing card using AMD's 9511.

BUS PHILOSOPHY

It became readily apparent that continuous advancement was taking place in the microprocessor field. Faster, easier to interface, and more sophisticated microprocessor were rapidly being introduced. It didn't seem wise to design the bus for a specific microprocessor which later will be superseded. In fact later versions of a software compatible microprocessor (Z-80 and 8080) may have different buses. A very easy way to upgrade a system is installing a more powerful processor. Since the processor is a small cost of the total system yet strongly determines performance, this is a very practical way of system upgrade. Thus the goal was to develop a bus that would be usable with several processors. In addition since at least two different microprocessor (6800 and 8080) would be used on the bus, the bus must be adaptable to at least these two processors.

Table 1 shows a comparison of several microprocessor and minicomputer buses. This list is not intended to be complete but merely to illustrate some of the important features of some well known buses. Obviously the most important requirement is to read and write into memory. Without this no program can be executed. As can be seen all but the Motorola use a separate read and write which indicates the operation to be performed (the address is also valid if either read or write is asserted). In the 6800 the read/write line together with #2 VMA can easily generate read and write controls. Therefore, this is the format used for the bus. The next requirement is some sort of wait control for use with slow memories. This is a simple control, which, when asserted by the memory, indicates memory is not ready. In both the 6800 and 8080 this control merely extends the memory cycle.

By making the peripherals appear as memory no separate I/O instructions are needed. This allows all the power of memory instructions to be used with peripherals. A 256 word memory address space is decoded on the processor board called I/O. This simplifies peripheral decoding since the most significant 8 bit are already decoded.

TABLE 1
BUS COMPARISONS

	ALTAIR/IMSAI 8080	MOTOROLA 6800	DEC LSI-11	44 PIN BUS
ADDRESS	16 lines	16 lines	16 lines	16 lines
DATA	16 lines (separate input and output)	8 lines (bi-directional)	16 lines (bi-directional and multiplexed with address)	8 lines (bi-directional)
READ AND WRITE CONTROL (MEMORY AND I/O)	Memory read Memory write Ready Input operation Output operation Write Data bus in 3 memory protect lines	R/W VMA #2 Wait	Read Write Reply Sync Write/byte Bank select 7	Read Write Wait I/O
INTERRUPT	8 vectored interrupt lines Interrupt acknowledge Interrupt enable Interrupt request	Halt Bus available	Interrupt request Interrupt acknowledge - daisy chained Event	Interrupt request Interrupt acknowledge-daisy chained Non-maskable request
OTHER	Reset about 16 others	Reset TSC DBE	Initialize Refresh Halt 2 power/up/down lines	Reset TTL clock

The interrupt structure is more complicated to standardize on. Microprocessor companies are rapidly putting more of this function into the processor. There are two important items to consider in this area. The first is priority interrupts. This insures the most important peripheral has priority in interrupting the processor. Another item is vectored interrupt. This allows the peripheral device to tell the processor the address of the peripheral handling routine. This latter feature is probably more important since it alleviates the hassle of software polling if more than one peripheral can assert an interrupt request. However in cases where high speed peripherals are used the first feature may be important.

The bus has both priority and vectored interrupt. The interrupt request line is a line which any peripheral can ground. This causes the processor to assert an interrupt acknowledge. This line is then daisy chained on the bus. The device nearest the processor requesting service will not pass the interrupt request but will gate its handling address on to the data bus. In addition there are two interrupt levels (non-maskable and maskable) which allows the processor to have two priorities. Another version of the bus may be implemented latter. This requires slightly different bus wiring since each interrupt request is brought into a priority encoder where the interrupt request priority is compared against the processor priority. If the priority is greater than the processor priority an interrupt request is generated to the processor. This must be implemented on a separate card since there is not enough spare pins on the bus. Implementation of this is very straightforward using some of the new IC's. However this feature isn't felt to be necessary unless our systems become significantly more complex.

Another important feature is direct memory access (DMA). This allows a peripheral device to take control of the bus for high rate transfer to and from memory. This is implemented in DMA request and DMA grant. Priority is determined by the device physically nearest the processor. Again any peripheral can assert a bus request (DMA request) by grounding the bus request line. The processor will then remove itself from the bus and generate a bus grant. This then is daisy chained much as the interrupt request is daisy chained.

INDIVIDUAL CARDS

In the following section each individual card will be discussed. All I/O devices are treated as memory locations. Table 3 shows the address assignments for the I/O devices and their associated status words. The name of the principle designer of each card is given in parentheses when a card is discussed. A great deal of the design was taken from articles. These designs were then modified, improved or changed to conform to our bus.

8080 Card (Castro). The design of the 8080 processor is basically taken from Intel's 8080 user's manual.¹ A block diagram of the 8080 card is shown in figure 1. The 8224 clock chip and the 8228 control chip are used in the design. The address lines are buffered with 8T97's. 3K of ROM and 1K of RAM are also included on the card. 2708's are used for the ROMs and 2102's are used for the RAM's.

To provide memory address for I/O devices the upper 8 bits of the address lines are AND'ed together to give 256 I/O addresses. The bus I/O line is asserted whenever the upper 8 bits of the address lines are all "ones." This places the I/O addresses from FF00 to FFFF. It should be noted that the 6800 card places the I/O at a different place in memory because of architectural differences. The bus I/O line is also asserted whenever an I/O instruction is executed. The design of the 8080 card also includes provisions to allow the software program to be stepped. The design of this hardware is taken from a Designer's Casebook article.² The hardware required only one 7474 and 2 NAND gates plus two toggle switches and one pushbutton switch.

The 8080 card was designed to allow the monitor program which is in the 2708 ROM's to be located anywhere in memory. To cause the processor to go to the monitor, the processor is fooled into thinking that the next instruction is an unconditional jump to the monitor program. The design hardware to cause this jump is shown in figure 2. When the push button is pushed the address lines and bus I/O line are turned off. The 82S123 ROM is then turned on when the bus read is asserted. Also the 74163 counter is incremented

TABLE 2
BUS PIN ASSIGNMENTS

PIN	SIGNAL
1	GND
2	A0
3	A1
4	A2
5	A3
6	A4
7	A5
8	A6
9	A7
10	A8
11	A9
12	A10
13	A11
14	A12
15	A13
16	A14
17	A15
18	-5 V
19	I/O
20	-12 V
21	TTL Clock
22	+5 V
A	-
B	D0
C	D1
D	D2
E	D3
F	D4
H	D5
J	D6
K	D7
L	+12 V
M	READ
N	WRITE
P	WAIT
R	BUS REQUEST
S	-
T	INTERRUPT REQUEST
V	NON-MASKABLE INTERRUPT REQUEST
W	INTERRUPT ACK IN
X	INTERRUPT ACT OUT
Y	RESET
Z	BUS ACK IN
	BUS ACK OUT

TABLE 3

ADDRESS SPACE

6800	Monitor (2K)	F800-FFFF
6800	Monitor RAM (128)	EF80-EFFF
6800	I/O space (256)	F400-F4FF
8080	Monitor (3K)	E000-EBFF
8080	Monitor RAM (1K)	EC00-EFFF
8080	I/O space	FF00-FFFF

I/O Address Assignments

Keyboard (P1A)	08-0B
TVT Status Word	F8-F9
Cassette #0 (UART)	00-01
#1	02-03
#2	04-05
#3	06-07
Paper Tape Reader (P1A)	0C-0F

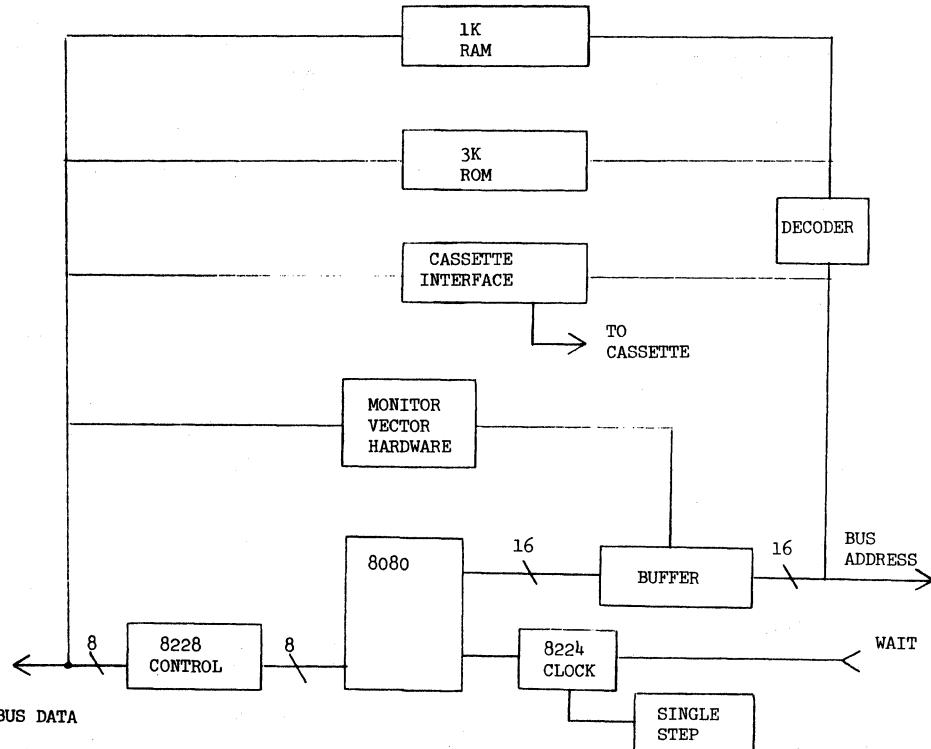


Figure 1. 8080 processor card.

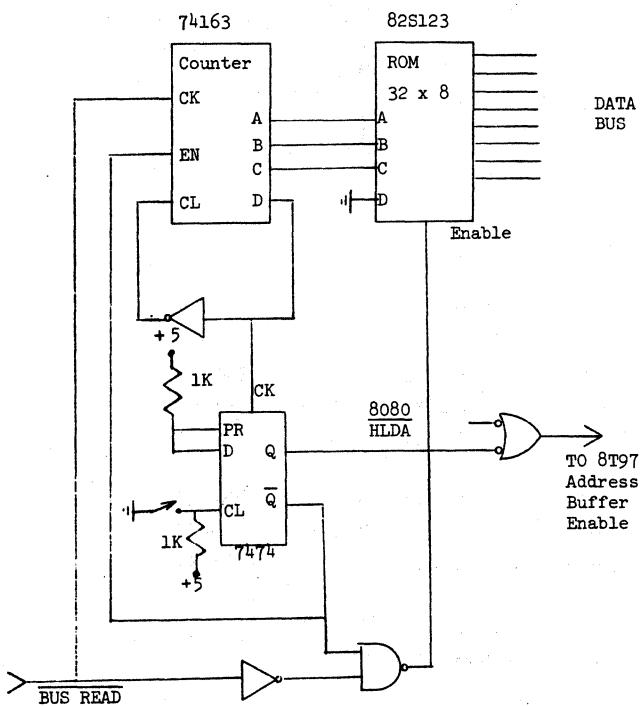


Figure 2. Monitor vector hardware.

after each bus read. The ROM contains the jump instruction and the jump address. When the 74163 counter gets to address 8 the ROM is turned off and the address lines and bus I/O line are reasserted. The purpose of this hardware is to allow programs to reside at the bottom of memory and the monitor to be located somewhere else. In my system the monitor is located at E000.

The 8080 design plus ROM and RAM occupy only half of the space of the 9.5 inch by 4.5 inch Vector card. A cassette interface card was designed but because of the empty space I decided to place the cassette interface on the 8080 card. The circuit is the same as one used for the cassette interface card.

TVT card. The design of the television display card was taken from an article, "Build a Television Display,"³ with minor modifications.

The display is arranged as 32 characters by 16 lines. We used 2102's for the display storage. Since the display only requires 512 bytes, two pages of display are available. The display is addressable as ordinary memory and is located from F000 to F3FF. The TVT card also has a I/O status word located at F8. The status word indicates when the display is available for modifications and which page the display is using. Also the page can be changed via the status word.

We used a different rf up-converter than Mr. Gantt. Figure 3 shows the circuit we used. The oscillator is tuned for channel 3.

The Cassette Interface. The cassette interface uses Don Lancaster bit boffer circuit⁴ to implement the Kansas City standard. For a UART an Intel 8251 was used. However one person used the Motorola 6850. The addresses used for the cassette data and control are shown on table 3.

8K RAM Card (Mitchell). The 8K RAM shown in figure 4 uses the 1K static 2102's. The card uses 64 RAM chips and 8 IC's are used for interfacing.

6800 Processor Card (Heaberlin). This card is similar to the processor circuits discussed in the Motorola 6800 applications manuals⁵ (see figure 5). Initially it was decided to use back-to-back monostables rather than a crystal

clock. This configuration is tentative slightly faster in that the processor may continue immediately after concluding a DMA cycle or a wait period (to be discussed later.) As is well known the 6800 requires a two phase, non-overlapping clock. During phase 1 the address and R/W is asserted while during phase 2 the data becomes valid. Obviously for slow memories or other slow I/O devices phase 2 must be extended to allow adequate read and write times. This is provided by the wait signal. This extends phase 2 until the wait signal is removed by memory or peripherals. In a similar manner phase 1 can be used

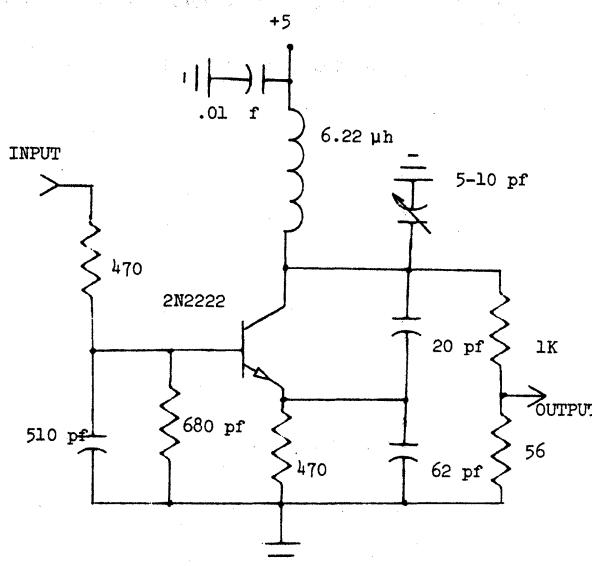


Figure 3. Channel 3 rf up-converter.

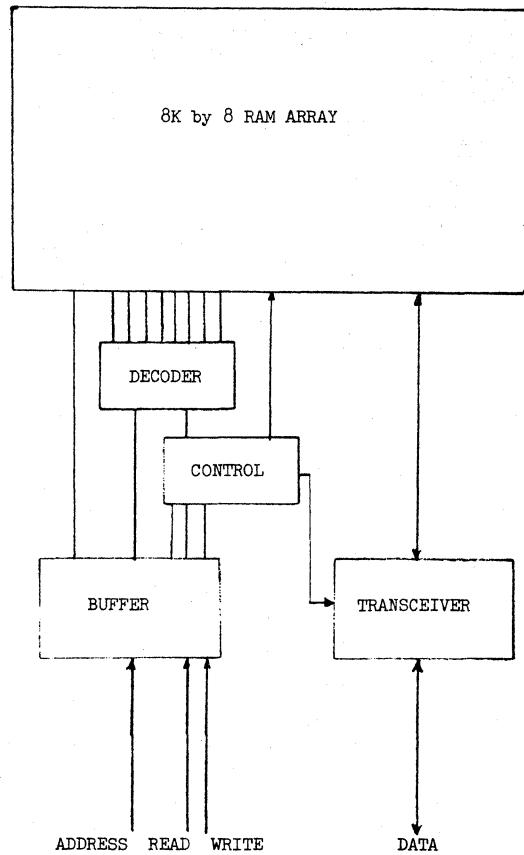


Figure 4. 8K RAM card.

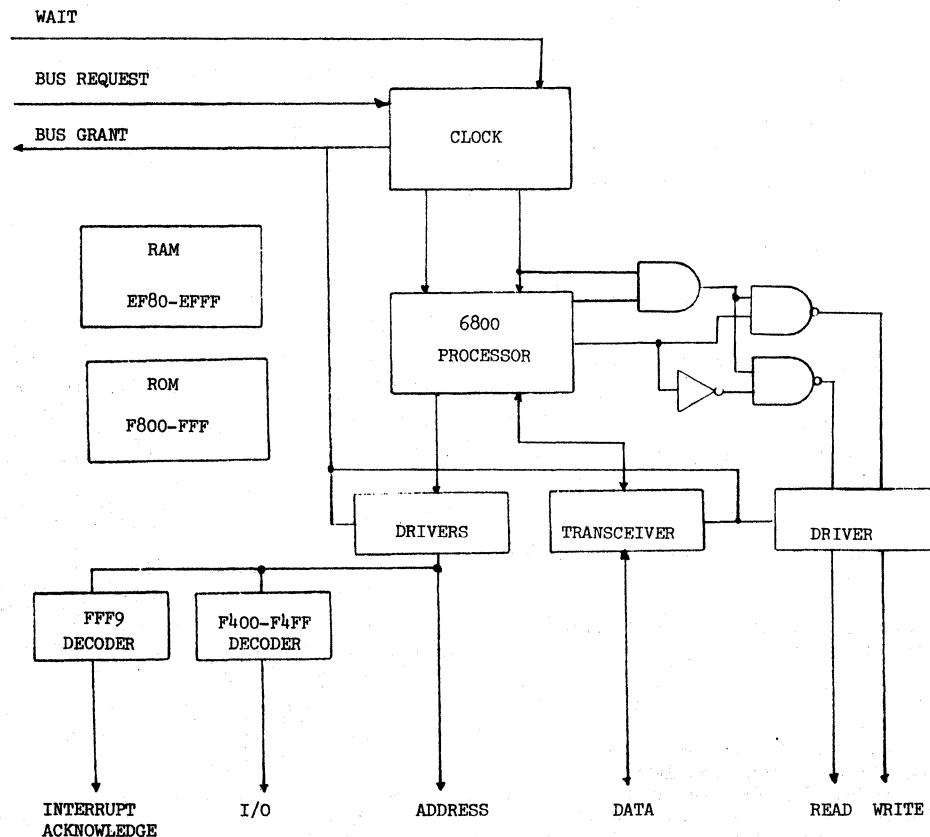


Figure 5. 6800 processor card.

for DMA cycles. In this case when a DMA cycle is requested the address is removed from the bus and the bus grant signal is asserted from the processor during phase 1. This then allows another device to have control of the bus. Since the 6800 is a dynamic device neither the DMA cycle or wait period can be extended beyond 5 microseconds. This method for DMA and slow memories is standard with 6800 systems.

As has been mentioned previously the 6800 has a read/write line and chip select line which is usually $\bar{D}2\text{-VMA}$. In this design we generate a read signal by AND'ing the read/write line with $\bar{D}2\text{-VMA}$. A write signal is similarly generated by first inverting read/write and then AND'ing it with $\bar{D}2\text{-VMA}$. This provides a read and write superficially close to the 8080. However in the 8080 write cycle the data from the processor becomes valid before the write is asserted while in the 6800 the data will become valid about 200 nsec. after the assertion of write. In addition, in the 6800 the data is removed as soon as the write is removed. In the 8080 the data is held about 300 nsec. after write is removed. Thus the timing in the 6800 write mode is a lot more relaxed. This may seem to be a problem and in some cases where data must set up before a write strobe is asserted additional timing circuitry on the peripheral card will be required. However many of the new memory and other circuits have very relaxed timing requirements. They require the data to be valid only during the last part of the write cycle and no data hold times.

The other part of this design different from standard 6800 designs is the interrupt structure. As has been pointed out an interrupt acknowledge is generated from the processor card. The 6800 processor of course has no interrupt acknowledge output. However if the address FFF9 is decoded then this can be used as an acknowledge since the processor interrogates this address when responding to interrupts. Peripheral devices will place the lower 8 bit address of the handler routine on the bus in response to the interrupt acknowledge.

There is also incorporated drivers on all output lines from the processor card except the daisy chained lines (bus grant and interrupt acknowledge). This allows many devices to be connected on the bus.

CONCLUSION

From experience a budget of 500 dollars would be adequate to provide a person with a good homebrew system (see table 4). This system would consist of a TVT interface, cassette interface, keyboard interface, 8K memory, processor and power supply. Of course the above cost only covers hardware and doesn't include labor. Many hours are spent in fabrication of cards and chassis. All our cards were fabricated using wire-wrap techniques. However a solder type 44 pin connector was used for the bus. These connectors were inexpensive, but in retrospect the extra cost of wire-wrap connectors are worth the extra cost.

TABLE 4

COSTS

COST	CARDS & SOCKETS	ICs & DISCRETES	TOTAL
TVT	26.00	55.00	81.00
8080	27.00	126.00	153.00
Cassette	20.00	23.00	43.00
6800	17.00	94.00	111.00
8K RAM	32.00	118.00	150.00

Cesar Castro. Electronic Design Engineer, Naval Ocean Systems Center, San Diego, California.
BSEE 1967 San Diego State University.
MSE 1971 Purdue University.

Allen Heaberlin. Electronic Design Engineer, Naval Ocean Systems Center, San Diego, California. BSEE 1968 University of Missouri at Rolla, MS Information Systems, University of California at San Diego.

REFERENCE

1. Intel Corp., Intel 8080 Microcomputer Systems User's Manual, September 1975.
2. Wakerly, John F., "Circuit Steps Program for 8080 Debugging," Electronics, pp. 110-111, August 5, 1976.
3. Gantt, C. W. Jr., "Build a Television Display," Byte, pp. 16-21, June 1976.
4. Lancaster, Don, "Build the Bit Boffer," Byte, pp. 30-39, March 1976.
5. Motorola Inc., M6800 Microprocessor Applications Manual.

16-BIT AND 32-BIT ADAPTATIONS OF THE S-100 BUS STANDARD

Gary McCray
188 Western Avenue
Livermore CA 94550

I represent no company and am just an average (if there is such a thing) computer hobbyist, like most of you. I don't like writing papers or giving speeches and, quite frankly, I'd much rather be at home doing amazing things on my computer right now. However, there is a proper time for everything and if we, the hobbyists, are to derive maximum benefit from future developments, something has to be done now.

The best single thing that has happened to this hobby, aside from the development of the microprocessor itself, was the establishment and general acceptance of a standard buss: "Hobbyist Standard", "S-100", or "Altair - Imsai" buss. It has provided a fertile ground for manufacturers of peripheral devices, permitting them to market a single product such as memory I.O., C.P.U.'s, and so on, rather than having to produce complete systems that must include all the necessary components. This has resulted in a competitive market which has served to keep prices down, quality high, and inexpensive, easy-to-work-with software readily available...All working to the hobbyists advantage.

Now, there is an unfortunate trend for some of the larger manufacturers to produce complete systems which totally ignore the "S-100" buss, making outside updating and modification of their systems extremely difficult and/or expensive at best. The Processor Technology "SOL" is a notable exception, providing "S-100" card adaptability and many worthwhile features, but still not permitting the use of another C.P.U. There is no practical way to change the C.P.U. in any of these single board systems, necessitating the purchase of an entire new system, should such a change be desired--an expensive and certainly not hobbyist-oriented process. Furthermore, these single manufacturer designs do not encourage outside competitive development reducing variability and keeping the prices up: a great deal for the manufacturer (if he can sell it) and a rotten deal for the hobbyist.

Obviously some sort of standard is very worthwhile and, with a little bit of work, it can be seen how the "S-100" buss can be turned into a very powerful thing, whose potentials extend far beyond hobbyist use while still encompassing it.

The "S-100" has the particular advantage of complete de-multiplexing, providing separate data in, data out and address lines--a feature which makes trouble-shooting and design magnitudes easier. Interfacing is straightforward and, further, this de-multiplexed approach makes it possible to adapt virtually any 8 bit C.P.U. regardless of its multiplexed timing and control characteristics.

As for specific complaints about the "S-100" buss: the major one, noise, has been virtually eliminated by the introduction of noise-free mother boards such as the Morrow "Sigma" active termination board or the Crommenco "Z Buss". The second complaint of being slow due to the de-multiplexing is not really relevant to the vast majority of uses to which these mini-computer emulation systems are put. We would readily accept that a system to be maximized as an ultra-high speed scientific or industrial (dedicated) controller might be better implemented on other than the "S-100" buss. Even in this, though, faster microprocessors, memory and interface chips are making any time difference negligible.

So, OK, the "S-100" buss is a great thing, but how is it possible to implement a bigger C.P.U. with sixteen or more bit words. There are at least three approaches to this. One is being currently implemented by Alpha Micro Technology in their "CM-16", a pseudo "LSI-11" almost emulation. They use the "S-100" buss as is and divide sixteen bit memory addresses into two bytes and ship them to sequential spots in memory, requiring two actual moves. All this is handled very neatly by on board logic. It really is a great design and they have some excellent software to support it. It makes use of the vast majority of "S-100" cards and costs considerably more than the "LSI-11", which is unquestionably superior and has the best software support available. Be that as it may, it is a great design and does use the "S-100" buss.

I do not feel, however, that this approach is satisfactory for general higher order development of the "S-100" buss. The necessary byte-word exchanging greatly increases the complexity of the hardware, software and design criteria as well as slowing over all transfers right back to 8 bit system levels.

The second approach, which has several advantages over the first, is bank addressing. In this scheme, the micropro-

cessors' internal microcode toggles an address line changing the memory or I/O board addressed to implement byte-word shifts. This technique is much faster than the previous sequential system and doesn't require nearly the hardware. If one wishes to use more than 32 K of 16 bit or 16 K of 32 bit words, however, it requires the reassignment of some "S-100" spares as bank address lines to enable and disable the various memory banks. The major disadvantage, as I see it, to this approach is that, first, it requires microprocessors that are microprogrammable which will limit potential C.P.U.'s. Secondly, the C.P.U.'s themselves will still be functionally more complicated than plain "S-100" C.P.U.'s. Thirdly, real-time troubleshooting, design and debugging, both electrically and software-wise, will be increased in degree of difficulty over "S-100" levels. In all honesty though, this system is viable, and in the future most microprocessors, certainly most 16 bit and larger ones, will be microprogrammable. I would still like to see a system that would permit us to operate in this bank mode as cards become available, also, in our old 8 bit mode, and in a larger scale format that permits "S-100" like large scale 16+ bit cards to operate in a manner exactly like the 8 bit ones in our systems now, and requiring no soft or hardware tricks.

What we need is a way to implement these higher order systems that will provide simple and cheap design and manufacturing criteria to encourage their production and yet still be able to use "S-100" cards. For that matter, why not try to design in a little future development capability and try for a potential 32 bits of data. This could be accomplished in bit-slice now and in microprocessors in less than three years. While we're at it, we might as well increase our potential addressing capability to a full 24 bits, allowing a maximum possible of nearly 17 million words (8, 16 or 32 bit) to be addressed. There are already 92K and 256K bit chips on the market, so this is not an unreasonable future potential. This greater than current technology approach has the advantage of not getting us stuck with our 16 bit systems like we are with our 8 bitters now. Or are we?

I've spent many hours staring at my "S-100" system, wondering how I could stuff in a Texas Instruments "9900" C.P.U. but it wasn't until I built a bare "S-100" motherboard and power supply for our local computer club that it hit me. We can have all those "S-100" cards and have our 16 and 32 bit super systems, too. Look at the boards; they only have contacts under two-thirds of them. Actually, there is room under an "S-100" card for a 160 pin edge connector, still permitting standard edge guides to guide standard "S-100" cards into the connectors. With the addition of 2 divider bars (reducing our total to 156 available pins), the "S-100" card is securely locked into the same place it always was.

Simply leave all the "S-100" buss line assignments where they were, the first 8 data in, first 8 data out, 16 address lines, power supply, clocks and control lines. But what about those other 56 lines? We already have 8 data in and 8 out, and with the addition of 24 more in and 24 more out, plus the addition of 8 more address lines, we have used up our 56 lines and now have a total capability of 32 bits of data in, 32 bits of data out and 24 address lines.

Thus, simply by retrofitting an "SS-160" or "Super Standard 160" motherboard to our system, we gain the fantastic potential of 32 bits of data, 32 bit instructions and 17 million word addressing. Now that's a standard we can live with for awhile. Furthermore, our current system operation is completely unaffected, permitting updating only as we see fit. In order to provide a standard, the actual position of each extra data and address line would have to be fixed by convention, and certainly the second 8 bits of our data word, both in and out, and the extra 8 address bits should all be on one side of the board, permitting manufacturers to make 16 bit only cards, should they so desire. Further, a few, 4 or 5, of the "S-100" spare lines should be given permanent "SS-160" control assignment, for such things as recognition of 16 and 32 bit transfers, etc.

A second piece of hardware should also be made initially available to permit the use of "S-100" memory cards on the higher portions of the data words found on the "SS-160" buss. This could take the form of a "data reassignment edge connector". Specifically, a short P.C. card that plugs into the "SS-160" buss and extends all the lines (except data) up to an "S-100" connector at its top; the data lines would be brought from the appropriate places on the "SS-160" card to fill in the appropriate word. This would permit two "S-100" memory cards to be addressed the same and to address the high and low order bits of 16 bit words. This could be carried out later for 32 bit systems if necessary.

The "SS-160" approach has several less obvious advantages. First, they will fit in our current "Altairs", "Imsais", "Bytes", etc. virtually without modification. Second, we can add specific "SS-160" C.P.U.s., memory, I/O, etc. only as we see fit. In no circumstance would an "SS-160" device cause some other "S-100" device to be nonfunctional.

Say we have the following hypothetical system that we wish to change to an "SS-160". (An "Imsai with 32 K of memory, a VDM-1 and a Morrow, cassette, Keyboard I/O) First, we install an "SS-160" 20 connector motherboard. Our system operates exactly as before. Next, we add an "XX9900" or "XXX-11" "SS-160" C.P.U. and reassign half of our memory with reassignment edge connectors. Now, with an absolute minimum of expenditure, we have gone to a full 16 bit system with 16K of 16 bit word addressing. We can leave the I/O the same for now; our software can change easily from byte to word, and the vast majority of the time is still saved because all of our memory and register transfers are now 16 bits. Later we can add an "SS-160" 16 bit I/O device to further maximize software, speed and so on.

"SS-160" memory, I/O, front panels and C.P.U.'s. would undoubtedly be marketed first in 16, then in 32 bit format, at a rate and price that we could far better afford than if we are relying on separate manufacturers, producing separate systems with limited potential.

But first we need the system, and if it is going to happen, it is going to be us that make it so. If, and I stress only if, sufficient interest in this scheme is shown, will anything real be done. Otherwise, we will remain completely at the whims of the big manufacturers. The "S-100" buss gave us the right to demand more once, let's not give it up to the inflexible all in one computers now. We can only benefit from a standardized system.

In summation, let me point out that this "Super Standard 160" system encourages all current forms of "S-100" development such as the normal 8 bit systems, the hybrids such as the "CM-16" and the even greater potential bank addressed devices, while still making it possible to implement the most efficient of all "SS-160" devices as desired.

If you feel that the "Super Standard-160" has sufficient value to justify it and, especially, if you would like to implement it yourself, please write: "SS-160", c/o Dr. Dobbs Journal of Computer Calisthenics and Orthodontia, Box E, Menlo Park, Ca. 94025.

If sufficient interest is shown I, personally, will pursue the matter as best I can and am reasonably well assured of sufficient small manufacturer interest to feel confident that a production retrofit as well as a complete system could be forthcoming. I reiterate, such an effort will be made only as the result of sufficient interest being shown by you. Now is your chance to shape the future of our hobby.

STANDARDIZATION OF THE S-100 BUS: TIMING AND SIGNAL RELATIONSHIPS - A PROPOSED STANDARD

Tony Pietsch
Proteus Engineering
1717 E. Oakdale Ave.
Pasadena CA 91106

About four months ago I decided to put the TMS9900 microprocessor chip on the "S-100" bus. I had just finished interfacing some high speed, high resolution (12 bit) ADC's and DAC's to the IMSAI 8080 bus, so I thought I had a good idea as to what should go where when. Since there seemed to be an unlimited variety of interfaces and memory that were available as "S-100 bus compatible", it seemed to be the way to go. After laying out a preliminary design, I went down to my local Byte Shop and perused their schematics of all the boards that they sold (and some that they didn't) and discovered quite a number that weren't exactly compatible. In addition, I was shown a number of combinations of existing processors and memory and interfaces that wouldn't "work together". I then set out to find the lowest common denominator of the S-100 bus and its timing. I intended on setting up a standard way in which processing was to be handled so that only a few cuts and patches could make everything work together. The results of my standardization procedure are to be regarded only as a commentary on what might make the S-100 bus a more comfortable place to work.

The first thing I noticed when I looked for a smallest common signal set was that nearly every possible subset of signals was used by someone. This meant that to reach the widest range of existing boards, timing and control would have to be defined using the existing redundancies. Signals peculiar to the 8080A were not used in my standardization procedure so the widest range of processor boards would be able to comply with its definition. Some signals on the Altair bus were combined and given simpler, more specific names.

Proper decoding of the timing on the S-100 bus is not difficult once certain relationships are made clear. Current processors using the bus do a data transfer in two parts; a valid address time and a valid data time.

For timing of data transfers eleven signals are commonly used; SINP, SOUT, SMEMR, SWO, MWRITE, PDBIN, PWR, PSYNC, ϕ_1 , ϕ_2 , PRDY and PWAIT. The four major data transfer states are decoded as shown in table 1. An example of the data transfer bus relationships is shown in figures 1 thru 4. Note that only one of the transfer states can be enabled at any one time.

TABLE 1

- 1) Memory Read
 - A. ENREAD = $\overline{\text{SINP}} \cdot \text{PDBIN}$
 - B. ENREAD = $\text{SMEMR} \cdot \text{PDBIN}$ (recommended)
- 2) Memory Write
 - A. ENWRITE = MWRITE (recommended)
 - B. ENWRITE = MWRITE + ($\overline{\text{SOUT}} \cdot \text{PWR}$)
- 3) I/O Read

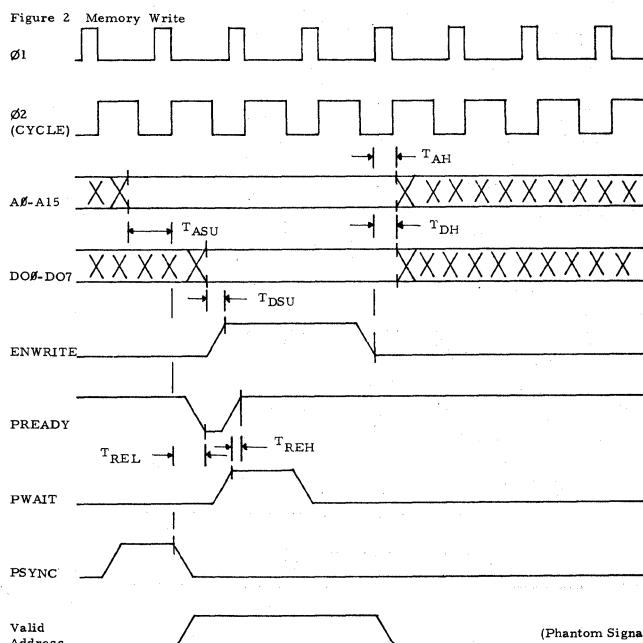
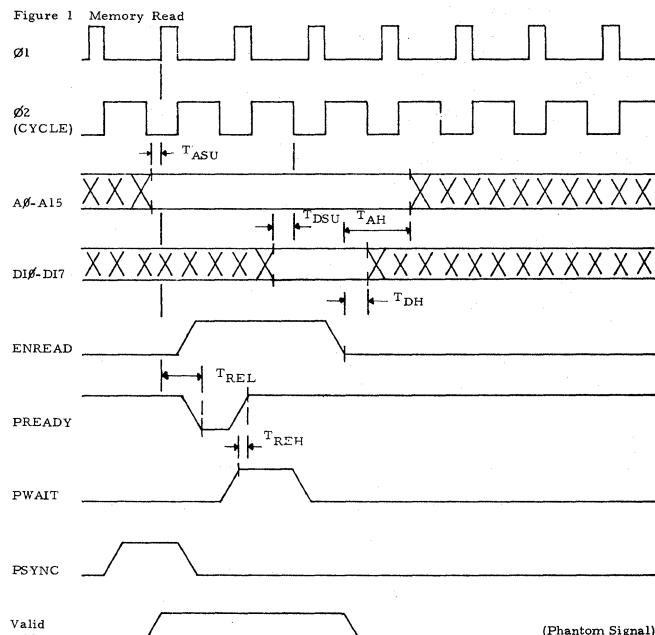
$\text{ENIN} = \text{SINP} \cdot \text{PDBIN}$
- 4) I/O Write

$\text{ENOUT} = \text{SOUT} \cdot \overline{\text{PWR}}$

SWO is sometimes used in conjunction with an on-board data direction line.

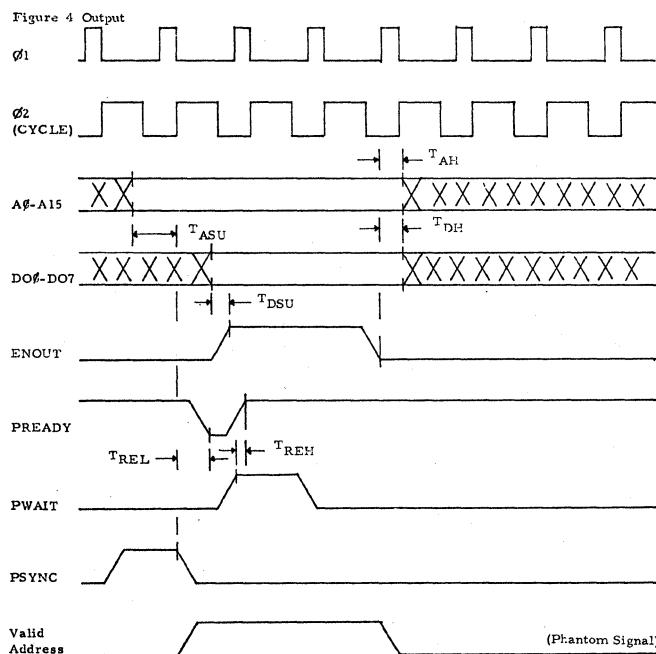
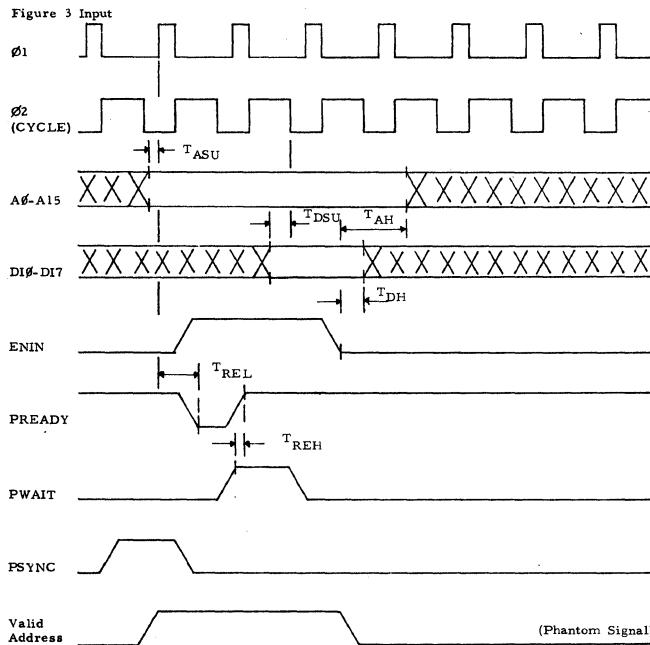
The signals PSYNC and ϕ_1 are used to indicate a valid address. Valid address is a phantom signal (not defined on the S-100 bus) that is used primarily for generating wait state timing. The valid address signal should be decoded true on

the positive going edge of ϕ_1 when PSYNC is high (true). If a wait state is to be generated, the PRDY signal line must be pulled low at this time (most 8080 processor boards - the ones that use the 8224 clock generator - allow at most 110 nsec for response and settling). The valid address signal may also be used in enabling some buffers, and if used in this manner should go false on the negative transition of the data transfer enable (ENIN, ENOUT, ENREAD, or ENWRITE).



If a single wait state is to be generated, the PRDY line should be pulled low on occurrence of a valid address (as above) and released when the processor line PWAIT goes high. This generates exactly one wait state for almost any microprocessor. Alternatively, one or more wait states

can be made by using the negative going transition of ϕ_2 (CYCLE) to count an appropriate number of machine cycles and release the PRDY line at this transition. In this way (using the negative edge of ϕ_2) it will be certain that the correct number of wait states is generated. I would like to rename ϕ_2 as CYCLE as not all processors complete a machine cycle at the end of a second phase.



The data valid time is defined during the data transfer states EN_{IN} , EN_{OUT} , EN_{READ} , and EN_{WRITE} . Timing generated on the S-100 bus by a processor board should allow for both level sensitive and transition sensitive devices. To ensure good transfer for transition sensitive circuits, data and address lines should remain stable for at least 25 nsec (T_{DH} , T_{AH}) after the enable condition goes false. To allow

full settling during a read (or input) operation, data should be stable on the bus at least 25 nsec before the processor samples it. Since two existing processors on the bus (8080 and Z-80) sample at or shortly after the negative going transition of ϕ_2 , data should be stable 25 nsec before ϕ_2 negative transition. To ensure proper board enable decoding the address bus must be stable at least 25 nsec before the valid address condition is true. The choice of 25 nsec was made to allow for the propagation delay of most decoding circuitry in use. A summary of timing relationships is given in table 2.

TABLE 2

S-100 Bus Timing Requirements

	min	max
T_{ASU} (Address set-up)	25 nsec	
T_{REL} (Ready high-to-low)	25 nsec	
T_{REH} (Ready low-to-high) ¹	25 nsec	
T_{DSU} (Data set-up)	25 nsec	
T_{AH} (Address hold)	25 nsec	
T_{DH} (Data hold)	25 nsec	

¹ Ready low-to-high may be made either with respect to the positive-going transition of P_{WAIT} or the negative-going transition of ϕ_2 (CYCLE).

To allow certain interfaces to operate correctly, a stable time base is needed. This should be the crystal controlled 2 MHz CLOCK signal.

The main pitfall of the Altair bus seems to be that it is specific to the 8080A. Signals like SSTACK, SS, and RUN are unnecessary, and others could be condensed into more specific functions (Valid Address, Memory, I/O, Read/Write, etc.). Most wasteful is having eight separate data input and data output lines. In general, however, the S-100 bus is workable but inconvenient for use with 16 bit processors. If it is to be a true standard, a better definition of signal timing must be accepted. The signal timing I have defined is the best that I know of, but I may have missed something. If nothing drastic is found wrong then I would recommend that all future S-100 bus compatible designs be based on this standard.

DMA OPERATION PROTOCOL IN THE S-100 BUS ENVIRONMENT

Terry Walker
 Cromemco, Inc.
 2432 Charleston
 Mountain View, Ca
 (408) 964-7400

The need exists for a uniform protocol of DMA operation in S-100 bus systems. This paper strives to fill this need by showing the conditions necessary for compatible operation of several DMA devices in one computer mainframe. Topics covered include CPU output signals, DMA memory control cycles, common implementation problems, recommended DMA control protocol, and hardware design examples. The primary purpose of this paper is to promote standardization of DMA methods on a form compatible with both static and dynamic memory units and 8080 and Z80 CPU's. No attempt is made to cover other CPU types, and it is assumed that the memory in use does not require wait states for the DMA device in question.

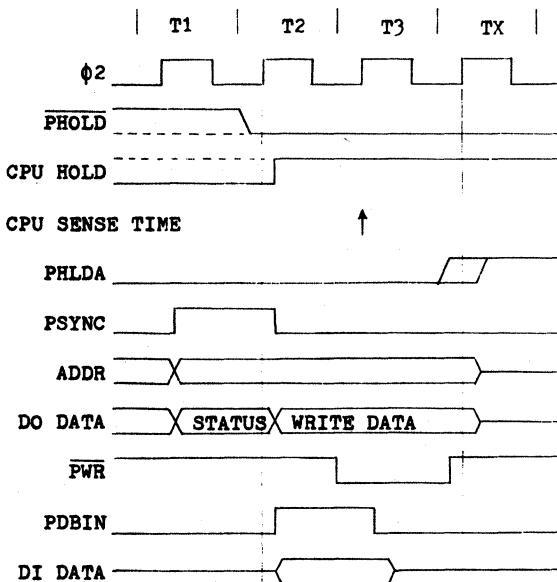


FIGURE 1. 8080 DMA INITIATION SIGNALS

Figure 1 shows, for example, the relevant bus signals for an 8080 CPU as used in an Altair or Imsai machine at the time of DMA initiation. The extra clock period delay of PHOLD recognition caused by the synchronizing FF causes PHLDA to go high about 2 clock periods after the last possible time for PHOLD recognition. Since PHLDA is timed from the ϕ_1 positive going edge, the delay specification for this output permits the PHLDA positive edge to occur on either side of the ϕ_2 positive edge. The subsequent ϕ_2 negative going edge is the first reliable time for sensing the PHLDA state. Shortly after PHLDA goes high at the end of the memory cycle, the CPU floats its address and data busses. The control outputs remain in their inactive state.

Figure 2 shows the corresponding signal changes occurring at the time of DMA termination. Again note that the CPU does not pull down PHLDA and begin operation until two clock cycles after the PHOLD line goes positive.

Figure 3 shows the relevant bus signals for a Z-80 as used in a CROMEMCO ZPU at the time of DMA initiation. In this case no PHOLD synchronization FF was necessary, so the CPU may only wait one clock cycle before changing PHLDA. When PHLDA goes high, it follows the ϕ_2 positive edge by the CPU propagation delay. The CPU floats its address and data lines at the same time as the PHLDA bus change.

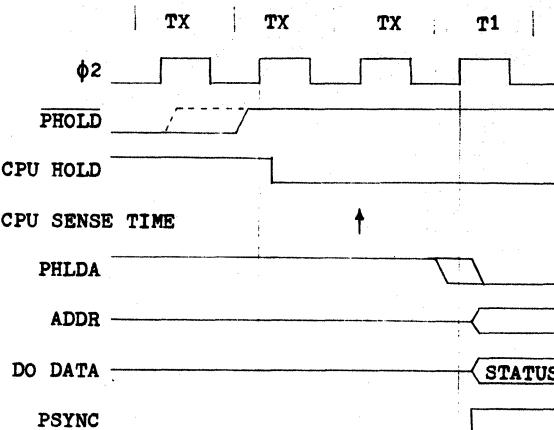


FIGURE 2. 8080 DMA TERMINATION SIGNALS

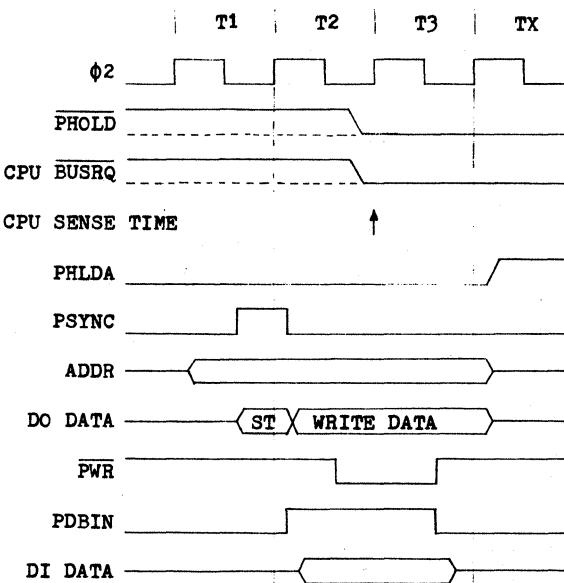
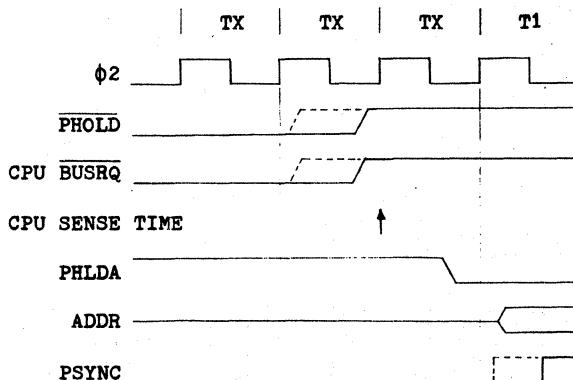
FIGURE 3. Z80 DMA INITIATION SIGNALS
(CROMEMCO ZPU)FIGURE 4. Z80 DMA TERMINATION SIGNALS
(CROMEMCO ZPU)

Figure 4 shows the corresponding signals at the time of DMA termination. In contrast to the case of Figure 2, the CPU takes PHLDA low and begins operation only one half clock cycle after PHOLD goes high. The PHLDA change occurs relative to the ϕ_2 negative going edge, preceding the resumption of CPU operation by one half clock cycle.

The important items in both cases are that PHOLD is sensed at the ϕ_2 positive going edge, PHLDA may be reliably sensed only at the ϕ_2 negative going edge, and a delay of more than two clock cycles may occur between PHOLD going high and PHLDA going low.

If we look at the various memory cycles of the 8080 and ZPU, we see that the basic DMA memory cycle of Figure 5 encompasses the necessary signals in a similar form. The DMA Enable signal turns ON the address, data, status, and control bus drivers on the DMA unit. Simultaneously, it pulls low the ADDR DSBL, DO DSBL, CONT DSBL, and STATUS DSBL bus lines to

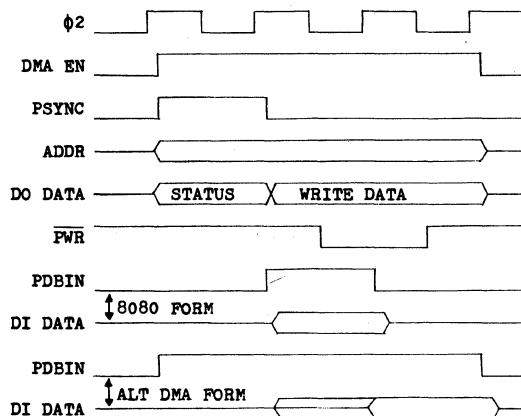


FIGURE 5. BASIC DMA MEMORY CYCLE

disable the CPU outputs. Control of the S-100 bus immediately passes to the DMA control unit for the time that DMA Enable is high. The DMA controller must provide solid drive for all bus lines released by the CPU. Take special note, however, that ϕ_1 and ϕ_2 always remain under control of the CPU card. At the time control transfer occurs, and for about 50 ns after each event, erroneous signal pulses may appear on the S-100 bus lines. These usually occur on control lines such as PSYNC and PDBIN which are active high, and therefore usually low at the transfer time. These glitches occur because the normal TTL gate interpretation of a floating input is a high level. Momentarily, all drivers on a bus line may be off, and attached logic gates pull up the bus line to a high level. This tendency is the major drawback of the S-100 bus signal definitions, and complicates the problem of successful DMA operation. The case when two bus line drivers may both be ON is less serious, as their outputs will assume the logical AND function of the desired signals, and tend to stay low if either one is low. One method of reducing the effect of these glitches on memory units using PDBIN to read data is to turn ON PDBIN immediately when beginning the DMA operation. This appears as the alternate DMA form in Figure 5. However, this should not be required for a properly designed DMA suitable memory card.

Various signals shown in Figure 5 may be compromised in one way or another, and still the majority of available memory units will work. Fully static memories, for example, do not generally require PSYNC or pulsing of PDBIN to read, but dynamic memories generally require one or both. Advance status on the DO data bus generally is not used during DMA by memory boards and can be omitted. The general read cycle is longer than necessary, though shortening the write cycle can cause problems with data setup or hold times.

Figure 6 shows a shortened DMA cycle possible for reading single data bytes only. The PSYNC and advance status may be omitted at the risk of incompatibility. Both of the preceding figures do not show an allowance for wait states since it can generally be assumed that memory fast enough for the DMA function desired will be used. Occurrence of wait states during a multibyte DMA transfer requiring data at definite times (as in the TV Dazzler) would cause loss of data synchronization. The system user must be assumed to have some sense in these matters. Therefore I generally don't construct the DMA controller to use PRDY signals, and all figures in this paper are shown without wait states.

Alternate methods of synchronous DMA operation for appropriately designed memory units may use pulses on the status or other bus lines to initiate

memory cycles. However, there is insufficient commonality among available units to recommend this approach.

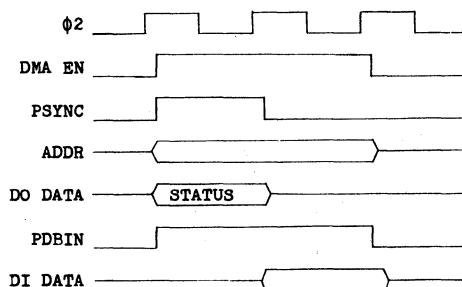


FIGURE 6. SHORT DMA READ CYCLE

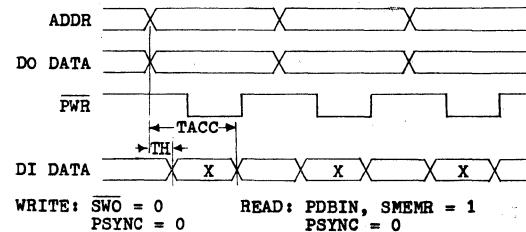


FIGURE 7. ASYNCHRONOUS DMA CYCLES

Asynchronous DMA memory cycles can generally be shown as in Figure 7. This type of operation is usually possible only with fully static memory units, although under carefully controlled conditions some dynamic memory units can support it. In this method, a synchronously clocked address sequence (and data if writing) is output to the bus. When reading, the data for the desired address appears after the access time TACC, and persists until the hold time TH after the next address change. Regions of indeterminate data appear as X. Writing is done with a PWR pulse in the middle of each time cell. This type of DMA was used in the TV Dazzler for reasons of speed and because the desired data rate was asynchronous with the system ϕ_2 clock. This is not the best choice because of incompatibility with higher density dynamic RAM's, and should be avoided if possible. At the time of the Dazzler design 2 years ago, the predominant memory type in use was full static, so incompatibility was not seen to be a problem at that time. The Dazzler operates properly with some dynamic memory units because it supplies a PSYNC pulse synchronized to ϕ_2 during each data read interval. A better choice would be use of a DMA controller permitting synchronous memory cycles as in Figure 5.

Several important implementation problems can occur when setting up a DMA system. One of these is the data hold time on an 8080 after a PWR positive going edge. This time is reduced if the write cycle precedes a DMA interval, and can cause some memory boards to fail. The failure appears to be caused by the DMA unit, but in fact results from the change in CPU timing. Propagation delay in the front panel MWRIT logic exacerbates the problem, as does too many logic gates in the memory board MWRIT signal path. Two cures are use of faster logic on the memory board, and latching incoming data with a pulse edge.

A second problem is the status latch in some 8080 systems. Figure 8 shows a simplified block diagram of the CPU card on my early model Altair. During DMA intervals with all CPU disable lines pulled low, the 8212 still functions. Coincidences of PSYNC pulses on the bus with ϕ_1 clock pulses cause entry of DMA data present on the DI bus into the status latch. Then when control transfers back to the CPU, the status latch word is erroneous, and may be nearly any random 8 bit binary number. Naturally, this may mess up some S-100 bus cards with insufficiently discriminatory design after DMA operation. Again, the problem appears to come from the DMA unit but in fact is located elsewhere. There is no cure without changing the CPU or peripheral card designs.

A third problem is the general class of bus line glitches. Some of these originate in the CPU card bus drivers, and some come from the close proximity of long parallel bus lines. Of particular importance here is the occurrence of a 2V high 10 ns glitch on

the ϕ_2 clock line when PHLDA changes states and the DMA takes or releases the bus. This glitch can mis-clock the DMA state controller and indirectly mess up the memory unit in use. A recommended fix is a 20 ns RC network in series with ϕ_2 where it comes into the DMA controller clock buffer.

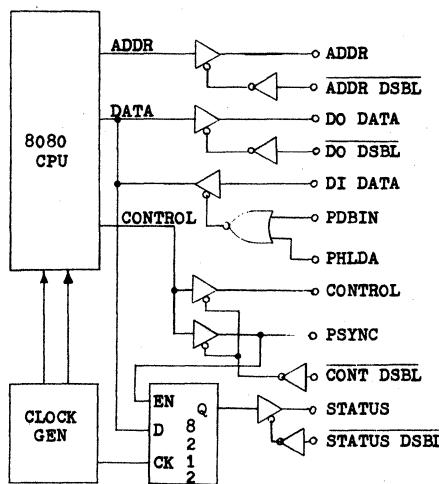


FIGURE 8. STATUS LATCH PROBLEM BLOCK DIAGRAM

A final problem is that some peripheral interface cards may malfunction when presented with the rapidly changing DMA data on the bus. In this case, they should use PHLDA as an input to indicate that DMA operation is in progress and to disable them.

Proper DMA operation in the S-100 bus system necessitates several protocol requirements. Firstly, transfer of control between two DMA units requires a means of signalling the proper time. Given a system containing two DMA controllers properly chained for priority, assume that the unit with lower priority is in operation. Now, if the higher priority unit desires service, it must send that command down the priority chain, and wait for an acknowledgment that the bus is available. I chose to use the PHOLD line for this job, as with proper timing it may serve several purposes. When the higher priority DMA finds the PHOLD line already low, it waits until the lower priority DMA releases it before taking control.

When using PHOLD in this way, attention must be paid to the special conditions when the CPU may detect the PHOLD transfer pulse, and begin operation at the same time as the higher priority DMA. Loss of program execution sequence can result. The PHOLD synchronizing FF required on 8080 CPU cards causes an extra clock cycle of delay, so the DMA controller must allow for this in its state sequencing.

In order to prevent inadequate resolution of priority caused by the finite DMA priority chain propagation time, a request synchronizing FF is recommended on each DMA controller. This FF reclocks DMA action requests with the ϕ_2 positive going edge, causing at least one half clock cycle to be available for the priority ripple. This is adequate for small systems with TTL controllers, and allows for ϕ_2 clock skew between DMA controller cards.

Finally, the DMA controller section which meters out memory cycles must produce only full cycles. If the memory cycles are abruptly terminated by the priority input going low, for instance, the resulting short pulses could cause loss of data in a dynamic memory unit. This problem incurs greater importance as systems shift from static to dynamic memory devices. If the DMA controller operates in well ordered, integral cycles only, then dynamic memories will not present any data integrity problems.

Figure 9 shows a recommended signal sequence for DMA bus acquisition. The request signal REQ begins action by going high. At the next ϕ_2 positive going edge, SREQ is generated and causes the priority output PROUT to go to logic low. The DMA controller then waits for a negative going ϕ_2 edge at which it finds both PHOLD and the priority input PRIN at a high level. When this occurs, QA goes high, causing an open collector driver to pull down PHLDA. Two clock cycles later the controller begins looking for the hold acknowledge line PHLDA to be high. This

delay allows for the PHOLD delay on the CPU card. DMA action is enabled on the ϕ_2 negative going edge when this occurs.

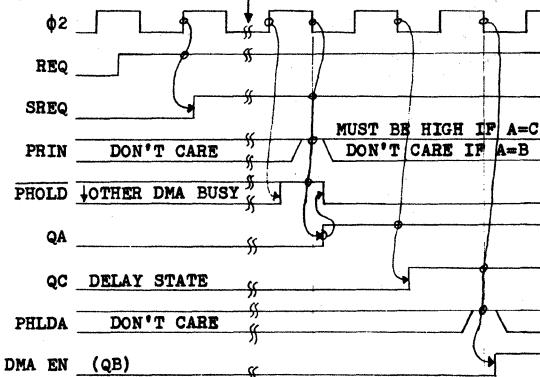
CONTROLLER WAITS UNTIL BUSY DMA (LOWER PRIORITY)
SIGNALS DONE BY RELEASING PHOLD

FIGURE 9. DMA BUS ACQUISITION

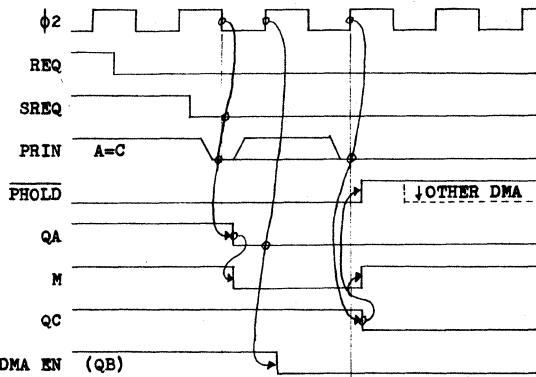
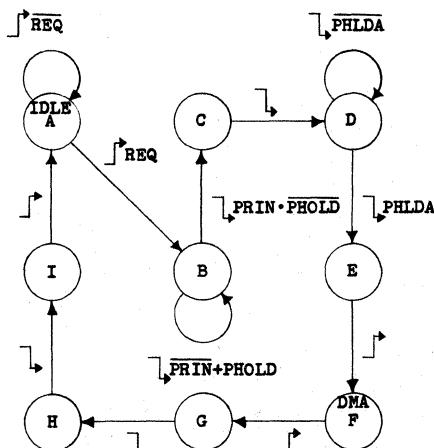


FIGURE 10. DMA BUS RELEASE

Figure 10 shows the corresponding sequence at DMA bus release. The action shown here may be initiated by either REQ going low or, if interruptible, by PRIN going low. Control is given back to the CPU in reverse order, and then PHOLD is released. Another DMA may pick up control as shown.

Figure 11 shows the state diagram for an example single cycle DMA controller. The arrows show the ϕ_2 clock edges on which the indicated state change may take place. Output signals corresponding to the various states are given in Table I. Figure 12 shows one possible hardware implementation for this state diagram. No warranty is given concern-

FIGURE 11. SINGLE CYCLE DMA CONTROLLER
STATE DIAGRAM

ing its suitability or proper functioning, as it was made up for this example. A J-K FF synchronizes the REQ input and controls operation of a "walking ones" shift register state generator. The logic one level propagates to the right, causing the various state transitions. The exclusive OR gates 10-12 control which ϕ_2 clock edge shifts the register, and the 74120 inhibits clock pulses when necessary to prevent state changes. A set of bus drivers controls the S-100 bus lines during DMA times.

STATE	PROUT	PHOLD OUT	DMA EN	PSYNC	PDBIN	PWR	CK PHASE
A	1	1	0	0	0	1	\uparrow
B	0	1	0	0	0	1	\downarrow
C	0	0	0	0	0	1	\downarrow
D	0	0	0	0	0	1	\downarrow
E	0	0	0	0	0	1	\uparrow
F	0	0	1	1	1	1	\uparrow
G	0	0	1	0	1	1	\downarrow
H	0	0	1	0	1	0	\downarrow
I	0	0	1	0	1	1	\uparrow

TABLE I. OUTPUTS FOR FIGURE 11

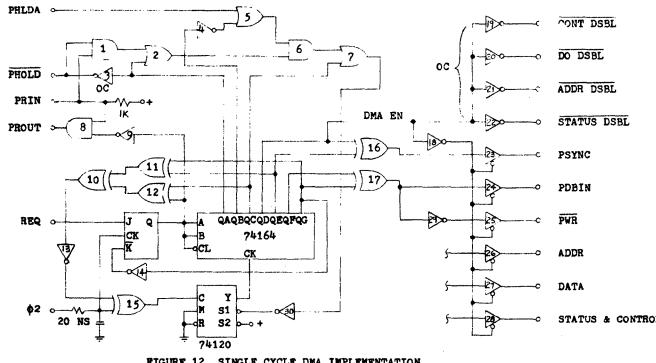


FIGURE 12. SINGLE CYCLE DMA IMPLEMENTATION

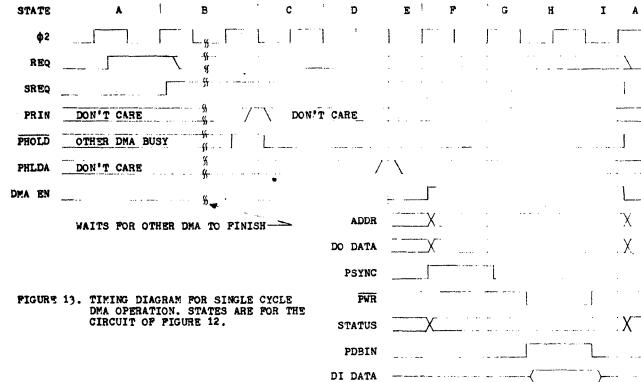


FIGURE 13. TIMING DIAGRAM FOR SINGLE CYCLE DMA OPERATION. STATES ARE FOR THE CIRCUIT OF FIGURE 12.

The timing diagram for this unit appears in Figure 13. It is essentially a concatenation of the operation of Figures 5, 9, and 10.

Figure 14 and 15 show state diagrams for a multiple memory cycle DMA controller that may be operated in either of two modes. It can be either interruptible or non-interruptible by PRIN according to the use at hand. The circuit of Figure 16 shows the implementation selected for this case. This circuit is very similar to the one in the TV Dazzler with two exceptions. First, OR gate 3 was added to permit non-interruptible operation when A connects to B, and second, the provision for a bus float state was omitted. Hindsight has shown that floating the control busses during control transfer is a bad thing, especially for dynamic memories. This can cause erroneous memory cycles and loss of data integrity. The best procedure is to minimize the bus transfer time. The 7495 operates as a bidirectional shift register, shifting on ϕ_2 negative going edges during bus acquisition and only according to the control gating. During bus release it shifts on ϕ_2 positive going edges. The resulting timing diagrams appeared as

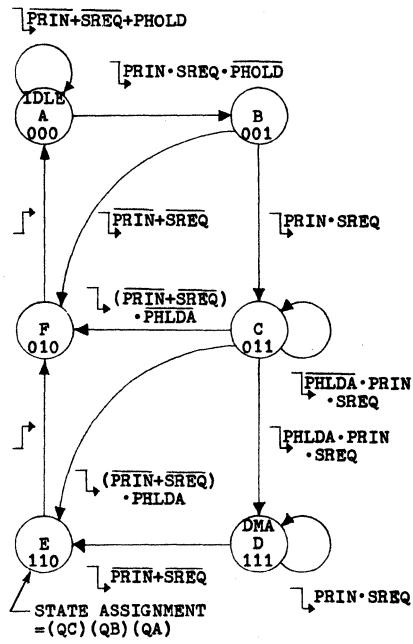


FIGURE 14. INTERRUPTIBLE DMA CONTROLLER STATE DIAGRAM

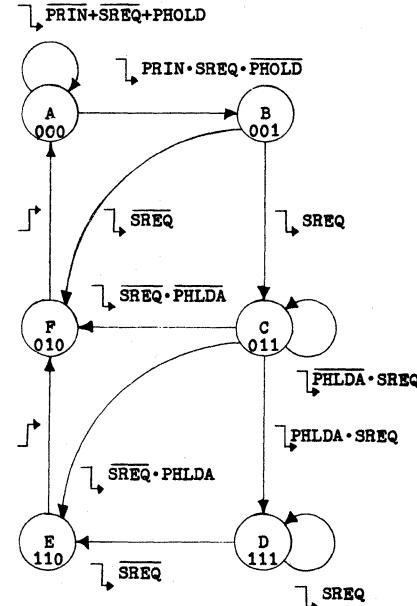
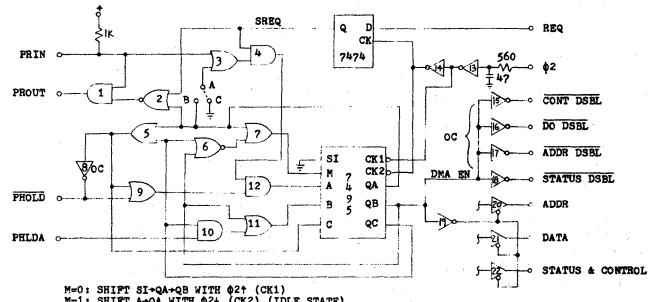


FIGURE 15. NON-INTERRUPTIBLE DMA CONTROLLER STATE DIAGRAM



M=0: SHIFT SI-QA-QB WITH ϕ_2^+ (CK1)
M=1: SHIFT A-QA WITH ϕ_2^+ (CK2) (IDLE STATE)
CPU CHECKS PHOLD AT ϕ_2^+ EDGE
CONNECT A TO B IF DMA PRIORITY INPUT IS NOT TO BE ABLE TO HALT THIS DMA UNIT

FIGURE 16. MULTICYCLE DMA CONTROLLER IMPLEMENTATION

Figures 9 and 10. The actual multiple memory cycle generation is performed in other circuitry elsewhere. Upon tracing the circuit operation, NOR gate 6 may appear to not serve any purpose, but in fact it clears an erroneous state condition which may occur when power is first applied.

In conclusion, techniques for handling the S-100 bus in a compatible manner for multiple DMA systems have been shown. Proper application of these techniques will permit reliable DMA operation with both static and dynamic memory units.

A BIOMEDICAL APPLICATION USING THE S-100 BUS STANDARD

William J. Schenker, M.D.
2086 Essanay Avenue
Walnut Creek CA 94596

Technology usually benefits by standardization. Standards imply a common ground upon which designers, maintenance personnel, and users alike can rely. Without standards, design costs increase as each model is developed on a custom basis. Maintenance and repair costs increase as the stockpile of replacement parts and device-specific test equipment proliferates, training time lengthens and down-times increase. Finally the accumulated impact of these added costs and delays are felt by the end-user, or purchaser.

The above considerations apply to the present state of computer technology. Much time and money is spent designing custom interfaces between computers and peripheral equipment. Trouble-shooting and repair are unnecessarily complicated by vendor variations. More than a few scientists who are computer endusers can recall occasions of their own project's delay because of these factors.

An opportunity to examine more closely the advantages of standardization presented itself in designing the interface of an 8080 microprocessor system to a Coulter model S blood cell counter. Our Coulter counter takes a sample of blood, analyzes it and prints out on standard IBM cards the numbers of red and white blood cells, the hemoglobin level, and four other parameters of clinical importance. It does up to 500 tests a day. Prior to use of our interface, the lab technologist was responsible for riffling through the deck of cards at the end of the day, looking for any abnormal value and then notifying the patient's doctor in such an event. To obtain more timely and reliable detection of abnormalities, we developed a microcomputer module which activates a buzzer and panel light whenever an abnormal specimen passes through the machine. Figure 1 shows the lab technologist at work on the Coulter S blood cell counter. The small box in the upper right is our S-100 module.



Initially a high-quality industrial grade 8080 system was used. (We obtained loan of the unit through the cooperation of the Lawrence Livermore Laboratory and its technology transfer program.) The basic unit, about \$3000, consisted of a standard

19 inch rack-mount, power, supply, fan, front panel controls consisting of power on/off and reset, and a 30 slot card-cage containing the following PC or printed circuit cards: CPU, controller, decoder, two 1K memories, paper tape reader and teletype interface. While performance of this system in our prototype was extremely reliable, several disadvantages became apparent. Their significance may be better appreciated by the following 3 fold analysis, which formed the basis for our choice of another 8080 system when designing the second version of the Coulter interface.

Design

One key to an effective computer bus architecture is redundancy. A computer bus is physically configured as rows of PC board (or card) connectors arrayed in parallel:

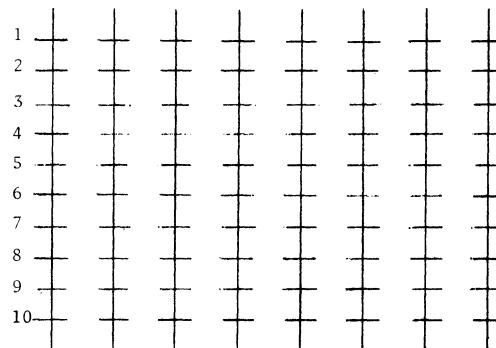


FIGURE 2

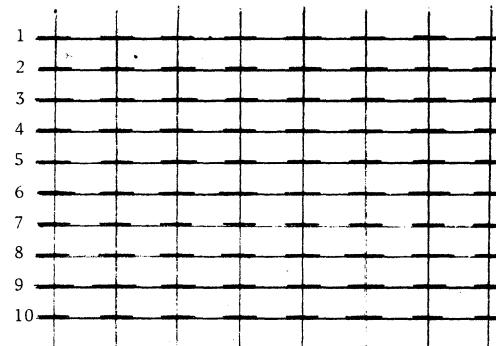


FIGURE 3

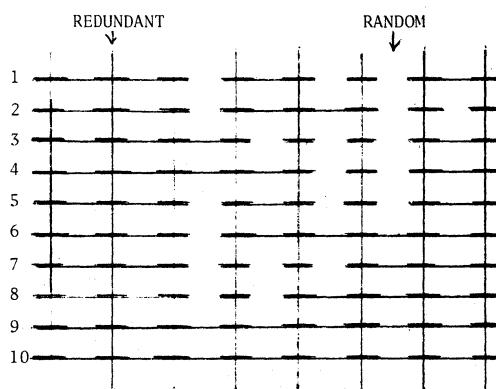


FIGURE 4

The significance of the redundant structure is considerable. It means no matter which connector you plug a board into, there will be no compatibility problem. It means each board's pin # 18 for example will "see" the same signal, for illustration purposes, say, the "input strobe" signal. The straight-forwardness of this architecture makes it very attractive to the engineer/designer. For he knows that no matter how complex his PC board designs are, at least he does not have to re-define his pin assignments. This convenience, multiplied by 50-75-100 pins cuts into design time to a significant degree, illustrating immediately the value of standardization. And, obvious as this advantage is, it has a significant effect on another factor.

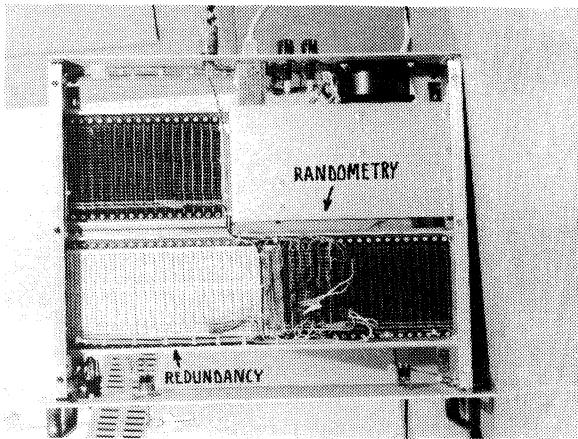
Availability

A bus architecture and signal discipline that is attractive and easy to use from a designer's point of view will be one more likely to be embraced in new designs. In other words an efficient, convenient design tends to be copied by other vendors, or to be "cross-vendored", thus becoming even more available in the market place. If this availability increases further, beyond a critical point, then its availability alone, regardless of its intrinsic worth, becomes a second important factor in the choice of a computer system. This then leads directly to the third and last major factor.

Cost

A product in greater supply will tend to be lower in price. Added to this is the cost savings resulting from quicker, easier designs. An additional cost savings is generated later by the ease and speed with which trouble-shooting personnel can approach a redundant bus structure. (They too can rely on pin # 18.) The resourceful technician or engineer can in addition devise test equipment to take advantage of the universality of such a bus architecture - one instrument can connect to any PC card or any bus connector to exercise and debug it. Further, increased product availability decreases cost in a subtle yet palpable manner by raising the level of expertise regarding the product line both at the vendor/jobber/supplier level and in the general technical community. It is easier to get the answer to an obscure problem with less delay.

And now, measuring up to the foregoing analysis, our first 8080 system shows the following characteristics:



In addition, the vendor has only one sales office in northern California, the system is not cross-vendored, and because it is not widely used, there is no ready reservoir of expertise we could tap locally. Finally, the cost of the basic system, \$3000, was a consideration in our decision.

We turned then to an 8080 system whose bus structure and signal discipline are cross-vendored by a half dozen firms, with more than 15 additional manufacturers offering interface boards that are bus compatible. This is the S-100 bus system. The system and its compatible products are readily available at over 15 retail suppliers in northern California alone, with about 200 to 250 others scattered throughout the country. Using these products, systems may be designed from a minimal configuration costing about \$300 in kit form, to a \$10-to-\$12,000 factory assembled and tested development system. The latter consists of: 65K of main memory, full front panel, dual floppy discs and cassette memory, PROM programmer, CRT, printer, B&W and color graphics, analog converters, and a variety of soft-

ware including effective operating systems and higher level languages. Also, each PC board is standardized to have its own on-board voltage regulators, considerably simplifying the bus power supply design. Of primary importance, however, is its bus architecture. Provided with 100 pins, hence its name, all pins are redundant, but 16 of them are uncommitted, giving flexibility for local designs, while still remaining within the standard bus discipline.

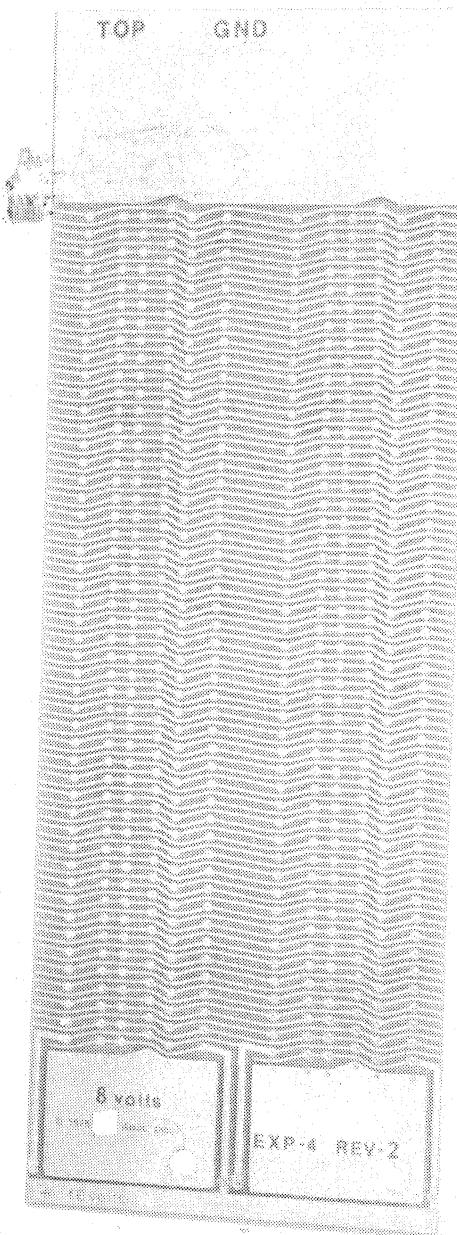
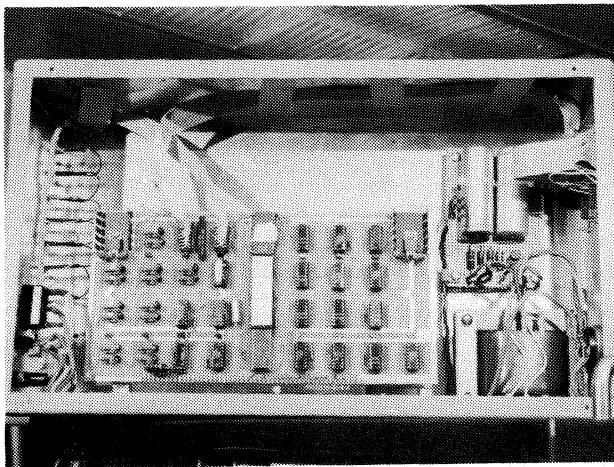
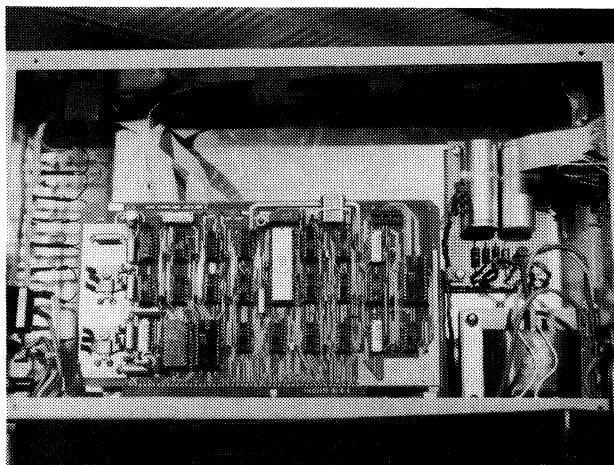
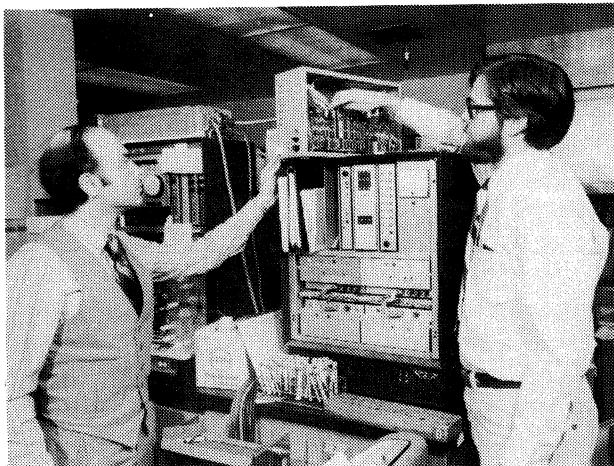


Figure 6 shows a shortened version of the full S-100 bus. It has provision for 4 connectors. Imagine if you will this board extended laterally to handle up to 22 connectors, and you will have the commonly-used version. This bus board, or motherboard as it is known to the engineer, shows total redundancy along the full 22-connector length. This is manifested by the parallel configuration of the printed circuit pattern extending horizontally.

Figure 7 shows our S-100 module in the clinical laboratory with Dr. Terdiman on the left and Mr. Bob Tuttle our design engineer on the right. Figures 8 and 9 respectively show the MPU board and our wire-wrapped custom interface board.

This bus system is now so widely used that expertise in the system is readily available from many vendors and retailers. An additional aspect of S-100 logistics is that, because the bus structure of all systems is identical, a design developed as a stand-alone system may be fully exercised and debugged by merely plugging it into a development system.



And now a balanced description of the S-100 system includes its disadvantages. Many of the early products, some still being marketed, were poorly engineered. Some manufacturers' recent products leave something to be desired either in design, operation, or because of significant overpricing. In this regard, aggressive advertising by these manufacturers would make the facts seem otherwise. There are also a few manufacturers who did not heed the bus discipline and consequently have marketed boards with compatibility problems. Many products lack adequate documentation. The lack of bi-directional data lines on the bus is considered a weakness by some. The motherboard needs re-designing to decrease noise and crosstalk problems when more than 10 boards are plugged in. (As an aside

with the appearance recently of at least two new motherboards this problem may have been solved.) Finally, the system is not designed to operate in a highly hostile environment such as one would encounter in heavy industrial, military, or space applications.

On the whole, though, the quality is good and generally improving, especially as knowledgeable buyers accustomed to high performance enter the market. In this connection you may have noticed I have avoided mention of specific vendors. Those of you who plan to dig deeper, however, will need to know brand names and specific products to fill the void. My considered advice is two-fold. 1. Visit an S-100 retailer in your area. Generally he will provide unbiased advice, since he deals in a cross-vendored market. 2. With the following proviso! Check the dealer's product racks. If you see mostly one brand name with only a scattering (or absence) of others, you've chanced on one of the somewhat thinly-disguised factory-franchised dealers. Caveat emptor!

In summary, then, this bus standard, the S-100, appears to be a cost-effective alternative to vendor specific products presently available in the small systems computer market.

William J. Schenker, M.D. is a practicing physician and a medical information systems scientist working with microcomputer technology. He co-organized a symposium in November, 1976 on the commercial viability of the S-100 bus system. He has at home an S-100 computer with CRT, electric typewriter, paper tape and cassette peripherals.

VACUUM

VARIABLE ARCHITECTURE COMPUTING MACHINE

by Tom Pittman and Bob Davis

© 1977 by Bob Davis and Tom Pittman

Box 23189, San Jose CA 95153

This paper is a description of the Variable Architecture Computing Machine (VACuM) developed and marketed by Davis Laboratories. We will also attempt to discuss some of the design objectives in its implementation, and how they affected trade-off decisions.

The intended application for VACuuM is for experimental architecture testing and the emulation of several popular mini and mainframe computers. It is expected to sell primarily to educators and hobbyists, as well as those with access to existing software. The major design objective was the ease of implementing a variety of popular computer emulators, while maintaining a reasonably low cost and high execution speed. It is fundamentally a micro-programmed computer, with as many of the archi-

tectural decisions left to the firmware as possible. The use of existing LSI bipolar components and the objective of reasonably high execution speed rules out making the microengine completely transparent; we will notice this in some of the trade-off decisions made in the design. Of course the microcode can implement virtually any conceivable machine, but some architectures are more easily achieved than others.

A general overview of VACuuM shows a system with a 24-bit address bus, an 18-bit memory data bus, and an internal processor word length of 8, 16, 32, or 36 bits (see Figure 1). The microprogram memory is 4096 64-bit words and is writable by the microprogram. The microstore RAM is overlaid by a 512 x 64 shadow PROM for bootstrapping the micro-

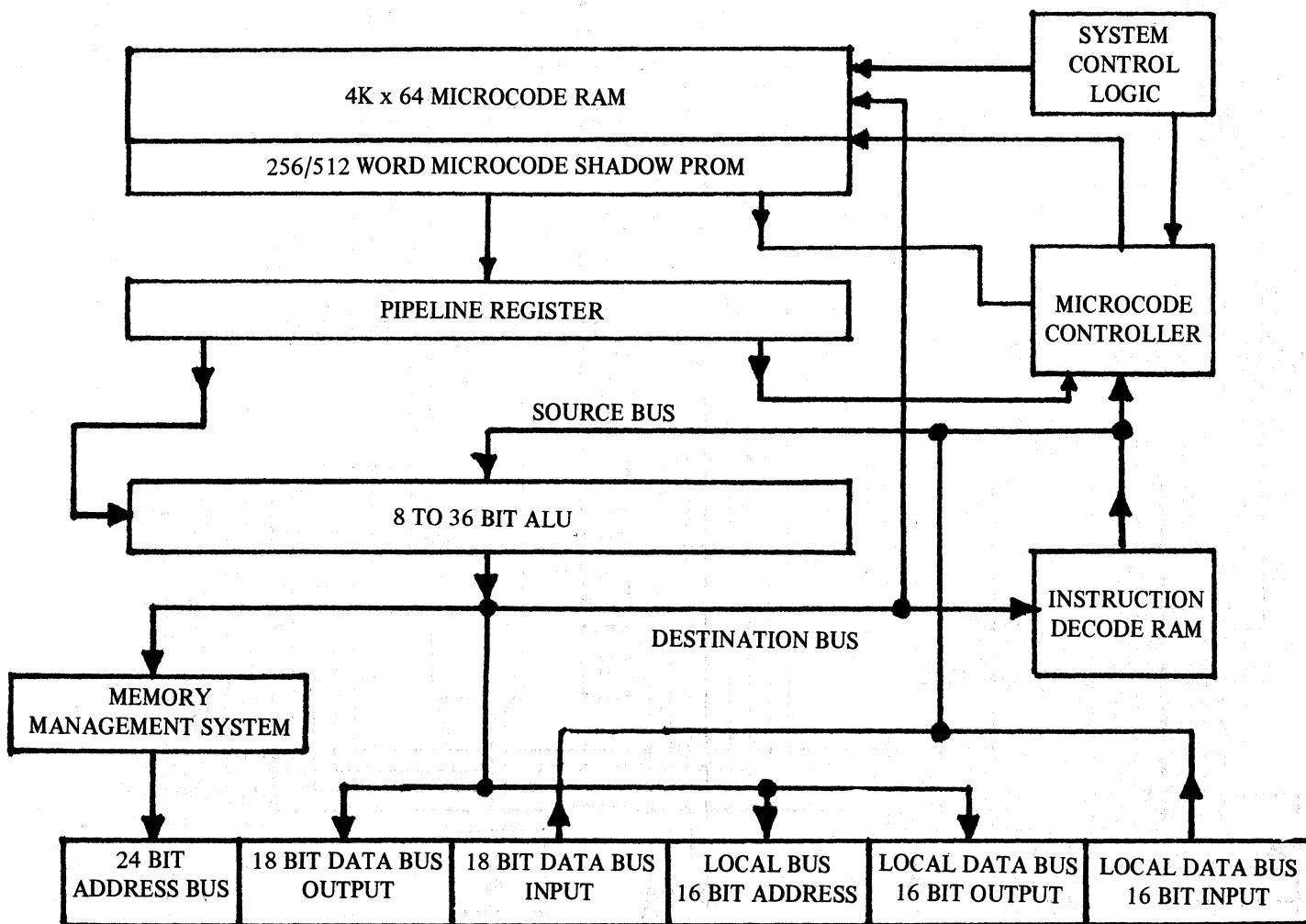


FIGURE 1. VACuum -- A Variable Architecture Computing Machine

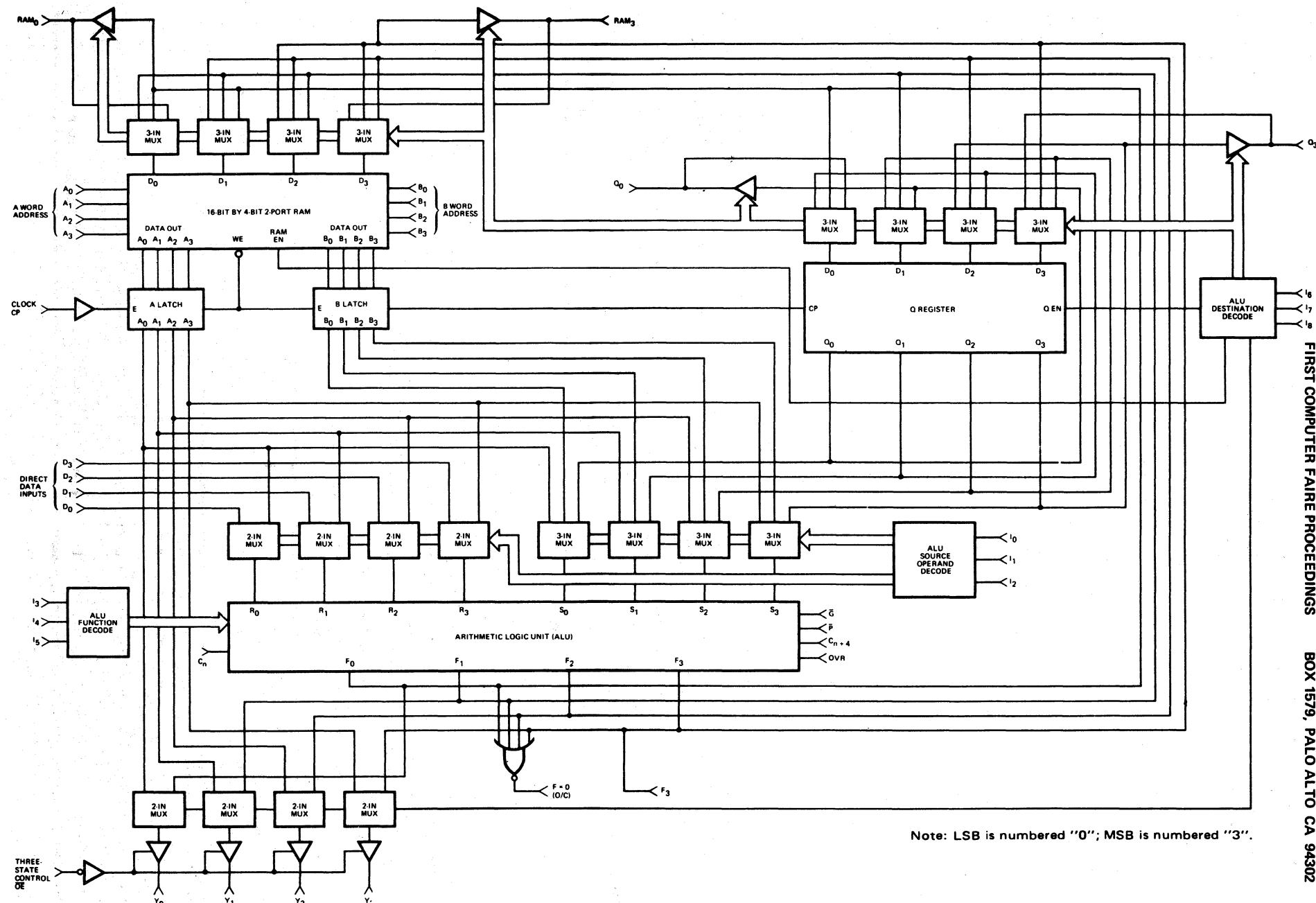


FIGURE 2. 2901 BLOCK DIAGRAM

FIGURE 3. 2901 INSTRUCTION DECODE

MICRO CODE			ALU SOURCE OPERANDS	
I_2	I_1	I_0	Octal Code	R S
L	L	L	0	A Q
L	L	H	1	A B
L	H	L	2	O Q
L	H	H	3	O B
H	L	L	4	O A
H	L	H	5	D A
H	H	L	6	D Q
H	H	H	7	D O

MICRO CODE			ALU Function	Symbol
I_5	I_4	I_3	Octal Code	
L	L	L	0	R Plus S
L	L	H	1	S Minus R
L	H	L	2	R Minus S
L	H	H	3	R OR S
H	L	L	4	R AND S
H	L	H	5	\bar{R} AND S
H	H	L	6	R EX-OR S
H	H	H	7	R EX-NOR S

ALU Source Operand Control.

ALU Function Control.

MICRO CODE			RAM FUNCTION		Q-REG. FUNCTION		Y OUTPUT	RAM SHIFTER		Q SHIFTER	
I_8	I_7	I_6	Octal Code	Shift	Load	Shift	Load	RAM ₀	RAM ₃	Q ₀	Q ₃
L	L	L	0	X	NONE	NONE	F \rightarrow Q	F	X	X	X
L	L	H	1	X	NONE	X	NONE	F	X	X	X
L	H	L	2	NONE	F \rightarrow B	X	NONE	A	X	X	X
L	H	H	3	NONE	F \rightarrow B	X	NONE	F	X	X	X
H	L	L	4	DOWN	F/2 \rightarrow B	DOWN	Q/2 \rightarrow Q	F	F ₀	IN ₃	Q ₀
H	L	H	5	DOWN	F/2 \rightarrow B	X	NONE	F	F ₀	IN ₃	Q ₀
H	H	L	6	UP	2F \rightarrow B	UP	2Q \rightarrow Q	F	IN ₀	F ₃	IN ₀
H	H	H	7	UP	2F \rightarrow B	X	NONE	F	IN ₀	F ₃	X
											Q ₃

X= Don't care. Electrically, the shift pin is a TTL input internally connected to a three-state output which is in the high-impedance state.

B= Register Addressed by B inputs.

Up is toward MSB, Down is toward LSB.

ALU Destination Control.

I_{210} OCTAL	0	1	2	3	4	5	6	7
ALU Source	A, Q	A, B	O, Q	O, B	O, A	D, A	D, Q	D, O
$C_n = L$ R Plus S $C_n = H$	A+Q A+Q+1	A+B A+B+1	Q Q+1	B B+1	A A+1	D+A D+A+1	D+Q D+Q+1	D D+1
$C_n = L$ S Minus R $C_n = H$	Q-A-1 Q-A	B-A B-A	Q Q	B B	A-1 A	A-D-1 A-D	Q-D-1 Q-D	-D-1 -D
$C_n = L$ R Minus S $C_n = H$	A-Q-1 A-Q	A-B-1 A-B	-Q-1 -Q	-B-1 -B	-A-1 -A	D-A-1 D-A	D-Q-1 D-Q	D-1 D
3 R ORS	A \vee Q	A \vee B	Q	B	A	D \vee A	D \vee Q	D
4 R ANDS	A \wedge Q	A \wedge B	0	0	0	D \wedge A	D \wedge Q	0
5 \bar{R} ANDS	$\bar{A} \wedge Q$	$\bar{A} \wedge B$	Q	B	A	$\bar{D} \wedge A$	$\bar{D} \wedge Q$	0
6 R EX-ORS	A \forall Q	A \forall B	Q	B	A	D \forall A	D \forall Q	D
7 R EX-NORS	$\bar{A} \forall Q$	$\bar{A} \forall B$	\bar{Q}	\bar{B}	\bar{A}	$\bar{D} \forall A$	$\bar{D} \forall Q$	\bar{D}

+ = Plus; - = Minus; V = OR; \wedge = AND; \forall = EX-OR

Source Operand and ALU Function Matrix.

program memory. Dynamic Address Translation (DAT) is provided for main memory in blocks of 1024 words. There are 17 addressable registers in the ALU and numerous other registers, memories, and controls which we shall discuss in detail later. The CPU is designed primarily in low-power Schottky, although most components have been selected for the availability of full-speed Schottky equivalents. The memories are static NMOS, but they may be readily replaced by high-speed equivalents as they become available, or bypassed entirely.

ALU

The Arithmetic-Logic section of the processor is organized around the 2901 bit-slice components (see Figure 2). These are 4 bits wide and include 16 general registers, a "Q" register used in double-length operations, an 8-function ALU, and an independent shifter.

Nine bits from the microinstruction word define the ALU operation. Three bits select one of eight pairs of source operands, as shown in Figure 3. Three bits select one of eight arithmetic or logical functions, and the remaining three bits determine the destination of the ALU result and control the shifter. Because each of these three fields are encoded, there are very few superfluous or useless instructions for the 2901.

The shifter is supplemented by a set of selectors and gates between the 2901s to implement rotates through or around the shift link, logical shifts with zero fill, and double-length operations as shown in Figure 4. Additional gating supports the boundary conditions for each of the four standard word lengths, which may be dynamically selected in the microcode. For non-standard wordlengths, either the boundary conditions may be simulated in the firmware, or some of the 2901s may be omitted and replaced by jumpers, but of course this only works for word lengths which are multiples of 4, the 2901 bit size. Actually the wordlength is a problem only in the handling of shifts which link the ALU and the Q register into a double-length register.

A two-level look-ahead carry generator is connected to the 2901s to minimize the propagation delays in the adder.

The four standard wordlengths were selected for their similarity to popular existing mainframes: 36 bits is needed for emulation of the 7094, the 1108, and the DEC-10; 32 bits is required for the 360/370 and its imitations: 8- and 16-bit word lengths are useful in partial word operations in the 360/370, as well as in the emulation of a host of popular minicomputers and microprocessors. The wordlength selections are applied to several functions in addition to the end-around shift links. When the selected wordlength is less than the maximum 36 bits, any test for sign (most significant bit), carry out,

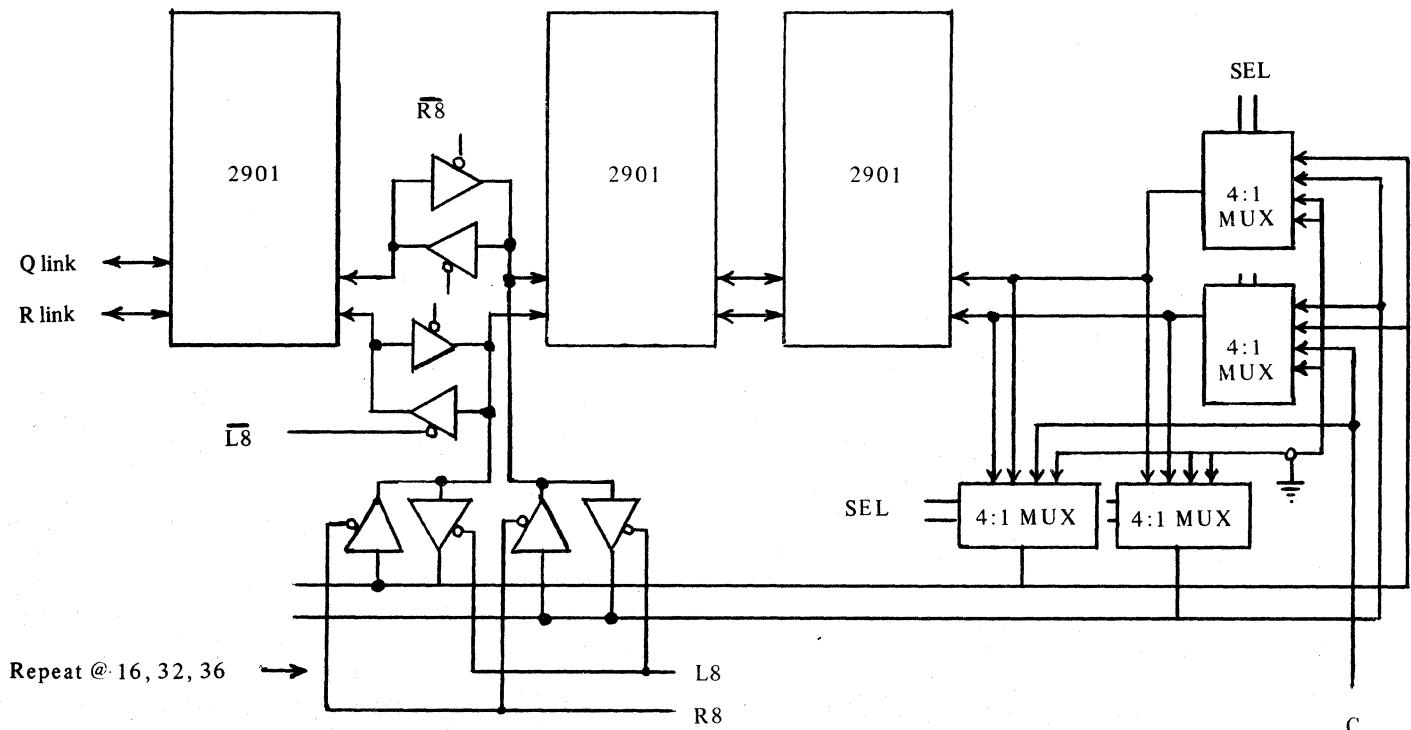


FIGURE 4. ALU SHIFT LOGIC

overflow, or zero measures only that part of the ALU defined to be part of the word length and ignores the excess.

INTERNAL BUSSES

The 36-bit internal busses route the data through the system. These are designated "source" and "destination" in deference to their relation to the 2901s.

The source bus drives the D inputs to the 2901s. It also provides for a computed Jump address in the microprogram, as we shall see in the discussion of the 2909 sequencer. The source bus is driven by one of the following:

- Memory Data input latches.
- I/O data Input.
- DAT Mapping RAM read data.
- Decode/scratch RAM read data.
- Decode/scratch RAM data register.
- Microinstruction word constant field.
- Byte Swapper.
- Microinstruction memory read data registers.
- Control bit words.
- Interrupt priority encoder.

Note that only the memory data is the full bus width, and all others drive only the lower 16 bits of the bus or less. A more complete description of the nature of these source registers will be given later.

The Destination bus is generally driven by the outputs of the 2901s. It feeds the inputs to the following registers or functions:

- Main Memory Address register.
- DAT Mapping RAM Data register.
- Main Memory Data register.
- Decode/scratch RAM data register.
- Decode/scratch RAM address.
- I/O Output Data register.
- I/O Address register.
- Byte Swapper.
- Microinstruction memory write data registers.
- Microinstruction address register.
- Register Selection Latches.

Note here also that the bus may be wider than some of these destinations; the excess bits are ignored. Some of the destinations do not latch the bus data, but are active only while the data is present. These include the byte swapper, and the address of the Decode/scratch RAM.

MAIN MEMORY

In order to support a 32- or 36-bit word length on an 18-bit main memory data bus, a multiplexing scheme is used to pack two memory words into one processor word, as shown in Figure 5. The bits were

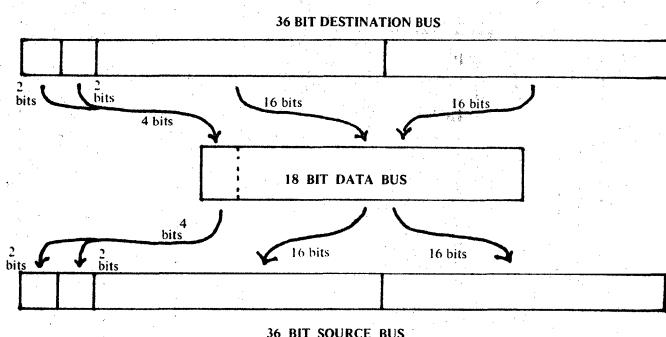


FIGURE 5. MEMORY DATA MULTIPLEXOR

scrambled slightly to simplify the conversion between 32- and 36-bit packing: the low 16 bits of each memory word goes to either half of the low 32 bits of the internal bus, and the other two bits on each word are packed into the high four bits. Thus if 36 bits are not required, all of the components supporting this word length may be conveniently deleted from the circuit. Two control signals are provided to latch from the memory data bus either the low half or the high half.

Similarly, for writing a long word back out to the memory bus, separate controls select either the upper or lower half.

The memory address is developed from a 24-bit address register which is loaded from the destination bus. The low ten bits of this register pass directly to the external address bus when enabled (see Figure 6).

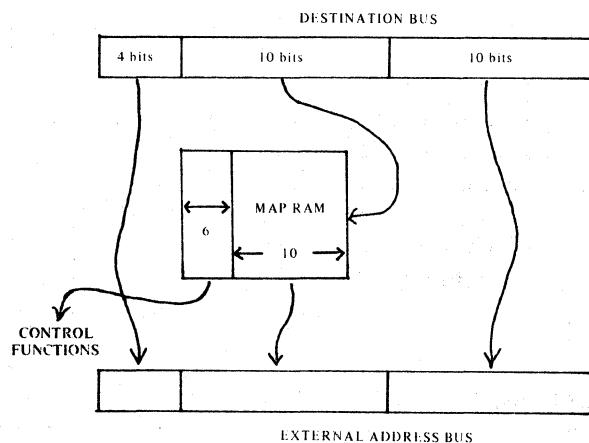


FIGURE 6. MEMORY MANAGEMENT

The next ten bits serve as the address to the Mapping RAM and the RAM outputs provide the corresponding ten bits of the external address. The upper four bits pass directly to the external address bus. Within a block of 1,048,576 memory locations (20-bit address space) 1024 1K pages may be defined. Note that these are 18-bit words, not bytes. Users inclined to access the maximum 32 megabytes are obliged to

add some extra firmware controls to distinguish banks, since the mapping RAM ignores the high four bits. It turns out that the position of the pagination boundary is identical to that of the DEC-10 if it is assumed that each word of emulated DEC-10 memory corresponds to two words of VACuuM main memory.

The Mapping RAM is 16 bits wide, divided into two parts: ten bits define the actual page address, and six bits serve the virtual memory control functions. Five of the six bits have particular hardware significance associated with memory management; the sixth bit is unassigned. The five functions are:

Write Protect. A memory Fault interrupt occurs if this bit is set and a write is attempted into this memory page.

Privileged. A Memory Fault interrupt occurs if both this bit and the User Mode flag bit are set in any access to this memory page.

Absent. A Memory Fault occurs on any access to this page if this bit is set. This bit would be set for all memory pages which do not have instances in real main memory but only reside on mass storage.

Unreferenced. This bit is unconditionally set to 1 on every access to this memory page. It is intended that this bit would be used in implementing LRU swapping algorithms.

Dirty. This bit is set to 1 on every Write access to this memory page. Of course a memory page which has not been written into need not be recopied to the mass storage in swapping.

A memory fault occurs at the time the address is stored into the address register. This sets a condition flag which may be tested in the microcode to abort the memory access cycle before any actual write (in this case of write protect), or at least before any use is made of the read data. The memory faults are also channelled into the priority interrupt encoder.

The memory paging RAM may also be bypassed entirely without modification by disabling it and using the data register directly.

INTERRUPTS

VACuuM is in complete control at all times, and all interrupts are implemented in the firmware. Such normally hardware-related functions as power-fail, Halt, and Reset are actually funnelled into the interrupt priority encoder, where they may be served by the microcode, to emulate the desired function. The individual interrupt conditions may be masked (disabled). Interrupt service consists of detecting the request, either through an indexed jump with OR (in the 2909s), or by means of a conditional branch, then jumping to the service routine.

Other causes of an interrupt are the internal timer, memory fault, the decoding the CPU's own I/O address, and ten unassigned inputs.

TEST CONDITIONS

There are fifteen conditions which may be tested for conditional branches in the microcode. These are the latched Carry flag from the selected wordlength, zero, sign, and overflow from the selected wordlength, a processed form of the G (generate) and P (propagate) signals which drive the look-ahead carry unit to signal positive or negative result, a collected memory fault condition, the individual memory faults, and interrupt pending.

MICROINSTRUCTION WORD

The 64-bit microinstruction word is divided into several distinct fields, each controlling some particular operation or function of the microengine. It is a horizontal microprogram format, but most of the fields are encoded to minimize the number of duplicated or useless instruction codes. Only one field of the microinstruction serves multiple functions, and all other fields are directly connected to the respective control logic in the processor.

The significance of the various fields is as follows:

(16 bits) *Jump Address/Constant/Strobes.*

This field is defined in one of three modes depending on the use of the instruction. Where they are required in the ALU, constants up to 16 bits may be included in the microinstruction word. These are intended for mask generation, fixed main memory address assignment (as in interrupt vectors), or other needs of the microprogram. If a constant is not needed, the same field doubles for a Jump address to the microprogram sequencer. This is only 12 bits, with the remaining four bits in the field being used to select a second ALU register. Alternatively the 12 bits are decoded to select one of 32 auxiliary strobes and one of the 64 state latches with its data. The strobes supply the additional selection for the source and destination busses (beyond the three bits each in another part of the microinstruction word), as well as the pulses to latch data off the external busses or to set up temporary internal conditions. The State Latches implement the selection of the word length, the privileged access mode for the memory management hardware, and other infrequently changed state conditions, as well as providing the static controls to drive the external busses, select memory cycle controls, etc. It is presumed here that the

microengine is sufficiently faster than the external memory that these controls will be required to remain static for several microcycles, during which time the processor may be off performing unrelated operations.

(8 bits) *ALU Register Select* (See Figure 7)
 This field, together with the optional four bits overlaid by the constant, defines the selection of the register file in the 2901 for each of the two inputs to the ALU section of it. In many cases (including most cases where a constant is used) only one register need be designated, thus minimizing the impact of the field redefinition on the microcode. The two register selection inputs of 2901s are each driven by four-input multiplexors, which each in turn select one of the two register selections in the microword or one of two four-bit latches loaded from the destination bus during macroinstruction decode time. Thus each side of the ALU is fully definable, either fixed in the microprogram, or dynamically selected as a function of the data processed by the ALU (at instruction decode or otherwise).

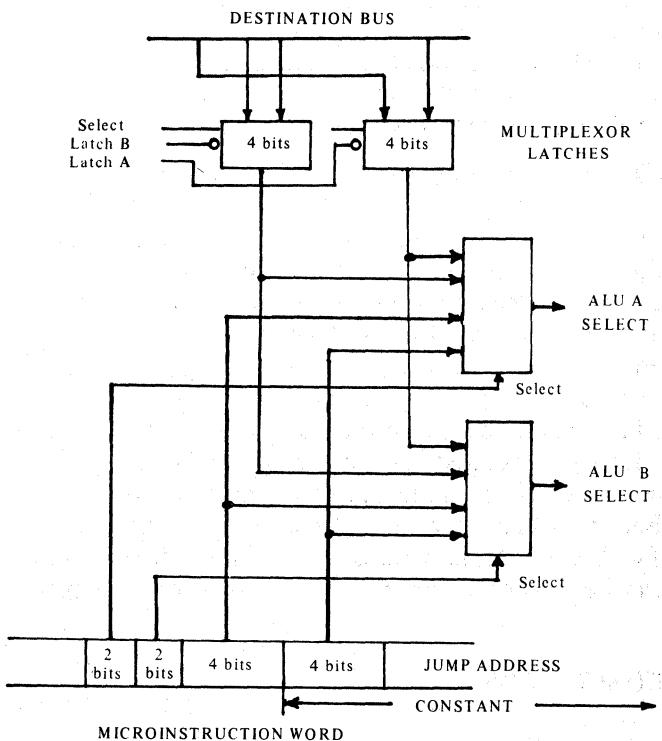


FIGURE 7. ALU REGISTER SELECTION

(9 bits) 2901 Instruction

These bits were described earlier.

(3 bits) Source Bus Selection

These three bits select one of seven source registers or memories to drive the Source Bus to the 2901.

Additional sources are selected by the auxiliary strobes overlaid by the Constant/Jump Address.

(3 bits) Destination Bus Selection

These three bits select one of seven Destination registers or memories to receive the contents of the Destination bus driven by the 2901s. Additional destinations are selected by the auxiliary strobes overlaid by the Constant/Jump Address.

(7 bits) Carry Control

The 2901 adders require the definition of the carry into the least significant bit. The carry out of the most significant bit may be latched in the conditions register (see Condition Control, below), but it is also preserved for subsequent re-use in multiple-precision operations in either its true or complement form. The carry-in may be selected from either the true or complement of the saved carry, or a constant one or zero. All nine of the four-bit carry outputs from the 2901s are made available to facilitate packed decimal arithmetic, though they must be examined individually.

(4 bits) Shift Control

Part of the nine bits of the 2901 instruction define the operation of the shifter (left/right/none, long/short), and this field determines the conditions at the word ends. Specifically, at each end of both the main register shifter and the Q shifter is a multiplexor to select from zero, the carry flag, and the opposite end of either the main shifter or the Q shifter. Thus are possible both single and double-length shifts in either direction, either as logical, or as a circular shift through or around the carry.

(6 bits) Condition control

Besides the four bits needed to select one of the sixteen conditions, this field contains a bit to select the true or complement of the condition, and one to latch a new set of conditions or not.

(4 bits) Sequence Control

This field selects one of sixteen sequence control functions for the 2909 microprogram sequencer. These are discussed more fully later.

(4 bits) Miscellaneous strobes

Included here are the two strobes for latching the register selections off the Destination Bus, and an enable for the auxiliary strobe and state latch field.

MICROPROGRAM SEQUENCER

The sequence of microprogram instructions is controlled by the 2909, a four-bit slice having a program counter/incrementer, a four-deep stack for subroutine

return addresses, and selection logic for choosing between either of them or the direct or latched inputs, as illustrated in Figure 8. The selection of the 2909 function is controlled by the outputs of a 32 x 8 PROM, which encodes the 4-bit sequence field of the microinstruction word in the microword together with the condition tester output (see Figures 9 and 10). The following are the available sequence operations:

- Jump to 000
- Next instruction
- Conditional Jump
- Conditional Indexed Jump
- Conditional Subroutine Call
- Conditional Indexed Call
- Conditional Return
- Conditional Loop
- Start Loop
- Conditional Exit Jump
- Loop
- Conditional Repeat
- Return
- Call
- Jump

Indexed Jump

These operations define nearly every conceivable combination of sequence allowed by the 2909 for complete flexibility in microprogram sequence control. Effectively, the address of the next instruction may come from any of the following sources:

- Power-up (location FFF)
- (location 000)
- OR of Priority encoder (4 bits)
- Source Bus, which may be driven by the Decode RAM
- Microinstruction Register
- Next instruction in sequence.

FIRMWARE

Particular care was taken in the design of the microengine to facilitate the most likely kinds of microprograms. The following is a description of some of the programming considerations:

Instruction Decode

The Decode/Scratch RAM in the processor is intended to rapidly decode up to ten bits of an instruction opcode, converting it into a jump address in the microprogram. While it may have been faster to place this RAM directly between the memory data input register and the direct inputs of the 2909, it seemed better to sacrifice a cycle or so to allow the ALU the opportunity to mask off unwanted bits, so that other parts of the RAM may be used for other purposes. This also permits the Opcode to be positioned,

if need be, by shifting. If any of the 2901 registers are to be addressable by the macroprogram, it is necessary to isolate the register selection bits of the Macro instruction, and catch them in one or both of the register selection latches.

Signed Arithmetic

While it is easiest in this processor to implement Two's Complement arithmetic, Sign-magnitude and One's Complement may be effected with a minimum of difficulty. Sign-magnitude arithmetic requires that the signs be analysed and stripped off, then restored to the result. One's Complement addition or subtraction requires two microcycles: one to do the addition or subtraction, and a second to add in the end-around carry. Sign extension in shifts also requires two microcycles: one to set the Carry with the most significant bit, a second to shift it into the data word.

Decimal Arithmetic

The cost of providing full decimal adders or automatic decimal correction in the binary addition was prohibitive. Instead the intermediate carries are made available and the microprogram must sequence through the digits of the decimal word, testing each carry out while analysing the bits of the sum, and making the correction one digit at a time.

Multiplication and Division

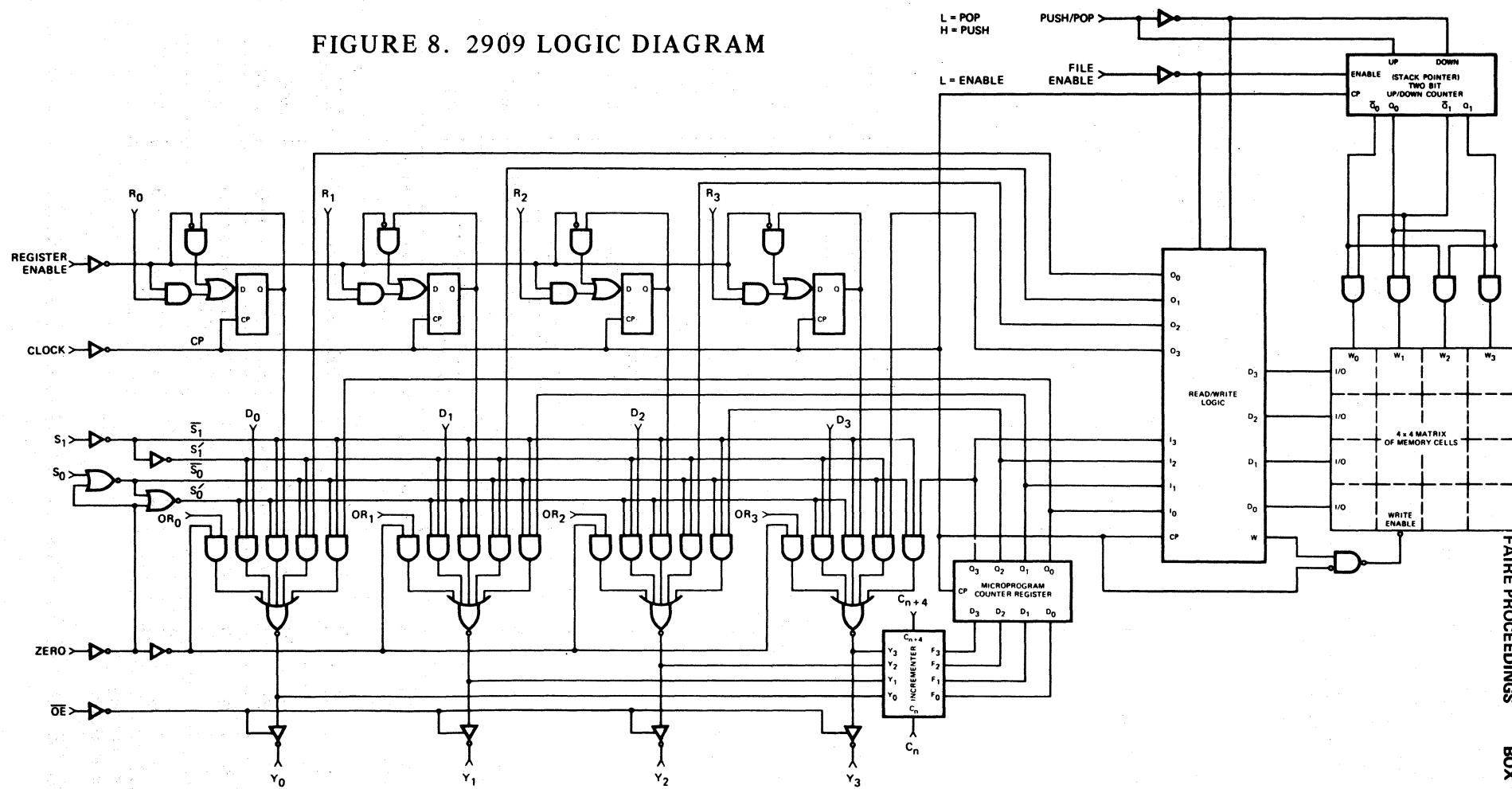
The combination of the Q register with the main register file in the 2901 makes these iterative procedures quite simple. The basic loop of a multiply consists of two microinstructions: one to do the add and/or shift, the other to decrement the loop counter and iterate. This was preferred over a single microinstruction capability which would have required additional hardware to maintain the iteration counter with its associated test for zero. Division similarly requires two or three cycles in the iteration. If Booth's Algorithm is to be used to accomodate Two's Complement signs, one or two extra cycles can be expected for the analysis of the extra bits.

CONCLUSION

It was the purpose of this paper to show both the feasibility and some of the implementation details of the Variable Architecture Computing Machine. It promises to be an exciting new capability in small computers.

This paper is a revision of that presented by the author with Bob Davis of Davis Laboratories at the First West Coast Computer Faire on April 17, 1977. At this writing the hardware is in early prototype and not all the bugs have been worked out.

FIGURE 8. 2909 LOGIC DIAGRAM



Note: R_i and D_i connected together on Am2911 and OR_i removed.

FIGURE 9. 2909 CONTROL

Address Selection			Output Control					
OCTAL	S ₁	S ₀	SOURCE FOR Y OUTPUTS	SYMBOL	OR _i	ZERO	OE	Y _i
0	L	L	Microprogram Counter	μPC	X	X	H	Z
1	L	H	Register	REG	X	L	L	L
2	H	L	Push-Pop stack	STK0	H	H	L	H
3	H	H	Direct inputs	D _i	L	H	L	Source selected by S ₀ S ₁

Z = High Impedance

Synchronous Stack Control

$\overline{\text{FE}}$	PUP	PUSH-POP STACK CHANGE
H	X	No change
L	H	Increment stack pointer, then push current PC onto STK0
L	L	Pop stack (decrement stack pointer)

H = High
 L = Low
 X = Don't Care

CYCLE	S ₁ , S ₀ , $\overline{\text{FE}}$, PUP	μPC	REG	STK0	STK1	STK2	STK3	Y _{OUT}	COMMENT	PRINCIPLE USE
N N+1	0 0 0 0 —	J J+1	K K	Ra Rb	Rb Rc	Rc Rd	Rd Ra	J —	Pop Stack	End Loop
N N+1	0 0 0 1 —	J J+1	K K	Ra J	Rb Ra	Rc Rb	Rd Rc	J —	Push μPC	Set-up Loop
N N+1	0 0 1 X —	J J+1	K K	Ra Ra	Rb Rb	Rc Rc	Rd Rd	J —	Continue	Continue
N N+1	0 1 0 0 —	J K+1	K K	Ra Rb	Rb Rc	Rc Rd	Rd Ra	K —	Pop Stack; Use AR for Address	End Loop
N N+1	0 1 0 1 —	J K+1	K K	Ra J	Rb Ra	Rc Rb	Rd Rc	K —	Push $\mu\text{PC};$ Jump to Address in AR	JSR AR
N N+1	0 1 1 X —	J K+1	K K	Ra Ra	Rb Rb	Rc Rc	Rd Rd	K —	Jump to Address in AR	JMP AR
N N+1	1 0 0 0 —	J Ra+1	K K	Ra Rb	Rb Rc	Rc Rd	Rd Ra	Ra —	Jump to Address in STK0; Pop Stack	RTS
N N+1	1 0 0 1 —	J Ra+1	K K	Ra J	Rb Ra	Rc Rb	Rd Rc	Ra —	Jump to Address in STK0; Push μPC	
N N+1	1 0 1 X —	J Ra+1	K K	Ra Ra	Rb Rb	Rc Rc	Rd Rd	Ra —	Jump to Address in STK0	Stack Ref (Loop)
N N+1	1 1 0 0 —	J D+1	K K	Ra Rb	Rb Rc	Rc Rd	Rd Ra	D —	Pop Stack; Jump to Address on D	End Loop
N N+1	1 1 0 1 —	J D+1	K K	Ra J	Rb Ra	Rc Rb	Rd Rc	D —	Jump to Address on D; Push μPC	JSR D
N N+1	1 1 1 X —	J D+1	K K	Ra Ra	Rb Rb	Rc Rc	Rd Rd	D —	Jump to Address on D	JMP D

X = Don't care, 0 = LOW, 1 = HIGH, Assume C_n = HIGH
 Note: STK0 is the location addressed by the stack pointer.

FIGURE 10. NEXT CYCLE REGISTER CONTROL

LARGE SCALE COMPUTERS FOR THE HOBBYIST

David C. Wyland
Raytheon Semiconductor
350 Ellis St.
Mountain View CA 94042

INTRODUCTION

Full scale processors of the IBM 360/370 class will soon be desired by the serious hobbyist and are practical due to the availability of low cost LSI components. Machines of this class will be desired as the natural memory size limits of current microprocessors are reached, and they can be built for a CPU parts cost of less than \$500.00

Current microcomputer memory sizes appear to track those of minicomputers with a 2 year lag. A well developed minicomputer system will have 64 K bytes of memory at this time, with a few large systems having 256 K. A well developed microcomputer system will have 16 K, with a few large systems having 64 K.¹ Because of the driving force of the semiconductor industry, the amount of memory purchasable per dollar increases by a factor of four every two years, and this trend should continue for at least two and possibly three more increments.² This means that the cost of a 16 K memory today will buy 256 K of memory four years from now, and possibly a full megabyte six years from now. Since the range of tasks that a computer can perform is limited by memory size, microcomputer memory sizes will continue to expand in the same manner as in the minicomputer and large computer case. This brings us to a problem: all currently popular microprocessors have a natural memory limit of 64 K bytes. This is because they have 16 bit program counters and address registers. This can be overcome, but it requires complex memory mapping hardware and software to allow the microprocessor to access a larger memory space.

The problem of large memory space access is solved in larger machine designs of the IBM 360 class. The IBM 360 has 32 bit registers and a 24 bit address bus. This allows direct addressing to 29 megabytes and address calculations to 4 gigabytes (as is useful in virtual memory applications using large disks). The IBM 360 is a straightforward example of microprogrammed computer design and lends itself to emulation by the hobbyist using bit slice components and microprogram design techniques. By virtue of its design, the IBM 360 has a unique possibility for interface to conventional microprocessors such as the 8080, using these microprocessors as the I/O Channel controllers. An advantage of emulating the 360 or a subset of it is that there is a wealth of literature available on this machine, and "cross assemblers" are immediately available.

IBM 360 GENERAL CHARACTERISTICS

The IBM 360 lends itself well to emulation by the hobbyist. It is a multi-register, byte oriented design and is similar in many ways to the 8080. The IBM 360 has 16 general purpose 32 bit registers, a 24 bit program counter, and a 64 bit status word which includes the 24 bits of the program counter. Each of the 16 general purpose registers can be used as an accumulator or an index register. The instruction set is rich, including 32 bit multiply and divide, 8, 16, and 32 bit operations, and byte string operations; however, these instructions are all implemented with microprogram software and any appropriate, usable subset can be implemented.

The IBM 360 uses byte addressing, and its instructions have 5 formats as shown in Figure 1. Instructions are two, four, or six bytes long with a single byte for the operation code, similar to the 8080, 6800, and 6502, etc. microprocessors. Data words are 8, 16, or 32 bits long, with

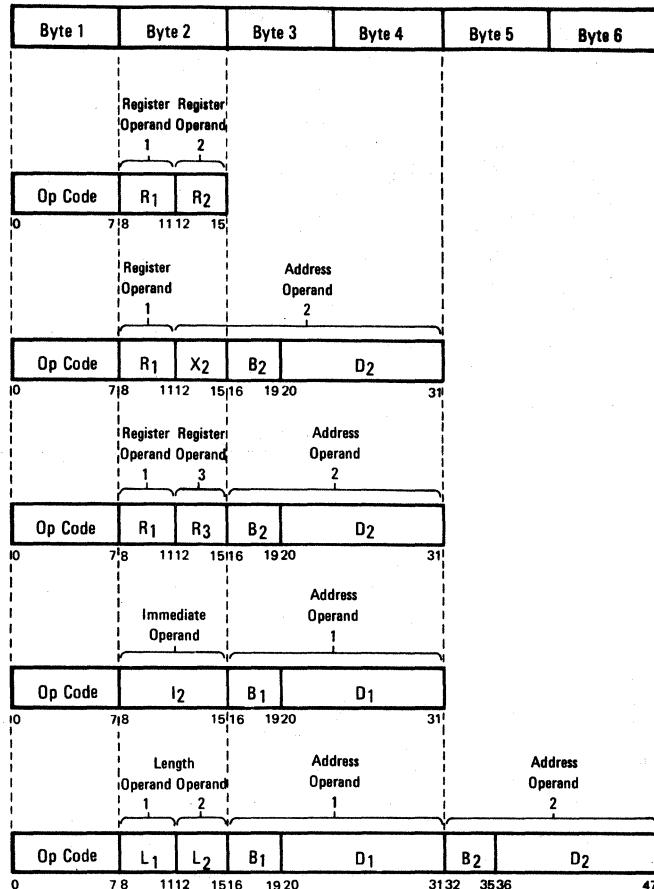


Figure 1. IBM 360 Instruction Formats

arithmetic operations on 16 and 32 bit words. A set of byte string instructions are provided for the 8 bit data, accommodating strings of 1 to 256 bytes. The single 8 bit operation code in each instruction specifies the instruction format and the meaning of following modifier in bytes, resulting in a straightforward microprogram design.

I/O operations on the IBM 360 are unique: they are performed by special purpose microprocessors called I/O Channels. Each I/O Channel is a completely independant microprocessor system with attached I/O devices, which IBM calls I/O Control Units. A specially designed 8080 CPU card on an S100 bus would make an excellent I/O Channel, with standard S100 I/O devices as I/O Control Units. An IBM 360 can have up to seven I/O Channels, but typically has one or two. Each I/O Channel has access to main storage via DMA. Each I/O Channel can have up to 256 I/O devices attached. I/O operations therefore specify the desired I/O operation with an 8 bit byte to identify the channel and an 8 bit byte to identify the device. As a result of this approach, there are only four I/O instructions in the IBM 360: (1) Start I/O, which sends an address to the identified I/O channel processor and device. This is the address in main storage of a string of I/O commands for that device. (2) Halt I/O stops the identified device, (3) Test I/O provides a status word to the CPU indicating the status of the identified device, and (4) Test Channel provides status to the CPU of the condition of the addressed channel processor. If an S100 Buss unit is used as the I/O Channel microprocessor, this means that all I/O operations would be accomplished in detail by the S100 microprocessor software.

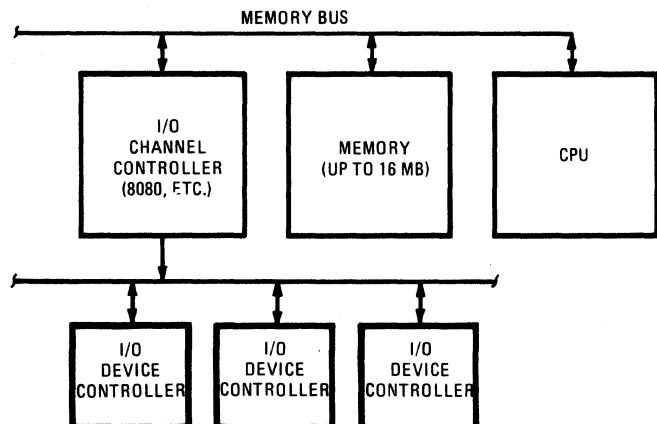


Figure 2. IBM 360 System Block Diagram

IBM 360 EMULATION DESIGN

A block diagram of the IBM 360 is shown in Figure 2. The CPU, Memory, and one I/O Channel Controller are shown. The memory bus could be similar to the current S100 bus with 8 bit data and a 24 bit address rather than the current 16 bit address. The interface between the I/O Channel and the I/O devices can be a standard S100 bus. Both the CPU and the I/O Channel access the Memory, with the I/O Channel having priority DMA access.

A block diagram of the CPU section is shown in Figure 3. The key items are the General Registers block, consisting of 8 four bit slices (2901's), the Program Counter using 2911 slices to provide both the counter and register functions, the Instruction Register and its associated data and register select multiplexers, and the microprogram control section consisting of an operation decode ROM, a microprogram sequencer, and the microprogram control store. This results in

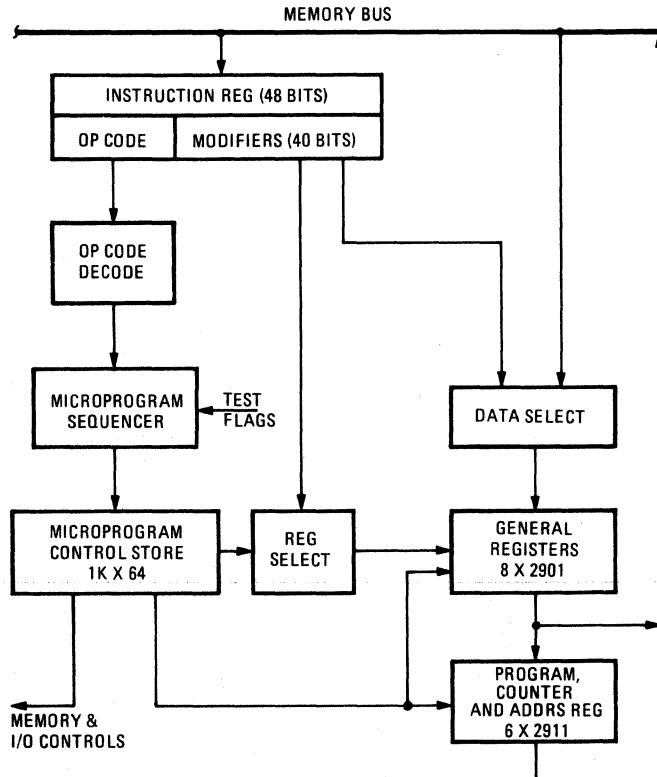


Figure 3. IBM 360 CPU Block Diagram

approximately 80 chips including 20 TTL chips and assuming 4K static MOS RAMs for the microprogram control store. The cost of these chips is approximately \$380 at today's prices, including \$160 for the control store.

Note that MOS RAMs are used in the control store in this case. This allows a low price for the control store at some expense in speed. With this arrangement, microcycle times of 500 nanoseconds are practical, which is also comparable with 8080 timing and the access/cycle time of many currently available RAMs. Remember that the major improvement in changing architectures is in the ability to address large memories and to perform 16 and 32 bit arithmetic in one instruction.

RAM is used for the microprogram store to allow development and debug of the microprogram, and to allow microprogram extensions to the basic set. It also has the advantage of allowing instant design changes. This RAM can be loaded by suitable connection to I/O Channel Controller 0 with a corresponding special microprogram load routine in the I/O Channel local ROM. Microprogram design is straightforward: the Instruction Register is loaded, the Op Code Decode is used to select the corresponding microroutine, the routine is executed, and the Program Counter is incremented in preparation for the next instruction.

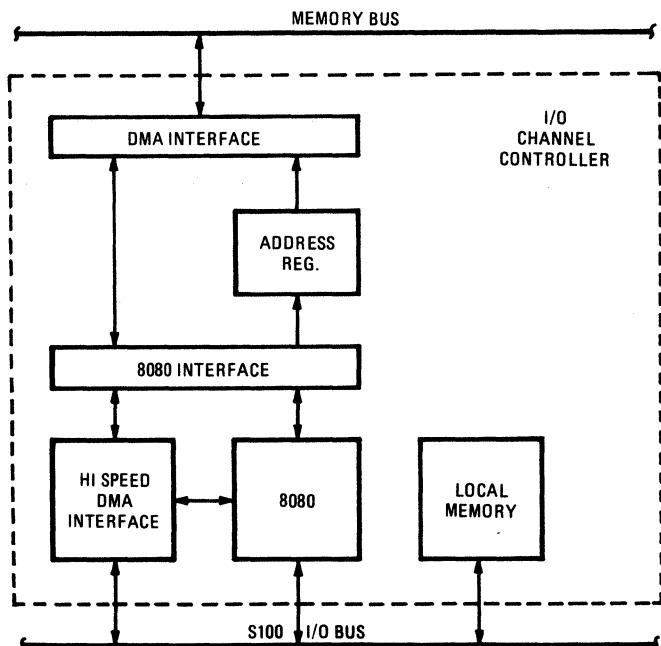


Figure 4. I/O Channel Controller Block Diagram

A block diagram of an I/O Channel Controller design is shown in Figure 4. An 8080 is used as the processor with a DMA interface and main storage address register for accessing main storage, an 8080 interface to match the 8080 timing, and an S100 bus for the I/O devices. The program to interpret the 360 Channel Command Words is contained in the local memory, along with local data buffering.

SUMMARY

The concept of large scale computing for the amateur computerist may seem astonishing; however, astonishment is the working environment in this field. Large scale machines of the IBM 360 class are not only possible but, considering the low costs now possible, highly probable. In the process of overcoming natural memory limitations of current microprocessors, machines of this class provide very significant

performance improvement at realistic cost levels.

It should be noted that the IBM 360 design was chosen for description because of its relative simplicity of architecture and its compatibility with current bit slice components. Many will argue that other machines, such as Burroughs 5500, etc., are significantly better in various respects, and I applaud these sentiments. The controversy over computer architecture is, and should be, as active in large scale machines as it is in the microprocessor world.

REFERENCES

1. Results of a current Homebrew Computer Club survey (1/19/77):
Total systems in operation: 167 (100%)
Total with 16 K or more: 52 (31%)
Total with disks: 16 (9.5%)
2. This year will see volume production of 16K dynamic RAM chips, and 64K RAMs are in design. IBM has fabricated CCD chips, which is the same technology being employed in 16K RAM design, of 500K bits. It is therefore reasonable to project that 256K bits per chip is practical, and perhaps one megabit per chip may be achieved.
3. Reference data on the IBM 360 is available from IBM in their manual Order No. GA22-6821.

BIPOLAR MICROPROCESSOR MICROPHOBIA

John R. Mick
Advanced Micro Devices
901 Thompson Place
Sunnyvale CA 94086

Introduction

The MOS Microprocessor has created a computing revolution that brings the personal computer within the grasp of every household in America. Even more important in bringing this revolution to reality is the availability of large, low-cost memory chips that are controlled by the microprocessor. Certainly until now, the hobbyist has concentrated on using one of the various MOS microprocessors as the heart of a home computing system.

In addition to finding a market in the personal computing field, the currently available hobby equipment has found a very popular OEM segment in the low-end computing systems marketplace. As this segment of the market grows, many consultants providing such equipment and services will soon find that the MOS microprocessor "runs out of gas" for their application. With some frustration and with a burning desire to save their existing 8080 software, these creative engineers will seek a new hardware solution for the low-end computing market. Three cheers for Bipolar Bit Slice Microprocessors, such as the Am2900 Family. These devices will provide the necessary performance improvement required in low-end systems and yet because of their microprogrammability will allow for the emulation of each and every Microprocessor.

What Is a Bipolar Microprocessor

Most small minicomputers today are being designed using a technique called microprogramming. In microprogrammed systems, a large portion of the system's control is performed by a PROM rather than large arrays of gates and flip-flops. This technique frequently reduces the package count in the control section of the minicomputer and provides a highly ordered structure in the controller, not present when random logic is used. Moreover, microprogramming makes changes in the machine instruction set very simple to perform. Figure 1 illustrates a typical system architecture. There are two sides to the system. At the left side is the control circuitry; at the right side is the data manipulation circuitry. The Am2901 Microprocessor devices contain the ALU, scratchpad registers, data steering logic (all internal to the Am2901's), plus left/right shift control, and carry lookahead circuit. Data is processed by moving it from the main memory (not shown) into the Am2901 registers, performing the required operations on it and returning the results, if required, to main memory. Memory addresses may also be generated in the Am2901's and sent to the memory address register (MAR). In this diagram, the Am2930's are used to perform this function. The four status bits from the Am2901 ALU most significant device are captured in the status register after each operation.

The logic on the left side of the Figure 1 drawing is the control section of the microprocessor. Here, the Am2911 and Am29811 are used. The entire

system is controlled by a memory, usually PROM, which contains long words called microinstructions. Each microinstruction contains bits to control each of the data manipulation elements in the signal. There are, for example, nine bits for the Am2901 instruction lines, eight bits for the A and B register addresses for the Am2901, two or three bits to control the shifting multiplexer at the end of the Am2901 array, and bits to control the shifting multiplexers at the ends of the Am2901 array, and bits to control the register enables on the memory address register, instruction, and various bus transceivers. Bits in a microinstruction are also applied to all the data elements and the entire system is clocked; thus, one small operation (such as data transfer or a registered add) will occur.

A "machine instruction" such as a mini-computer instruction or an Am9080A instruction is performed by executing several microinstructions in sequence. Each microinstruction therefore contains not only bits to control the data hardware, but also bits to define the location in PROM of the next microinstruction to be executed.

An 8080 Emulation

Once the small system designer realizes that his MOS microprocessor has "run out of gas", he will soon realize that an emulation is in order. Suddenly, a whole realm of trade-offs are faced in the design of such an emulator. In addition, once the hobbyist realizes the flexibility of a microprogrammed microprocessor, he will become intrigued with the new "hands on" capability at his disposal. As Mr. David Wyland has suggested in his paper for this session, why not emulate an IBM 360 for your home hobby computer.

The intent of this paper is to give the hobbyist audience in attendance some feel for the capability as well as the complexity of the emulation. The 8080 microprocessor has been selected for this example. An 8080 emulation actually consists of emulating more than just the 8080 device itself. In addition to emulating the 8080, the design must also emulate the 8228, and a portion of the 8224. Thus, the emulator design has the same input and output lines and controls as the 8080/8224/8228 set of devices has. For example, the emulation has a 16-bit wide address bus, a 16-bit wide data bus, the control outputs (MNR, MMW, IOR, IOW, HLDA, WAIT, IE), and the required control inputs (HOLD, READY, IMT, RESET). After considerable study, it was decided that the 8080 emulator should use a 16-bit wide ALU that would be realized using four Am2901 Four-bit Bipolar Microslices. Their internal registers were assigned as the 8080 registers, including the program counter and the stack pointer. The four Am2901's are grouped into two pairs, a high-order pair and a low-order pair. This representation enables an easy access to the 16-bit double length registers BC, DC, and HL, as well as the PC and SP registers, which are also 16 bits wide. Since the

8080 utilizes an 8-bit data bus and most of the internal registers can be used as an 8-bit working register, it is necessary to have easy access to all the registers at the lower eight bits of the ALU. This is accomplished by mirroring the register pairs as CB, ED, and LH, as adjacent pairs of registers in the Am2901. Then, when single 8-bit wide operations are performed in the low-order Am2901's, the high-order Am2901's are updated by using a byte swapping technique. This is usually performed during the fetch of the next instruction, and therefore does not slow the emulator in any way.

The microprogram portion of the 8080 emulator consists of a microprogram memory with 56-bit wide words. The total amount of microcode required to emulate the 8080 and its associated devices is 352 words. The microprogram control unit is a classic controller using three Am2909 Microprogram Sequencers. The design was performed to demonstrate the feasibility and ease with which an MOS microprocessor can be emulated.

The major purposes of the emulator design was to demonstrate the use of the Am2900 Family components, to demonstrate microprogramming, and to show the use of the AMD microprogramming assembler, AMDASM. Although speed was not the first consideration in this design, the emulator needs about half the number of clock cycles as does the 8080 and it can run with a maximum clock frequency of approximately 5MHz. This means that a speed improvement of about 3:1 to 5:1 is realized when compared with a standard 8080/8224/8228 microprocessor chip set. Additional speed improvement could be realized by performing overlap fetch of the next machine instruction. In total, the design utilizes circuits to implement the full chip set capability.

Additional Emulation Benefits

One of the key reasons for emulating an existing microprocessor is to save existing software. However, one of the major benefits from such an emulation results from the ability to implement new machine instructions not currently found in the MOS microprocessor. For example, the 8080 emulator can be microprogrammed such that new instructions can be performed. For example, additional microcode can be written so that such machine instructions as multiply, divide, square root, and so forth, can be executed. This can save a considerable number of machine instructions, a large amount of software, and can improve the performance of an MOS microprocessor a hundred fold. It is this microprogrammability of new instructions that provide the emulator designer with the challenge. The advantage, of course, is that new software can take advantage of these new machine instructions, and significantly increase the throughput of the machine. Still, the old software is saved and can be executed on the microprogrammed emulation without any modification.

Summary

The purpose of this paper has been to introduce the hobbyist to the challenge of microprogrammed machines in home computing as well as its applicability to design for the small OEM user. One of the uses for such microprogrammed microprocessors is the

emulation of existing microprocessors to improve their performance while maintaining compatibility with existing software. A more exciting application of Bipolar Microprocessors in the hobbyist arena lies in the challenge of designing new machines with large scale computer capability for home use. For example, why not dedicate microcode for the basic machine instruction set, then add microcode for floating point capability, then add microcode for a higher level language such as FORTRAN, then add microcode for . . .

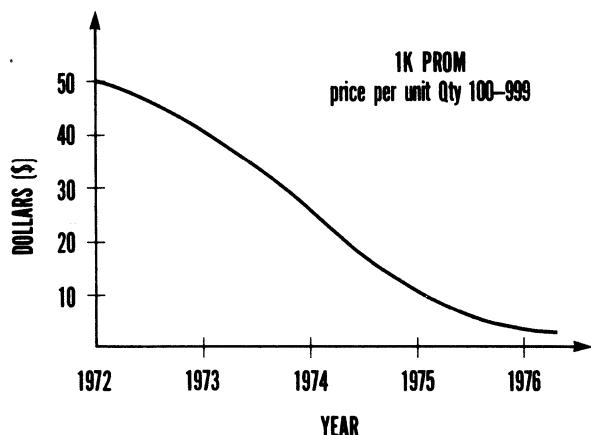
JRM/nls

MICROPROGRAMMING FOR THE HOBBYIST

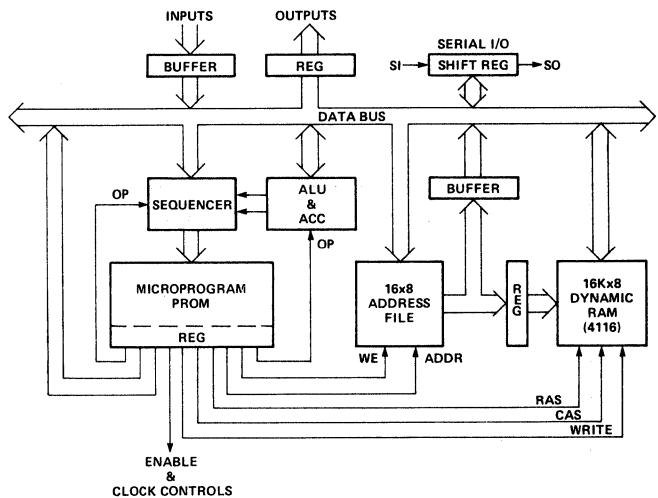
John Birkner

Monolithic Memories
Sunnyvale, California

In recent years, the MOS microcomputer revolution has made hobbyist computing a reality. While all of the excitement and attention has been focused on the microcomputer, an evolution in microprogrammed systems has been quietly taking place. The continued increase in MSI/LSI functions, the availability of larger and larger PROMs and the rapid price attrition of both has slowly transformed virtually all new high performance system design to microprogrammed techniques. What the hobbyist will be surprised to find is that state-of-the-art microprogrammed hardware design is well within his capability. In fact, microprogram hardware design has become a trivial task of interconnecting building blocks.



An 8-bit microprogrammed computer can be constructed as shown below using as little as thirty ICs. A 100nsec microcycle is easily achievable. Instruction decode is accomplished by direct vector from 8-bit macro instruction into micro-space. Dynamic RAM controls signals RAS, CAS and WRITE are under direct microprogram control.



This computer can read a keyboard, control a floppy disc and drive a television raster scan, all under microprogram control.

The PROMs

Bipolar PROMs are available in 4-bit-wide or 8-bit-wide organizations. They range in speed from 45nsec to 100nsec, and in size from 1/4K to 8K. Organizations are as follows:

Advantages

Performance is the name of the game in microprogrammed systems. The microcontrolled system executes many operations in parallel across a wide microinstruction (24 to 64 bits) offering higher speed than the fixed instruction microcomputer.

Where blazing speed is not required, speed can be used in place of complexity. For instance, reading a keyboard, handling floppy and also a CRT can be accomplished in sequential fashion by a single processor as opposed to individual processors and controllers.

Disadvantages

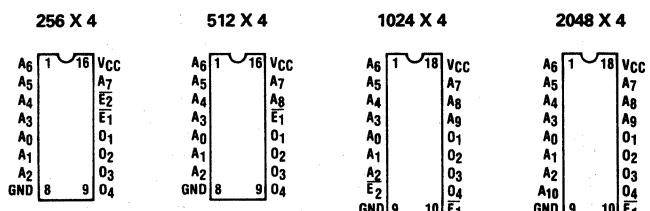
There is very little software available for microprogrammed systems. Until inexpensive microcode assemblers are available, the hobbyist will have to homebrew his own. Microcode assemblers are available, however, on many timesharing computer services. Monolithic Memories provides MICROAID™ for its 6700 series products, while Raytheon provides RAYASM for its 2900 products.

The Microprogrammed Computer

The microprogrammed computer is composed of five basic elements:

- Control Store PROMs
- Data Loop
- Control Sequencer
- Program Store
- I/O

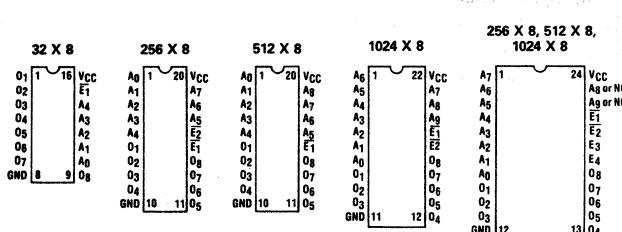
4-BIT-WIDE



FEATURES

- High Speed
- High Density
- Compatible Pin Configurations for Upward Expansion
- Used in Microprogram Stores

8-BIT-WIDE



FEATURES

- Convenient 8-Bit Format for Byte-Oriented Applications
- Used for:
 - Program Stores
 - EPROM Replacements
 - Table Look-ups
 - Character Generators

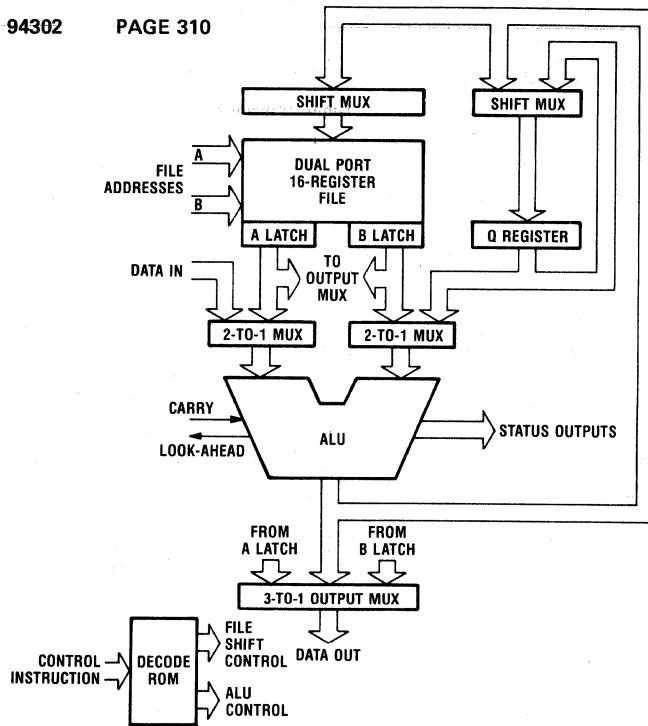
Choosing between 4-bit-wide and 8-bit-wide PROMs is a multi-variable problem depending on some of the following:

- Word Depth
- Access Time
- Package Size
- Cost Per Bit
- State of the Art Bit Densities
- Power Consumption
- Multiple Sourcing
- Programming Yields
- Access to PROM Programmer
- Upward Socket Compatibility

The Data Loop

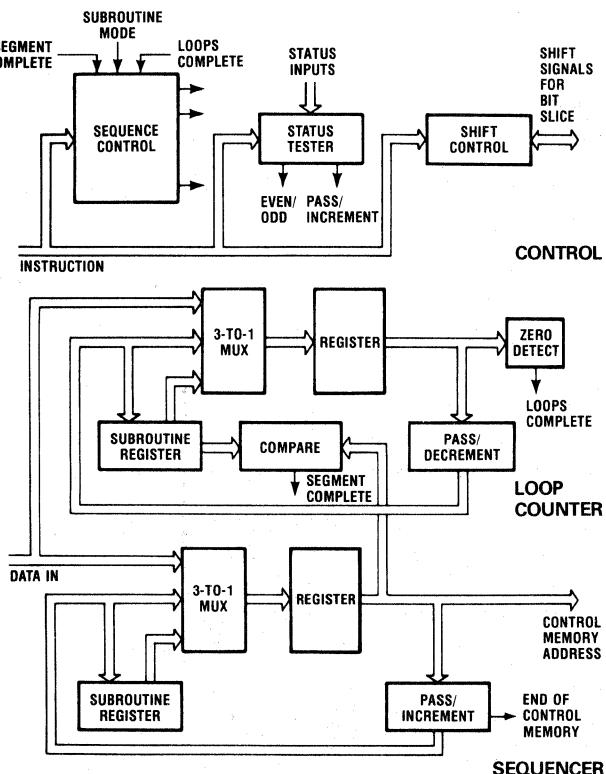
The data loop can be as simple as an ALU and an accumulator. The most popular ALU is the 74181, although the 74S381 is gaining. The 74181 is a 4-bit ALU in a 0.6 inch wide 24-pin package and contains all the function combinations you will ever need. The 74S381 contains a reduced number of functions in a 20-pin, 0.3 inch wide package. AMD offers an important variation of the part, which outputs CARRY and OVR for use as the most significant chip. The 74S281 is a 24-pin ALU with an Accumulator/Shift Register on board. An "LS" version is also available. This part efficiently compacts the basic ALU, accumulator and shift functions required by a medium performance data loop and, thus, is a versatile part for the microprogramming hobbyist.

The bit slices offer microprogrammers an ALU, accumulator/shift register and a multiport file in a single 40-pin package. The Monolithic Memories' 6701 and the AMD 2701 are now in the \$20 range, well within the hobbyist's reach. The Texas Instruments 74S481 should be available soon. Shown below is the Monolithic Memories' 6701.



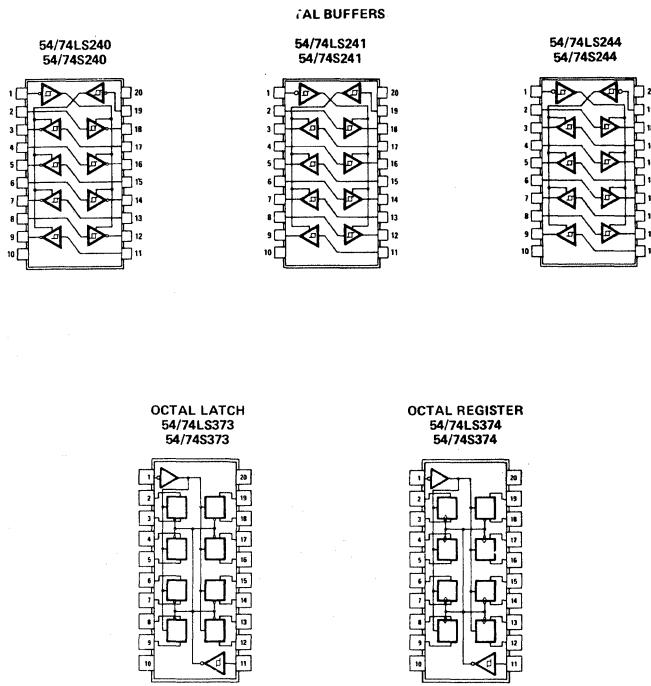
The Sequencer

The microprogram control memory address is driven by a sequencer which can be as simple as a 74S163 4-bit counter or as complex as the 67110 microprogram controller. Other sequencers include the AMD 2909/11, the T.I. 74S482, the Signetics 8X02, the Intel 3001 and the Fairchild 9408. The Monolithic Memories 67110, shown below, is a complete controller in that it provides a loop counter, shift linkage, status monitor, branch control and one level of subroutine stack.

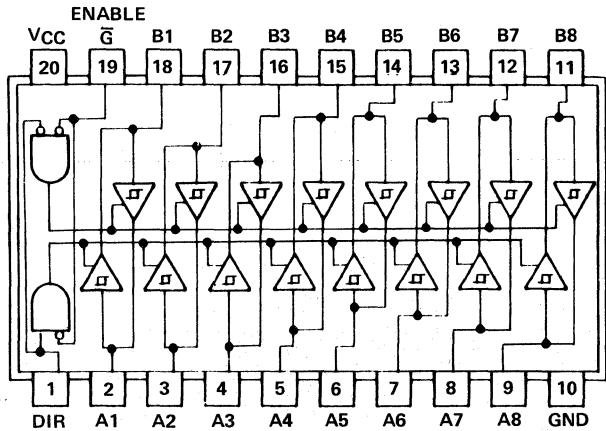


Interface

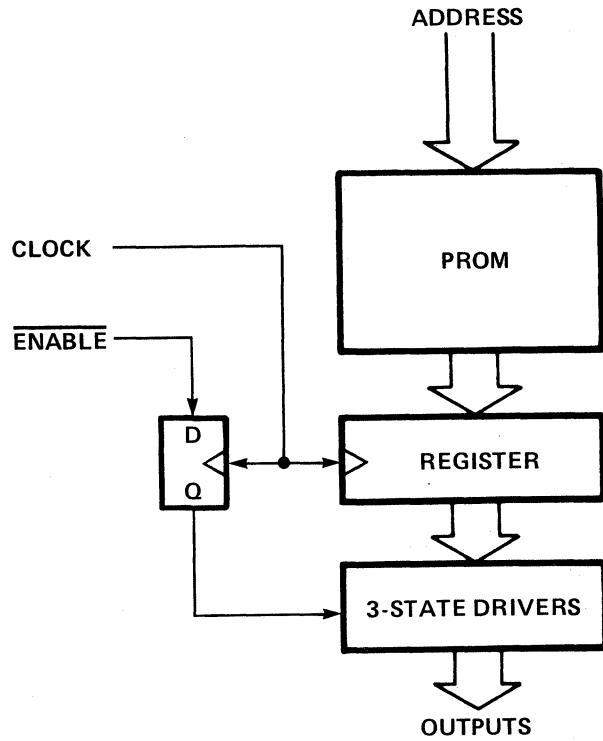
Buffers like the 74S240 and 74LS240 family provide the drive and isolation necessary for interconnecting microprogram building blocks together. Registers and latches are also necessary to hold data in time as instruction operands, address operands or simply data in transition. Shown below are the octal buffers, latches, and registers in their convenient 0.3 inch wide 20-pin skinny DIPs.



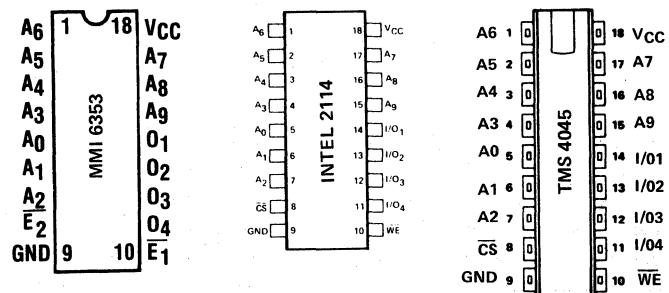
A spiffy new octal transceiver, also in the skinny DIP, promises to be a big winner for bidirectional data buffering.

**What's Coming?**

New PROMs, FPLAs and RAMs promise to make microprogrammed architecture increasingly competitive in medium and high-performance computing. Shown below is a Registered PROM which saves space and power while increasing speed in pipelined microprogram control stores.



RAMs in PROM pin-outs like the Intel 2114 and the T.I. TMS4045 will make writeable control store convenient for the hobbyist.



A concept of "Structured Logic" is beginning to form which embodies the principle of constructing computers entirely from RAMs, PROMs, FPLAs, Registers and Buffers. This structured approach to architecture will offer more flexible and more powerful solutions, not only to industry, but to hobbyists as well. And after all, wouldn't we all like to be the architects of our "home-brew" computers?

HAM RTTY: ITS EVOLUTION AND FUTURE

Robert C. Brehm
Promedics Data Corporation
Box 7454
Menlo Park CA 94025

Today's RTTY amateur has available a choice of equipment and techniques that allow him to develop sophisticated radioteletype and computer linkages which were only dreamed of several years ago. Moreover, with the ever increasing usage of RTTY on FM repeater networks across the country, the feasibility for radio-linked computer time sharing is not only plausible, but is on its way to becoming a reality.

Today I would like to provide you with a general introduction into amateur RTTY by describing the chronological background of RTTY developments over the past 15 years which lead up to the current state of the art using microcomputers. Then, if you will allow me the liberty of looking into my crystal ball, I will try and give you a prediction of what the marriage of microprocessors and RTTY will bring in the future.

Amateur RTTY - What is it?

In order to understand amateur RTTY it is helpful to understand where it fits in to the amateur radio operator's repertoire of transmission capabilities. The FCC allows hams to transmit information via morse code (CW), amplitude modulation (AM), single sideband (SSB), frequency modulation (FM), RTTY, Slow Scan TV, and several other more exotic methods not commonly used today. As you can see the amateur has a variety of methods to communicate with his fellow man. When compared to the total ham population, RTTY hams account for only 2-3% of all operators, yet this number has grown rapidly in the past year and should reach 10-15% (20,000 to 30,000) by 1980.

In addition to the variety of operating methods available to amateurs, there is a large spectrum of frequencies which can be used for transmission. RTTY activity has traditionally used 80 meters (3.6 MHZ) for medium distance communication out to about 800 miles; it has used 20 meters (14.1 MHZ) for worldwide communication, and 2 meters (146 MHZ) for local and repeater RTTY work. More recently 2 meters has also been used for RTTY transmission through the amateur radio satellite, OSCAR 6 which revolves around the earth.

As you can see, RTTY transmission takes place in two frequency spectrums, HF or frequencies below 30 MHZ and VHF or frequencies above 30 MHZ (specifically 146-147 MHZ). The methods of signal generation and reception are slightly different for VHF and HF although the end result is the same. Below 30 MHZ, the primary method of signal generation is called frequency shift keying or FSK. This method involves changing the carrier frequency each time a mark or space pulse is encountered during a character's transmission. Using FSK, a constant frequency carrier is transmitted for a mark pulse, with a space pulse causing the carrier to shift down by 850 HZ. More recently, common practice has changed the shift to 170 HZ due to the increasingly crowded ham band signals which cause interference within the wider 850 HZ shift. The other common method of RTTY transmission on HF utilizes audio frequency shift keying, or AFSK. With this method, a pair of audio tones is fed into a single sideband system, utilizing a well-suppressed carrier and operating in the lower sideband mode. The shift of the audio tones, usually generated with 2125 (mark) and 2295 (space) HZ signals, again presents a 170 HZ signal to a copying receiver.

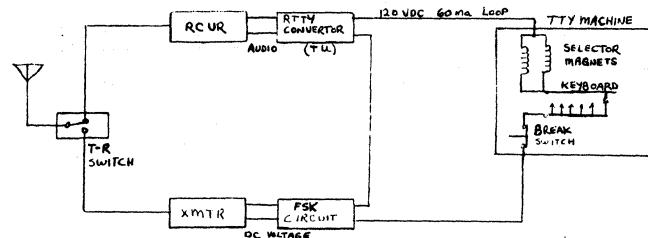
On the receiving end of things, both FSK and AFSK generated signals sound exactly alike, and it is unlikely that you could tell them apart. On VHF, the audio tones are usually fed into an Am or FM transmitter using either 850 HZ or 170 HZ shift, depending upon which shift is being used in the particular area.

The similarity between AFSK signals used in RTTY equipment is similar to the AFSK used in most computer-telephone-modem links. These similarities are shown in Table I.

Now that you have a general idea of how RTTY is generated and where it is used, let's take a look at some of the conventional equipment used in amateur stations.

RTTY Equipment

First let's look at a block diagram of a typical RTTY station.



In this setup, the receiver detects the signals, and converts them to audio signals, and feeds them to the RTTY convertor or Terminal Unit (TU). The convertor takes the signals, processes them through tone selective filters or a phase locked loop system and finally modulates or keys a high voltage 120VDC "loop" adjusted for 60mA of current required by the selector magnets of the teleprinter. The keyboard of the teleprinter is also connected in series with the loop supply; consequently, whenever a key is depressed, it breaks the loop in a predetermined sequence, which causes the printer to print that character. In addition, when the transmitter is on, depressing a key will cause the BAUDOT encoded character to be transferred into a FSK generator which can modulate the transmitter.

Transmitting and receiving are conceptually easy to understand although, depending upon the accessories that are hooked into the system, the switching circuitry can get quite complicated.

Many amateurs started with simple setups as shown in Figure 1; however, they soon learned that adding accessories could significantly enhance both receiving and transmitting operations. Early in the RTTY game, most additions to the RTTY station were mechanical in nature. Paper tape perforators and transmitter distributors (tape readers) were probably the first added luxuries. These items allowed you to punch a tape for transmission later at full speed. It is common practice to punch a tape while the other person is sending to you, so that when he is done, you can immediately start sending back to him. Another mechanical feature which proved to be quite popular was the stunt box available in the Model 28 machines. This mechanism allowed the printer to decode or "recognize" certain characters or combinations of characters in such a manner that a switch closure (or opening) could take place. This feature, in conjunction with an option called answer-back (part of an item called a WRU or who are you?), allowed a person's machine to answer or transmit back a programmed message when queried with a preprogrammed recognition sequence. Some Model 28s are even equipped with back space and reverse line feed so you can "roll up" what has already been typed. Imagine coming home to a spindle of paper all rolled up and printed!

Many of the above mechanical features were coupled with a very popular option called autostart, which is short for automatic starting. This feature allows your TU to be activated by a steady mark tone lasting a given duration (usually about 3 seconds or more). Although employed to a limited extent on the low bands, autostart RTTY is used quite extensively between club or net members using the VHF bands for local communications. The principle involved with autostart is simple. As an example, suppose you have a receiver monitoring your favorite channel, and while you are out at a swapmeet, one of your friends calls you on the radio to leave you a message. When he transmits a continuous mark tone for 3 seconds, your TU responds by turning your printer motor on and subsequently prints your friend's message and shuts itself off again... all automatically. When you arrive home, the message is there waiting to be read. This feature is obviously very useful, and many amateurs monitor a given frequency 24 hours per day in order to be in constant communication with their friends.

	RTTY Communication	Computer Communication
Transmission Media	Ether/Radio waves	Hardwire/Audio Signals
Speed	45.5 BAUD or 5 chars/sec commonly called 60WPM	110, 300 BAUD or 10 to 30 chars/sec
Code	5 level BAUDOT 32 lower case "letters" 32 upper case "figures"	8 level ASCII 128 characters commonly used
Frequency Shift	850 HZ 2125, 2975 HZ signals or 170 HZ 2125, 2295 HZ signals Half Duplex only	200 HZ 1800 through 2200 HZ Half or Full Duplex

Communication Characteristics of RTTY Versus Computer
Table I

The 1960's

These mechanical marvels were quite the state of the art in the mid and late 1950's. But around 1960, with the ever-increasing use of the transistor by amateurs, many of the functions previously performed by mechanical means were being performed by electronic circuits.

Probably one of the first mechanical items to give way to electronic circuitry was the ubiquitous polar relay. This relay was used in the polar circuitry of terminal units where current flowed in one direction for space and in the opposite direction for mark. Thus the relay could key a local loop supply for a teleprinter when driven by tone decoders. This electronic "advancement" in modern technology, utilizing such components as SCRs and switching transistors, got rid of noisy clicking relays which often got out of adjustment or were plagued with dirty contacts. More importantly, however, it also sparked the beginnings of the evolution of solid state technology in RTTY applications.

Several other changes took place during the 1960's. Tube terminal units became more sophisticated, then finally gave way to solid state equivalents during the late 1960's. AFSK generation was upgraded to crystal controlled oscillators using digital divider ICs for increased accuracy. Mechanical stunt boxes now had digital equivalents that could recognize selected character strings during a transmission. All in all the evolution that took place in the 60's was not revolutionary, in that no real new operating techniques or innovations of technological brilliance took place. The new circuitry developed used state of the art components but the operational methods used by RTTY hams didn't really change.

The 1970's

The 1970's have brought us not only a digital revolution in consumer and industrial products, but also in RTTY, both for high frequency and VHF application. Figure 2 illustrates this evolution in both areas of RTTY operation.

In 1970 the ST-3 and ST-5 terminal units were pioneered by Irv Hoff. Although designed primarily for VHF, they have been used extensively for HF also. In 1971 the ST-6 was born. This TU represents a solid state attempt to incorporate the circuitry developed in advanced tube TU's in the late 60's. It incorporated mostly op amps of the 709/741 series. Even today this TU is the most popular and widely used on the HF bands.

Very few major technological differences have taken place in HF RTTY TU technology since the advent of the ST-6. Most changes have involved accessories which interface with the ST-6 and provide more convenient operation. In 1971 speed converters using discrete components were introduced, as were preprogrammed digital message generators. In 1973 a crystal controlled digital audio synthesizer was described for AFSK generation with SSB transmitters. 1973 also brought several new accessories to the RTTY community, including video displays for BAUDOT, digital keyboards, digital autostart units, and Morse code to RTTY converters using discrete TTL circuits. 1974 was also a good year for technological developments in digital HF RTTY and the introduction of UARTs in RTTY circuitry. The Phase locked loop also became popular in several TU designs taking circuitry ideas modeled after the PLLs used in computer modems. It appeared that technical developments were really beginning to accelerate, but little did anybody know what was to happen next.

It was in 1975 when RTTY technology really exploded, just shortly after the introduction by MITS of the Altair 8800 microcomputer. Programmable logic had now been introduced into RTTY and this would prove to be the most revolutionary happening in RTTY since its beginning in 1951.

In 1975 several magazine articles described programmable RAM message generators. Also UARTs and FIFOs were becoming very popular. In late 1975 we began to observe the influence of microprocessors as evidenced by several articles which outlined microprocessor controlled RTTY stations both on VHF and HF.

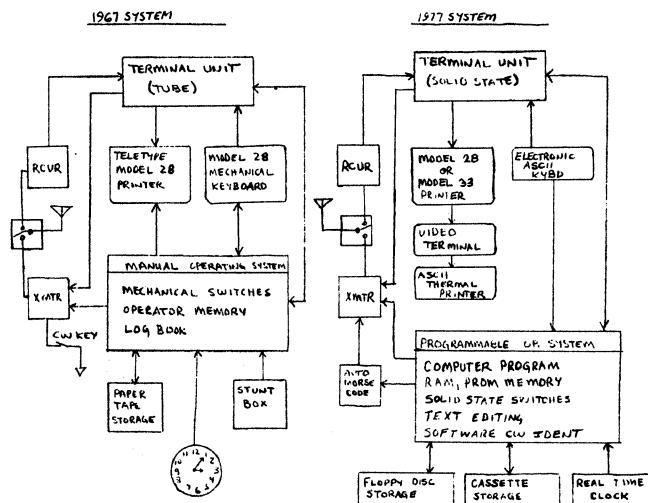
This exciting trend has continued to accelerate at an increasing pace as the availability of hardware and software technology becomes more widespread. Hal Communications has introduced an 8080 based RTTY terminal; digital Selcal circuits are springing up everywhere; ASCII to BAUDOT and BAUDOT to ASCII code converters have been written up in several magazines with commercial units now available; and, to top it all off, the grand prize winner at the World Altair Computer Conference was a computer controlled RTTY station! We have certainly come a long way from the days of polar relays and tube converters!

So here we are in 1977, the year of the microprocessor in RTTY; where to go now? Let's look back for a second and compare a typical advanced RTTY station of 1976 with an advanced station in 1967. These systems are illustrated in Figure 3 and

a comparison matrix is shown in Figure 4.

1970	Mainline ST-3, 4 Mainline ST-5	VHF TUs of modern design
1971	Mainline ST-6 Digital MSG gen. Speed Converters Digital AFSK gen.	First good low band TU Digital accessories come into wide use
1972	PLL TU-VHF Audio Digital synthesizer	PLL technology used
1973	Hal video display Digital keyboards Morse to RTTY Converter Digital autostart	
1974	Prom Memory for RTTY ID RTTY MSG gen. Using UART	
1975	MITS introduces Altair Programmable RAM MSG gen. UT-4 (FIFO) PLL TU Solid state TTY keyboard uP control of RTTY STN	
1976	DT-600 Digital ST-6 Hal MP unit Digital Time Clock BAUDOT/ASCII Digital Selcals	Morse to RTTY - uP ASCII/BAUDOT WACC Winner - RTTY Station
1977	RTTY used with uP Intelligent terminals uP control of repeaters uP controlled RTTY stations - wide use Computer oriented repeaters with A/D converters for time, weather, temperature, etc.	

Figure 2



System Diagram

Figure 3

The Future

OK, so what's left, you may ask. If you let me take the liberty of gazing into my crystal ball, here is what I see. First, I can see quite clearly the ever-increasing use of micros like the 4040, 8008, 8080, 6800, SC/MP, Z80, etc., in amateur designed circuits dedicated to RTTY. By late 1977 the majority of RTTY amateurs should have some type of micro in use in their station, even if only to look at! Many of the micros

will be used to provide intelligent terminals which can be used for text editing before transmission and for preparing data for storage on a tape cassette.

It is also reasonable to expect microprocessor controlled radio repeaters in 1977 for both RTTY and voice applications.

	1977 Station	1967 Station	
Terminal Unit (RTTY Convertor)	Solid State with analog and digital circuitry. \$300	Advanced tube design (homebrew) \$150	
Keyboard	Electronic with ASCII Serial output (multiple speed). \$100	Mechanical-probably part of model 28. (one speed)	
Printer	Video with 16x64 or 12x80 format \$200 ASCII or thermal printer \$700	Model 28 printer \$500	
Operating System	Computer programs, RAM memory, text editor, real time clock \$800	Mechanical switches, operator memory, log book, stunt box, mechanical clock \$50	
Data Storage	Floppy discs, cassettes \$700	5 level paper tape, reperforators, transmitter distributors \$300	
Morse Code Rx/Tx	Reception and generation under program control up to 1000 WPM	Operator ear, mechanical or electronic key-up to 60 WPM. \$100	
Software	Greeb Text Editor- <u>Doctor Dobb's Journal</u> ASCII to BAUDOT BAUDOT to ASCII	Operator memory	
System Cost	Full System \$2100 Limited System using BAUDOT I/O \$1100	\$1100	

Systems Comparison Matrix

Figure 4

These will be interfaced with A/D converters which could give time, temperature, weather reports, etc., when certain access codes are received by the computer. In addition, I foresee more sophisticated remote monitoring of repeater technical parameters, where voltages, temperature, and currents are transmitted in ASCII, when inquiry codes are received by the repeater. I also can see the increased use of magnetic tape and floppy discs. These will be used for contest record keeping, QSL info., magazine indexes and personal accounting records. Also in 1978 "canned" software for your favorite computer routines will be readily available from your ham or computer dealer. In addition, with the wide spread use of video discs and audio terminals, more computer amateurs will be transmitting and receiving video messages and symbols. Image processing and computer graphics will be fully developed on RTTY frequencies. But perhaps the most exciting event to take place will be the integration of VHF repeaters and computers.

I foresee the wide spread linkage of computers with radio repeaters for computer time sharing. Such a system could be structured as shown in Figure 5.

Each remote user would have a transmitter/receiver and terminal. He would have a choice of 5 transmit/receive frequency pairs to use for conversation with the computer. If one was being used, he could use another or if all were in use, he would have to wait until one was free. This system is analogous to the timeshared computer networks operating around the country which have multiple phone lines for input.

The computer could use the recently introduced 16/8 microprocessor board which allows real time hardware multiprocessing so that several users could be executing programs simultaneously and the computer could be transmitting and receiving on all channels simultaneously. In this manner, many people could use the capabilities of a computer which has a full complement of accessories without having to settle for a machine with limited

resources. In addition the repeater's capability to cover a large area could foster dissemination of software and computer techniques at a nominal cost.

As an adjunct to the above system, it is also possible to foresee the linkage, via radio waves and phone lines of computer repeaters in such a manner as to allow message switching and transmission of information across the United States. Thus you could turn on your 5 watt transmitter, type in an appropriate destination code and talk with another ham 3000 miles away!

Obviously FCC approval is going to be required to implement these ideas, but that confirmation will come providing there are enough concerned people behind it. And that is where all of you come in! If you are interested in what the future may bring in the RTTY and microprocessor world of amateur radio, join a radio club with a license training program, pass you test and get on the air! There are many RTTY repeater members around the country and most of them would be more than happy to help you get on the air - just ask one!

If you wish to learn more about RTTY try reading 73 Magazine which has had more RTTY articles over the last 10 years than any other amateur radio magazine. Other sources of information include Kilobaud, Ham Radio Magazine and the RTTY Journal. But perhaps the best way to learn about RTTY is to get on the air, practice the technique, become an expert and then you can give the next evolution of RTTY speech in 1978.

After the speech a short demonstration of RTTY operating techniques using microcomputers was given by David Alterkruse, W6RAW.

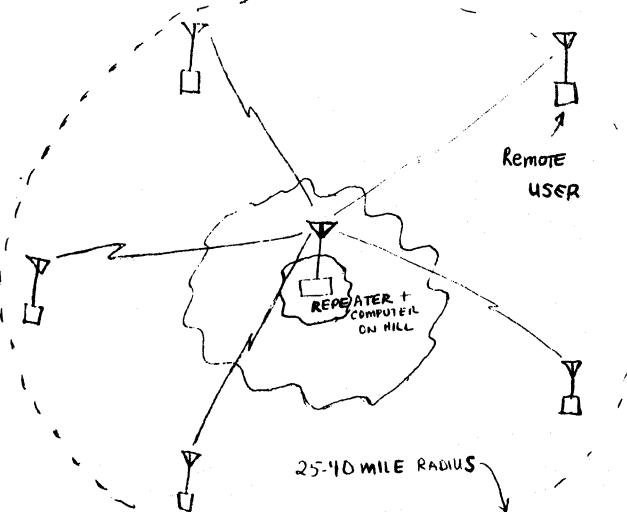


Figure 5

GENERATE SSTV WITH YOUR SWTPC 6800 MICROPROCESSOR

Clayton W. Abrams, K6AEP
1758 Comstock Lane
San Jose CA 95124

Soon after completion of my SWTPC 6800 System I thought now what can I do with it? Since my hobby is ham radio coupled with an interest in Slow Scan TV (SSTV) the answer was obvious. I decided to start in a small way with a program to load my WØLMD SSTV keyboard (ref 1). This project was quite successful, and I next attempted a more ambitious one to generate SSTV in the CPU main memory. It became obvious to generate SSTV I must first slow the microprocessor way down to accomplish the proper timings. I started by writing a short feasibility program which generated timing pulses and video in the proper relationships. I fed these pulses into a SSTV modulator which uses only three IC's and is external to the SWTPC 6800 system. This program performance exceeded my expectations; however, it had one large drawback. The picture dot patterns had to be entered a byte at a time into the main memory, which was a big job. This article describes my third program which automatically loads memory and generates SSTV pictures.

I think a short technical discussion about how I accomplish this is in order. I first decided on a few basic specifications which were taken from my WØLMD keyboard design which were:

1. A SSTV picture will contain 30 ASCII characters in a 5 x 7 dot matrix pattern
2. A SSTV picture dot will consist of three scan lines (vertical) and 1 dot (horizontal)
3. A SSTV picture will consist of 117 scan lines
4. A SSTV scan line will consist of 7 eight bit bytes (6 data, 1 sync)
5. A SSTV picture in the SWTPC 6800 memory will consist of 819 bytes (117 lines x 7 bytes/line).

As you can see from these specifications the memory requirements are quite negligible and my first feasibility program fits easily into the 4k of memory provided with the basic SWTPC 6800 system.

Well we now have a set of specifications, now how is the programming accomplished? The only programming language which makes sense in this application is assembly language. The 6800 assembly language instructions coupled with the use of MIKBUG makes programming the SWTPC 6800 system an easy task. My first job was to flow chart the entire program, which is a highly recommended practice for anyone attempting such a project.

Now lets examine in general how the seven picture bytes are used to generate the various sync pulses or picture dots.

00	00	F0	80	00	00	FF
----	----	----	----	----	----	----

Figure 1

Picture dots (hexadecimal) in memory

Figure 1 is an example of a typical SSTV scan line in memory. The first operation the program does in the transmit mode is to load from memory the first byte into an accumulator. The accumulator in the 6800 is a powerful register which can be used to add, compare or shift, etc. This byte is then compared to see if it is FF. If it is, then it must be a sync pulse. If not, it must be a data byte. The accumulator is then shifted to the left a bit at a time. If the shifted bit is a one then bit 0 of the parallel (PIA) interface is turned on. If the bit is zero then the interface bit is turned off. If the byte is a sync byte (FF) then interface bit 2 is turned on for 5 or 30 milliseconds (for horizontal and vertical sync pulses, respectively). Sounds easy? Well it is. All that is required is to connect these pulses to a SSTV modulator and out comes SSTV pictures. Obviously, programming delays must be executed between all steps. Three program constants control all of these delays. One for the horizontal line frequency, and one for the sync pulse durations and one for the number of scan lines. By manipulating these constants any number of scan lines can be transmitted (up to 117) at any frequency, which is only limited by the CPU cycle time and memory speed.

The generation of the SSTV is quite easy, the biggest trick is to place the correct dots in the picture at the correct location. This was accomplished by use of a translate table and a dot table. The translate table is used to tell the program where in the dot table the correct dots are located. The dot table is the 5 x 7 ASCII dot matrix array of bytes. The character dots were taken from the specification sheet of the 2513 character generator ROM chip (ref 2). Not all characters

(ASCII) were coded in this program due to their uncommon usage. The translate table in this program allows for expansion of 13 dot patterns without rewrite of my generator routines. The reader can code any of the patterns to suite his needs. In order to accomplish this task I must explain in detail how the various tables are coded.

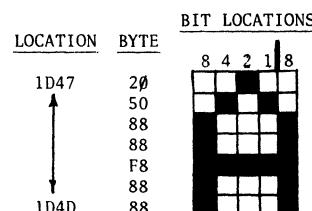


Figure 2. Dot pattern and hex representation for the letter A

As you can see, you are only limited by your imagination. With a little ingenuity a set of graphic-like dot patterns can be created. The translate table in memory is contiguous with the dot table. The low order address of the translate table is equivalent to 6 of the 8 bit ASCII code. For example, take the letter A. The ASCII code for A is 41 hex. If you strip off the 8 and 4 bits the code remaining is 01 hex. If you look in translate table location 1D01 (hex) you will find a 47. This value means that the address of the dot bytes for A are located at 1D47 in memory. You will find in the translate table (1D00 to 1D3F) a number of 04 characters. This value is a blank character and is placed on numbers with ASCII codes between 21 and 2D. The se 13 characters are rarely used so we save them for use in calling special graphics. The character codes (ASCII) 5B-5F will load characters with an inverted letter V (see photos). You now know how the program operates, now let's use it.

The program makes a few assumptions about how your 6800 system is configurated:

- The PIA card is in location 7 (address 801C)
- MIKBUG or equivalent is used, with the following:
 - E07E output a string of characters
 - E1D1 output a single character
 - E1AC input a single character
- At least 8K of memory
- PIA bit 0 used for data pulses and bit 2 for sync pulses

Some of the program constants which can be modified to fit your specific requirement are:

Memory Location	Present Value	Name
031A	75 (hex)	Number of lines/ Picture (117) Half frame=45 (hex)
02E3	80 (hex)	Horizontal line freq 80=15 hz (USA) 70=16 2/3 (Europe)
02E4	5D (hex)	Sync Pulse Width Horiz=5 msec. Vert=30 msec.

The program can be executed by loading MIDBUG location A048 and A049 with 0000 and typing G. The first routine executed is 'LOAD'. This routine loads the picture buffers as you type in. The first message printed on the TV terminal is:

PICTURE FORMATS 0-5

1 2 3 4 5

The program is now asking you to load 5 lines of 6 characters each. You first type in the picture number you want to load followed by 30 characters. The first character of each line will be placed under its corresponding line number.

You can now load all 6 pictures or type a ASCII letter to end the process. The next message printed on the TV terminal

screen is:

```
SELECT LOOPS-PICTURES  
LOOP MAX=9  
?
```

The program is now asking you to type in the number of times each SSTV picture is transmitted. For example a 3 loop would transmit approximately 24 seconds (3×8) of SSTV (each picture requires 8 seconds to transmit). The program will then respond with a slash after you enter the loop number. It will then wait for the picture you have just loaded (0-5). You can enter up to 7 loops/pictures and the process can be terminated by entering an ASCII letter for the loop number. A SSTV picture can be looped up to 9 times. This type of programming will provide over 8 minutes of SSTV 7 loop/ pictures the SSTV transmission is executed. After transmission the program branches back to the load routine. This completes the description and operation of the program. If you require more information please write. All letters with enclosed return postage will be answered. The following is a SSTV modulator which I use. The circuit was constructed on a Vector board and installed in a mini box. The design is quite straight forward and can be duplicated with little cost and difficulty. The timing pulses are interfaced to the computer by a 7400 NAND gate. White letters on a black background or black letters on a white background are selected by a SPST switch. The video and sync pulses are mixed by 4 diodes, 1 IC and 1 transistor. This video then drives a 566 function generator. The output triangular waveform is shaped into a sine wave by the output active filter.

I hope you will enjoy using this SSTV generator. Over the air reports have indicated that the video is considerably more readable under QRM conditions than my WØLMD keyboard. The use of micro-processors is an obvious choice for the generation of SSTV. Considering that my special purpose SSTV keyboard is complex (44IC's) and my SSTV 6800 generator runs on an unmodified SWTPC 6800, proves that the micro-processor is the way to go for SSTV.

CW OPERATOR'S UTOPIA - AUTOMATIC TRANSMISSION AND RECEPTION

Ivar Sanders, W6JDA
1161 Ribier Ct.
Sunnyvale CA 94087

INTRODUCTION

This presentation is a description of two algorithms. One algorithm generates Morse code at selectable rates in response to data entered by an operator using an alphanumeric keyboard. The other algorithm provides a visual display (on a CRT, teleprinter, or other output device) of received Morse code data; this reception algorithm automatically adapts itself to a wide range of code speeds. These algorithms are based on two articles in the October, 1976 issue of *Byte* magazine: "Add This 6800 Morser to Your Amateur Radio Station," and "If Only Sam Morse Could See Us Now." Although the programs listed in these articles are written for the 6800 microprocessor, this presentation does not assume the use of any specific processor. Instead, the algorithms are presented as more general flow charts which can be easily implemented on any machine. Also, no attempt has been made to include routines which are unique to specific systems, such as keyboard input or display output routines.

BACKGROUND

Although some hams may argue about the merits of using a machine to transmit and receive Morse code instead of the greatest computer of all - man's mind, many techniques have been developed to accomplish the task. The algorithms presented here work remarkably well and provide the ham operator with a novel (and sometimes useful) way to operate CW. For example, the transmission routine allows an operator to type a response to another station's comments while still listening, and then transmit the comments later in perfectly-sent CW. The "canned" messages required in many contests can also be stored away and transmitted as needed, thus allowing additional time to fill out log books, etc. The reception routine is an excellent tool for code practice, allowing the verification of hand-received code. Another interesting application of these algorithms is as a pseudo RTTY device. For example, stations could conceivably communicate using only keyboards and CRT displays and not even listen very carefully to the CW, and the data rate could be faster than the common 60 to 100 word-per-minute RTTY rates.

MORSE CODE STRUCTURE

So how do we get our expensive homebrew computers to speak in Morse code? Well, let's begin by looking briefly at the structure of the data format we call Morse code. The timing relationships for perfect code are based on the length of time a telegraph key is held closed to produce a dot (or short) Morse code element (Fig. 1). Assuming that a dot lasts one unit of time, a dash is defined as three units of time. Similarly, the key-open time between dot and dash elements within a character is one unit of time. So there we have the timing for a single Morse character. Now, how do we distinguish one character from another, and how do we distinguish one work from another? By varying the key-open time of course! For perfectly sent code, the time between characters is three units (same as a dash) and between words is seven units. However, there may be

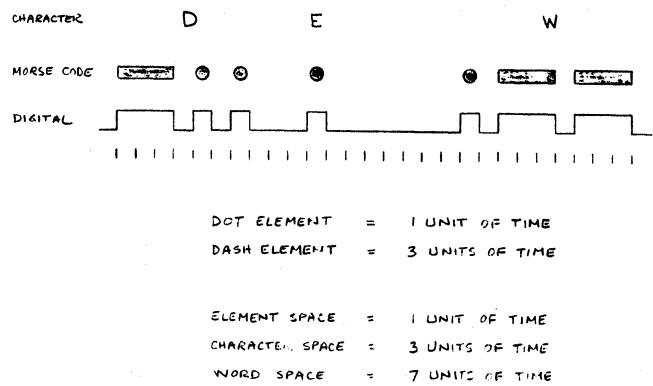


FIG. 1. MORSE CODE STRUCTURE.

instances where we want to increase the time between characters and between words. Code practice is a good example; it is often desirable to send individual characters at a faster rate than the overall code speed. The transmission algorithm described provides this capability.

TRANSMISSION ALGORITHM

Now that we understand Morse code's structure, let's take a closer look at what we have to do to generate CW with our home computers.

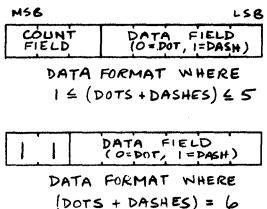


FIG. 2. COMPUTER'S REPRESENTATION
OF MORSE CODE DATA —
USED BY THE TRANSMISSION ALGORITHM.

Data Representation

The first problem we encounter is the representation of Morse code characters in our computers, since Morse characters have a variable number of elements. The representation we have chosen (Fig. 2) for the transmission algorithm divides an 8-bit byte into two fields: a data field and an element count field. For characters having five or fewer elements, the count field contains a binary number in the range one (001) to five (101), denoting the number of elements within the Morse character. The data field contains a binary representation of the dots and dashes making up the character where "0" represents a dot and "1" represents a dash. The character is right-justified within the data field, and the first element is in the LSB. But what about those special characters that have six elements? Since there are no characters that contain seven elements, the MSB of the count field is a "don't care" bit for characters of six elements, and the LSB becomes available for use as the MSB of the data field. Therefore a 11 code in the two MSBs uniquely specifies a six-element Morse character. Using this encoding method, we see that there is a unique 8-bit representation for every Morse character with the exception of word spaces and the error character (eight dots). These two exceptions are handled as special cases; a word space is arbitrarily assigned a code of all zeros, and the error code is assigned a code of all ones.

Hardware Requirements

In terms of hardware requirements we are assuming that one bit of a parallel output interface is dedicated to driving a relay or transistor which keys the transmitter. Another bit of the parallel output is used to drive a loudspeaker which is used as a CW monitor.

Generation Subroutine

Now let's look at the flow chart (Fig. 3) to see how we use this information to generate CW. Throughout this discussion we are assuming that the Morse code transmission routine is really a subroutine which is called by some other program as needed to generate a single Morse character. For example, the calling program might also manage a FIFO (first-in-first-out) buffer which is loaded with typed characters from an ASCII keyboard and unloaded one character at a time with calls to the generation subroutine.

After we call the subroutine, the first thing we have to do is convert the ASCII character we want to transmit into the previously described digital Morse representation. We perform this conversion with a look-up table which we have previously stored in the computer. We use the character's ASCII code to point to a specific memory location within the conversion look-up table, the contents of this location being the digital Morse representation of the character.

Next, we check the data to see if it is one of our special cases - a word space. If not, then we extract the count field from the byte and store it separately as an element count. In the case of our other special character - the error code - we force the element count to a value of eight and the data byte to all dots. Hence, we now have a two-byte representation of the Morse character, one containing the total number of elements and the other containing a one or zero (dash or dot, respec-

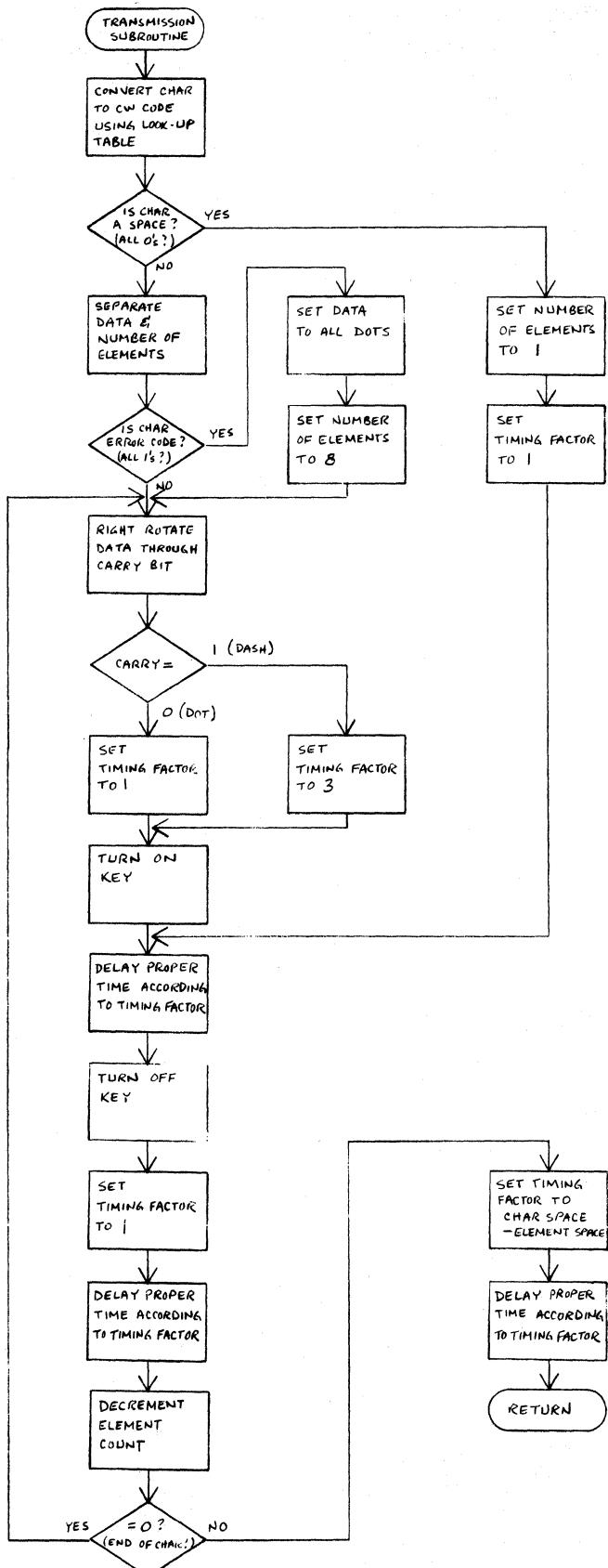


FIG. 3. TRANSMISSION SUBROUTINE.

tively) for each element.

Now let's rotate the data byte to the right by one bit through the carry (Fig. 4). This places the first element to be transmitted in the carry bit, which we test to determine whether we are to send a dot or a dash. If a dot, then we set a timing-factor register equal to one; if a dash, then we set the timing factor equal to three. In either case, we close the key at this point.

Next, we enter a delay loop which produces a time delay equal to the timing factor times the length of time it takes to transmit a single dot at the code speed we have selected for operation. We are assuming here that the "dot time" used in the delay loop has been specified and loaded into the computer by the operator some time previous to calling the generation subroutine.

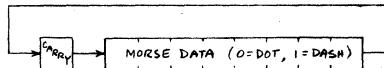
Upon exiting the delay loop, we open the key and set the timing factor to one. Then we enter the delay loop again, and generate an inter-element space.

Now that we have produced a single element of the Morse character plus the inter-element space following it, we check to see if there are other elements to be generated by decrementing the element count and testing to see if the count equals zero.

If it is not zero, we go back and rotate the data byte through the carry bit again and generate the next element.

Once the character is complete, the timing factor is set equal to the intercharacter space time minus one inter-element unit of time and enter the delay loop again. For perfect Morse, the timing factor would equal two at this point. However, if we want to transmit individual characters faster than the total CW rate, then a value greater than 2 would be used here. Upon completion of this final time delay, we return to the master program from the generation subroutine.

What do we do with space characters? First, we generate a "phantom" dot without closing the key by setting the number of elements equal to one and setting the timing factor to one. After producing this one unit time delay, we reenter the algorithm and produce one more unit time delay as though we had actually sent a Morse element. Then we decrement the element count, exit from the generation loop, and produce an inter-character space. For perfectly-sent Morse, we have produced a space equal to four "dot times". When added to the three-unit space produced after the previously-generated character, we find that we have produced a seven-unit space - exactly what is required for an inter-word space.



DATA IS ROTATED RIGHT THROUGH THE CARRY BIT, THEN THE STATE OF THE CARRY BIT IS TESTED TO DETERMINE WHETHER A DOT OR DASH ELEMENT IS TO BE GENERATED.

FIG. 4. MORSE ELEMENT GENERATION.
RECEPTION ALGORITHM

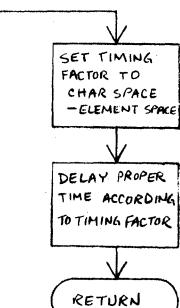
The task of Morse code reception is somewhat more difficult than the task of Morse code transmission, largely because of the irregularities and ambiguities associated with each operator's "fist". Let's face it, some hams transmit CW which is far from ideal in its timing. The algorithm we are going to look at allows for some irregularities in the received code. However, it just does not work as well as a trained human operator who knows the intended context of the received message. This algorithm automatically adjusts itself to the incoming code speed and provides a visual display of the received data if the structure of the incoming CW is reasonable good.

Hardware Requirements

The reception routine assumes that the received CW is available at one bit of a parallel input interface, where the state of the bit indicates whether a mark (key closed) or space (key open) is being received. This input bit can be derived from receiver audio by filtering and rectifying the audio or through a phase locked loop set up to detect the desired audio frequency. October 1976 Byte contains several circuits which should work fine.

Reception Routine

The reception routine contains several parts (Fig. 5). Let's begin by looking at the audio sampling routine. First, we wait for a period of time specified by a delay time value. This delay time is varied by another part of the algorithm and provides the timing necessary to automatically lock onto any CW speed. Then we sample the audio input and check to see if a change has occurred in the mark/space status. If there is no change from the previous sample, then we increment a value which we call the time count. This time count value indicates the



number of audio samples we have taken for the currently received mark or space. It is at this point that we send a data character to the output device if the device is ready and if there is data in our output buffer. We remain in this waiting and audio sampling loop until we detect a change in the mark/space status of the received audio. When a change is detected, we set a data ready flag and check to see if we have just completed a mark or a space interval. If the audio was a mark before the change, then the time count is stored in the space register. At this point, the new mark/space status is stored, and the time count value is set to one.

This time, as we go through start, the data ready flag is set. Now, if the element we have just received is a mark, then we apply three rules to see if the mark is dot or a dash. If the mark is more than twice as long as the previous mark received, then the mark is a dash. If the mark is less than half as long as the previous mark received, then the mark is a dot. If the mark is between half and twice the previous mark, then the mark assumes the same state as the previous mark.

Assume for now that the mark is a dot. We shift both the dot and dash registers (Fig. 6) to the left by one bit. (These registers are used later to determine the specific character we are receiving.) Then the dot register is incremented - effectively inserting a one into the LSB. Now we get to the part of the algorithm which adapts the delay time value to the speed of

the code being received. We have arbitrarily decided that the number of audio samples for a perfect dot would be about four. So we compare the mark time value which we have stored for this dot with the delay time value and see if we need to increase or decrease the sampling rate. If the mark time is less than three delay times, then the delay time is decremented to speed up the sampling rate. If the mark time is greater than five delay times, then the delay time is incremented to slow down the sampling rate. If the mark time is between three and five, we conclude that we are probably sampling at the correct rate and the delay time is not changed. Finally, the data ready flag is reset and we jump to the sampling routine.

So what happens if the mark is a dash? First we calculate the character space value which we let equal 3/4 of the mark time; this value is used later in the algorithm.

Then we shift the dot and dash register left by one bit, just as we did in the case of a dot, only this time we follow it up by incrementing the dash register rather than the dot register. Finally, the word space value, which is also used later, is calculated as twice the mark time, the data ready flag is reset, and we jump to the sampling routine.

Now suppose that a space has been detected rather than a mark. We compare the space time with the character space value stored previously. If the space time is less than the character space value, then we conclude that the space is an inter-element space, and we jump to the sampling routine after resetting the data ready flag. If, on the other hand, we find that the space time is greater than the character space value, then

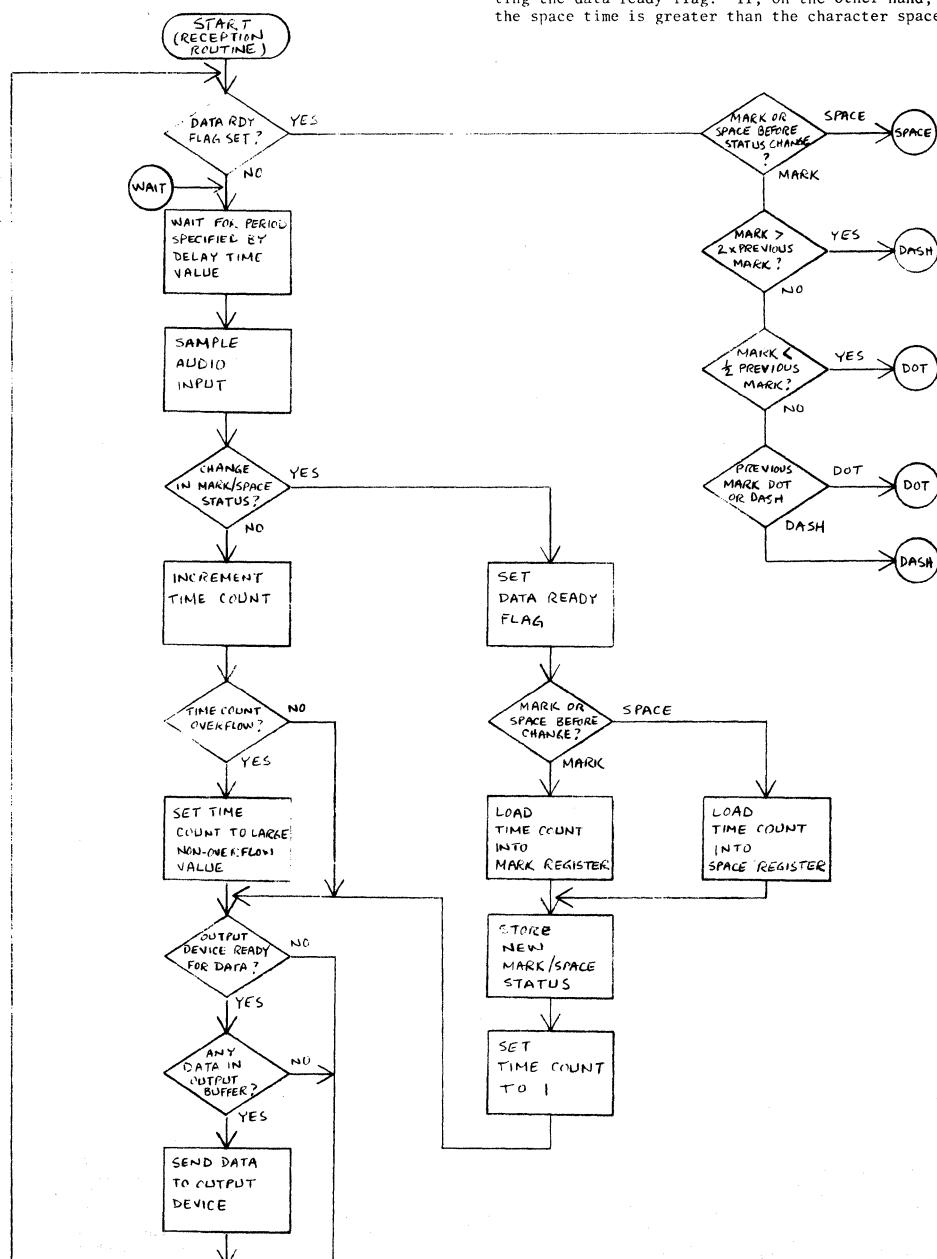
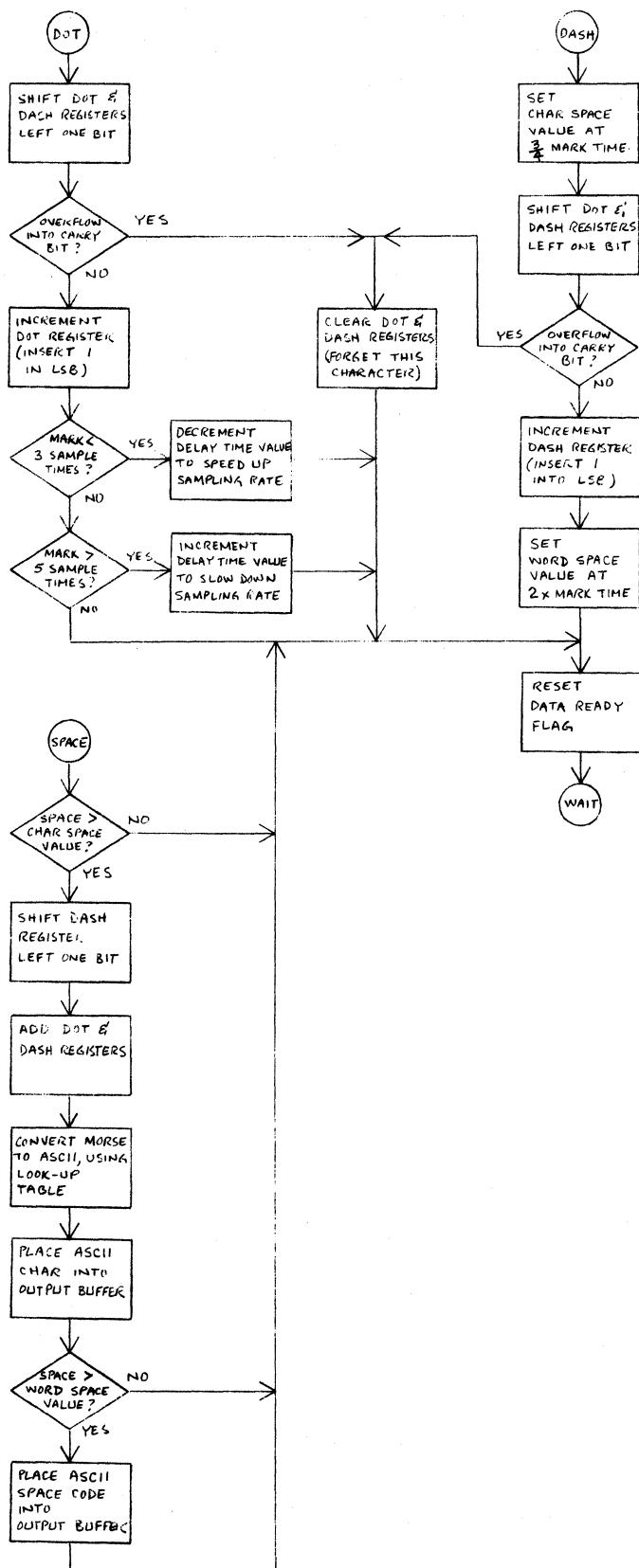


FIG. 5. RECEPTION ROUTINE.



we conclude that the character we have been receiving is complete and ready to be converted to ASCII. The conversion process begins by shifting the dash register to the left one more time, and then adding the dot and dash registers together (Fig. 6). The result is a binary value which uniquely represents a specific Morse code character. To convert this binary value to an ASCII code, a conversion look-up table is used. The ASCII code is then placed in the output buffer. As a final step, the space time is compared to the previously derived word space value. If the space time is greater than the word space value, then an ASCII space character is also placed in the output buffer before resetting the data ready flag and returning to the sampling routine.

SUMMARY

So there we have it, Morse code transmission and reception algorithms to help us combine our amateur radio and home computer hobbies. Of course, many additions and improvements can be made to these algorithms. How about combining a Sweepstakes serial number subroutine with the Morse transmission subroutine? How about automatic log keeping? Or a subroutine for use with the reception algorithm which looks for our call letters and records a message during our absence? If we had a transceiver whose frequency could be computer controlled, it might even be possible to let the computer run an entire contest for us! Have fun!

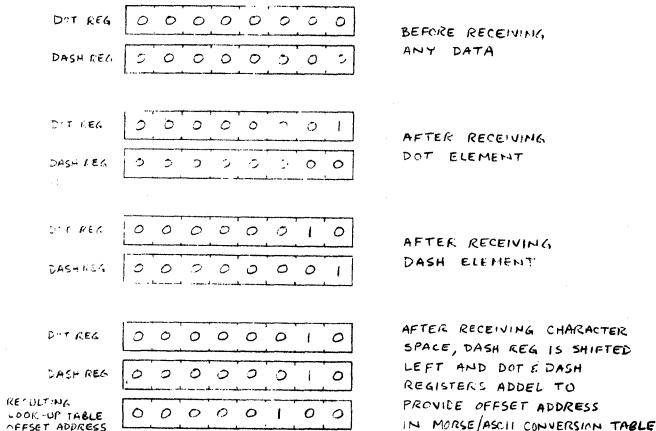


FIG. 6. STATE OF DOT AND DASH REGISTERS DURING RECESSION OF A MORSE "A" (- -).

MICROPROCESSOR CONTROL OF A VHF REPEATER

Lou Dorren, WB6TXD
107 W Bayshore Blvd
San Mateo CA 94401

In most repeater systems, control functions are generally performed by hard wired integrated circuitry. This means that each function is managed by a group of integrated circuits whose specific job is to do that function only. In a repeater system, with a reasonable number of controls performing very simple management functions, as many as 60 to 70 integrated circuits could be required. Most such systems use a touch tone decoder and some form of logic that decodes the various control numbers and turns on function outputs to operate the various controls of the repeater. Changes in these systems require a redesign of the circuit or a change in the output wiring.

With the advent of microprocessors a whole new era of repeater control and management becomes possible. Here we will describe a repeater control system built for WR6ABM around a Motorola 6800 microprocessor. This repeater control system performs all of the supervisory and management functions using a software system instead of the hardware systems described above. In this repeater control system, the microprocessor, which is essentially a computer, has a series of outputs connected to the repeater controls. Signals appearing at the inputs of the microprocessor program it to perform the various test and control functions desired.

This system can be described according to the functional blocks which do the work. Figure 1 shows the system block diagram. First is the central processor unit (CPU), which contains the microprocessor integrated circuit, a scratch pad or working stack memory, a monitor or control memory and an asynchronous communications device for talking to the data terminal or teletype machine. The CPU is contained on a single printed circuit card which is interconnected with the other cards in the system via plug-in sockets held in a standard card frame. The microprocessor card used as the CPU in this system is the MADIC, manufactured by the Seiveteck Corporation of San Mateo, CA., who also make the other cards used in the system.

The second functional block is the random access memory. In this case, we used 8,000 words of random access memory, contained in two printed circuit cards. Each card consists of 32 1Kx1 static memory chips, as well as the address decoding ICs and the data and address line buffer ICs. In this memory is stored the operational program for the repeater. Changes to this program are effected through the data terminal or teletype, making it unnecessary to rewire the circuitry in order to change control functions. All such changes can be made over the air, or by phone line.

The third functional block is the peripheral interface adapter (PIA). Each one of the PIA cards serves some input or output function of the repeater. For instance, there is one card called the COR PIA. This card connects the receiver COR lines and the transmitter key lines to the CPU. It tells the processor when a receiver is keyed up and also keys up the transmitters as instructed by the CPU.

The touch tone decoder and real time clock make up the fourth functional block. Both are contained on the same card. A hybrid touch tone decoder built around a Telenetics 7516 and a real time digital clock built around a National MM5309 are used. Also contained on this card is a touch tone encoder for outputting high speed touch tone commands and data and for self testing. This utilized a Motorola MC14410. The decoded touch tones are presented to the CPU for appropriate action, while the real time clock permits the processor to time events and make calculations involving the time of day.

Fifth in the series of functional blocks are the audio matrix cards. Each card consists of a 8 x 4 audio matrix with input and output buffers. The audio matrix is connected to the audio outputs of every receiver and tone generator and to the audio inputs of all the transmitters, local speakers and telephone lines. The audio matrix can be commanded by the audio tone DVM PIA to switch between the various inputs and outputs.

The tone generator, contained on a single card, is the sixth block in the system. It contains a pair of digital synthesizers for the generation of sine wave tones. It also contains a three and one-half digit digital voltmeter which is commandable and which can be used for telemetry data taking by the CPU. The synthesizer and voltmeter controls are fed from their own PIA registry. The audio matrix is also controlled by this PIA.

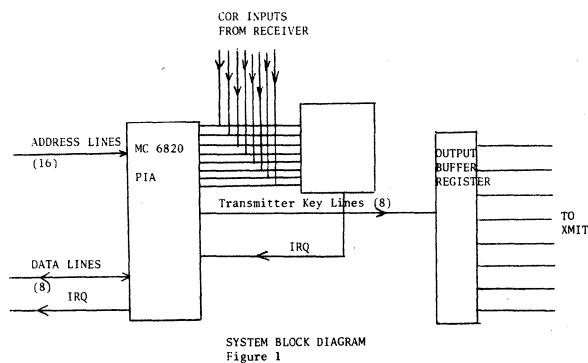
Last in the series of functional blocks is the system clock. This clock, located on its own card, gives the necessary non-overlapping phases for operating the microprocessor. It operates at 1 MHz. Additionally the clock card contains a baud

rate generator to generate the necessary frequency for the teletype interface and an interrupt request time generator which produces a 50 Hz signal used during various supervisory functions of the repeater. The clock for the audio synthesizer and DVM are also supplied by this card.

Unlike most repeaters, there is no COR delay sensor or detector and there is no transmitter key with hang time or identifier. All of these functions are in the software that is generated with the microprocessor. As an example, if the command has been entered in the repeater to permit using the VHF transmitter, the microprocessor is constantly waiting for what is known as an interrupt signal. When this signal appears, the microprocessor looks at all of the PIA's to determine which one requested the interrupt. If the COR PIA is the one requesting the interrupt, the microprocessor then looks to see which receiver was keyed up. If it finds the VHF receiver keyed up, it then looks in the memory to see if all the necessary parameters are present to permit keying up the transmitter, i.e., has the repeater on command been entered. If all the conditions are fulfilled in the software of the microprocessor it keys up the transmitter and continues to keep the transmitter keyed up until the receiver drops. When the receiver COR drops, the microprocessor calculates the hang time before it drops the transmitter so that the transmitter stays up for a pre-determined amount of time by software.

The microprocessor lends itself to many other aspects of repeater control. Identification and Morse code generation are some of the more interesting aspects. The Morse code is generated by an algorithm in the microprocessor memory. Each character of the Morse code is encoded into an 8 bit word. With this type of software, any CW message can be generated. In addition, the CW messages can be changed without having to do reprogramming or rewiring of ID memories. It can be changed merely by changing the software, as can all the other repeater functions. With the microprocessor control system, the level of sophistication of repeater control is limited only by the programmer's ability and the amount of memory available.

In the WR6ABM system, the computer performs over 100 different functions at any given time. For example, the computer executes 20 thousand instructions to output an identification. All of the timing for the Morse code CW is determined by the microprocessor so that its rate and weight can be modified. One of the commands that exists in the WR6ABM computer is what we call the Banana Boat Special - Morse code with a slightly Southern accent. Block diagrams of the various circuit cards are shown in Figures 2 through 4. As can be seen, the microprocessor opens a whole new era of FM repeater control, allowing increased capability and functionality as well as individualism in design, together with the advantages of remotely changing the capabilities of the repeater by software. It also brings to the amateur knowledge of a new and very exciting field of semiconductor electronics, the microprocessor.



AMATEUR RADIO AND COMPUTER HOBBYIST LINK VIA RTTY REPEATER

Alan Bowker and Terry Conboy

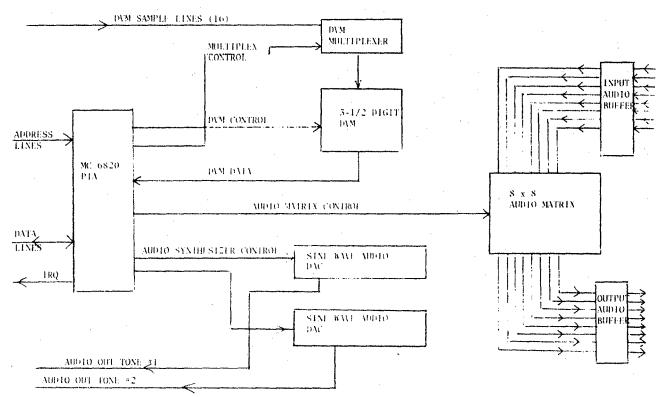


Figure 2

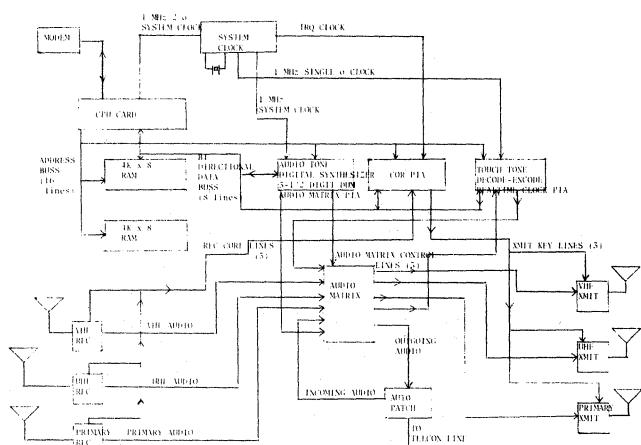
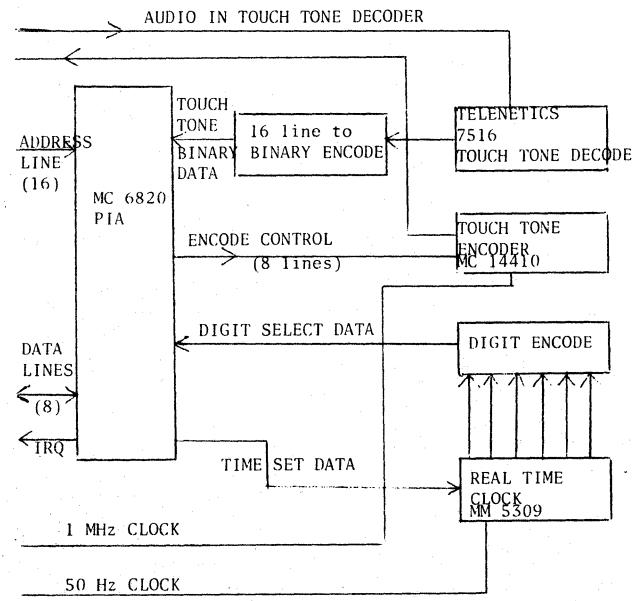


Figure 3



TOUCH TONE CLOCK
Figure 4

Many people interested in computers and communications wonder where a good communications link is that can provide an information interchange between computer hobbyists as well as those just interested in communicating.

The answer is the growing use of the amateur radio teletype repeater WR6ACR. This repeater is utilized for the promotion of radio teletype as a means of communications for anyone who is interested. And lately the interest has been computers!

The repeater is located on San Pedro Ridge just north of San Rafael. The frequency is 147.93/147.33 MHz. Those wishing to use the repeater should use narrow band FM transmission with a two-tone audio keying shift of 185 Hz. (2125-2290). Since the signal is FM, signals from miles around will get through perfect copy almost all the time.

A simple T.U. (terminal unit) that decodes the two tones and keys the printer loop supply is available at low cost. The unit consists of a phase lock loop tone decoder, a function generator to create the two tones when keyed and associated circuitry for autostart.

Printers are readily available and very low cost. A Teletype Model 15 or 19 is available for less than \$100 and can be used with your computer providing a conversion program is used to convert from ASCII to the Baudot code which is used exclusively on amateur radio.

Should you want more information call: Alan Bowker at (415) 453-1853, San Rafael, or if you live on the Peninsula call Terry Conboy at (415) 364-3107, Redwood City.

THE NEW MICROPROCESSOR LOW COST DEVELOPMENT SYSTEMS

Phil Roybal
M/S 470
National Semiconductor
2900 Semiconductor Drive
Santa Clara CA 95051

While microprocessor prices have dropped dramatically in the past few years, the cost of the prototyping hardware that the user needs to get his design off the drawing has stayed relatively fixed. Since most sales were to electronics companies who were used to paying thousands of dollars for a piece of lab equipment, the market was not price sensitive. That is, the people who needed the gear would pay the price, and those who didn't need it wouldn't buy and if the price were cut in half.

Then, about a year ago, a dramatic change began to occur in this market as the plunging prices of microprocessor components finally brought them into the range of hobbyists and non-traditional "springs, levers, and gears" users. These groups were both characterized by the fact that the old prototyping approach was not suitable for them: they either could not or would not pay for the traditional microprocessor prototyping tools. Yet it was still necessary for these users to go through the same design steps that engineers had always gone through in the development of their systems. Without access to large prototyping equipment, the user was forced to start with a sack of chips and to build his own prototyping system at the very time that he was least equipped to do that. There had to be a better way, and the product that brought it came out of an analysis of what actually happens in the design process.

Microprocessor system design involves two separate phases: design of the application software, and design of the interfaces that connect the processor to the rest of the system. In the past, there were only two real approaches to these tasks for the engineer: He could take the Cadillac approach (if he could sway management), and invest in a vendor's prototyping system (Figure 1). This set him back several thousand dollars; but it gave him everything he needed to operate the processor, as well as a fair amount of software and a variety of supported peripheral devices. But it required a big investment for a small company or one not yet sure that microprocessors really were the wave of the future.

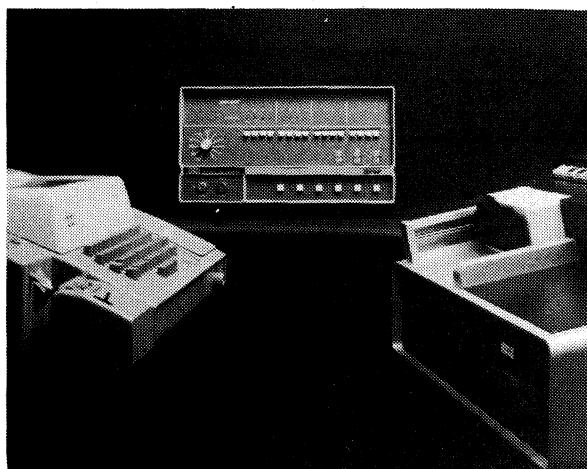


Figure 1. A full-scale Prototyping System in the \$10,000 price range.

If an engineer couldn't sell the full system idea, there was always the chip kit, offered by most microprocessor vendors to introduce their products. On the surface, it seems to offer most of what a designer needs to build up his prototype system. But let's look at what the engineer must do with those parts, and we'll see the shortcomings of the typical kit as a design tool.

Steps In The Design

The engineer normally approaches his design in the following way:

- o Layout the system elements

- o Interface the processor to the devices to be controlled
- o Create a small (1-2 step) program to move data between the processor and the external device, to verify the interface
- o Write a driver program, defining how the processor is to operate the device
- o Debug the driver
- o Write programs around the driver, to process the data and make the decisions about when and how the device should be used
- o Debug those programs.

As you can see, 85% of the work is software-oriented; and chip kits don't offer enough capability to help much. A typical kit is pictured in Figure 2. Although the kit offers the processor, support logic, memory and interfaces, there is no easy way to communicate with it.



Figure 2. The SC/MP Kit: Low-Cost and a good value, but how does the user talk to it?

Most manufacturers get around this problem by programming a ROM memory with a TTY driver and debug routine, so that a user can hook up a TTY to the kit and enter his program through the keyboard. Unfortunately, this only helps those who have terminals at their disposal. For those who don't, purchase of a TTY adds an extra \$1000 to the cost of a \$250 kit (and a year's lead time as well, waiting for the TTY).

An alternative approach for the kit user is to get himself a PROM Programmer, and then to write his program directly in the PROM. The PROM is then removed from the programmer, tried in the circuit, and hypotheses are made about why the program doesn't work. Corrections are devised, the PROM is erased, the new program is inserted, tried again, adnaseam. A tedious approach, because of the lack of ability to manipulate the program as it operates, quickly trying out patches until proper operation is achieved.

Is the user then stuck with a large initial investment to get into microprocessor design? Well, nothing will completely replace the big prototyping systems for users with a lot of hardware and software development to do. But many a user finds the big systems an overkill, at least at the beginning. Often, they offer more features than he can use. Or, he finds he could use several systems, so that different groups could work simultaneously. But his needs are bigger than his budget. What he needs is a basic tool that he can expand as his needs and budget expand.

In short, he needs

A Low Cost Development Tool

A designer needs the following minimum capabilities in a microprocessor prototyping tool:

- o Access to data and control lines from the processor
- o Ability to enter simple programs (preferably in octal or hex code) directly into main memory
- o Ability to manipulate and display the program, to control program execution, and to easily change the program as bugs are discovered

The preceding features might be considered the bare minimum for a useful development tool. In addition, the following features can greatly increase the usefulness of the tool:

- o Ability to expand the prototype in an orderly way to encompass all memory and interfaces required in the final application
- o Ability to interface a terminal for full keyboard input and hard copy output
- o Ability to edit and assemble source programs
- o Ability to interface inexpensive bulk storage (cassette, paper tape, etc.) for rapid program loading and saving
- o Ability to use a high-level language

All this capability must be packaged compactly enough to fit on a crowded bench. It must be rugged enough for continuous, reliable operation, and it must be relatively low in cost. Up to this time, no major microprocessor vendor has come up with a tool that meets all these criteria. Most of the kits on the market leave the engineer with a handful of hardware, a lot of questions, and not a little frustration. It was in response to this frustration, expressed both by users and by our own development engineers, that National designed the Low-Cost Development System (LCDS). The first LCDS has been built up around the new SC/MP (pronounced "scamp") microprocessor.

The SC/MP LCDS

The SC/MP LCDS, pictured in Figures 3 and 4, provides all of the basic features a designer needs. It gives him the tools to develop small software programs in a compact, low-cost package. And it is readily expandable to handle more complex system tasks involving many interfaces and large amounts of memory.

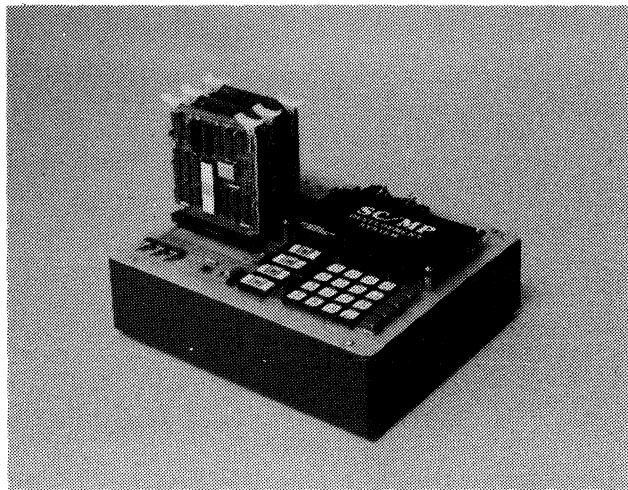


Figure 3. SC/MP Low Cost Development System (LCDS) as it would be set up to run SUPAK or NIBL Firmware Packages.

The LCDS consists of the following elements:

- A. Motherboard, which functions as both backplane and card cage for application cards, holds all control circuitry, and provides:

- o 16-key pad for data/command entry
- o 7 switches or keys for system control
- o 6 LED digits for data/status display
- o 20 mA current loop terminal interface
- o Firmware to control preceding items
- o Debug firmware for both stand-alone and terminal operation

- B. CPU/MEMORY CARD, containing the SC/MP microprocessor, support logic, 256 bytes of RAM, and socket for 512 bytes of ROM or PROM.

- C. Chassis, to support the motherboard.

This configuration, which comprises the minimal LCDS, may be extended by addition of memory and I/O cards which plug into the extra slots on the motherboard; and by means of an external card cage or other device, which connects to the data and control busses of the motherboard through a ribbon cable. To keep the system compact and inexpensive, +5 and -12 volt power is drawn from standard lab supplies. The system is otherwise self-contained.

So what can you do with it? Let's look at it in operation.

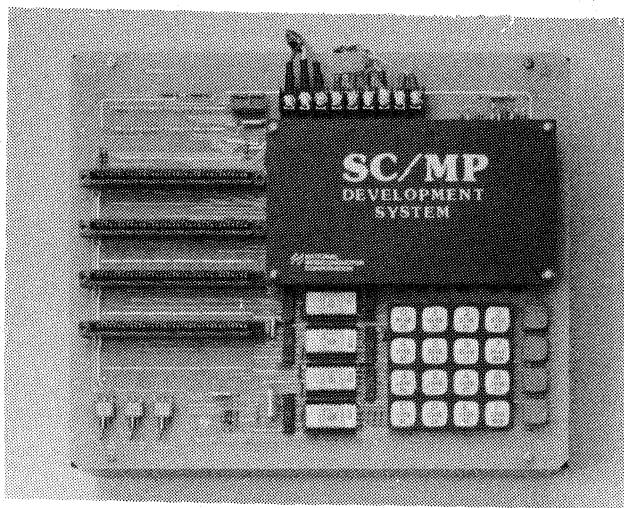


Figure 4. SC/MP LCDS: Details of controls and Data Entry Keyboard

To use the LCDS, the engineer first builds up an interface to the device to be controlled, using small wire-wrap cards which plug directly into the LCDS motherboard. He connects up the LCDS, powers up, and resets the system with the INIT key. Now he's ready to begin program development.

Taking the program that he has written in hex using the SC/MP Pocket Instruction Guide, the user enters his code into RAM on the LCDS through the keypad as follows:

KEY	DISPLAY	OPERATION
MEMORY ADDRESS	---- --	Put system in Memory Address mode.
0-F (4 digits)	XXXX YY	Enter address of start of program. Display shows the address entered (XXXX), and current contents of that location (YY).
F/LOAD MEM	XXXX--	Put system in data mode.
0-F (2 digits)	XXXX ZZ	Enter word of new program into RAM. Memory address increments automatically, and new

KEY	DISPLAY	OPERATION	Trying It Out
XXXY--		address is displayed as a prompt.	

The user continues in this manner until his program has been transferred to the RAM memory. He then presses 8/LOAD PC, enters the program's starting address, and presses RUN. The program will execute until the HALT or INIT keys are depressed or until it encounters a HLT instruction, at which time the display will show the contents of the Program Counter. In the SINGLE INSTRUCTION mode, the processor can be stepped through the program one word at a time to assist in debugging. When the system is in the operator mode (as opposed to the program execution mode) any memory address or processor register may be examined or altered.

Designing a Simple Control System With The SC/MP LCDS

To explore the process of developing a simple system around the LCDS, let us take the elementary example of a timer system, such as would be used in simple process control or home system control applications.

In our application, we wish to activate a relay for a period of about one second each time a delay is required. Let's see how our designer would go about building us such a system.

Building The Interface

The first step is to interface the processor with the item to be controlled: a relay. The SC/MP microprocessor has a special instruction for handling delay functions, and this instruction controls a line on the bus called DELAY. During a delay operation, this line goes high. At other times, it is low. We will use this as our control line, and interface it to the relay through a very simple driver circuit, as shown in Figure 5.

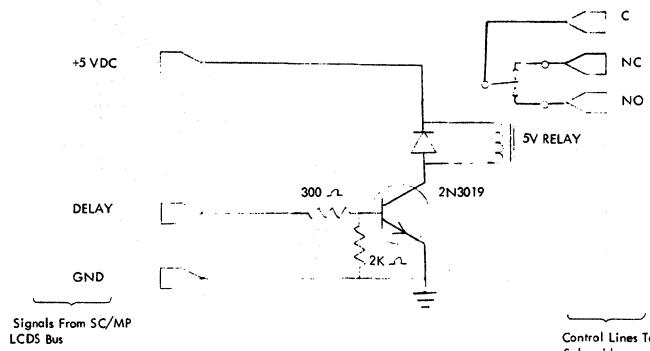


FIGURE 5
A simple relay interface for use with SC/MP LCDS.

Writing The Driver Program

Our delay program will be extremely simple. Because of this, we can indulge in sloppy programming practice and omit a flow chart. Our program will be activated by the user hitting the RUN button. It will turn the relay on for about one second, turn it off again, and halt. The program code looks like this:

OPERATION	CODE	COMMENTS
LDI 255	C4 FF	Load Accumulator With 255 (FF)
DLY 255	8F FF	Delay For .263 Seconds Assuming 4us Cycle Time
DLY 255	8F FF	Delay For .263 Seconds
DLY 255	8F FF	Delay For .263 Seconds
DLY 255	8F FF	Delay For .263 Seconds
HALT	00	Halt The Processor

Once the program is written, the designer goes to the LCDS and enters the program into RAM through the keyboard. Since the above program contains no labels or branches, it can go anywhere in memory without re-writing. The engineer simply selects a starting location and enters each byte of code consecutively into ascending locations. Once the code is entered, prudent practice dictates a quick check, so we go back to the starting location and step through the program, to check each memory cell.

When the program looks good, we insert our relay card into a slot on the LCDS motherboard, and we are ready for program evaluation. Cross your fingers, hold your breath, and press RUN. If things go according to plan, the relay will close for 1.052 seconds, open again, and the processor will halt.

And Then . . .

We now have a very simple delay routine which can be used as a subroutine for any program that needs it. By putting different values into the second byte of the instruction, or into the Accumulator, we can change the delay time. By putting a 3-to-8 decoder between the DELAY line and the transistor, and controlling it with SC/MP's user flags, we can have an 8-station timer. And expansion can go on and on, limited only by the imagination.

In this application, the SC/MP LCDS provided a convenient test bed with all the communication, memory, and buffering circuitry in place, ready to work on the user's application. The alternative would have been to build those functions from scratch - a tedious prelude to doing some useful work.

Expansion of The System

Those with a terminal available can get even more capability from the LCDS. By connecting an ASR-33 TTY to the current loop interface, a user can control the system from his keyboard and get a hard copy record of each of the above transactions. Additionally, he can set breakpoints for easier program debug, print the contents of blocks of memory, and load and punch paper tape for program storage. With the addition of a terminal, the user can now employ one of the standard SC/MP cross assemblers. He can write his program in assembly language, and the cross assembler will translate the program, give him an error listing, and punch the binary output into paper tape for loading into the LCDS through the terminal.

For those users who would like to expand their LCDS further and achieve complete independence from cross assemblers on larger machines, there are two software packages available for use on any standard LCDS equipped with at least 4K bytes of extra ROM or RAM storage.

The first expansion package is SUPAK. This package contains;

- o paper tape editor
- o interactive line assembler
- o P/N tape punch program

Using SUPAK, a programmer can now write source programs with the editor in standard SC/MP assembly language. A pass through the line assembler converts source code into machine language, which may then be run immediately on the LCDS. If the program operates properly, the code may then be output as a P/N tape, ready to feed into a PROM programmer. All source code written for SUPAK is fully upward compatible, so that it will assemble on the large macro cross assemblers without error.

The second package is NIBL, National's Industrial BASIC Language. This is a subset of Dartmouth BASIC, with special extensions to make it suitable for writing control and I/O programs. Since NIBL is implemented as an interpreter, it offers the advantages of instant error feedback and source code correction. It is easy to read and write, and many engineers find it ideal for verifying program logic, writing quick test programs, or even for incorporation into the final system.

For the hobbyist, NIBL has the advantage of being sufficiently close to BASIC that most of the games in the public domain will run on the system with little or no modification. Thus he can start with a small, inexpensive system, build it up to run games and entertain him, yet have the

power for real system development when he gets ready to have the processor actually do something.

Conclusion

SC/MP LCDS offers the designer or experimenter a convenient tool for the development and debug of simple control programs. Although the basic system is quite compact and inexpensive, it offers all the features that many users will require for their prototype developments. Yet, it expands readily through the addition of RAM, ROM/PROM, and interface cards to handle complex system problems. Additionally, since both the build-in keyboard and display are available to the user's program, he can actually incorporate the LCDS right into his prototype if he desires, saving both interface design and hardware fabrication time. Finally, because the LCDS allows the use of large amounts of memory (to 64K bytes), a terminal, and bulk storage, it encourages the development of resident software by the experimenter. Games, small operating systems, and even high-level languages may be implemented on the LCDS, facilitating its use as an educational tool or a home control system.

Although manufacturers will continue to offer kits as merchandising tools for their processors, a growing body of users will realize the limitations of these kits for serious development projects, and will demand more capable tools. A large percentage of these people cannot or will not buy the larger prototyping systems. Or if they do, they will still encounter a need for smaller systems in the lab for interface and program debug, while the big system is dedicated to major software developments. For this body of users, a Low Cost Development System is the ideal answer, since it gives the engineer on the bench the capabilities he needs, at only slightly more cost than a kit of parts.

A MEGABYTE MEMORY SYSTEM FOR THE S-100 BUS

Glenn E. Ewing, Senior Engineer
IMSAI Manufacturing Corp.
14860 Wicks Blvd.
San Leandro CA 94530

Abstract

The 65-kilobyte address space supported by the S-100 bus is inadequate for some applications otherwise within the capabilities of 8080/8085 microprocessors. The number of S-100 boards required to implement larger memory systems has also been a limiting factor. This memory size restriction is eliminated by the IMSAI "Megabyte Micro" memory system. It can be implemented within a single mainframe in any multiple of 16K bytes. The system consists of 16K, 32K and 65K byte memory boards and a memory management/interrupt controller board (IMM). The memory boards are of conventional dynamic design and can be used alone to implement memories of up to 65K. For larger memory systems the IMM board is required. The IMM expands the address space by driving four additional address lines (for a total of twenty) which at any one time maps four memory blocks of 16K bytes each into the CPU's 65K byte address space. The IMM maintains 16 software-presetable memory configurations to allow rapid reconfiguration in multi-task applications. It also provides for subroutine calls and returns to/from other configurations, read protect, and two types of write protect. The interrupt controller functions include vectoring to any memory location in the megabyte address space with automatic system reconfiguration and automatic CPU state save/restore. A real-time clock/rate generator and a "time of day" clock are also included.

A NEW APPROACH TO MICROCOMPUTER SYSTEMS FOR EDUCATION

Alice E. Ahlgren, Ph.D.
Cromemco, Inc.
2432 Charleston
Mountain View CA 94043

ABSTRACT

This paper briefly examines the future role for microcomputers in education and describes a unique, time-sharing, microcomputer system developed by Cromemco -- the Z-2 Education System -- to assist educators in obtaining cost-effective, computer hardware designed for classroom use. This new system provides a combination of economic and practical advantages not previously available through either large, centrally-located, time-sharing systems or through small, dedicated, single-user systems. The Cromemco Z-2 Education System requires low capital investment, provides time-sharing capabilities to support multiple users, provides classroom access to computer hardware, and provides the economy of shared resources. The Z-2 Education System is a special configuration of Cromemco's general purpose Z-2 computer system. Specific features are described in detail.

INTRODUCTION

The microcomputer industry has undergone phenomenal expansion in the past year. Until recently, however, the industry has focused largely on the industrial and home/hobbyist markets and has all but ignored one of the largest potential markets in the United States -- the education market. The reason for this failure is that neither educators, nor hardware manufacturers and computer scientists who entered the educational marketplace, really envisioned providing students with personal access to a computer. Yet, the availability of low-cost computers with increasingly smaller dimensions and with lower power requirements is possibly of greater importance to education than the advent of large computers has been. Unfortunately, the current educational effort is not sufficient to prepare students for this kind of advancing technology.

There is little doubt that the computer is an appropriate and valuable educational tool. However, what is really important in the implementation of new technology is both economic viability and practicability. Economic viability requires no explanation. The practicability of a piece of technology involves considerations "such as convenience and accessibility, equipment reliability, acceptability to the people who must use it, training required for its effective use, and cost relative to that of other appropriate alternatives." [1] Of course, improvements in technology will be of limited value unless they are matched by improvements in software and system design. The software development problem is, however, outside the scope of this discussion.

COMPUTER ALTERNATIVES FOR EDUCATION

Recently, educators interested in providing computer-oriented instruction have had only two alternatives -- use of large, centrally located, time-sharing systems or use of small, dedicated, single-user microcomputers. Both alternatives have serious economic and practical limitations. A centrally located, time-sharing system requires a large capital investment and also isolates students from the computer hardware -- which endows the computer with a mysteriousness which is undesirable. Small, dedicated systems have the advantage of avoiding both these problems. However, these single-user systems restrict the number of people who can interact with the system and, in addition, prohibit economies which can be gained (especially in classroom settings) through the use of shared resources.

Now, however, as a third alternative, Cromemco has developed a new microcomputer system which provides a combination of economic and practical advantages from both these approaches. This new, dedicated microcomputer system -- the Z-2 Education System -- requires low capital investment, provides time-sharing capabilities to support multiple users, provides classroom access to the computer hardware, and provides the additional economies gained through the use of shared resources. Thus, the Z-2 Education System (which is a special configuration of Cromemco's general purpose Z-2 computer system) was designed to meet a number of requirements

of school districts and colleges interested in using the microcomputer for instructional purposes.

As mentioned previously, specific major stumbling blocks, involving hardware, to the introduction of microcomputer systems into school districts have included equipment costs, limited system expansion capabilities, and the inability to provide access to more than one user at a time. Equipment costs have been of particular concern to educators operating with limited budgets. The purchase of a large, time-sharing system represents a major investment for most schools. Furthermore, the cost of upgrading the system to serve more users is frequently prohibitively expensive. For example, the San Jose School District, in response to a growing demand for computer facilities, investigated the possibility of upgrading their PDP-8E system. School District personnel discovered that close to \$50,000 would be required to upgrade the DEC system to 16-20 users, which far exceeded any support the District could expect. [2]

The District's solution to this problem was to purchase microcomputers, with a long-range goal of one microcomputer per school. The limitations of such a solution are immediately apparent, however. First, student access to the machine is severely restricted since only one student at a time can be scheduled to use the computer. Yet, hands-on access for each student is essential for learning about both computer hardware and computer programming. Thus, to effectively increase access to the machine, the school district would need to purchase additional microcomputers for each school. Such expenditures represent a fairly high equipment cost per student.

THE Z-2 EDUCATION SYSTEM

Cromemco's response to this problem has been to develop a time-sharing microcomputer system -- the Z-2 Education System -- which permits up to eight users to effectively use the computer almost simultaneously. The Z-2 is also designed to be completely expandable in order to meet the specific applications required in classroom situations. For example, one Z-2 microcomputer can support eight users and, if desired, a floppy-disk drive, a line printer, and an audio cassette interface. The resultant cost per student is significantly reduced while increasing the power and versatility of the system, increasing student access to the computer, and consequently increasing the instructional effectiveness of the system.

The expanded user access available through the Z-2 Education System is provided through a unique multi-tasking approach. Multi-tasking in the Z-2 Education System is accomplished both through a control program which selects which task to execute first and through Cromemco's unique bank select feature which provides the expanded memory capability required for multiple users. Essentially, the Z-2 system provides eight independent banks of memory, each with a capability of up to 64 kilobytes. For educational uses, each bank is assigned to one user who interacts with the computer through a remote CRT terminal or teletype terminal. Using the bank select capability, the control program has access to all eight banks. The control program then provides a prioritizing scheme which controls task execution.

In addition to providing access to multiple users, the Z-2 Education System has a number of other features which make it particularly suited to classroom use. These features include a fast, powerful CPU card based on the Z-80 microprocessor; a motherboard with 21 card sockets which allows for easy system expansion; and an extremely heavy duty power supply capable of meeting virtually any power need.

As stated previously, a significant feature of the Z-2 Education System is that it uses the Z-80 microprocessor, generally considered to be the standard of the next generation of microprocessors. The Z-80 has much more power and speed than previously popular chips such as the 8080 or 6800. It is also important that the new Z-2 uses an especially fast version of the Z-80, one having a 4 MHz speed (250-nanosecond cycle time). The higher speed (and recognized much more powerful instruction set) of this microprocessor means, of course, more throughput for the Z-2 than for any other present microcomputer. Such higher throughput is especially significant in a dedicated computer since, for example, it

often means that the system can do real time operations that until now were the province of much larger and much more costly computers. The higher throughput also means that the new Z-2 is ideal for the multi-user setup required for effective classroom use. Furthermore, although designed to work at 4 MHz, the Z-2's CPU card also has a 2-or-4 MHz clock rate switch so that the Z-2 can be used at a 2 MHz clock rate for compatibility with existing slower peripherals.

In addition to the powerful CPU card provided, the Z-2 system also uses the industry standard S-100 bus which gives wide peripheral compatibility. Consequently, the Z-2 system can support a number of peripherals which are of particular interest to educators -- such as a color graphics interface which permits an ordinary color TV set to be used as a full-color graphics terminal.

Since the Z-2 allows wide peripheral compatibility, the power supply was designed to provide abundant power for all foreseeable combinations of circuits and peripherals. To this end, the Z-2 provides 30A from 8V and 15A from both +18 and -18V.

In addition to providing such features, a microcomputer system designed for classroom use must be highly reliable. The need for reliable operation is particularly important to educational users who are relatively unsophisticated about computer hardware and who cannot afford large amounts of down-time during school hours. A number of features of the Z-2 were designed to provide this kind of reliability.

First, the front panel of the computer is totally free and clear of controls or switches of any kind. As a result, the computer is immune to accidental program or memory mishaps which might be caused by accidental switch flipping -- an occurrence particularly possible in a classroom situation involving multiple users with varying degrees of knowledge about computer hardware.

A mechanical feature that enhances operating reliability is a sturdy card retaining bar in the Z-2 that keeps circuit cards firmly in their sockets despite vibration, mechanical shocks, and other types of impacts. When the retainer is in place, cards simply can't be jostled out of their sockets.

Another important feature of the Z-2 is the unique ground-plane design on the 21-slot motherboard. The design reduces bus cross-coupling and ground-current noise by several decibels over conventional designs.

A strong convenience for instructional situations is the power-on, memory-jump feature which begins automatic program execution when the power is turned on.

CONCLUSION

The Z-2 Education System is only a first step toward Cromemco's goal of providing high-quality, cost-effective, reliable equipment to educational users. Certainly, the impact of microcomputers on education is just beginning. Perhaps two million elementary and secondary school students use computers. Yet most of these students have seen the computer only as an instructional tool for drill and practice tasks or as a problem solving tool using Basic. Opportunities for pursuing more creative and interesting tasks have been limited by the lack of access to computer equipment. Still, school districts are demonstrating an increasing awareness of the need for instructional programs in both computer programming and computer science. For instance, computer concepts are already rapidly invading the mathematics curriculum in a number of school districts in the Bay Area. In the San Jose School District, computer programming electives will be added to the curriculum within the next few years. The speed and extent to which these objectives can be accomplished, however, will depend in part on the quality, availability and cost of computer hardware designed for the education market.

ABOUT THE AUTHOR

Dr. Ahlgren received her Ph.D. in Communications from Stanford University. While at Stanford, she was in charge of document processing for a publication of the ERIC Clearinghouse on Educational Media and Technology. Since 1974, she has been responsible for the coordination and management of policy-related communications research. She has participated in studies evaluating the use of information

retrieval systems in public libraries, evaluating biomedical communication experiments, evaluating the effectiveness of cancer public education programs and examining academic employment opportunities for minorities in the field of communications. Dr. Ahlgren joined Cromemco, Inc. as Marketing Manager in 1977.

REFERENCES

- [1] Miller, Richard H., "The Prospect for Technological Innovation in Scientific and Technical Communication," (draft copy), Applied Communication Research, Inc., Palo Alto, California.
- [2] Grimes, Peter S., "San Jose Schools Discover the Microcomputer," (draft copy), San Jose Unified School District, San Jose, California.

A COMPUTERIZED PROM PROGRAMMER, PROM EMULATOR, AND CROSS ASSEMBLER SYSTEM

Richard Erickson, President
Sunrise Electronics
228 N. El Molino
Pasadena CA 91101

Microprocessors are finding a wide variety of applications. Upwards of \$250 million worth of microprocessors will be consumed in 1977 with an annual growth rate approaching 50%. The speed, flexibility and data handling power of the microprocessor is being used to make machines which are more intelligent and consequently more useful.

It is appropriate that one such application should be an intelligent PROM programmer, emulator, universal cross assembler and program editing machine which is used to develop and produce microprocessor based equipment. PROM programmers have traditionally filled the single need of loading programs into PROMs. PROM emulators have been sold separately and editing aids have been part of an expensive development system dedicated to a single type of microprocessor. Assemblers and cross assemblers, which convert assembly language code into machine code, have always required an expensive development system or a time sharing service for their use.

SYSTEM STRUCTURE

Combining these four functions into one machine, shown in Figure 1, takes full advantage of the microprocessor.

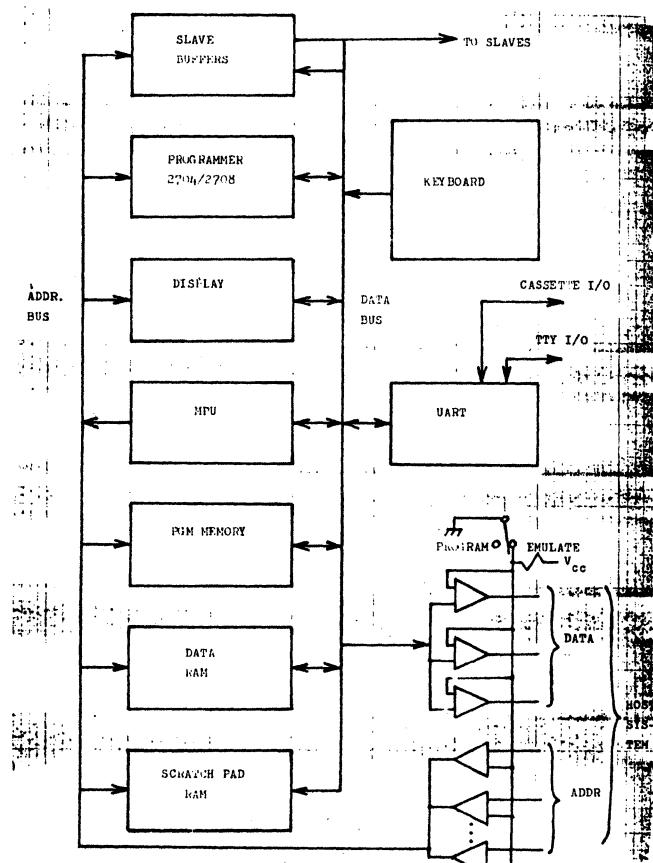


FIG 1.

System Block Diagram. The components of the system are the MPU, Program Memory, Data RAM, Scratch PAD RAM, Emulator Buffered Outputs, UART/TTY Driver and Cassette Driver, Keyboard and Control Switches, Display 2704/2708 Programmer and Slave Expansion Plug.

The functional blocks of the system are the MPU, Program Memory, Data RAM, Scratch Pad RAM, Emulator Buffered Outputs, UART/TTY Driver and Cassette Driver, Keyboard and Control Switches, Display, 2704/2708 Programmer and Expansion Plug which connects to optional slave modules for programming other types of PROMs.

The microprocessor is an RCA COSMAC 1802, an 8 bit CMOS device operating at 5 volts DC with a 1 MHZ crystal controlled clock. The resulting low power dissipation of the system permits the use of PC mounted power transformers which are inexpensive, light weight, small and dissipate little heat. This helps to reduce the cost, weight, and size of the entire machine.

The microprocessor program memory can consist of 2K of PROM or ROM. The basic machine comes equipped with one 512x8 program ROM which contains all the software required to perform the functions listed in Table 1. The user may modify the machine functions by writing a program in COSMAC object code and burning it in a new PROM. The new PROM may replace the existing PROM or it can be plugged into the unused PROM socket which is the second 1K byte of program memory. The standard data RAM is made up of 1Kx4 static RAMs. This RAM holds the data to be burned into the PROM during the programming cycle. The RUN/EMULATE switch is used to electrically disconnect the Data RAM from the rest of the system and connect it to the EMULATOR cable which plugs directly into a 2704/2708 PROM socket in the users system. To the users system, the EMULATOR cable looks just like a 2704/2708. An optional expansion kit includes two more 4804 RAMs and an adapter cable assembly which permits the emulation of 2048 words configured 2Kx8 or 4Kx4. The adapter cable assembly can be easily customer wired to emulate virtually any type of MOS, CMOS or TTL PROM or ROM. The adapter board has cable termination to permit up to 8 256x8 or 16 256x4 PROMs to be emulated at once.

During emulation, Memory Data is nondestructively read out and cannot be altered by the users system. Bipolar buffers, installed in sockets for easy replacement, protect the RAM from over voltages or shorts in the users system.

In the Emulate Mode, the microprocessor is in an idle state. When the RUN/EMULATE switch selects the RUN mode, the Data RAM is reconnected to the MPU. Load, modify, shift, move, dump and assemble operations take place in the RUN mode.

The UART is an 8 bit CMOS Universal Asynchronous Receiver Transmitter with its own crystal controlled clock. It interfaces the 8 bit data bus with 20 ma current loop TTY driver. The UART may be run at 10 or 30 characters per second. Latched error lights indicate parity, over-run or framing errors in incoming or outgoing ASCII data. The standard system software accepts BPNF formatted ASCII input and has a sophisticated data analysis capability which permits rubouts, skips and other tape anomalies to occur without producing load errors. A packed format loader is available as an option. The user may change the program to accept any other data format and perform any other type of data manipulation he desires.

An optional cassette drive is available. It is supplied complete with interconnect cables, test cassette and a pre-programmed PROM which may be plugged into the unoccupied PROM socket in the machine. This sub-system loads and unloads PROM data at the rate of 30 8-bit words per second. The data for a 512x8 PROM can be loaded from the cassette sub-system in under 20 seconds. Up to 50 512x8 PROM data blocks can be stored on each cassette using a keyboard entered identifying code for easy retrieval. This feature reduces a large data library to a handful of cassettes. The cassette drive greatly speeds the loading operations when using any of the available cross assemblers.

The keyboard and control switches permit the user to load, dump and modify data, program, emulate, assemble and enter specialized programs into unused RAM areas.

No training is required to operate the machine. The software consists of a simple operating system requiring a minimum of button pushing but having sufficient flexibility to allow the user to expand the machine by adding new programs. The steps in the procedures have names which correspond to the switch, button and display names on the machine. This allows an untrained, unskilled operator to load a tape, modify the data, burn in a PROM and punch a new tape by following the steps in the "Operating Instruction" card, Table 1.

OPERATING INSTRUCTIONS

1. Press the Reset Button and then the Run Button. The three digit display should show 000 and the three status lights should be off. The "Q" light should be on.
2. Enter 4 Keystrokes according to the following table.
3. To exit from any function, press Reset.

Function	Sub Function	1st Stroke	2nd Stroke	3rd Stroke	4th Stroke
1	Load From TTY to Data RAM (BPNF)	0	C	1	4
2	Dump From Data RAM to TTY (BPNF)	1	F	0	1
	Starting Address in Data RAM	X	X	X	X
3	Modify Data RAM	1	A	2	1
	Address of Word to be Changed	X	X	X	X
	New Data	X	X	NE	NE
4	Load Data RAM from Keyboard Auto	1	B	C	F
	Starting Address in Data RAM	X	X	X	X
	Data	X	X		
	Data	X	X		
5	Copy Master PROM to Data RAM	0	9	F	1
6	Burn in Dup. PROM from Data RAM	0	8	B	A
7	Locate and Display Address	1	F	8	7
	8 bit byte to be located - 1st occurrence	X	X	NE	NE
	Next Occurrence	X	X		
8	Move Block	1	F	9	9
	Starting Address of block to be moved	X	X	X	X
	Starting Address of New Location	X	X	X	X

TABLE 1.

Operating Instructions. Operation of the SMARTY is simple.

The operator need only apply power, press RESET and RUN and then enter the appropriate keystrokes.

The flow chart, Figure 2, shows the machine operation and operator interaction required to load a tape and burn in a PROM.

Pushing "Reset" and then "Run" puts the machine into the operating mode. The program quickly proceeds to a keyboard input mode. The operator keys in the address of the program to be run. This address is conveniently called out on the "Operating Instruction" card. For TTY load function, the operator keys in "OC14" which causes a program jump to the TTY input program and to start reading the tape.

To summarize, the operator pushes: "Reset"
"Run"
"OC14"

Turn on tape reader.

This simple operating system allows a number of program modules to be added to the system without changing the original program ROM. As new programming slaves, a floppy disc or the cassette drive are added, appropriate programs can be loaded into a new PROM which plugs into the unused PROM #2 socket inside the machine.

Whenever additional slaves or the cassette Data Recorder are ordered, a new instruction card and program PROM are provided. The new PROM contains a set of programs for operating the particular set of add on equipment attached to that system.

The three digit hexadecimal display is under program control. Keyboard entry data is displayed as entered. The display also indicated addresses and data during programming, data error codes and other data appropriate to the machines' operating mode.

The 2704/2708 family of 512x8 and 1Kx8 EPROMs are gaining wide acceptance as industry standards for microprocessor software development. The programming section of the SMARTY is designed for these PROMs.

SLAVES

A 40 pin expansion permits the attachment of numerous slave modules. Slaves are connected by "daisy chain" which permits the programming of many different types of PROMs without having to plug in new modules. The user selects the module to

be used by keying in the appropriate 4 Keystroke code shown on his operating instruction card. Currently available modules cover several EPROMs and TTL PROMs. Some of the modules planned for the immediate future have the capability of programming 4 or 8 PROMs at a time.

The final and perhaps most powerful feature of the system is its cross assemblers. The MPU is designed to execute a two pass cross assembler which we call DATA I/Q. This cross assembler is designed to produce object code from the users Assembly Language program. This object program is then resident in RAM and may be used immediately to simulate the PROM in the users system, to burn in a PROM, to punch a tape or write an object cassette for later use.

The Cross Assembler for generating 8080 object code is now available. We plan to product cross assemblers for the 6800, 2650, F8 and COSMAC in the immediate future.

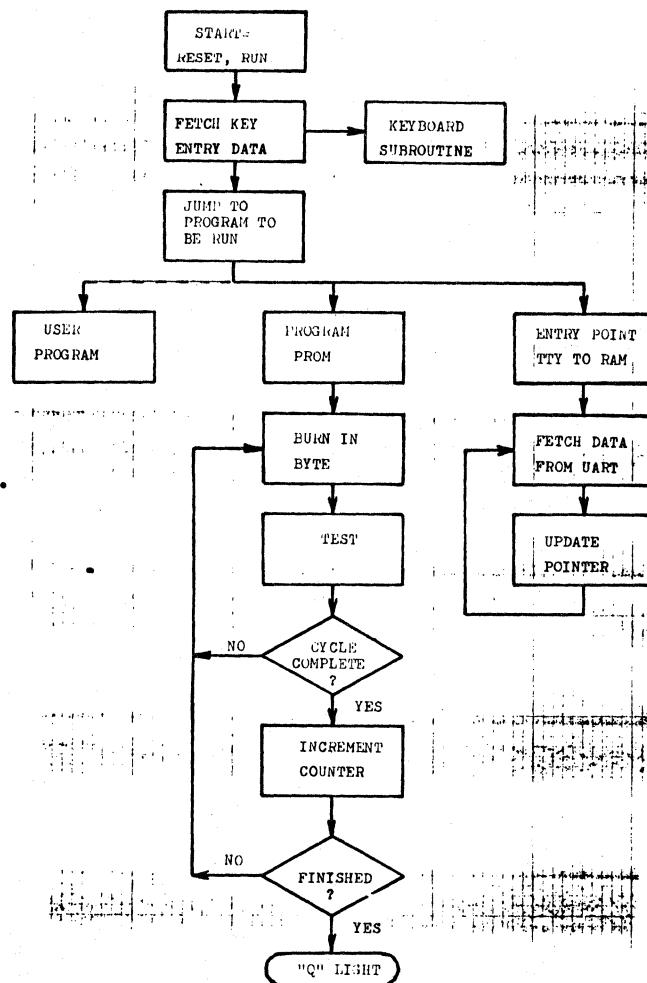


Figure 2 - Flow chart of monitor program.
Escape from any machine state is accomplished by pressing "RESET".

With this system the user can assemble a program, try it out by emulation, modify and debug it on his system and then burn in one PROM which will be correct. He can do this with any of the popular 8 bit microprocessors.

The SMARTY is a universal development tool as well as a versatile PROM programming system.

The microprocessor revolution has offered a marvelous opportunity for engineers to expand their useful knowledge base and to apply their creative skills to new and existing projects. Those of us who participated in the development of the SMARTY consider it our most rewarding achievement. I hope those of you who use it will enjoy working with it as much as we have.

The Model SM-1 as described in this article is priced at \$1,495.00. The Model SM-C Cassette Data Storage Unit is priced at \$995.00 complete. The Slave modules range in price from \$175.00 to \$595.00.

HOMEBREW EXHIBITORS*(as of 3:30 a.m. on 77 April 7)*

Dr. Franz Frederick
Room 112, Education Building
Purdue University
W. Lafayette IN 47907
(317) 749-2589

[Computer controlled "turtle"]

Ken McGinnis, M.D.
Box 2078
San Mateo CA 94401
(415) 343-5611

[Dual Homebrew System]

George Ledin Jr.
Computer Science Department
Harney Science Center
University of San Francisco
San Francisco CA 94117
(415) 666-6530

[Time share game playing & small computer building exhibit]

Laurence R. Upjohn
7297 Stanwood Way
Sacramento CA 95831
(916) 453-2476 x 3305

[Direct human interface!]

John S. French
846 Tamarack Ln.
Sunnyvale CA 94086
(408) 745-4763

[Homebrew Micro System]

Arthur Weigel
671 Iris Ave No. 14
Sunnyvale CA 94086
(408) 734-8102 x 550

[Self learning system]

Dave Arnold & Don Hanson
116F Escondido Village
Stanford CA 94306
(415) 497-4971

[100% Homebrew system]

Don R. Donfray
770 Lakehaven Dr.
Sunnyvale CA 94086
(408) 245-7100

[Complete Small Computer System]

Nishan Urkumyan
321 Cory Hall
University of California
Berkeley CA 94720
(415) 642-3730

[Computer driven locking system]

Joseph V. Jaworski
7051 Natal Drive No. 128
Westminster CA 92683
(714) 835-6000 x 426

[Low cost 6800 Hardware design] [Homebrew 8080 System]

Ben J. Milander
1439 Kinsport Ln.
San Jose CA 95120
(408) 256-1832

Dennis L. Wong
11266 Monterey Court
Cupertino CA 95014

[Micro disc Business Application]

Center for Contemporary Music
Mills College, Box 9991
Seminary and MacArthur Ave.
Oakland CA 94613
(415) 635-7620

[Oracle Booth!]

Donald C. Schertz
764 Toyon Drive
Monterey CA 93940
(408) 646-2982

[Multi voiced music synthesizer]

Steven S. Shaw
6520 Beadnell Way No. 2E
San Diego CA 92117
(714) 452-4016

[Computer Color Graphics]

Tony Williams & Jim Coe
PO Box 369
Mill Valley CA 94941
(415) 383-7074

[Computer generated & controlled audio]

Daniel L. Wright & Richard Fish
280 Perrymont Ave. Apt. 1
San Jose CA 95125
(408) 297-3000 x 3828

[6800 Based Home Micro System]

COMMERCIAL EXHIBITORS
(as of 2:20 p.m. on 27 Apr 7)

6502 PROGRAM EXCHANGE 2920 MOANA LN RENO NV 89509 (702)825-8413	BOOTSTRAP COMPUTER STORE 24 ST COMPONENT SHOP 3981 - 24 ST SAN FRANCISCO CA 94114 (415)282-3550	COMPUTER T-SHIRTS 114 W 17 ST NEW YORK NY 10011 (212)691-2821
ASSOC. FOR COMPUTING MACHINERY S. F. BAY AREA CHAPTERS BOX 60355 SUNNYVALE CA 94088 (415)851-7075	BYTE PUBLICATIONS, INC. 70 MAIN ST PETERBOROUGH NH 03458 (603)924-7217	CREATIVE COMPUTING MAGAZINE BOX 789-M MORRISTOWN NJ 07960 (201)766-7559
ACTION AUDIO ELECTRONICS WESTLAKE SHOPPING CENTER 323 S. MAYFAIR AVE DALY CITY CA 94015 (415)756-7440	BYTE SHOPS OF ARIZONA 813 N. SCOTTSDALE RD TEMPE AZ 85281 (602)894-1129	CROMEMCO 2432 CHARLESTON MT. VIEW, CA 94043 (415)964-7400
ADVANCED MICROCOMPUTER PRODS BOX 17329 IRVINE CA 92713 (714)968-3655	BYTE, INC. 1261 BIRCHWOOD DR. SUNNYVALE CA 94086 (408)734-9000	CUSTOM COMPUTER SYSTEMS 11 CREEKSIDE IRVINE CA 92715 (714)752-1591
ADVANCED TECHNOLOGY RESEARCH ASSOCIATES (ATRA) BOX 456 MINNEAPOLIS MN 55440 (612)377-7387	CALIFORNIA BUSINESS MACHINES 2211 THE ALAMEDA SANTA CLARA CA 95050 (408)244-7313	CYBERCON / SOLID STATE MUSIC 2102-A WALSH SANTA CLARA CA 95050 (408)246-2707
ALPHA MICROSYSTEMS 17875 SKYPARK NORTH, SUITE N IRVINE CA 92714 (714)957-1404	CALL COMPUTER & COMPUTER CONVERSOR CORP. 1961 OLD MIDDLEFIELD MT. VIEW, CA 94043 (415)964-9013	DAJEN ELECTRONICS 7214 SPRINGLEAF CT CITRUS HEIGHTS CA 95610 (916)752-0947
AMERICAN RADIO RELAY LEAGUE 225 MAIN ST NEWINGTON CT 06111 (203)666-1541	CENTER FOR THE STUDY OF THE FUTURE 4110 NE ALAMEDA PORTLAND OR 97212 (503)282-5835	DATA TERMINALS & COMMUNICATIONS 1190 DEIL AVE. CAMPBELL, CA 95008 (408)378-1112
ANDERSON JACOBSON, INC. 521 CHARCOT AVE. SAN JOSE, CA 95131 (408)263-8520	COMPONENT SALES, INC. 178-A BRANNAN ST SAN FRANCISCO CA 94103 (415)861-1345	DATAMATION 2530 BAYSHORE, #401 MT. VIEW CA 94043 (415)965-8222
APPLE COMPUTER 20863 STEVENS CREEK BLVD CUPERTINO CA 95014 (408)996-1010	COMPTEK REAL WORLD ELECTRONICS 30X 516 LA CANADA CA 91011 (213)790-7957	DAVIS LABORATORIES 2325 QUIKH AVE. BOX 2787 SANTA CLARA, CA 95051 (408)984-2930
APPLIED DATA COMMUNICATIONS 1509 E. MCFADDEN SANTA ANA CA 92705 (714)547-6954	COMPUTOCOLOR CORP. 5965 PEACHTREE CORNERS, EAST NORCROSS GA 30071 (404)449-5961	DAYTON ASSOCIATES BOX 454 BURLINGAME CA 94010 (415)347-9419
ARTEC ELECTRONICS 605 OLD COUNTY ROAD SAN CARLOS CA 94070	COMPUTALKER CONSULTANTS BOX 1951 SANTA MONICA, CA 90406 (213)392-5230	SAC SALES GROUP BOX 352 DAVIS CA 95615 (916)723-1050
ASSOCIATED ELECTRONICS CO. 1885 W. COMMONWEALTH, #G FULLERTON CA 92633 (714)879-7707	COMPUTER CONVENTIONS COMPANY 1600 WOOLSEY ST. BERKELEY, CA 94703 (415)845-3832	DIGITAL ELECTRONICS CORP. 415 PETERSON ST OAKLAND CA 94601 (415)532-2920
BELL & HOWELL SCHOOLS 209 W. JACKSON CHICAGO IL 60606 (312)939-8200	COMPUTER DECISIONS MAGAZINE HAYDEN PUBLISHING CO., INC. 50 ESSEX ST ROCHELLE PARK NJ 07662 (201)843-0550	THE DIGITAL GROUP BOX 6528 DENVER, CO 80206 (303)757-7133
BENDER PUBLICATIONS NORTHERN CALIFORNIA ELECTRONIC NEWS SOUTHERN CALIFORNIA ELECTRONIC NEWS BOX 3631 LOS ANGELES CA 90051 (213)737-6820	COMPUTER KITS, INC. 1044 UNIVERSITY AVE BERKELEY CA 94710 (415)845-5300	DIGITAL PROJECTS 1226 W. MAPLE ST LOMPOC CA 93436 (805)736-3766
BERG PUBLICATIONS 1360 SW 199 COURT ALOHA OR 97005 (503)649-7495	COMPUTER MAGAZINE IEEE COMPUTER SOCIETY 5855 NAPLES PLAZA, #301 LONG BEACH CA 90803 (213)438-9951	DIGITAL RESEARCH BOX 579 PACIFIC GROVE, CA 93950 (408)373-3403
BILLINGS COMPUTER CORP. BOX 555 PROVO UT 84601 (801)375-0000	COMPUTER POWER & LIGHT CO. 12321 VENTURA BLVD STUDIO CITY CA 91604 (213)760-0405	DIGITAL SYSTEMS 1154 DUNSMUIR PL. LIVERMORE, CA 94550 (415)443-4078
BLASTMASTERS, INC. & MULLEN COMPUTER BOARDS BOX 6214 HAYWARD CA 94545 (415)278-5122	COMPUTER ROOM OF SAN JOSE 124-H BLOSSOM HILL RD SAN JOSE CA 95123 (408)226-8383	DILITHIUM PRESS BOX 92 FOREST GROVE OR 97116 (503)357-7192
BOARD BYTERS BOX 17512 IRVINE CA 92713 (714)586-8133	COMPUTER STORE OF SAN FRANCISCO 1093 MISSION ST SAN FRANCISCO CA 94103 (415)431-0643	DXMAX BOX 310 MENLO PARK CA 94025 (415)323-0117
E & L INSTRUMENTS C/O IBM, INC. 30 W. EL CAMINO REAL, #C MT. VIEW CA 94040 (415)961-2828		

E, S & L INDUSTRIES 867 S ROSE/ANAHM92805 BOX 1260 SOUTH GATE CA 90280 (714)956-1477	IC MASTER & UPDATE UNITED TECHNICAL PUBS 1287 LAWRENCE STN RD SUNNYVALE CA 94086 (408)734-8214	MICRO-TERM INC. BOX 9387 ST LOUIS MO 63117 (314)645-3656
ECD CORP. 196 BROADWAY CAMBRIDGE MA 02139 (617)661-4400	ICOM DIVISION PERTEC COMPUTER CORP. 6741 VARIEL AVE CANOGA PARK, CA 91303 (213)348-1391	MICROCOMPUTER ASSOCIATES BOX 1167 CUPERTINO, CA 95014 (408)247-8940
EIDETIC DESIGNS / MEMCON 505 W. OLIVE AVE, #612 SUNNYVALE CA 94086 (408)737-2070	IMSAI 14860 WICKS BLVD. SAN LEANDRO, CA 94577 (415)483-2093	MICROCOMPUTER DEVICES 564 S. GREENWOOD AVE MONTEBELLO CA 90640 (213)728-2393
ELECTRONIC CONTROL TECHNOLOGY BOX 6 UNION NJ 07083 (201)686-8080	INFORMATION TERMINALS 323 SOQUEL WAY SUNNYVALE CA 94086	MICRODESIGN 8187 HAVASU CIRCLE BUENA PARK CA 90621 (714)523-8080
ELECTRONIC TOOL CO. 4736 W. EL SEGUNDO BLVD. HAWTHORNE, CA 90250 (213)644-0113	INTEGRAND RESEARCH CORP. 8474 AVE 296 VISALIA CA 93277 (209)733-9288	MICROMATION, INC. 524 UNION SAN FRANCISCO CA 94133 (415)398-0289
ESCON 1235 - 10TH ST BERKELEY CA 94710 (415)524-8664	INTEGRATED COMPUTER SYSTEMS 4445 OVERLAND AVE LOS ANGELES CA 90230 (213)559-9265	MICROPOLIS CORP. 9017 RESEDA BLVD., #204 NORTHRIDGE CA 91324 (213)349-2328
EXECUTIVE DEVICES 966 PENINSULA, #103 SAN MATEO CA 94401 (415)342-0690	INTELLIGENT COMPUTER SYSTEMS 777 MIDDLEFIELD RD., #40 MT VIEW CA 94043	MICROTRONICS BOX 7454 MENLO PARK CA 94025 (415)854-5630
EXTENSYS CORP. 592 WEDDELL DR. #3 SUNNYVALE CA 94086 (408)734-1525	INTERFACE AGE MAGAZINE 13913 ARTESIA BLVD BOX 1234 CERRITOS CA 90701 (213)926-6620	MIDWEST SCIENTIFIC INSTRUMENTS 220 W. CEDAR OLATHE KS 66061 (913)764-3273
FLOOD & ASSOC. 5378 BOUTE AVE SAN DIEGO CA 92122 (714)455-0190	INTERSIL, INC. 10900 N. TANFAU AVE. CUPERTINO, CA 95014 (408)996-5000	MINI-MICRO SYSTEMS MAGAZINE 5 KANE INDUSTRIAL DR. HUDSON MA 01749 (617)562-9305
GALAXY SYSTEMS 4376 AVENIDA CARMEL CYPRESS CA 90630 (213)644-9881	ITTY BITTY COMPUTERS BOX 23189 SAN JOSE, CA 95153	MINITERM ASSOC. BOX 268 BEDFORD MA 01730 (617)648-1200
GIMIX, INC. 1337 W. 37 PL. CHICAGO IL 60609 (312)927-5510	JADE CO. 2351 W. 144 ST LAUREL CA 90260 (213)679-3313	MINNESOTA MINING & MANUFACTURING COMPANY (3M) DATA RECORD PRODUCTS 3M CENTER ST PAUL MN 55101 (612)733-1100
GNOMON ASSOCIATES BOX 23601 PLEASANT HILL CA 94523 (415)689-2925	JENSEN TOOLS & ALLOYS 4117 N. 44 ST PHOENIX AZ 85018 (602)959-0021	MISSION CIRCUITS, INC. 41060-C HIGH ST FREMONT CA 94538
GODBOUT ELECTRONICS BOX 2365 OAKLAND AIRPORT CA 94614 (415)562-0636	JK ELECTRONICS C/O DAJEN 7214 SPRINGLEAF CT CITRUS HEIGHTS CA 95610	MOTOROLA SEMICONDUCTOR 4000 KOOPARK, #216 SAN JOSE CA 95117 (408)985-0510
HAL COMMUNICATIONS CORP. 807 E. GREEN ST BOX 365 UHBANA, IL 61801 (217)367-7373	KANEMATSU-GOSHU (USA), INC. 425 CALIFORNIA SAN FRANCISCO CA 94104 (415)788-3800	MOUNTAIN HARDWARE 569 MARION CT. BEN LOMOND, CA 95005 (408)249-2850
HAYDEN BOOK COMPANY, INC. MAIN OFFICE 50 ESSEX ST ROCHELLE PARK, NJ 07662 (201)843-0550	KILOBAUD MAGAZINE 73 PINE ST PETERBOROUGH NH 03458 (603)924-3873	MOVONICS 1922 ANNETTE LN BOX 1223 LOS ALTOS CA 94022
HEURISTICS, INC. 900 N. SAN ANTONIO, #C1 LOS ALTOS, CA 94022 (415)948-2542	LASSEN ELECTRONICS CORP. BOX 1429 FREMONT CA 94538	MR CALCULATOR CONTEMPORARY MKTG 39 TOWN & COUNTRY VILLAGE PALO ALTO CA 94301 (415)328-0740
HOBBY DATA FORENINGSS 67 FACK 20012 MALMO SWEDEN 40-971-777	LOGIC DESIGN, INCORPORATED BOX 3991 UNIVERSITY STATION LARAMIE, WY 82071 (307)742-7977	NATIONAL SEMICONDUCTOR CORP. 2900 SEMICONDUCTOR DR. SANTA CLARA CA 95051 (408)737-5543
HUH ELECTRONIC MUSIC PRODUCTIONS BOX 259 FAIRFAX CA 94930 (415)457-7598	LOGICAL SERVICES, INC. 711 STEERLIN RD. MT. VIEW, CA 94043 (415)955-8365	NEWMAN COMPUTER EXCHANGE 1250 N. MAIN ST. ANN ARBOR MI 48104 (313)994-3200
IASIS INC. 815 W. MAUDE AVE. SUNNYVALE, CA 94086 (408)732-5700	MECA 7344 WAMEGO TRAIL YUCCA VALLEY CA 92284 (714)365-7686	NORTH STAR COMPUTERS, INC. 2465 - 4TH ST. BOX 4672 BERKELEY CA 94710 (415)549-0858
IBEX 1010 MORSE AVE, #5 SUNNYVALE CA 94086 (408)739-3770	MICRO DESIGNS, INC. 499 EMBARCADERO OAKLAND, CA 94606 (415)465-1861	OHIO SCIENTIFIC INSTRUMENTS 11681 HAYDEN ST HIRAM OH 44234 (216)569-7945

OK MACHINE & TOOL CORP.
3455 CONNER ST
BROOKLYN NY 10475
(212)994-6600

OLIVER AUDIO ENGINEERING, INC.
7330 LAUREL CANYON BLVD.
N. HOLLYWOOD CA 91605
(213)765-8080

PACIFIC OFFICE SYSTEMS
2600 EL CAMINO REAL #502
PALO ALTO CA 94306
(415)321-3866

PAIA ELECTRONICS, INC.
1020 WILSHIRE BLVD
OKLAHOMA CITY OK 73116
(405)843-9626

PARASITIC ENGINEERING &
MORROW'S MICRO STUFF
BOX 6314 & BOX 6194
ALBANY, CA 94706
(415)547-6612

PAREC ELECTRONICS
5074 GEORGETOWN AVE
BOX A82327
SAN DIEGO CA 92138
(714)276-3255

PEOPLE'S COMPUTER CO.
1010 DOYLE ST.
BOX E
MENLO PARK, CA 94025
(415)323-3111

PERIPHERAL VISION
3248 S. EUDORA
DENVER CO 80222
(303)777-7133

PERSCI
4087 GLENCOE AVE.
MARINA DEL REY CA 90291
(213)822-7545

PERSONAL COMPUTING MAGAZINE
401 LOUISIANA SE, #G
ALBUQUERQUE NM 87108
(505)266-1173

PFEIFFER, E & L
BOX 2624
SEPULVEDA CA 91343
(213)368-3996

PMS PUBLISHING CO.
12625 LIDO WAY
SARATOGA CA 95070
(408)996-0471 OR 255-7894

POLYHEDRAL SYSTEMS
437-A ALDO AVE
SANTA CLARA CA 95050
(408)244-2155

POLYMORPHIC SYSTEMS
460 WARD DR
SANTA BARBARA CA 93111
(805)967-2351

PRIME RADIX, INC.
1764 BLAKE ST/80202
BOX 11245
DENVER CO 80211
(303)73-4895

PROCESSOR TECHNOLOGY
6200 HOLLIS ST.
EMERYVILLE CA 94608
(415)652-8080

PSA, INC.
3677 WEST 1987 SOUTH
SALT LAKE CITY UT 84104
(801)261-2529

R.D.C. ENTERPRISES
8352 STANFORD AVE
GARDEN GROVE CA 92641
(714)638-2094

R.H.S. MARKETING
DUTRONICS, DYNABYTE, PROMEDICS
INFORMATION TERMINALS CORP.
2233 EL CAMINO REAL
PALO ALTO CA 94306
(415)321-6639

REALISTIC CONTROLS CORP.
3530 WARRENSVILLE CTR RD
CLEVELAND OH 44122
(216)751-3158

RIGEL FOUR
BOX 1422
CAMPBELL CA 95008
(408)379-8137

RO-CHE SYSTEMS
7101 MAMMOUTH AVE.
VAN NUYS CA 91405
95228

ROM: COMPUTER APPLICATIONS FOR
LIVING (MAGAZINE)
RT. 97
HAMPTON CT 06247
(203)455-9591

S. D. SALES
BOX 28810
DALLAS TX 75228
(214)271-4667

S.F. CONVENTION PHOTOGRAPHERS
2 ST GEORGE
SAN FRANCISCO CA 94108
(415)981-3624

SCIENTIFIC PROGRAMMING
2213 JEFFERSON
BERKELEY CA 94703
(415)692-1600

SCIENTIFIC RESEARCH, INC.
1712 FARMINGTON COURT
CROFTON MD 21114
(301)721-1143
(800)638-9194 (TOLL-FREE, OUT-OF-STATE)

SHEPARDSON MICROSYSTEMS, INC.
20823 STEVENS CREEK BLVD
BLDG. C4-H
CUPERTINO CA 95014
(408)257-9900

SINAI-JOHNSON, INC.
317 DOUGLAS
BOX 5218
REDWOOD CITY CA 94063
(415)365-6263

SMOKE SIGNAL BROADCASTING
BOX 2017
HOLLYWOOD CA 90028
(213)462-5652

SOFTWARE TECHNOLOGY CORP.
BOX 5260
SAN MATEO CA 94402
(415)349-8080

SOUTHERN CALIFORNIA COMPUTER
SOCIETY & SCCS INTERFACE MAGAZINE
BOX 54751
LOS ANGELES CA 90054
(213)980-7343

SOUTHWEST TECHNICAL PRODUCTS
CORP. (SWTPC)
219 W. RHAPSODY
SAN ANTONIO TX 78216
(512)344-0241 OR 344-0242

STANFORD ELECTRICAL
ENGINEERING DEPARTMENT
STANFORD UNIVERSITY
STANFORD CA 94305
(415)497-4709

SUNRISE ELECTRONICS
228 N. EL MOLINO ST
PASADENA CA 91101
(213)963-8775

SUNSET TECHNOLOGIES
BOX 2411
GOLETA CA 93018
(805)968-7463

SYLVANHILLS LABORATORY
1 SYLVANWAY
BOX 239
STRAFFORD MO 65757
95695

SYMBIOTIC SYSTEMS
RT 1, BOX 266
WOODLAND CA
(916)758-1896

SYNETIC DESIGNS
1452 PROSPECT DR.
POMONA CA 91766
(714)629-1974

SZERLIP ENTERPRISES
1414 W 259 ST
HARBOUR CITY CA 90710
(213)325-9333

TARBELL ELECTRONICS
20620 S. LEAPWOOD AVE., SUITE P
CARSON CA 90746
(213)538-4251

TECHNICAL DESIGN LABS INC.
RESEARCH PARK, BLG H
1101 STATE RD
PRINCETON NJ 08540
(609)921-0321

TECHNICAL SYSTEMS CONSULTANTS
BOX 2574
W. LAFAYETTE IN 47906
(317)742-7509

TECHNICO
9130 RED BRANCH RD
COLUMBIA MD 21045
(800)638-2893

TELETYPE CORPORATION
5555 W. TOUHY AVE
SKOKIE, IL 60076
(312)982-3115

TELPAR, INC.
4132 MITCHELL RD
BOX 196
ADDISON TX 75001
(214)233-6631

TRIPLE I - PHI-DECK
4605 NORTH STILES
BOX 25308
OKLAHOMA CITY OK 73125
(405)521-9000

VANDEMBURG DATA PRODUCTS
BOX 2507
SANTA MARIA CA 93454
(805)937-7951

VECTOR ELECTRONIC CO., INC.
12460 GLADSTONE AVE
SYLMAR CA 91342
(213)365-9661

VECTOR GRAPHIC INC.
717 LAKEFIELD RD., SUITE F
WESTLAKE VILLAGE CA 91361
(805)497-0733

VIDEO TERMINAL TECHNOLOGY
BOX 60485
SUINNYVALE CA 94088
(408)255-3001

VITEK
1160 BARBARA DR.
VISITA CA 92083
(714)724-0210

WAVE MATE - COMPUTERS & SYSTEMS
1015 W. 190 ST
GARDENA CA 90248
(213)329-8941

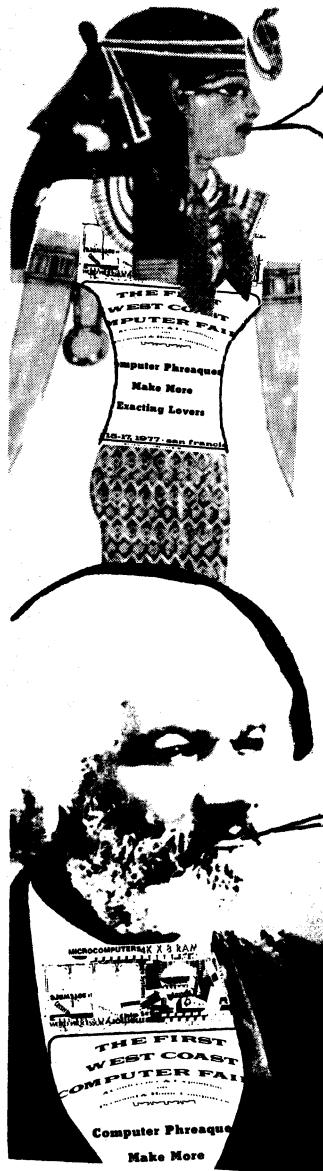
WESTERN DATA SYSTEMS
3650 CHARLES ST #G
SANTA CLARA CA 95050
(408)984-7804

WIZARD ENGINEERING
8205 RONSON RD. #C
SAN DIEGO CA 92111
(714)565-6021

WIZARD EYE STUDIO
NO. 21 COACHMAN COURT
701 WEST HAMPDEN
ENGLEWOOD CO 80110
(303)761-3546

XIMEDIA CORP.
1290 - 24 AVE
SAN FRANCISCO CA 94122
(415)566-7472

XYBEC
BOX 4925
STANFORD CA 94305
(415)296-3188



GEE, YOU WOULDN'T BELIEVE HOW MANY PEOPLE I HAVE TO TELL
TO KEEP THEIR HANDS OFF MY ASP SINCE I BEGAN WEARING
THE OFFICIAL FIRST WEST COAST COMPUTER FAIRE T-SHIRT!!

SHOW what you KNOW!!

Whether you're in the barroom or the boardroom, you may rest assured that The Official First West Coast Computer Faire T-Shirt (official attire for the Computer Faire) will bring to you sartorial splendor. This tri-color T-Shirt (top-o'-the-stack blue, Byte-this! violet, up-&running orange) will amaze your friends, restore hair, and prevent bullies from kicking sand in your face--if your friends are gullible, your alopecia was temporary, and you avoid beaches.

On the other back, who cares?

In separate laboratory tests, Dr Seymour Squintz (who was born outside of a log cabin in the Gaza Strip) stated: "When I go down to the beach of life [in my Official First West Coast Computer Faire T-Shirt] they may still kick sand in my mind's eye, but I see it as cause for the growth of a pearl rather than for the use of Visine."

The fact that Dr Squintz subsequently and tragically incurred major cataracts, vehemently denied he was an ardent voyeur, and accidentally committed reverse self-defenestration, is, of course, unfortunate. And we will probably stop using his testimonial.

In the meantime, whether you're a shareholder or a sharecropper, on the board of directors or simply bored of directors, buy a gross of these discreet garments, and flout them proudly. Equally at home under a motorcycle jacket or a Brooks Brothers, the Computer Faire T-Shirt will positively make you look stunning. Or maybe stunned.

In any case, keeping in mind that you are what you wear, and that, though this T-Shirt may be taken to the cleaners, you never will be, we think you will agree that the proposed investment of \$4 a shirt is modest--at least compared to this solicitation!

AFTER WEARING THIS
T-SHIRT, THEY BANNED
DATA BUSING IN BOSTON.

(Please note: Because this is a limited edition, Officials from the Franklin Mint will, at an unspecified time, uproot our cotton plants, shred our silkscreen, and set our copy writer adrift on a small raft, blindfolded, in downtown Billings, Montana.)

If you want to Act Now!, contact the Actors' Guild. If you merely want to set the stage, then buy the Computer Faire T-Shirt.

YES!! Send to me immediately (as ordered) The Official First West Coast Computer Faire T-Shirt(s). I understand they are additionally fabulous for their unique property of both cradling the wearer, and swaddling bystanders--innocent and guilty alike--with exquisite envy. (So that Computer Faire may prepare legal defense for actionable horripilatory incitement, please indicate your first intended bystander:
 Yves St Laurent, EE professor who flunked me, boss, janitor, janitor's tax accountant, maitre d' at MacDonald's, cat, computer club executive council, tech rep, cheeky proctologist, _____)

S, <input type="checkbox"/> M,	L, <input type="checkbox"/> XL SHIRT(S) @ \$4
CALIFORNIA RESIDENTS ADD 6% TAX	
1ST-CLASS POSTAGE & HANDLING (\$1/SHIRT)	
TOTAL AMOUNT ENCLOSED	
Please send your order & payment (check or money order), to: Computer Faire Box 1579, Palo Alto CA 94302 (415) 851-7664 Thank you.	

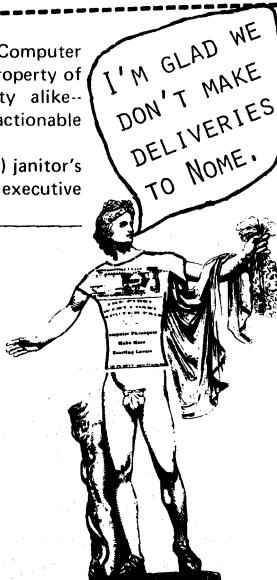
NAME _____

STREET _____

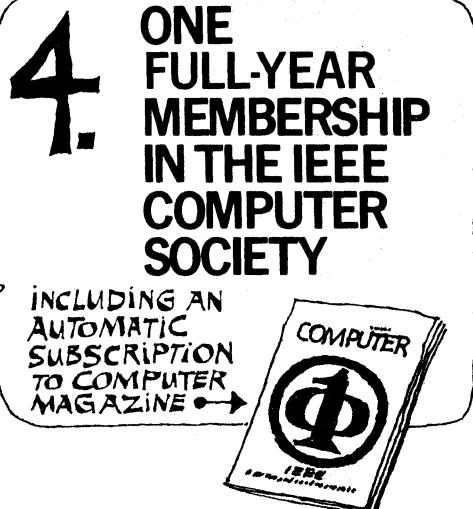
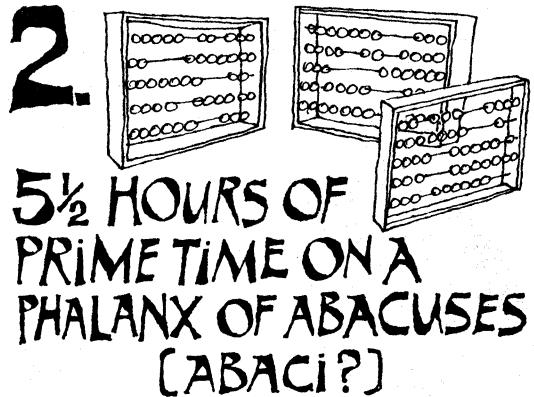
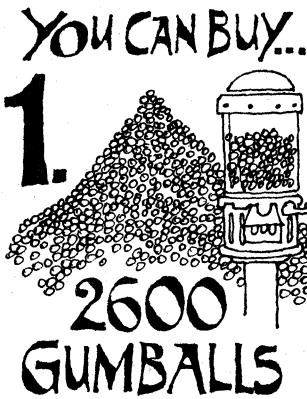
CITY _____

STATE _____ ZIP _____

THE FIRST
WEST COAST
COMPUTER FAIRE
A Conference & Exposition
on Personal & Home Computers



→FOUR← WAYS TO SPEND **\$26⁰⁰**



AND... ~~TO THE FIRST 2 MILLION WHO SIGN UP,~~
WE'RE THROWING IN A FREE COPY OF
MICROPROCESSORS & MICROCOMPUTERS...
A NEAT LITTLE COLLECTION OF REPRINTS FROM COMPUTER.

Organized and introduced by our technical editor, this volume carries a balanced selection of everything we've published in *COMPUTER* about micros over the past two years.

Fill out and mail the form below.

Find out why membership in the IEEE Computer Society is one of the best investments you'll ever make.

* Offer expires December 31, 1977.



IEEE COMPUTER SOCIETY

5855 Naples Plaza
Long Beach, California 90803



INSTITUTE OF ELECTRICAL AND
ELECTRONICS ENGINEERS

Quick—send me everything I need to know about joining the IEEE Computer Society. You can send me my free copy of "Microprocessors and Microcomputers" when I join.

Name _____

Address _____

Note: Offer expires December 31, 1977.

Please send a copy of the Computer Society Publications Catalog.

MAIL THIS FORM TO:

IEEE Computer Society 5855 Naples Plaza Long Beach, California 90803

TUTORIALS for computer people

from the
IEEE COMPUTER SOCIETY



MICROCOMPUTER 77 CONFERENCE RECORD

April 6-8, 1977 — 274pp.

A collection of over 50 papers describing the latest developments in microcomputer hardware and software as well as a wide range of microcomputer applications. Topics include communications and intelligent terminals, development systems, high-level languages, Basic for microcomputers, systems design, simulation and emulation, fault detection, and peripheral memories.

Non-members—\$20.00

Members—\$15.00

MINICOMPUTERS & MICROPROCESSORS — 271pp.

Review of recent developments in both minicomputer and microprocessor technology. Each chapter begins with a short section which introduces the topics to be discussed and also highlights the significant points found in the chapter. Topic headings include technological advances, microprogramming, minicomputer architectures, microprocessors, mini and micro computer system development, and interfacing and peripherals.

Non-members—\$12.00

Members—\$9.00

PROCEEDINGS OF THE SYMPOSIUM ON DESIGN AUTOMATION AND MICROPROCESSORS

February 24-25, 1977 — 110pp.

A collection of 19 papers examining the development of design aids for microprocessors and microprocessor-based systems. Sample topics include: capability requirements in a multimicro processors, hardware/software simulation environment; automatic design of multiprocessor microprocessor systems; automated design based upon micro-programmable bit-slice microprocessors; PLATO-PLA translator/optimizer; user requirements for digital design verification simulators; simulation hierarchy for microprocessor design; verification of hardware designs through symbolic manipulation; modeling for synthesis, the gap between intent and behavior; and several discussions of SARA.

Non-members—\$12.00

Members—\$9.00

SOFTWARE DESIGN TECHNIQUES (2nd Edition) — 282pp.

Intended for both beginning and experienced software designers, this book contains 18 key papers plus over 100 pages of original material by the editors, Profs. Peter Freeman of UC Irvine and Anthony I. Wasserman of UC San Francisco. Major subject categories covered include the importance of design and specification, differences between design and programming, basic design techniques (abstractions, representation, structure diagrams, metacode). Also includes a series of revealing case studies.

Non-members—\$12.00

Members—\$9.00

MIMI 76: PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON MINI AND MICROCOMPUTERS

November 8-11, 1976 — 244pp.

Forty-eight papers on hardware, software, systems, applications, and education presented by an international group of computer professionals. A sampling of the titles includes: A Timeshared Multi-User Approach to Microprocessor System Development; A Floating Point Computer for Generalized Spectral Analysis; A Resident Micro-Assembler for Microcomputers; LSI Beyond the MPU; Privacy Based Computer Design Using Microprocessors; and Microprocessor Instruction in Electrical Engineering and Electrical Engineering Technology.

Non-members—\$20.00

Members—\$15.00

DESIGNING WITH MICROPROCESSORS — 150pp.

This unique tutorial, given by Professor Tilak Agarwala of the University of Texas, Gerald Masson of Johns Hopkins, and Roger Westgate of Johns Hopkins at COMPCON Fall 76, deals with the principles and practice of microprocessor design. Covers such topics as chip architecture, microprocessor selection criteria, software aids, development systems, microprocessor applications, networks, bussing strategies, and distributed intelligence.

Non-members—\$10.00

Members—\$7.50



INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS

ISBN 0-930418-00-X