FST-2 COMPUTER

ASSEMBLY LANGUAGE PROGRAMMING COURSE

STUDENT WORKBOOK

Training Center

FAIRCHILD

COURSE DESCRIPTION

FST-2 COMPUTER ASSEMBLY LANGUAGE PROGRAMMING COURSE

COURSE LENGTH: 2 WEEKS (10 DAYS)

PREREQUISITES: SUCCESSFUL COMPLETION OF A SENTRY (BASIC) PROGRAMMING COURSE. THE STUDENT SHOULD ALSO BE FAMILIAR WITH BASIC COMPUTER CONCEPTS, BINARY AND OCTAL NUMBER SYSTEMS, COMPUTER ARITHMETIC, MACHINE LANGUAGE PROGRAMMING FUNDAMENTALS, SYMBOLIC LANGUAGE CONCEPTS AND FUNDAMENTAL PROGRAMMING TECHNIQUES IN ORDER TO OBTAIN THE MOST BENEFIT FROM THE COURSE, BUT IS NOT MANDATORY.

PURPOSE

SENTRY PROGRAMMERS LEARN HOW TO WRITE ASSEMBLY LANGUAGE PROGRAMS FOR THE FST-2 COMPUTER. ALTHOUGH THE BEGINNING OF THE COURSE IS AIMED AT THE PROGRAMMER WITH LITTLE OR NO EXPERIENCE, FOR THOSE WITH MORE EXPERIENCE, IT SHOULD CLARIFY AND REIEFORCE MANY ASSEMBLY LANGUAGE PROGRAMMING CONCEPTS ALREADY LEARNED.

THE FIRST PORTION OF THE COURSE WILL TEACH THE PROGRAMMER HOW TO WRITE GENERAL PURPOSE, NON-TESTER RELATED ASSEMBLY LANGUAGE PROGRAMS WHICH ARE EXECUTED IN THE "BACKGROUND" MODE USING THE MASTR OPERATING SYSTEM SOFTWARE. THE REMAINDER OF THE COURSE WILL TEACH THE STUDENT HOW TO WRITE TESTER RELATED ASSEMBLY LANGUAGE PROGRAMS WHICH ARE EXECUTED IN THE MASTR "FOREGROUND" MODE, AS WELL AS PROGRAMS WHICH CAN BE ENTERED THROUGH EITHER "FOREGROUND" OR "BACKGROUND" MODE, OR BOTH.

OBJECTIVES

AFTER COMPLETION OF THE COURSE, THE STUDENT WILL BE ABLE TO:

- o WRITE ASSEMBLY LANGUAGE PROGRAMS THE FST-2 COMPUTER INSTRUCTION SET AND ASSEMBLER SYNTAX
- o LOAD, ASSEMBLE, EXECUTE AND DEBUG GENERAL PURPOSE ASSEMBLY LANGUAGE PROGRAMS IN THE "BACKGROUND" MODE USING THE MASTR OPERATING SYSTEM SOFTWARE
- o WRITE, ASSEMBLE, AND LOAD ASSEMBLY LANGUAGE PROGRAMS WHICH COMMUNICATE WITH THE SENTRY AND ARE CALLED BY A FACTOR LANGUAGE PROGRAM
- o WRITE, COMPILE, LOAD, EXECUTE AND DEBUG SIMPLE FACTOR LANGUAGE PROGRAMS WHICH CALL AND EXECUTE ONE OR MORE ASSEMBLY LANGUAGE PROGRAMS THROUGH THE "FOREGROUND" MODE UNDER CONTROL OF THE MASTR OPERATING SYSTEM SOFTWARE

THE STUDENT WILL DEMONSTRATE ATTAINMENT OF THE COURSE OBJECTIVES BY SUCCESSFULLY COMPLETING ALL HOMEWORK, LAB ASSIGNMENTS, QUIZZES, AND A WRITTEN END-OF-COURSE EXAMINATION.

FST-2 (MASTR)
ASSEMBLY LANGUAGE PROGRAMMING COURSE

COURSE OUTLINE

DAY 1

COURSE ORIENTATION

Introductions
General information
Course objectives and outline review
Familiarization with course manuals

INTRODUCTION TO DIGITAL COMPUTERS

Brief history of computer development
Contemporary computer applications
The hierarchy of computer programming languages
The basic units of the digital computer

FST-2 COMPUTER SYSTEM DESCRIPTION

Computer system configuration
Computer hardware capabilities
Computer software capabilities
Memory management under MASTR - overview
FST-2 CPU simplified block diagram analysis for programmers

OPERATING PROCEDURES

Description of computer system controls and indicators
System turn-on/turn-off procedures
Operating procedures for system peripherals
Loading and initialization of system software
Creating, assembling, and executing an assembly language program

LAB

Demo of system operating procedures
Student performs associated lab procedures

DAY 2

BASIC PROGRAMMING

The stored program concept
Basic assembly language program syntax
Basic assembly language program layout - simplified
FST-2 CPU machine word formats - general
Detailed description of FST-2 CPU instruction word repertoire
Program writing by class

LAB

Demo of manually loading and executing a machine language program at the computer control panel
Student performs associated lab procedures

DAY 3

PROGRAMMING NON-DMA I/O

Detailed description of I/O instruction repertoire for non-DMA peripherals
Program writing exercises by class

PROGRAMMING ASSIGNMENT

Assign I/O program writing assignment #6
Assign program writing assignment #7 (ECHO program)

LAB

Demo of I/O program writing assignment #6
Demo of program writing assignment #7 (ECHO program)
Student writes and executes I/O program writing assignment

DAY 4

PROGRAMMING DMA I/O

Detailed description of I/O instruction repertoire
for DMA peripherals
Analysis of example programs
Program writing by class

PROGRAM DEBUGGING

Detailed description of DEBUG program usage

LAB

Demo of DEBUG program usage
Students work on program writing assignment #7 (ECHO program)

DAY 5

MASTR OPERATING SYSTEM MEMORY ORGANIZATION

Overview of memory organization using memory map
Overview of system tables, globals, transfer vectors, and library routines

REVIEW OF ASSEMBLY LANGUAGE PROGRAM LAYOUT

General review
Detailed explanation of program header

PROGRAM EXECUTION IN BACKGROUND MODE

Program structure for "background mode" only
Parameter passing in "background" mode

PROGRAMMING ASSIGNMENT

Add programming requirements to ECHO program as defined by program writing assignment #8

LAB

Demo ECHO program with program writing assignment #8 added
Students work on program assignment

DAY 6

PERIPHERAL I/O USING IOCS

Overview of ICCS
Writing and reading of data
IOCS control functions
Program writing by class

PROGRAM ASSIGNMENT

Rewrite the ECHO program as defined by program writing assignment #9

LAB

Demo ECHO program with program writing assignment #9 added
Students work on program assignment

DAY 7

INTERRUPT I/O PROGRAMMING

Overview of peripheral I/O using interrupt programming

GENERAL REVIEW

LAB

Students continue work on EHCO program

DAY 8

PROGRAM EXECUTION IN FOREGROUND MODE

FACTOR EXEC statement with parameter passing
Program structure for "foreground" mode only
Fetching parameters in assembly language program
Returning parameters to FACTOR program

TESTER I/O

Overview
Detailed description of addressing "short" registers
Short registers word formats - overview
Detailed description of addressing "long" registers
Long register word formats - overview
Generating tester I/O mnemonics
Program writing by class

PROGRAMMING ASSIGNMENT

Assign program writing assignment #10 (WEIR program)

LAB

Demo of programming assignment #10
Students work on program assignment #10

DAY 9

REVIEW

Review of tester I/O

LAB

Students continue working on program writing assignment #10 (WEIR program)

DAY 10

PROGRAM EXECUTION WITH DUAL ENTRY POINTS (FOREGROUND/BACKGROUND)

Program structure when using dual entry points

MISCELLANEOUS

Examine other routines and/or techniques used in operating system which would be of value to user programmer

COURSE REVIEW

Review of course subjects as necessary

COURSE CRITIQUE

DISCUSSION WITH APPLICATIONS ENGINEER

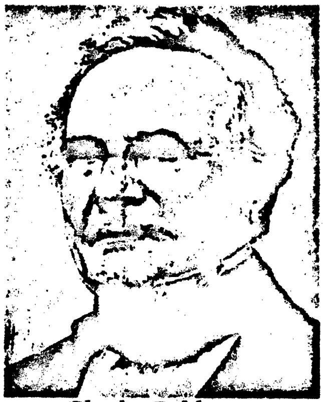
LAB

Finish any lab projects not completed and turn in to instructor

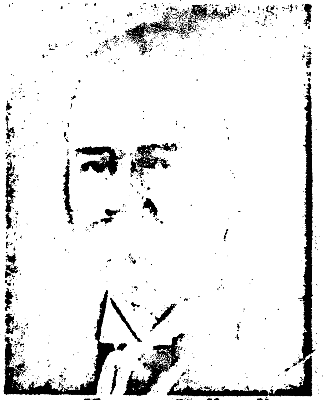
INTRODUCTION
TO
DIGITAL COMPUTERS

Blaise Pascal

Gottfried Leibniz



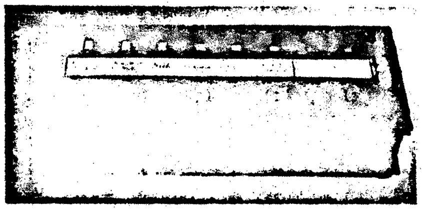
Charles Babbage



Herman Hollerith

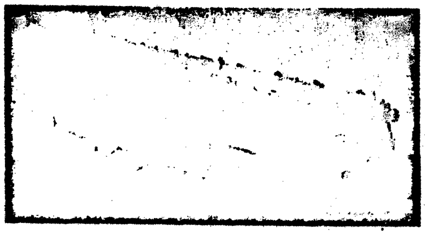
From Abacus to Computer

Most of us think of the computer as being the unique product of twentieth century technology. Yet many of the elements which are inherent in today's computers are centuries old. The abacus, developed about 3,000 years ago, was the first digital counting machine. Since then, many other "machines and engines" were developed—all of which led to the ultimate development of the modern electronic computer. Here are just a few:



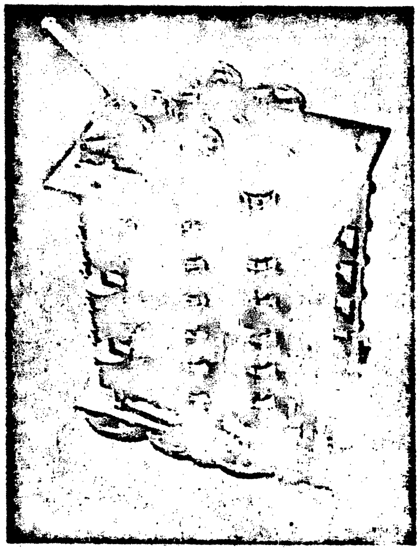
The Arithmetic Machine—1642

In the seventeenth century Blaise Pascal developed the first true calculating machine, using a technique which still is used in modern computers. A leading mathematician and philosopher in France, Pascal conceived his arithmetic machine in 1642 when he was only 19. The machine was operated by dialing a series of wheels bearing the numbers 0 to 9 around their circumferences.



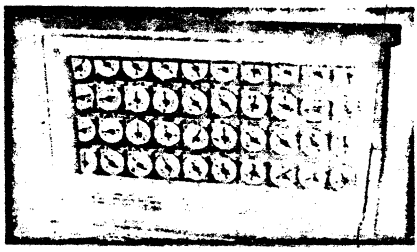
The Calculating Machine—1694

Just over fifty years later Gottfried Leibniz, also a renowned mathematician and philosopher, devised a crude machine to mechanize the calculation of mathematical tables. His calculating machine was the first machine to multiply and divide directly. More complex than Pascal's arithmetic machine, it was designed to mechanize the calculation of trigonometric and astronomical tables.



The Difference Engine—1822

This was the first of several difference engines built in the nineteenth century. Developed by Charles Babbage, a British mathematician, it accumulated differences to produce tables for navigation, astronomy and even insurance. It was capable of generating tables to a 20-place accuracy. Out of his work on the difference engine, Babbage came up with the first idea for a computer, a machine which could handle any sort of mathematical computation automatically. His "analytical engine", although never built, included all those essential parts of a computer: a stored program, an arithmetic unit and a section for data entry and output.



The Census Machine—1890

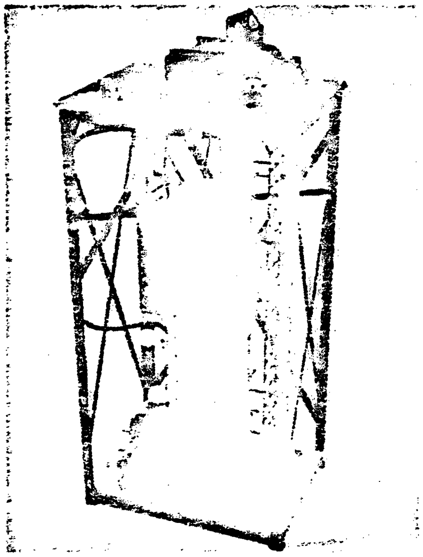
Dr. Herman Hollerith, a statistician from Buffalo, N.Y., solved a problem of major importance for the U.S. Census Bureau when he designed his electric tabulating machine in the 1880's. The problem was this: at the rate the population was growing, the eleventh census in 1890 would be obsolete before it was tabulated. Hollerith's machine solved the problem by being able to tabulate the massive amount of data electrically. The machine consisted of three parts: a tabulator which used a clock-like counting device (shown), a sorter box with compartments which were electrically connected to counters in the tabulator, and a pantographic punch, one of the first devices used to punch data onto cards.

The year 1890 marks the date the first major statistical machine was built and put into large-scale use. It was this invention of Hollerith's that launched the information-handling revolution. Afterward, many others followed who also made significant contributions leading to the development of the computer in the 1940's.

IBM®

Computing Pioneers: Part II

In an earlier issue, DP DIALOG traced the evolution of calculating devices from the abacus to the late 19th century. A number of readers asked that the story be continued. So, here are a few more of the events that led to the modern computer.



Automatic punch card sorter (c. 1910).

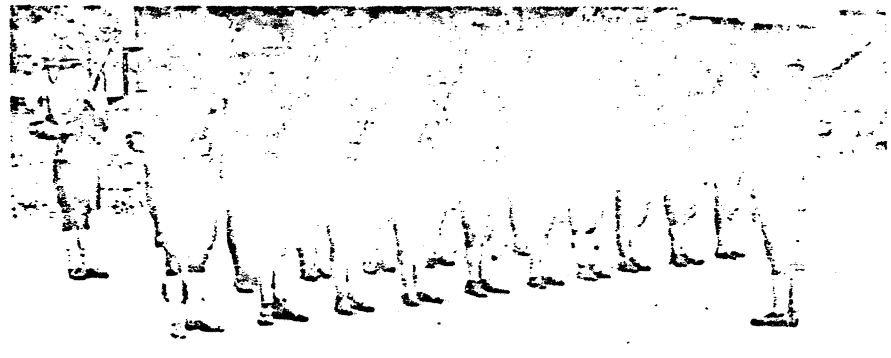
By 1890, a growing nation and its expanding industrial economy were producing numbers, figures and statistics in profusion. No longer was the census merely a matter of counting heads, for example. By 1890, it had been expanded to include statistics on immigration, race, health, literacy and employment. It was clear the government needed an efficient way to tally this wealth of information.

Herman Hollerith, a Census Bureau statistician, solved the problem with the first electrical tabulation machine. Fast and accurate, it used cards in which holes were punched to represent vital statistics. Systems like it earned growing acceptance throughout the next thirty years, although they were modified and speeded up to handle the ever-increasing needs of the government, business and scientific communities. To accommodate more information for business use Hollerith increased the size of the punched card itself. As his design model, he chose the dollar bill of the time.

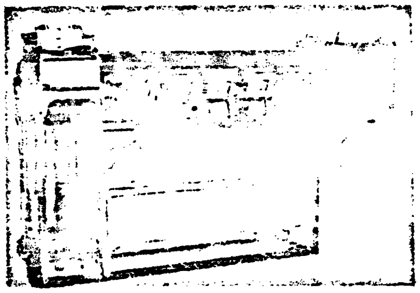
Between 1900 and 1910, railroads began using "Hollerith machines" to tabulate waybills. Insurance companies, with actuarial statistics to correlate and mortality predictions to make, were quick to see the advantages of mechanical tabulation. And public utilities, with countless customer records to maintain, also turned in growing numbers to machine accounting. More sophisticated methods—such as cost accounting and sales analysis followed.

With the United States' entrance into World War I, the Wilson administration set up a plethora of public agencies to control transportation, communication, manufacturing and distribution. Under Bernard Baruch, the War Production Board established committees on commodities as diverse (and unlikely) as baby buggies, biscuits and crackers, and pocketknives. To operate effectively, these agencies needed rapid access to vast quantities of information, and thus installed large numbers of tabulating machines.

By the thirties, many large firms had established a "tab" department, but it remained for the government to undertake the largest bookkeeping job ever. The Social Security Act of 1935 made it necessary to maintain employment records on 26 million people. To handle this task, a production line punched, sorted, checked and filed 500,000 cards a day.



During World War I, the Army used punched-card sorters in the first large-scale application of psychological testing.

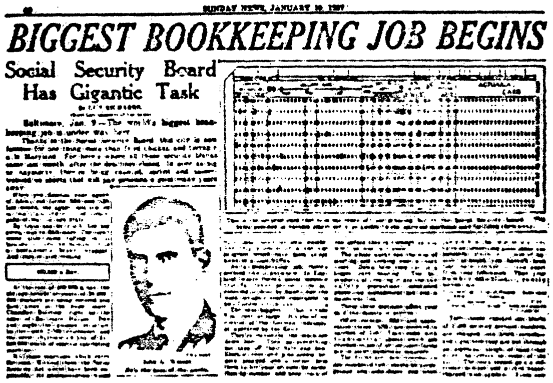


This high-speed card-sorting machine (c. 1920), handled 400 cards a minute.

By the mid-thirties technology had advanced to the point that mechanical reading, writing and arithmetic were available—but separately, as individual functions of distinct machines.

From the 1937 master's thesis of MIT student Claude Shannon came a way of using symbolic logic to improve electrical switching circuits. In one example, he showed how to automatically add two numbers using only relays and switches. Although any numbering base could be used, Shannon said, the circuit would be greatly simplified by adopting the base two.

PHOTO: COURTESY OF THE NEW YORK NEWS



SUNDAY NEWS, JANUARY 10, 1937

BIGGEST BOOKKEEPING JOB BEGINS

Social Security Board Has Gigantic Task

Baltimore, Jan. 9—The world's biggest bookkeeping job is under way here.

This article said of the IBM collators, "They're incredible. They do everything but take off their hats and bow."

That same year, working independently, George Stibitz of Bell Labs built such an adder in his home. He called it the "Model K", after the kitchen table on which it was constructed.

At about the same time, Wallace Eckert of Columbia University used a mechanical programmer to link together different kinds of punched card and accounting machines to allow complex astronomical calculations.

Such developments foreshadowed many of the advances of the next decade. As early as 1937, Harvard graduate student Howard Aiken had proposed that a new kind of calculating machine be built. It was later to become known as the Mark I, the first automatic, general purpose digital calculator.

CONTEMPORARY COMPUTER APPLICATIONS

HIERARCHY OF COMPUTER PROGRAMMING LANGUAGES

MACHINE LANGUAGE

A COMPUTER PROGRAM WRITTEN IN THE "NATIVE" LANGUAGE OF THE CPU HARDWARE WHICH INSTRUCTS THE CPU TO PERFORM BUILT-IN HARDWARE DESIGNED FUNCTIONS.

ASSEMBLY LANGUAGE

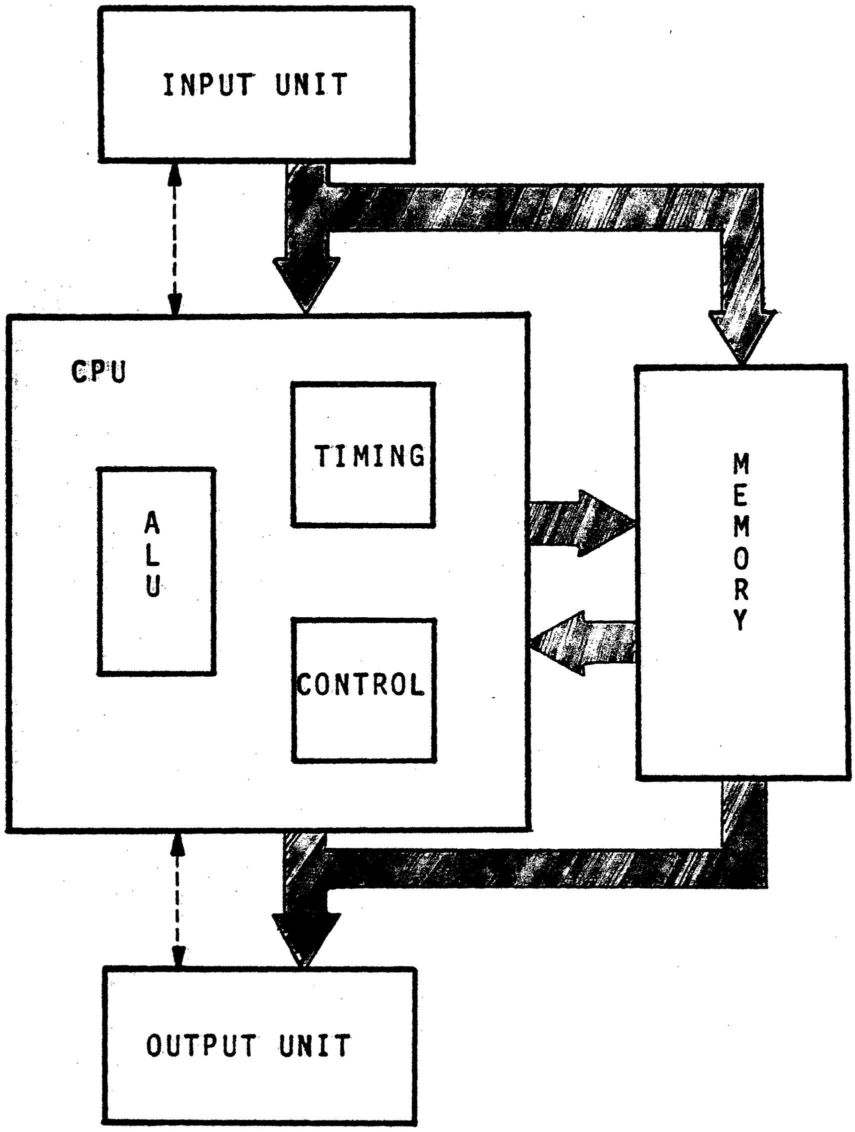
A PROGRAM WRITTEN IN A "SYMBOLIC", ENGLISH LIKE LANGUAGE RATHER THAN THE COMPUTER'S OWN MACHINE LANGUAGE. THE ASSEMBLY LANGUAGE PROGRAM IS TRANSLATED BY AN "ASSEMBLER" PROGRAM, RESULTING IN A MACHINE LANGUAGE PROGRAM FOR ACTUAL EXECUTION: THE TWO PROGRAMS ARE DIRECTLY RELATED WITH ONE ASSEMBLY LANGUAGE INSTRUCTION RESULTING IN ONE MACHINE LANGUAGE INSTRUCTION WORD. SOME ASSEMBLER "DIRECTIVES" WILL RESERVE BLOCKS OF COMPUTER MEMORY STORAGE SPACE, HOWEVER.

COMPILER LANGUAGE

A PROGRAM WRITTEN IN A HIGH LEVEL, GENERALLY ENGLISH LANGUAGE. THE TRANSLATOR, OR COMPILER, WILL TRANSLATE THE PROGRAM INTO ONE OR MORE EQUIVALENT MACHINE LANGUAGE INSTRUCTION WORDS. MANY COMPILER LANGUAGES ARE UNIVERSALLY ADOPTED AND CAN BE USED FOR MANY DIFFERENT COMPUTERS; EACH COMPUTER MANUFACTURER'S COMPILER, HOWEVER, IS DESIGNED TO GENERATE MACHINE LANGUAGE INSTRUCTION WORDS WHICH ARE UNIQUE TO THEIR OWN COMPUTER.

COMPARISON OF COMPILER/ASSEMBLY/MACHINE LANGUAGES FOR FST-2 COMPUTER

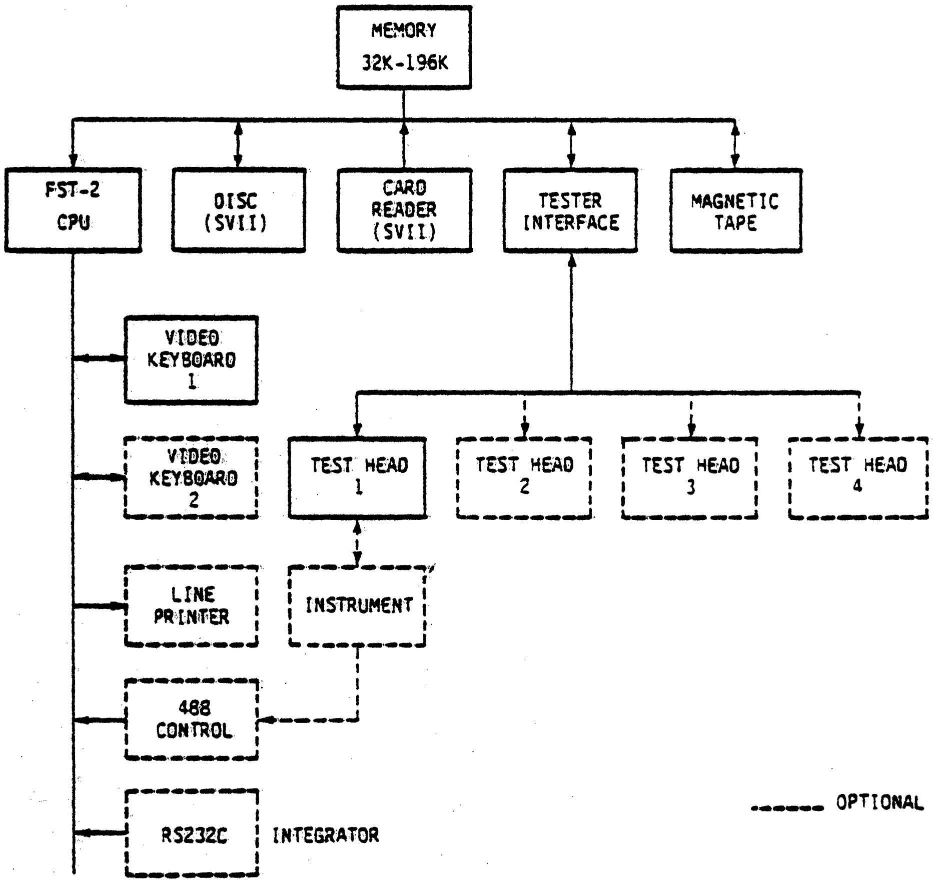
COMPILER LANGUAGE	ASSEMBLY LANGUAGE			MACHINE LANGUAGE	
				ADDRESS	CONTENT
A = 1;		LDA	A	00500	24000600
B = 2;		ADD	B	00501	20000601
C = 3;		ADD	C	00502	20000602
D = A + B + C;		STA	D	00503	14000603
	A	DATA	1	00600	00000001
	B	DATA	2	00601	00000002
	C	DATA	3	00602	00000003
	D	DATA	0	00603	00000000



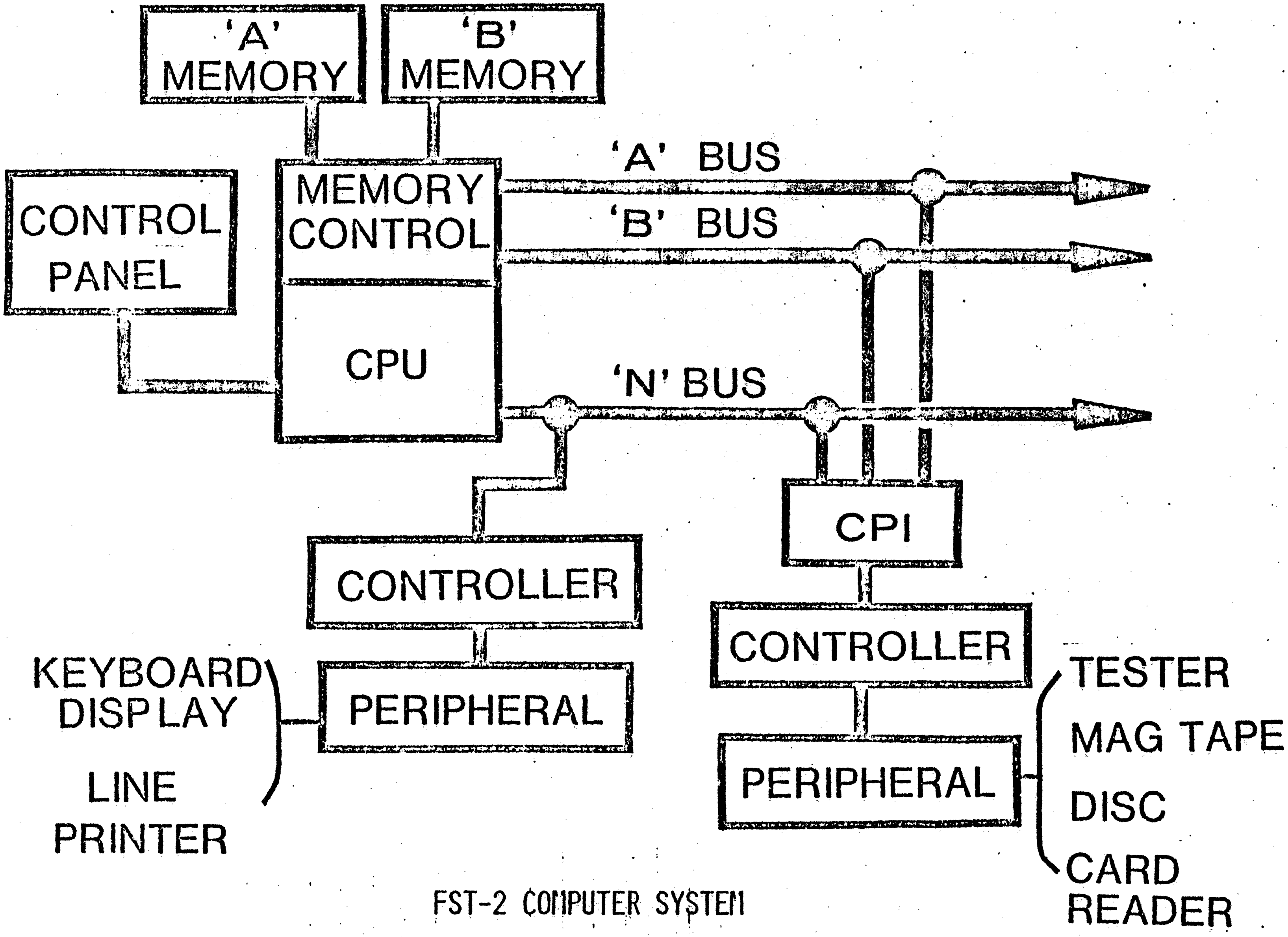
THE BASIC UNITS OF THE DIGITAL COMPUTER SYSTEM

FST-2

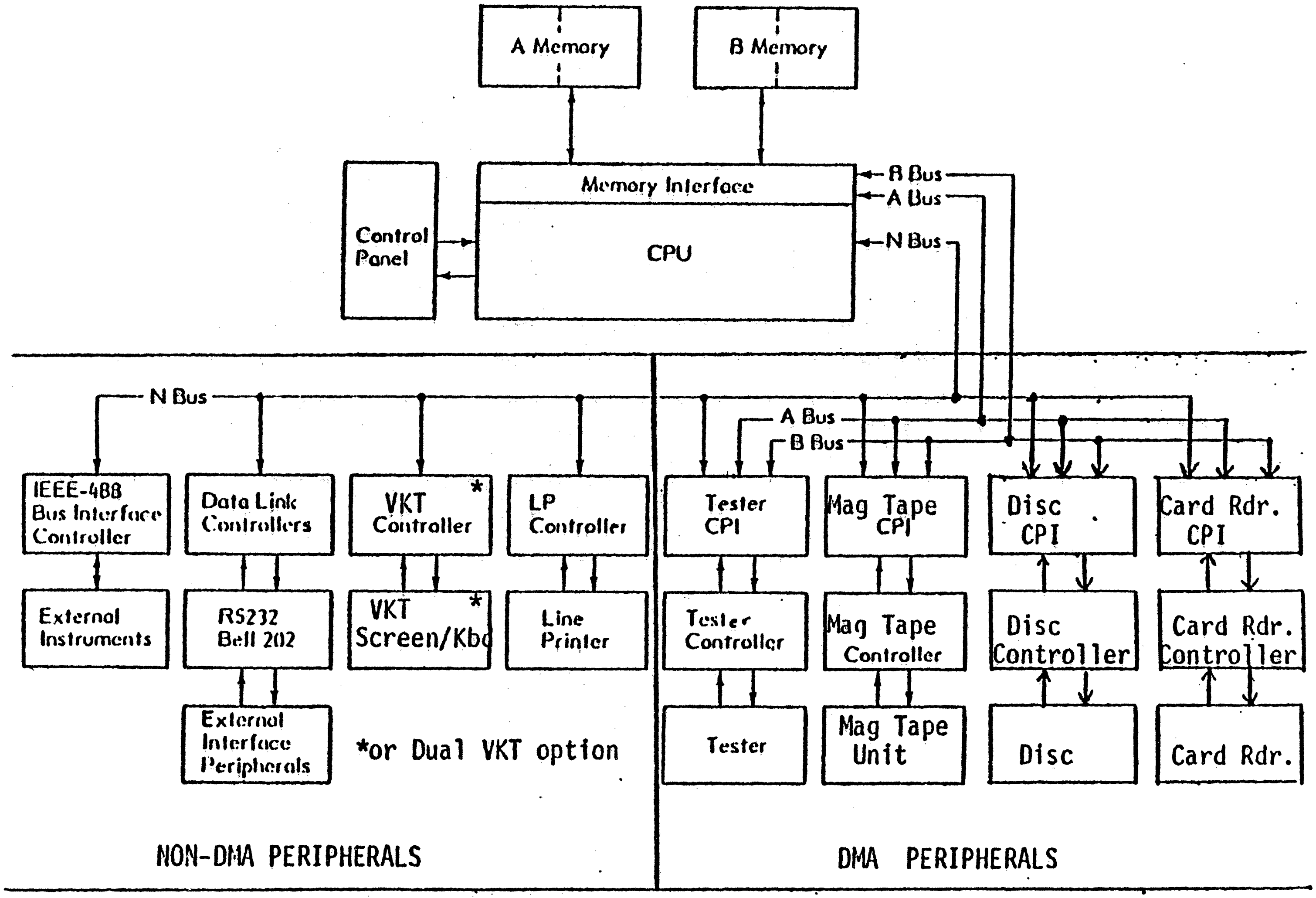
COMPUTER SYSTEM DESCRIPTION



Information Flow Through Peripherals for MASTR Operating System



FST-2 COMPUTER SYSTEM
SIMPLIFIED BLOCK DIAGRAM



FST-2 System Configuration

FST-2 COMPUTER HARDWARE FEATURES

- LARGE MAIN MEMORY USED FOR PROGRAM STORAGE
  - USES SEMICONDUCTOR MEMORY (196K 24-BIT WORDS MAX)
  - MEMORY CYCLE TIME OF 1.75 MICROSECONDS

- TWO MEMORY BUSSES INTERFACE MEMORY TO CPU AND PERIPHERALS
  - EACH MEMORY BUS HAS DMA (DIRECT MEMORY ACCESS) WITH 16 DMA CHANNELS PER BUS
  - DMA TRANSFER RATE OF 571,428 WORDS PER SECOND PER BUS

4 bits of Address for Different Peripheral address of UKT ⇒ DISC →

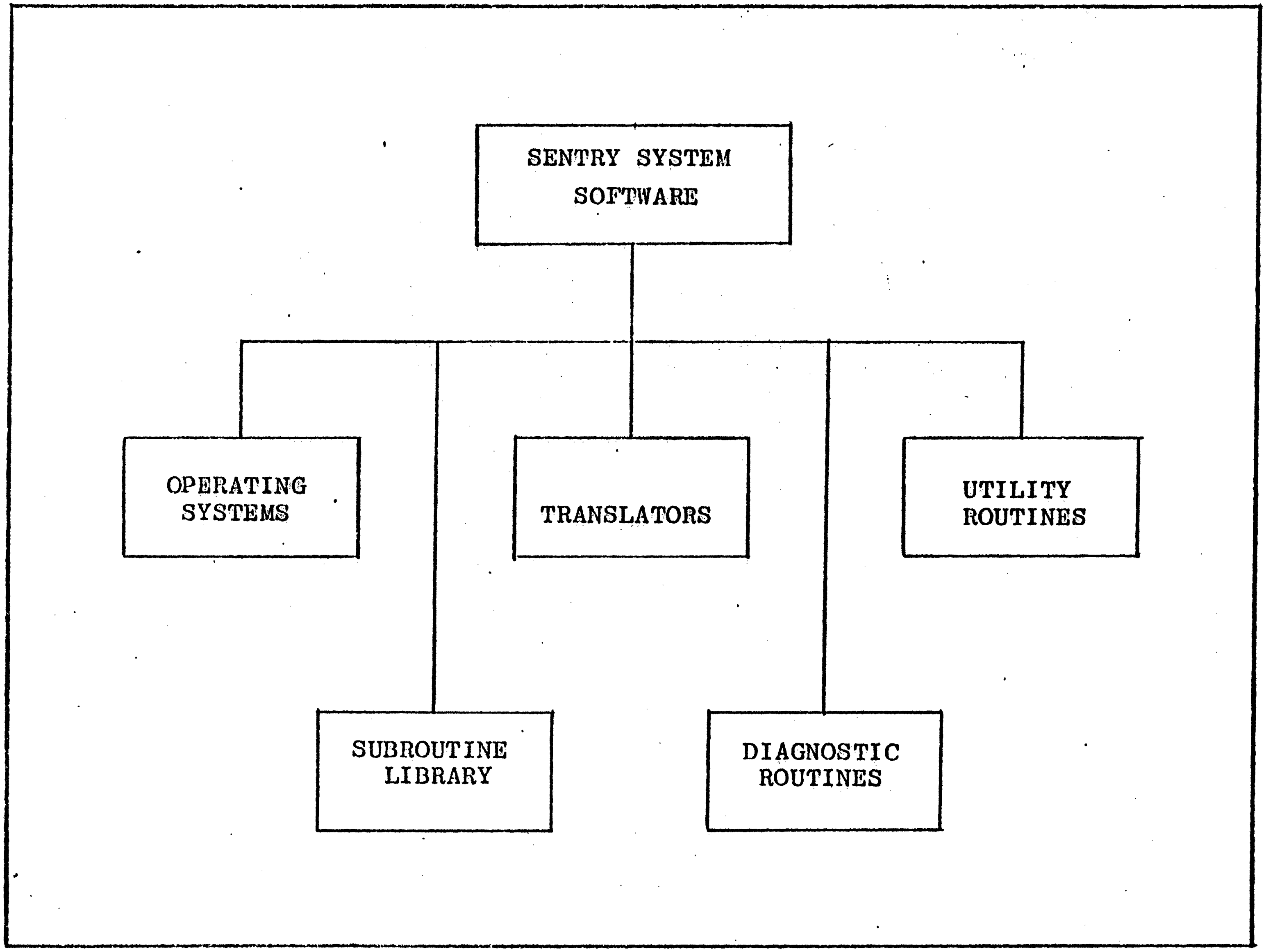
- INSTRUCTION WORD MODIFICATION USING:
  - SINGLE LEVEL INDIRECT ADDRESSING
  - INDEXING OPERATIONS
  - RELOCATION REGISTER
  - VARIOUS FORMS OF OPERAND ARITHMETIC

358
1.750

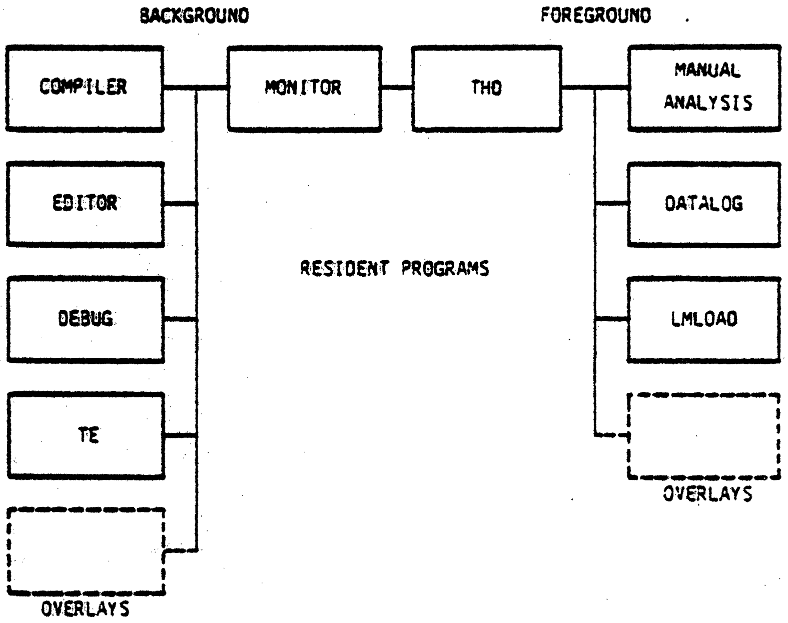
- EIGHT HARDWARE INDEX REGISTERS PROVIDE:
  - INSTRUCTION WORD OPERAND MODIFICATION
  - COUNTING AND COMPARE OPERATIONS

- HARDWARE MULTIPLY AND DIVIDE CIRCUITRY

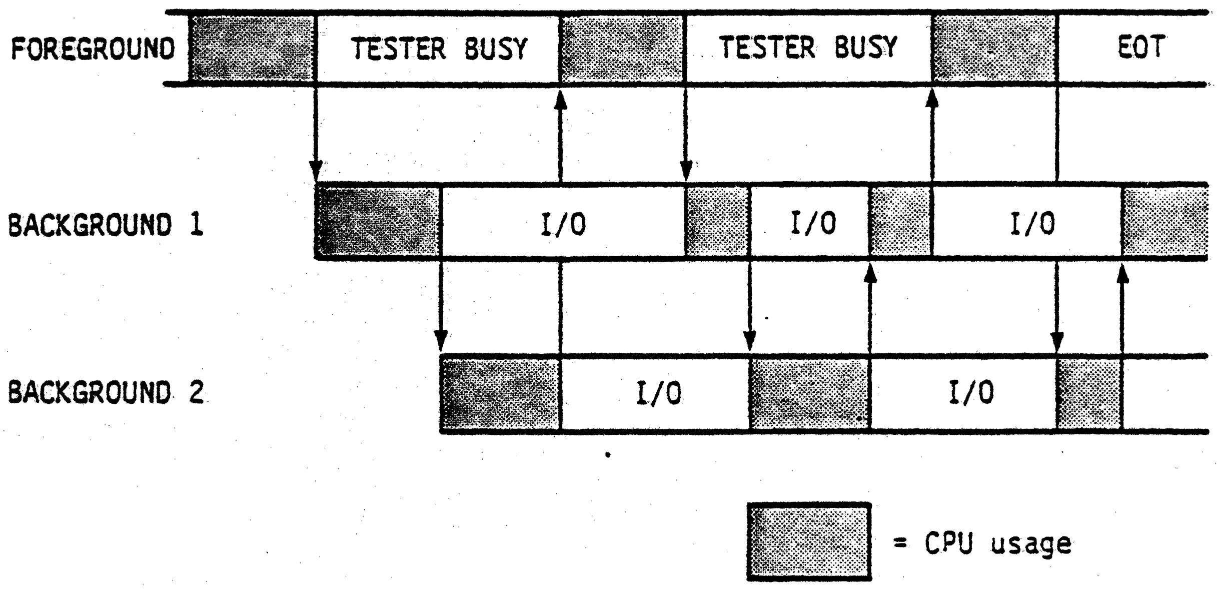
- PRIORITY INTERRUPT STRUCTURE
  - CPU ALLOWS UP TO 63 INTERRUPTS ⇒ 778
  - EACH INTERRUPT HAS AN INDIVIDUAL INTERRUPT SERVICE ROUTINE POINTER LOCATION IN CPU MEMORY
  - MULTIPLE INTERRUPTS FOR EACH PERIPHERAL ARE POSSIBLE


SENTRY SYSTEM SOFTWARE
OPERATING SYSTEMS
TRANSLATORS
UTILITY ROUTINES
SUBROUTINE LIBRARY
DIAGNOSTIC ROUTINES

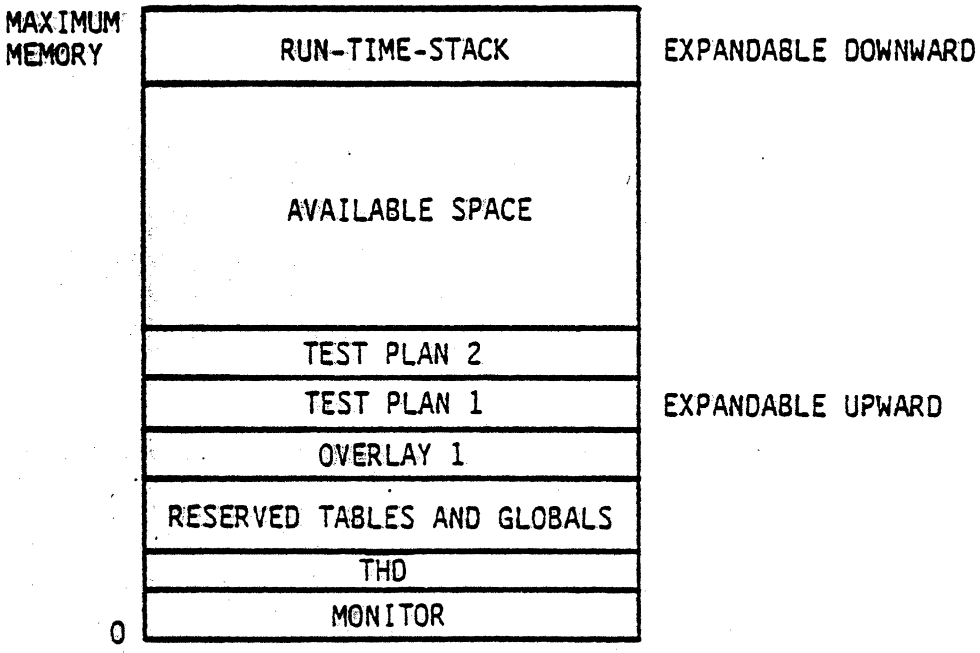
MEMORY MANAGEMENT UNDER MASTR



MASTR Operating System Program Map



Foreground/Background Switching Under the MASTR Operating System



Software Boundaries Under the MASTR Operating System

```
<MASTR

*NAME

 1-3-78    16:33

     NAME    TYPE     JOB    F-SIZE  ADDRESS   M-SIZE KEEP BSY STAT

   1 MACTAB  S.      ←←←←      256    515213B     256   Y
   2 IOATAB  S       ←←←←      260    511115B     260   Y
   3 LMSAVE  OVLY    ←←←←     2546    521213B    5714   N         1
   4 LMLOAD  OVLY    ←←←←      854    652433B    3926   Y
   5 EDIT    OVLY    ←←←←     6371    747771B    6371   Y
   6 COMPIL  OVLY    ←←←←     9585   1113348B   12585   Y
   7 G6800   TP      684K     4393   1420053B    4393   N         1
   8 MA      OVLY    ←←←←     2746   1524568B    2746   N         1
   9 MNEMON  U       684K      513   1577508B     513   N

 39784 WORDS LEFT         23 ENTRIES LEFT
```

THE FOLLOWING DEFINITIONS APPLY TO ACTION AUTOMATICALLY TAKEN AGAINST FILES IN MEMORY BY THE SYSTEM WHEN INSUFFICIENT MEMORY SIZE EXISTS TO LOAD A NEW FILE.

RELEASE - REMOVE FROM MEMORY AND PURGE MACTAB ENTRY

BUMP - REMOVE FROM MEMORY BUT KEEP MACTAB ENTRY TO ALLOW FASTER RELOAD

KEEP - NEVER RELEASE OR BUMP EXCEPT WITH RELEASE COMMAND

STAT - ATTACHED TO GIVEN STATION. DON'T BUMP UNLESS TEST PLAN OR MOD FILE.

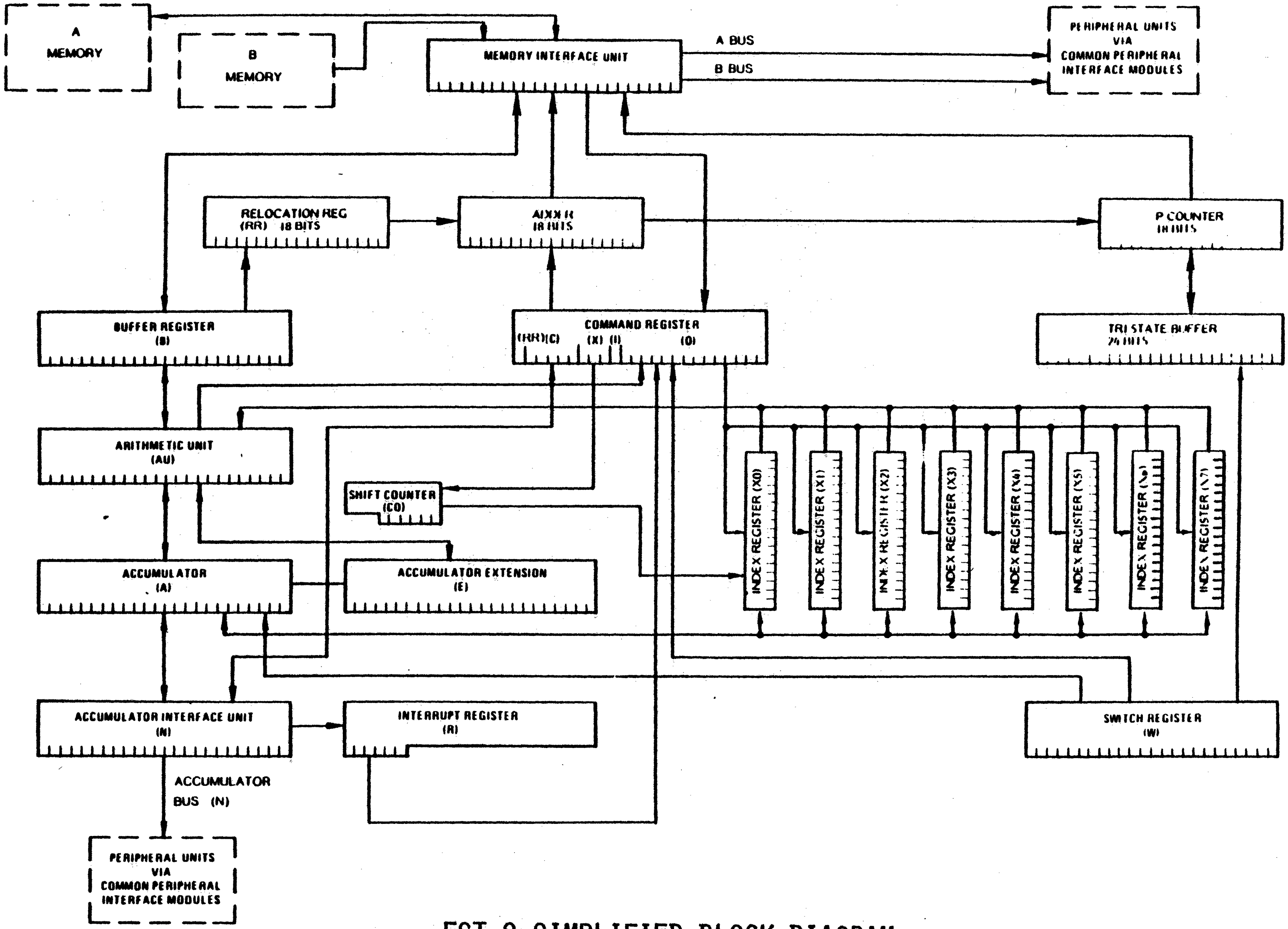
ATTACH/KEEP RELATIONSHIP

	ATTACHED[1]		NOT ATTACHED	
KEEP (Y)	NEVER RELEASE[2] NEVER BUMP		NEVER RELEASE NEVER BUMP	
DON'T KEEP (N)	OVLY	TP/MOD[3]	OVLY	TP/MOD
	NEVER RELEASE NEVER BUMP	BUMP ONLY NEVER RELEASE	RELEASE	RELEASE

1) A STATION NUMBER IN THE 'STAT' COLUMN INDICATES THAT THE GIVEN FILE IS ATTACHED TO THE SPECIFIED STATION.

2) RELEASED ONLY VIA USER RELEASE COMMAND.

3) OVLY = ASSEMBLY LANGUAGE F.G. OR B.G. PROGRAM
   TP = FACTOR PROGRAM
   MOD = LMLOAD 'LMI' FILE



FST-2 SIMPLIFIED BLOCK DIAGRAM

ACCUMULATOR BUSS USES

DATA TRANSFERS

1. CPU selects a peripheral controller to perform a specific function by transmitting SPU instruction word via the 24-bit accumulator buss.

2. When the specified peripheral recognizes it's Unit Address, it then transmits it's general status back to the CPU via the four most significant bits of the accumulator buss.

3. a. If the peripheral is a non-DMA type and the general status was "not busy" then 24-bits of data are parallel transferred between the CPU Accumulator Register and the peripheral (the direction of the data transfer is determined by the individual SPU instruction).

   b. If the peripheral is a DMA type and the general status was not busy then the 14 least significant bits of the Accumulator Register are transferred to the peripheral's CPI (the 14 bits are interpreted as a Data Control Block address pointer - the DCB determines the amount of data and the direction of transfer via the memory buss).

PRIORITY INTERRUPT SYSTEM

1. If the Interrupt System is properly armed and an "interrupt" occurs, the 16 least significant bits of the Accumulator buss are examined by the CPU to resolve interrput priority.

2. After "priority" is resolved and a specific peripheral is given "exclusive" control, the 6 least significant bits are used to transmit the interrupt address of the peripheral back to the CPU and are placed into the Interrupt Register.

	INTERRUPT	
PERIPHERAL	PRIORITY	ADDRESS
TESTER	9	12 - 21
DISC	8	07
MAG TAPE	7	10
CARD READER	5	04
LINE PRINTER	3	06
VKT DISPLAY	2	03
VKT KEYBOARD	1	02
DATA SET (COMM LINK)	11	30 - 37
488 BUS (I BUS)	6	40 - 43

MEMORY INTERFACE SYSTEM

- TWO MEMORY BUSSES (A AND B) PROVIDE FOR DATA TRANSFER BETWEEN MEMORY AND THE CPU, OR MEMORY AND DMA PERIPHERALS

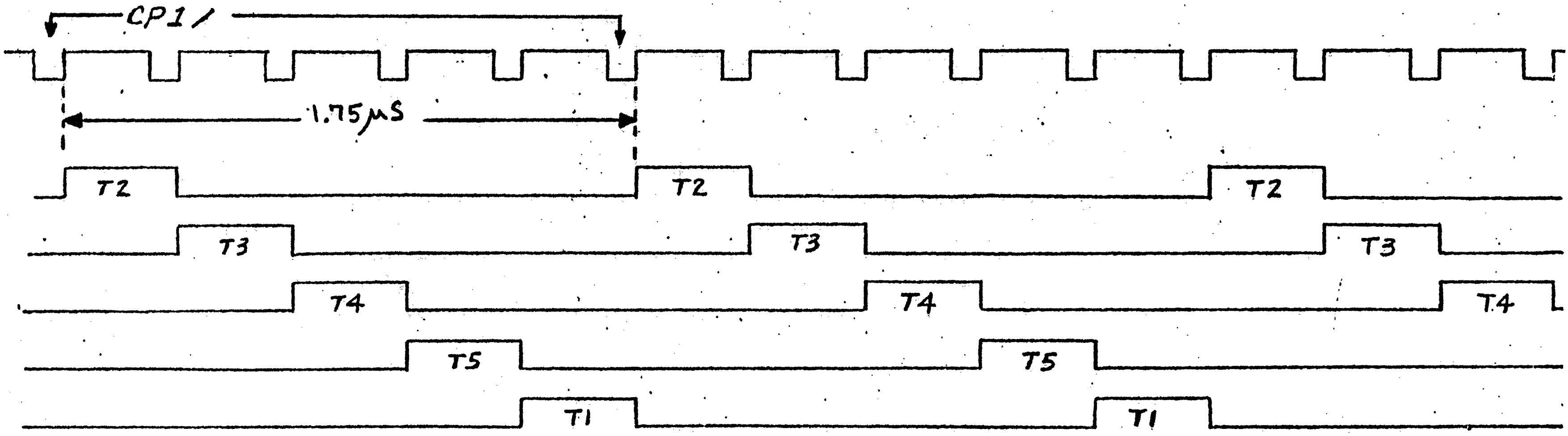
- A DATA TRANSFER BETWEEN MEMORY AND THE CPU (ONE WORD AT A TIME) IS ACCOMPLISHED BY EXECUTING A "MEMORY REFERENCE INSTRUCTION"

- A DATA TRANSFER BETWEEN MEMORY AND A DMA PERIPHERAL IS ACCOMPLISHED BY EXECUTING AN SPU (SELECT PERIPHERAL UNIT) INSTRUCTION WHICH ADDRESSES A DMA PERIPHERAL. THIS CAUSES A BLOCK OF DATA TO BE TRANSFERRED, AS DEFINED BY A DCB (DATA CONTROL BLOCK), WITHOUT ANY FURTHER INTERVENTION BY THE CPU OR PROGRAM

- DATA TRANSFERS BETWEEN MEMORY AND CPU/PERIPHERALS ARE INITIATED ON A PRIORITY BASIS, WITH THE 16 LEAST SIGNIFICANT BITS OF EACH MEMORY BUS USED TO RESOLVE PRIORITY AS FOLLOWS:

PERIPHERAL	MEMORY PRIORITY
DISC	12 (high)
MAG TAPE	8
CARD READER	5
TESTER	1
CPU	0 (low)

FST-2 TIMING



T2 = MEMORY ACCESS TIME

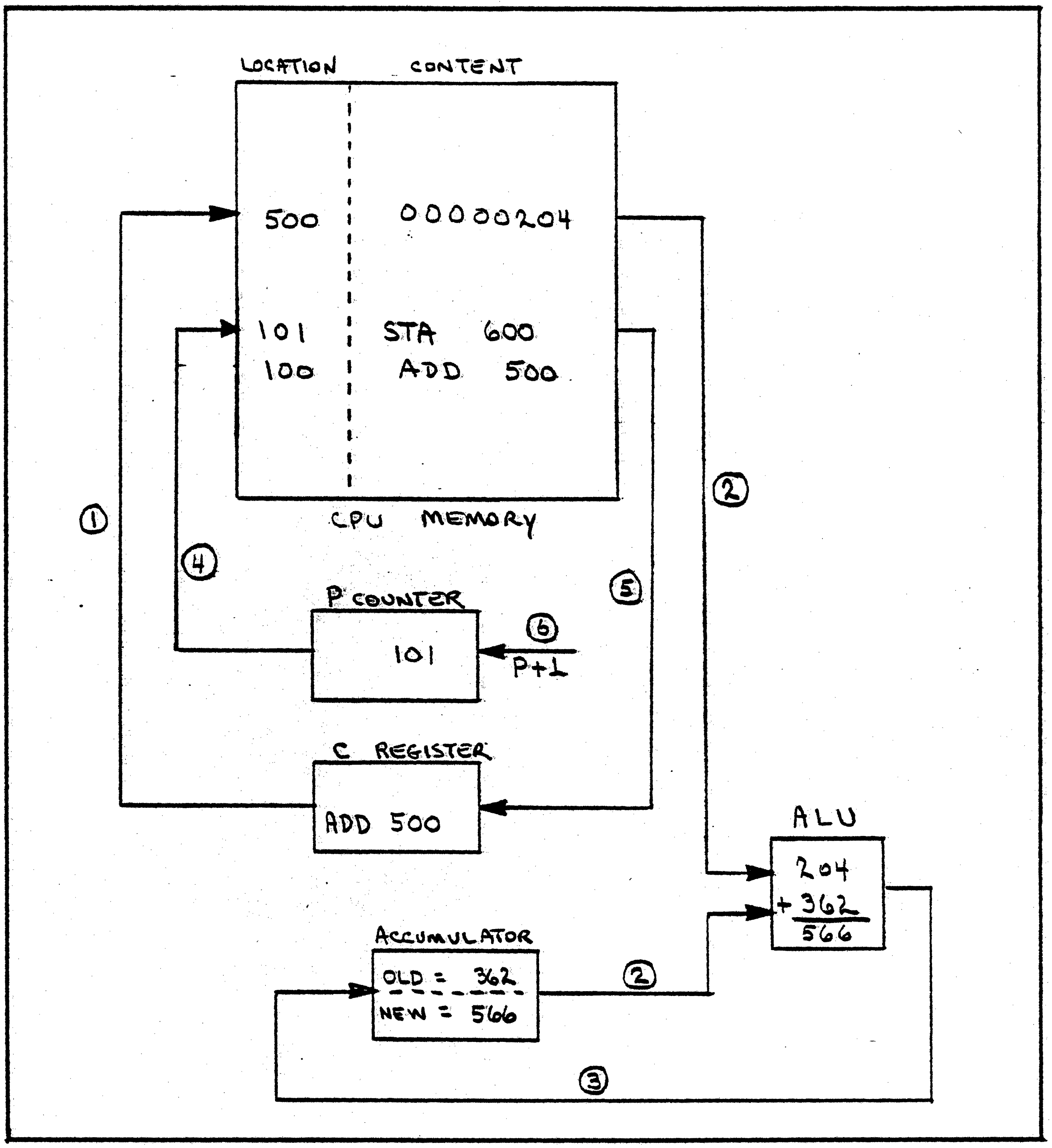
T3 = MEMORY ADDRESS

T4 = PERIPHERAL SELECT TIME

T5 = MEMORY WRITE

T1 = MEMORY READ

CP1/ = CLOCK PULSE 1



"ADD" INSTRUCTION EXECUTION SEQUENCE

① INSTRUCTION DECODE AND OPERAND FETCH
② EXECUTE INSTRUCTION OPERATION
③ STORE RESULT IN ACCUMULATOR
④ FETCH NEXT INSTRUCTION (ADDRESS SPECIFIED BY P-COUNTER)
⑤ LOAD INSTRUCTION INTO C-REGISTER
⑥ ADD +1 TO P-COUNTER

FST-2 COMPUTER OVERFLOW CONDITIONS

DEFINITION:

The OV indicator will be turned on at the CPU front panel when executing an arithmetic instruction (i.e. ADD, DADD, SUB, DSUB, MUL, DIV, AOM, SOM) which causes an "arithmetic overflow" in the accumulator or a core memory location.

An "arithmetic overflow" is one in which the arithmetic operation attempts to produce a number which exceeds the maximum positive or negative value allowed by the FST-2.

EXAMPLES:

1. Valid overflow conditions:

```
37777777        40000000
     + 1             - 1
---------      ----------
40000000        37777777
```

NOTE: In the above examples, the maximum positive or negative values allowed by the CPU were exceeded and a sign change occurred.

2. Non-overflow conditions:

```
77777777        00000000
     + 1             - 1
---------      ----------
00000000        77777777
```

NOTE: In the above examples, even though a sign change did occur, the resulting answer did not exceed the maximum positive or negative numbers allowed by the FST-2.

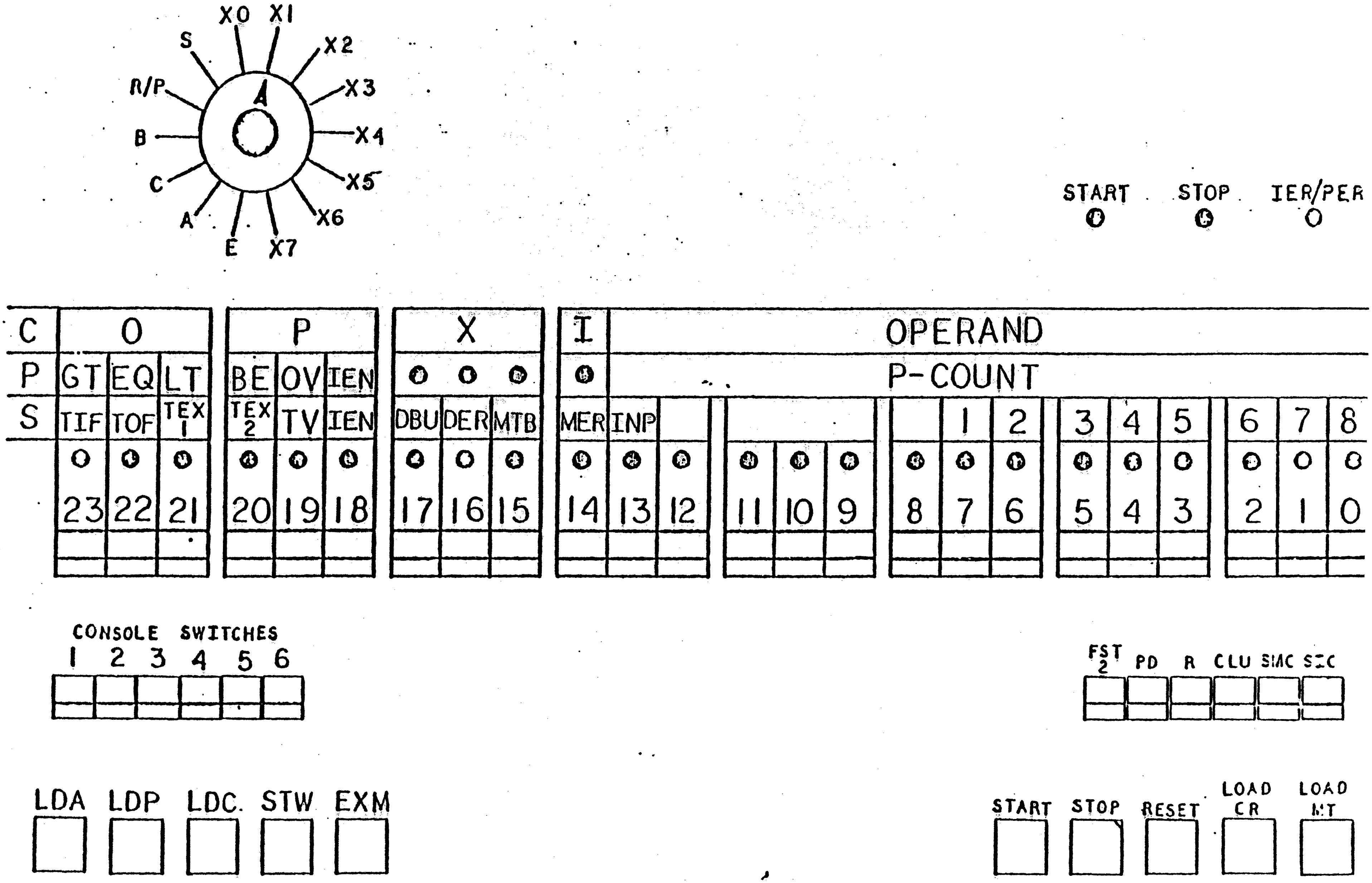
CREATING AND EXECUTING
ASSEMBLY LANGUAGE PROGRAMS
UNDER MASTR

BACKGROUND EXECUTION (VIA A MASTR KEYBOARD COMMAND)

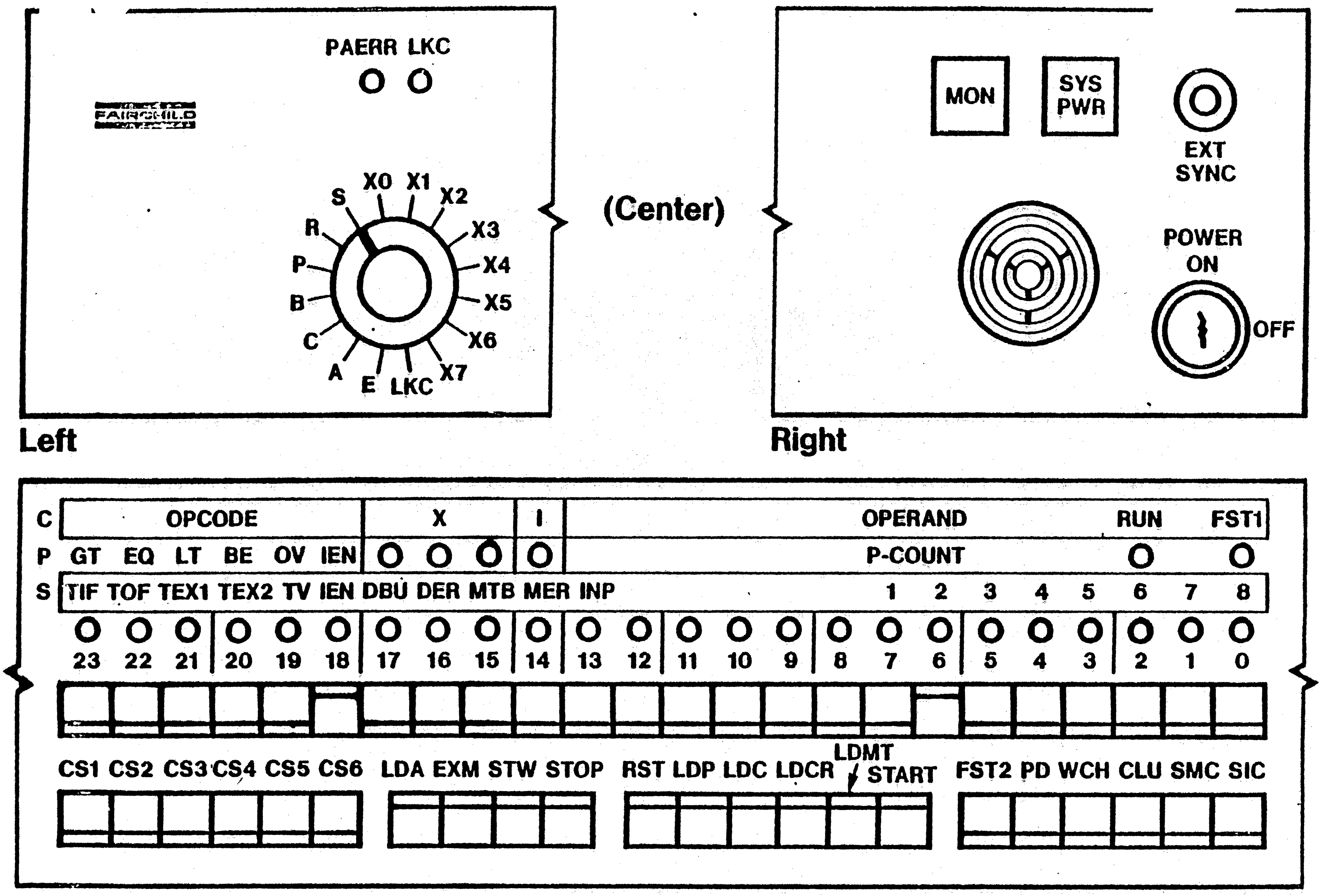
```
1.  *JOB 'USER'
2.  *COPY CR '*NAME'
3.  *ASM '*NAME' '=NAME' LIST LP SYM XREF
4.  *CREATE 'NAME' COREIMAGE '=NAME'
5.  *NAME (PARAMETER LIST)
```

FOREGROUND EXECUTION (VIA A FACTOR 'EXEC' STATEMENT)

```
1.  *JOB 'USER'

2.  *COPY CR '*ALP'
    *ASM '*ALP' '=ALP' LIST LP SYM XREF
    *CREATE 'ALP' COREIMAGE '=ALP'

3.  *COPY CR '*FACTOR'
    *COMPILE '*FACTOR' 'FACTOR' LIST LP XREF

4.  *LOAD 'FACTOR' STATN

5.  *START STATN
```



C	O			P			X			I	OPERAND													
P	GT	EQ	LT	BE	OV	IEN	○	○	○	○	P-COUNT													
S	TIF	TOF	TEX 1	TEX 2	TV	IEN	DBU	DER	MTB	MER	INP							1	2	3	4	5	6	7	8
	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

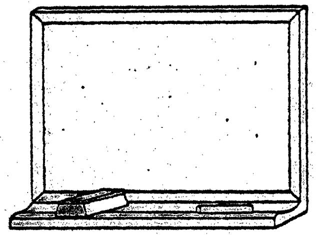

PAERR LKC
FAIRCHILD
S X0 X1 X2 X3 X4 X5 X6 X7 LKC E A C B P R
(Center)
MON
SYS PWR
EXT SYNC
POWER ON
OFF
C OPCODE X I OPERAND RUN FST1
P GT EQ LT BE OV IEN P-COUNT
S TIF TOF TEX1 TEX2 TV IEN DBU DER MTB MER INP 1 2 3 4 5 6 7 8
23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
CS1 CS2 CS3 CS4 CS5 CS6
LDA EXM STW STOP
RST LDP LDC LDCR LDMT START
FST2 PD WCH CLU SMC SIC

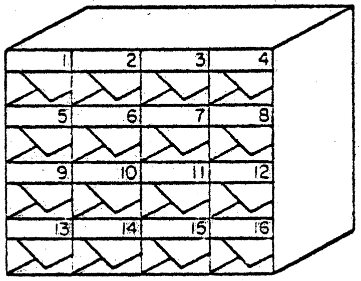
Left

Right

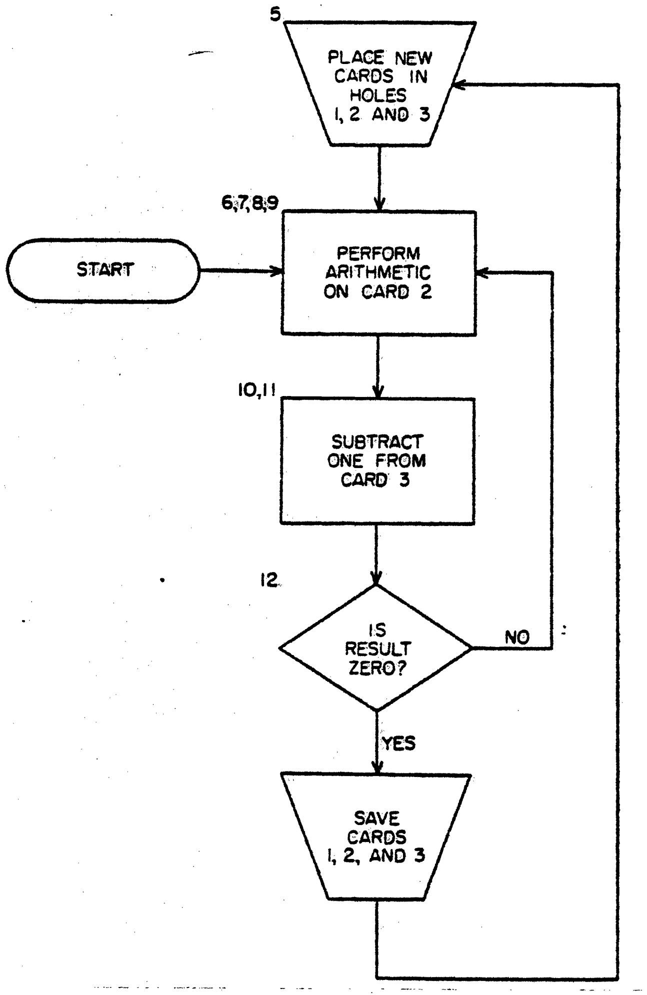
Center

THE STORED PROGRAM CONCEPT





CARD NUMBER	*CONTENT*
1	200
2	106
3	5
4	1
5	Place new cards in holes 1, 2, and 3
6	Erase slate, write number from card 1
7	Multiply by number on card 2
8	Round off answer two places
9	Replace number on card 1 with number from slate
10	Erase slate, write number from card 3
11	Subtract number on card 4
12	If result is zero go to card 15
13	Write number from slate on card 3
14	Go back to card 6
15	Remove cards 1, 2, and 3 and save for boss
16	Go back to card 5



FLOWCHART OF PIGEON-HOLE ANALOGY

LOCATION	CONTENT	COMMENT
1	XXXX	Principal Temp Storage
2	XXXX	Interest Rate Temp Storage
3	XXXX	Term of Loan Temp Storage
4	0001	
5	READ CR 3, 1	Read 3 Cards and Store Contents Beginning In Location 1
6	LS 24	Clear Accumulator (Logical Shift 24)
7	LDA 1	Load A From Location 1
10	MUL 2	Multiply By Content of Location 2
11	SR 2	Shift Right 2 Places
12	STA 1	Store A in Location 1
13	LS 24	Clear Accumulator (Logical Shift 24)
14	LDA 3	Load A From Location 3
15	SUB 4	Sub Contents of Location 4
16	BZ 21	If AC = 0, Branch to Location 21
17	STA 3	Store A in Location 3
20	BRU 6	Branch Unconditionally Location 6
21	WRITE LP 3, 1	Write Contents of Location 1,2,43 On Line Printer
22	BRU 5	Branch Unconditional to Location 5

```
LABEL     INSTRUCTION

PRIN      XXXX
INTRAT    XXXX
TERM      XXXX
SUBYR     0001
INITAL    READ CR 3, PRIN
NEXT      LS 24
          LDA PRIN
          MUL INTRAT
          SR 2
          STA PRIN
          LS 24
          LDA TERM
          SUB SUBYR
          BZ ANSWER
          STA TERM
          BRU NEXT
ANSWER    WRITE LP 3, PRIN
          BRU INITAL
```

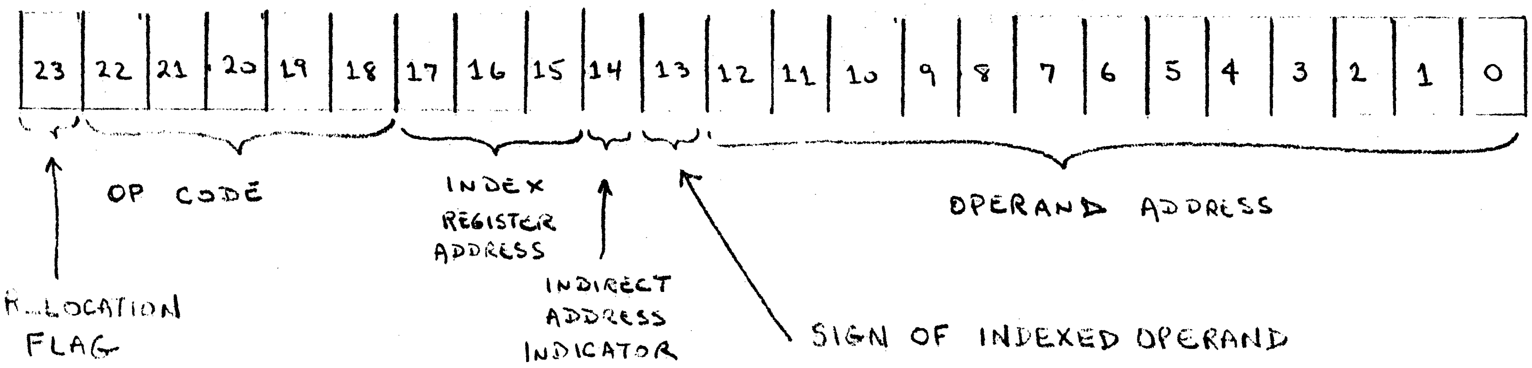
See Masth operating Manual Appendix A

```
            REL
0    FWORD  DATA   0
1,2         TEXT   'ANAME'        PROGRAM NAME
3           DATA   77B            DEFINES GENERAL OVERLAY
4           DATA   '1.0'          REVISION NUMBER
5           DATA   LWORD-FWORD    PROGRAM SIZE
6           BSS    6
14   RSENT  PZE    0
15          BRU*   RSENT
16          BSS    2
20   RLENT  PZE    0              RELEASE ENTRY POINT
21          BRU*   RLENT
22   BGENT  PZE    0              BACKGROUND ENTRY POINT
23          BRU    BGSTRT         BRU TO START OF BACKGROUND PROG
24   FGENT  PZE    0              FOREGROUND ENTRY POINT
25          BRU*   FGENT
     *
            DATA   2
            TEXT   '10/17/79'     PROGRAM RELEASE DATE
     *
     BGSTRT BSM    READ           BRANCH TO READ SUBROUTINE
     NEXT   CLA    0              CLEAR ACCUMULATOR
            LDA    PRIN           LOAD ACCUMULATOR WITH PRINCIPAL
            MUL    INTRAT         MULTIPLY BY INTEREST RATE
            SR     2
            SL     2
            STA    PRIN           STORE NEW PRINCIPAL
            CLA    0              CLEAR ACCUMULATOR
            LDA    TERN           LOAD "A" WITH TERM OF LOAN
            SUB    SUBYR          SUBTRACT ONE YEAR
            BZ     ANSWER         BRANCH IF TERM OF LOAD COMPLETE
            STA    TERM           STORE NEW TERM
            BRU    NEXT           COMPUTE INTEREST FOR NEXT YEAR
     ANSWER BSM    WRITE          BRANCH TO WRITE SUBROUTINE
            BRU*   BGENT          EXIT PROGRAM THRU BACKGROUND ENTRY
```

```
*  "READ" SUBROUTINE
READ    PZE    0              ENTRY POINT TO READ SUBROUTINE
         .
         .
         .
        BRU*   READ           EXIT READ SUBROUTINE
*  "WRITE" SUBROUTINE
WRITE   PZE    0              ENTRY POINT TO WRITE SUBROUTINE
         .
         .
         .
        BRU*   WRITE          EXIT WRITE SUBROUTINE
*
PRIN    DATA   0              PRINCIPAL TEMP STORAGE
INTRAT  DATA   0              INTEREST RATE TEMP STORAGE
TERM    DATA   0              TERM OF LOAN TEMP STORAGE
SUBYR   DATA   1
*
LWORD   EQU    *
*
        END                   END OF PROGRAM
```

BASIC ASSEMBLY LANGUAGE PROGRAM LAYOUT
(BACKGROUND OVERLAY)

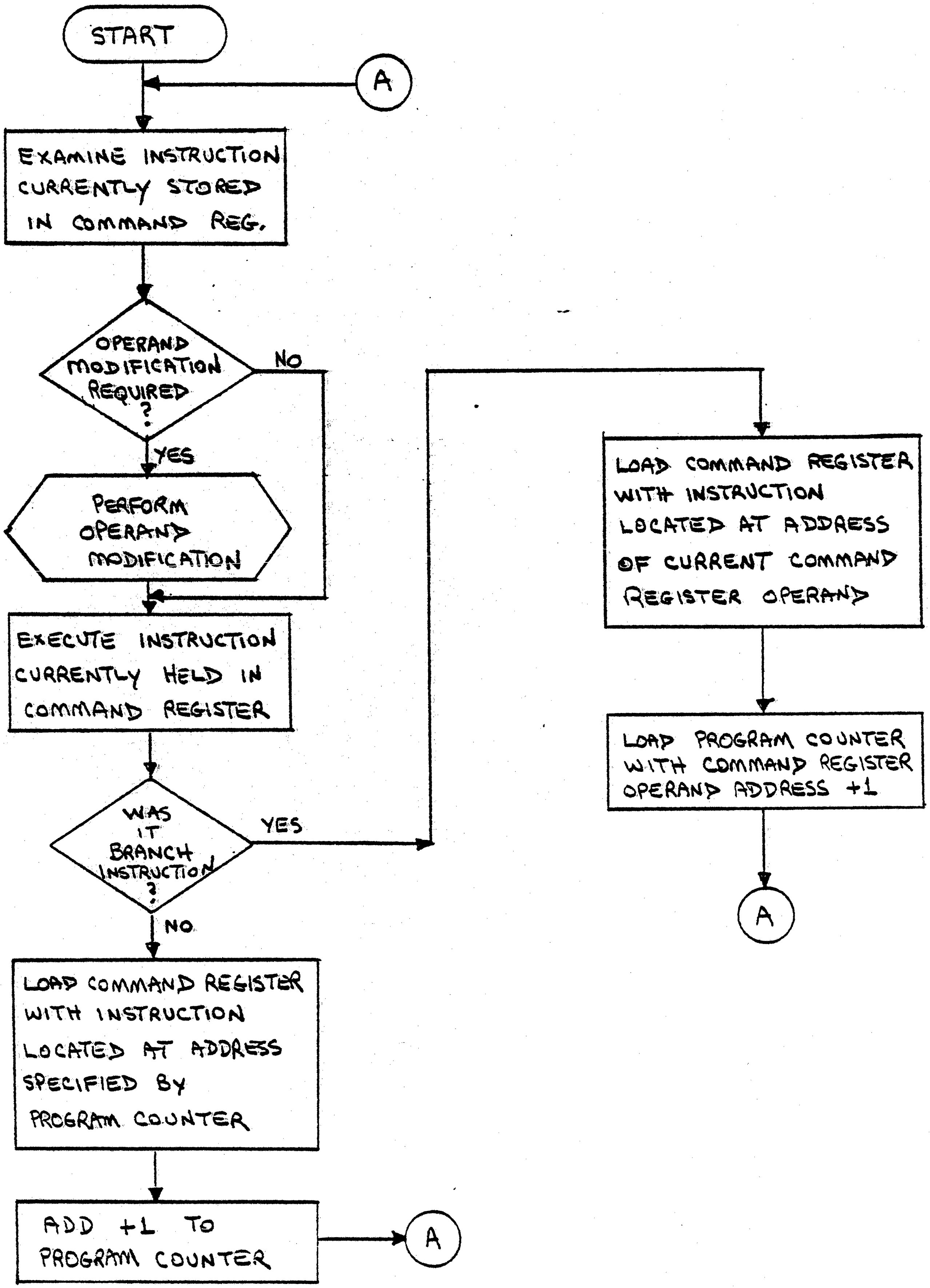
```
        REL
*  EQUATE TABLE - IF USED
CS1     EQU    12B
         .
*  22-WORD HEADER
FWORD   DATA   0
         .
         .
BGENT   PZE    0            BACKGROUND ENTRY POINT
        BRU    BGSTRT
         .
*        .
   END OF HEADER
*  START OF PROGRAM MAIN BODY
BGSTRT  LDA    N
         .
         .
         .
         .
         .
        BRU*   BGENT        BACKGROUND EXIT
*  END OF PROGRAM MAIN BODY
*  SUBROUTINES - IF USED
NAME    PZE    0            SUBROUTINE ENTRY
         .
         .
        BRU*   NAME         SUBROUTINE EXIT
*  END OF SUBROUTINES
*  DATA TABLE - IF USED
D100    DATA   100
0100    DATA   100B
LWORD   EQU    *
        END                END OF PROGRAM
```



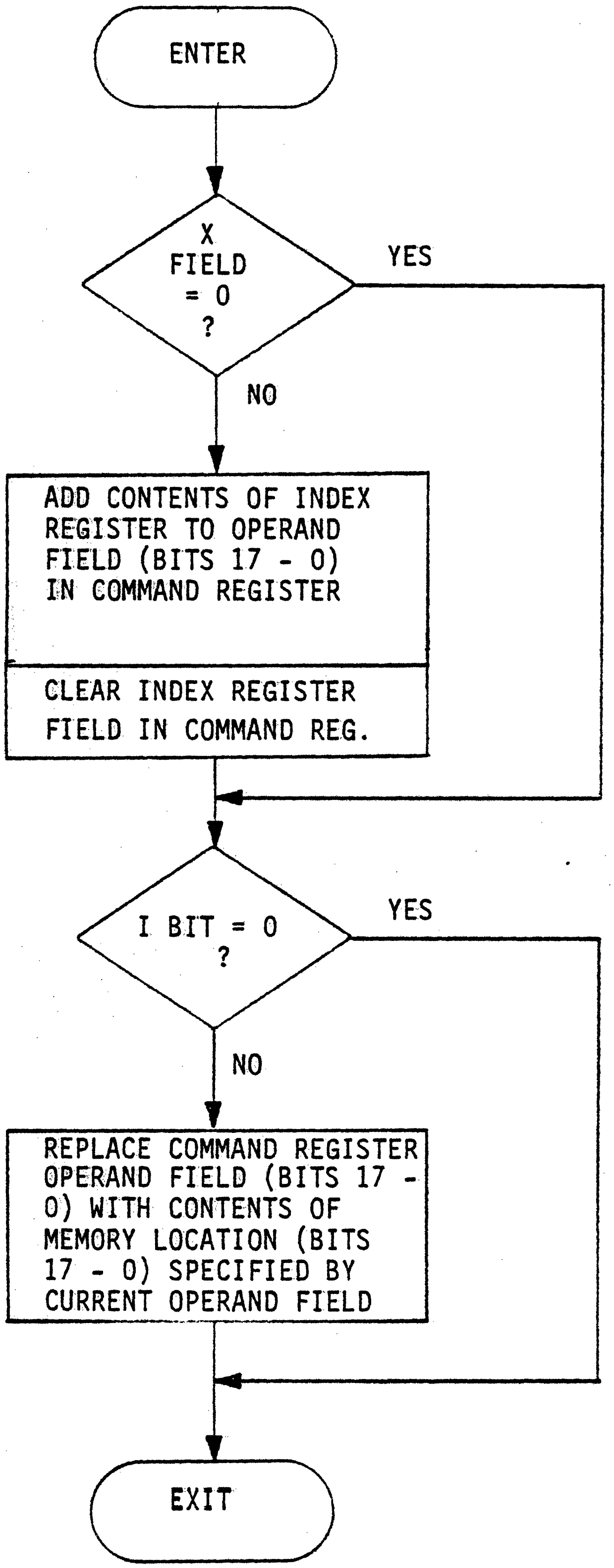
FST-2

MEMORY REFERENCE INSTRUCTION BIT CONFIGURATION

INSTRUCTION EXECUTION FLOWCHART


START
A
EXAMINE INSTRUCTION CURRENTLY STORED IN COMMAND REG.
OPERAND MODIFICATION REQUIRED ?
NO
YES
PERFORM OPERAND MODIFICATION
EXECUTE INSTRUCTION CURRENTLY HELD IN COMMAND REGISTER
WAS IT BRANCH INSTRUCTION ?
YES
NO
LOAD COMMAND REGISTER WITH INSTRUCTION LOCATED AT ADDRESS SPECIFIED BY PROGRAM COUNTER
ADD +1 TO PROGRAM COUNTER
A
LOAD COMMAND REGISTER WITH INSTRUCTION LOCATED AT ADDRESS OF CURRENT COMMAND REGISTER OPERAND
LOAD PROGRAM COUNTER WITH COMMAND REGISTER OPERAND ADDRESS +1
A

MEMORY REFERENCE INSTRUCTION
OPERAND MODIFICATION FLOWCHART

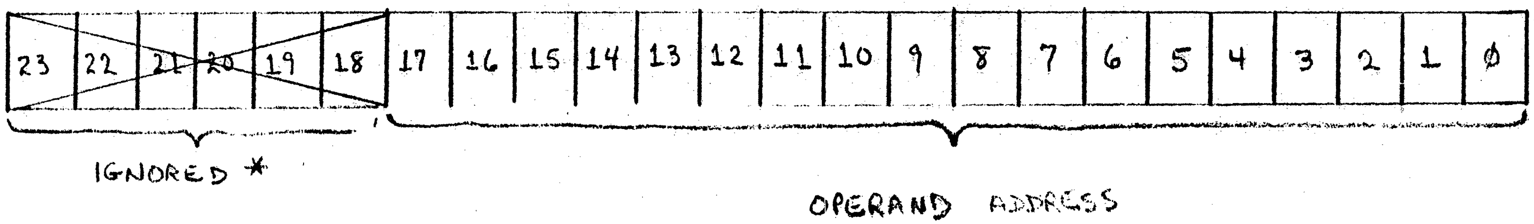

ENTER
X FIELD = 0 ?
YES
NO
ADD CONTENTS OF INDEX REGISTER TO OPERAND FIELD (BITS 17 - 0) IN COMMAND REGISTER
CLEAR INDEX REGISTER FIELD IN COMMAND REG.
I BIT = 0 ?
YES
NO
REPLACE COMMAND REGISTER OPERAND FIELD (BITS 17 - 0) WITH CONTENTS OF MEMORY LOCATION (BITS 17 - 0) SPECIFIED BY CURRENT OPERAND FIELD
EXIT

INDIRECT ADDRESSING

```
START        *
        ≀
        ≀
LOOP    LDE*    POINTER
        BSM     OUTPUT      CALL OUTPUT SUBROUTINE
        AOM     POINTER
        LDA     POINTER
        CAM     STOP
        BL      LOOP
        ≀
        BRU     START
        ORG     1000B
POINTER DATA    TABLE
STOP    DATA    TABLE+9
TABLE   DATA    10,20,30,40,50,60,70,80,90,100
        ≀
        END
```

INDEXING

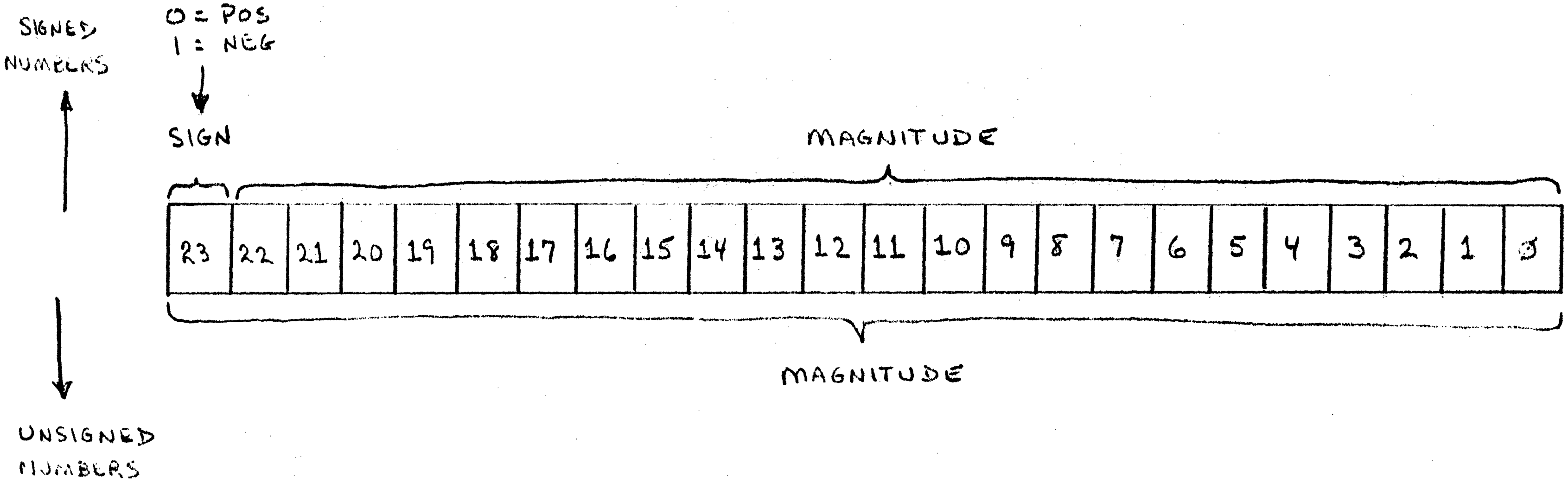
```
BEGIN   LDX     7,0
        LDX     6,10
         ⌇
LOOP    LDA     TABLE,7
        BSM     OUTPUT       CALL OUTPUT SUBROUTINE
        ATX     7,1
        BL      LOOP
         ⌇
        BRU     MAINPR+1
OUTPUT  PZE     0            START OF OUTPUT SUBROUTINE
         ⌇
        BRU*    OUTPUT       END OF OUTPUT SUBROUTINE
*
* OTHER SUBROUTINES LOCATED HERE
*
        ORG     1000B
TABLE   DATA    10,20,30,40,50,60,70,80,90,100
         ⌇
        END
```



(BRU HAS SPECIAL USE)

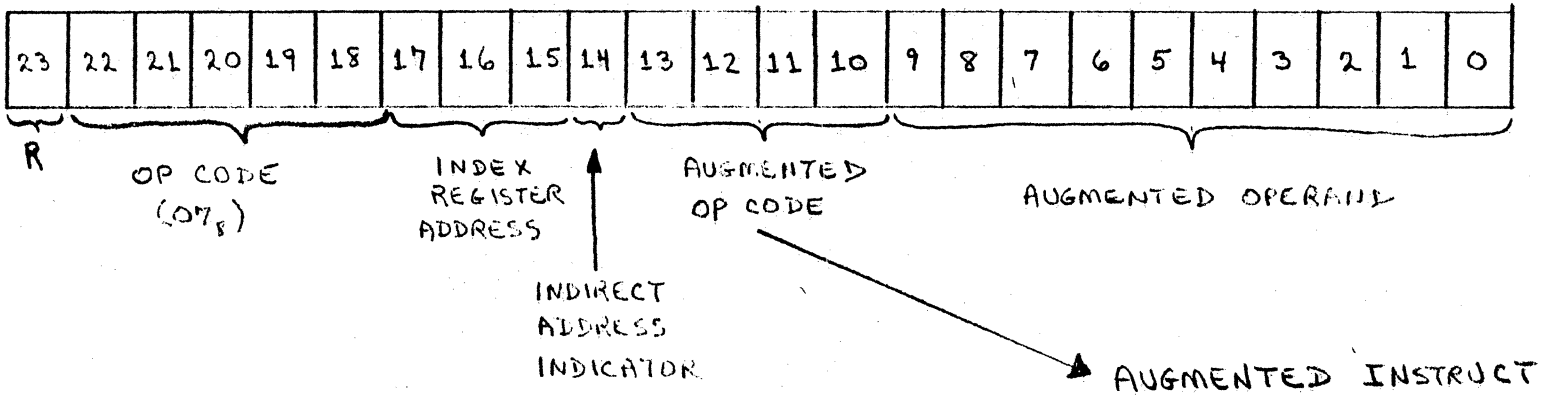
FST-2
ADDRESS WORD
BIT CONFIGURATION

	POS	NEG.
OCTAL =	00000000 → 37777777	77777777 → 40000000
DECIMAL =	0 → 8,388,607	-1 → -8,388,608



OCTAL = 00000000 → 77777777
DECIMAL = 0 → 16,777,215

FST-2
DATA WORD
BIT CONFIGURATION



AUGMENTED INSTRUCTIONS

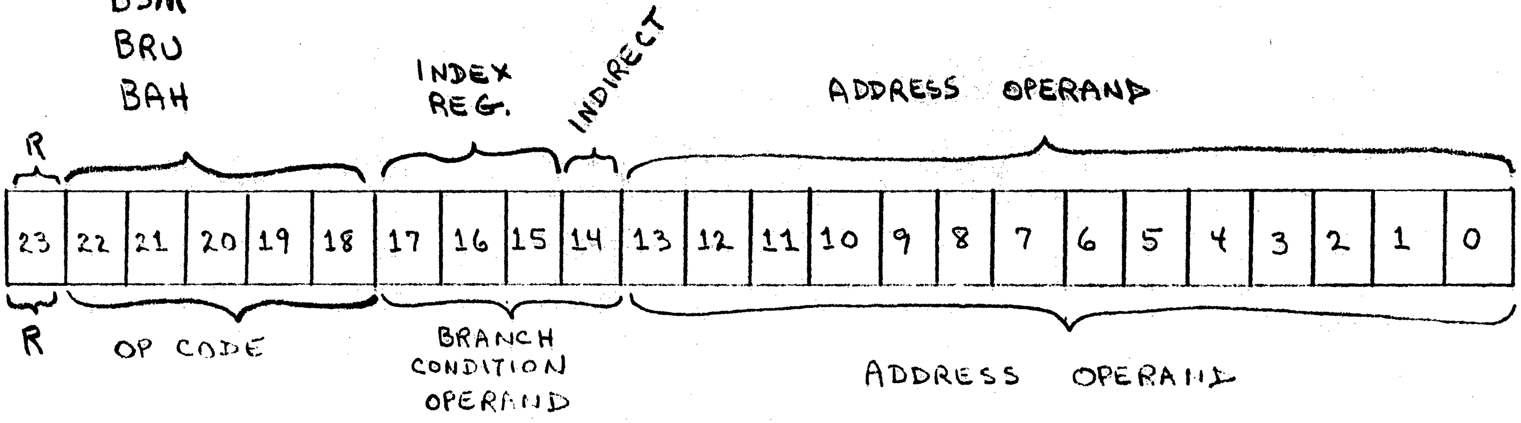
LXA
TCA
DTC
RSR
EXC
SST
RST
IEN
IDA

DSN
SR
LS
SA
SL
DSR
LDS
DSA
DSL

FST-2
AUGMENTED INSTRUCTION
BIT CONFIGURATION

UNCONDITIONAL

BSZ
BSM
BRU
BAH



CONDITIONAL

BAT
BOI
BOS

FST-2
BRANCH INSTRUCTION
BIT CONFIGURATION

FST-1 INSTRUCTION SUMMARY

DATA TRANSFER INSTRUCTIONS

LDA(*) M(,X)	-	Load A from memory
LDE(*) M(,X)	-	Load E from memory
EXC	-	Exchange contents of A and E Registers
DLD(*) M(,X)	-	Double load A & E from memory
LDX(*) X,M	-	Load X with value of memory address (content of address if indirect)
LXA X	-	Load X from A
LAX X	-	Load A from X
STA(*) M(,X)	-	Store A in memory
STE(*) M(,X)	-	Store E in memory
DST(*) M(,X)	-	Double store A & E in memory
STX(*) X,M	-	Store X in memory
RSR	-	Read Switch Register into A

ARITHMETIC INSTRUCTIONS

ADD(*) M(,X)	-	Add M to A
DADD(*) M(,X)	-	Double add M and M+1 to A and E
SUB(*) M(,X)	-	Subtract M from A
DSUB(*) M(,X)	-	Double subtract M and M+1 from A and E
MUL(*) M(,X)	-	Multiply M x A
DIV(*) M(,X)	-	Divide A and E/M
ATX(*) X,M	-	Add value of memory address to X (content of address if indirect)
AOM(*) M(,X)	-	Add one to M
SOM(*) M(,X)	-	Subtract one from M
TCA	-	Two complement A
DTC	-	Double twos complement A and E

SHIFT INSTRUCTIONS

SR J(,X)	-	Shift A Right (arithmetic)
DSR J(,X)	-	Double shift A and E Right (arithmetic)
LS J(,X)	-	Shift A Right (logical)
LDS J(,X)	-	Double Shift A and E Right (logical)
SL J(,X)	-	Shift A Left
DSL J(,X)	-	Double Shift A and E Left
SA J(,X)	-	Shift A around
DSA J(,X)	-	Double Shift A and E around
DSN J(,X)	-	Double Shift Normalize A and E

LOGICAL INSTRUCTIONS

AND(*) M(,X)	-	AND A with M
OR(*) M(,X)	-	Inclusive OR A with M
EOR(*) M(,X)	-	Exclusive OR A with M
RUM(*) M(,X)	-	Replace Under Mask (M masked by E)

STATE CONTROL INSTRUCTIONS

SST 0,1,2,3,4,5,6,7,8,9	-	Set State
IEN	-	Interrupt Enable
RST 0,1,2,3,4,5,6,7,8,9	-	Reset State
IDA	-	Interrupt Disable

TRANSFER OF CONTROL (BRANCH) INSTRUCTIONS

BOS K,M	-	Branch on State to M
BAT K,M	-	Branch on A Test to M
BOI K,M	-	Branch on Indicator to M
BSM(*) M(,X)	-	Branch and Store Return at M
BRU(*) M(,X)	-	Branch Unconditionally to M
BSZ(*) M(,X)	-	Branch and Store Return at Zero
BAH(*) M(,X)	-	Branch to M and Halt

MISCELLANEOUS INSTRUCTIONS

CAM(*) M(,X)	-	Compare A with M
NOP	-	No operation

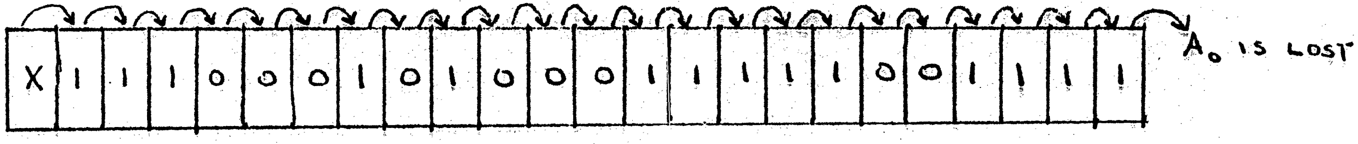
FST-2 INSTRUCTION SUMMARY

All of the above instructions except BSZ

Also includes the following new instructions:

CLA	-	Clear the Accumulator
LRA	-	Load Relocation Register from Accumulator
LAR	-	Load Accumulator from Relocation Register
STM1	-	Set FST-1 mode
STM2	-	Set FST-2 mode

SHIFT RIGHT



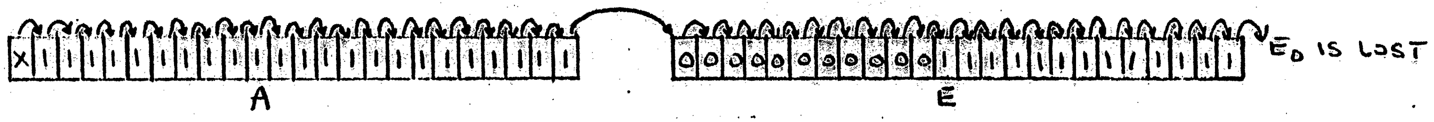
EXAMPLE 1: <u>SR 5</u> (ASSUME A_{23} IS A '0')

0	0	0	0	0	0	1	1	1	0	0	0	1	0	1	0	0	0	1	1	1	1	1	0

EXAMPLE 2: <u>SR 5</u> (ASSUME A_{23} IS A '1')

1	1	1	1	1	1	1	1	1	0	0	0	1	0	1	0	0	0	1	1	1	1	1	0

DOUBLE SHIFT RIGHT



EXAMPLE 1: DSR 12 (ASSUME A_{23} IS A '0')

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1

A

1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1

E

EXAMPLE 2: DSR 12 (ASSUME A_{23} IS A '1')

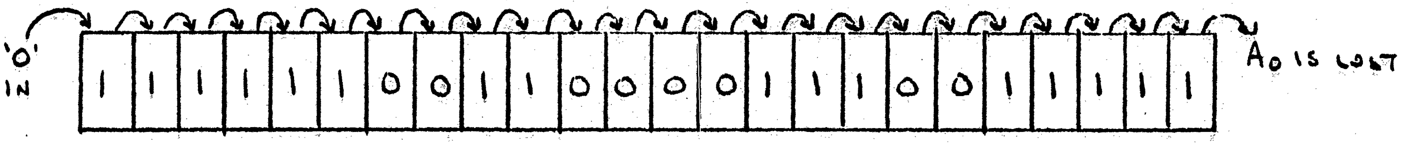
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

A

1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1

E

LOGICAL SHIFT (RIGHT)

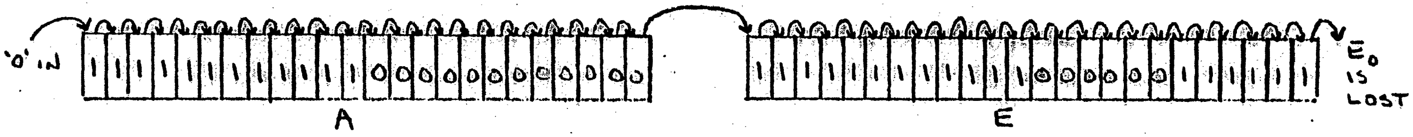


1	1	1	1	1	1	0	0	1	1	0	0	0	0	1	1	1	0	0	1	1	1	1	1

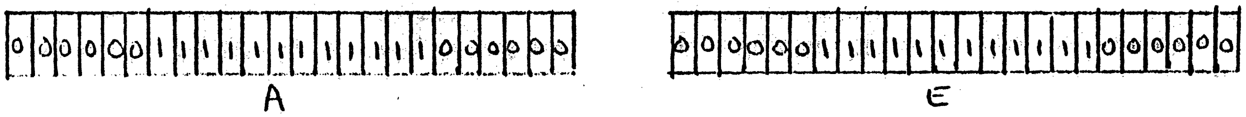
EXAMPLE: LS 4

0	0	0	0	1	1	1	1	1	1	0	0	1	1	0	0	0	0	1	1	1	0	0	1

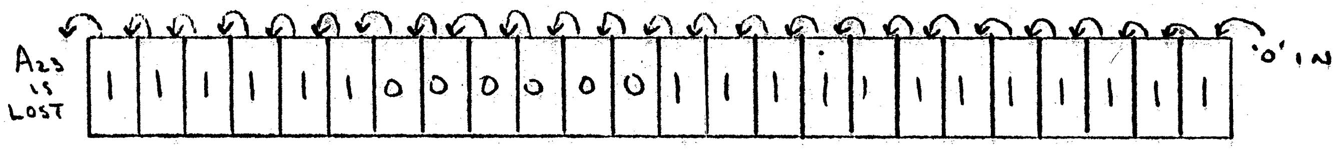
LOGICAL DOUBLE SHIFT (RIGHT)



EXAMPLE: LDS 6



SHIFT LEFT

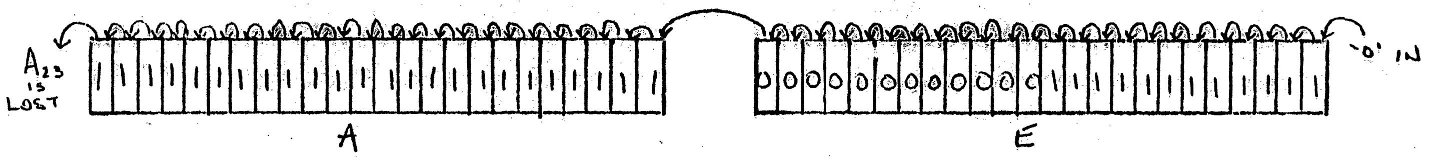


1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

EXAMPLE: SL 6

0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

DOUBLE SHIFT LEFT

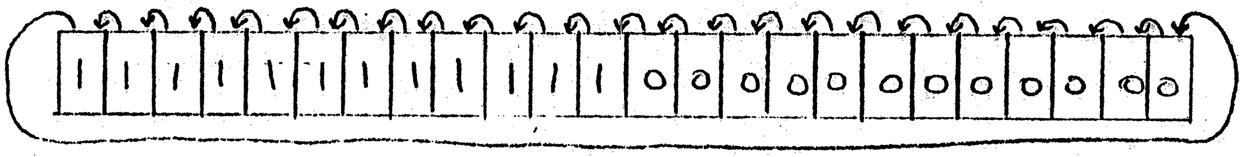


A	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
E	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

EXAMPLE: DSL 6

A	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
E	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

SHIFT AROUND

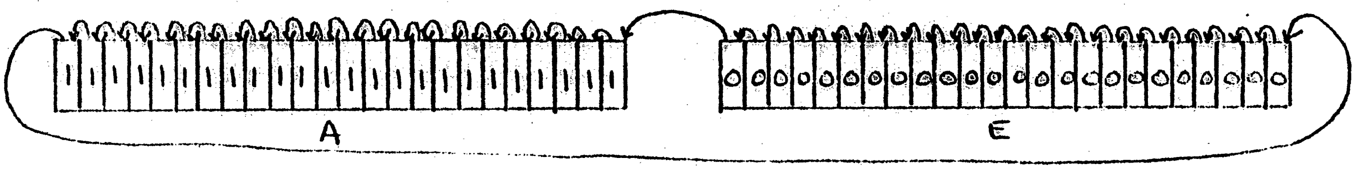


1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

EXAMPLE: SA 6

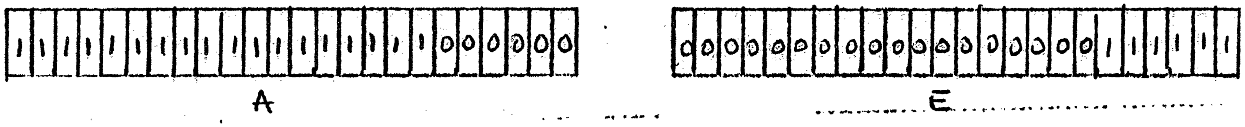
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

DOUBLE SHIFT AROUND (LEFT)



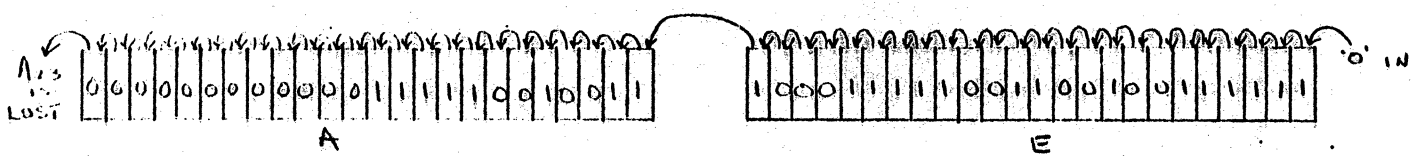
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

EXAMPLE: DSA 6



1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

DOUBLE SHIFT NORMALIZE



EXAMPLE 1 : DSN 10

0	0	1	1	1	1	1	0	0	1	0	0	1	1	1	0	0	0	1	1	1	1	1	0

A

0	1	1	0	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0

E

EXAMPLE 2: DSN 24

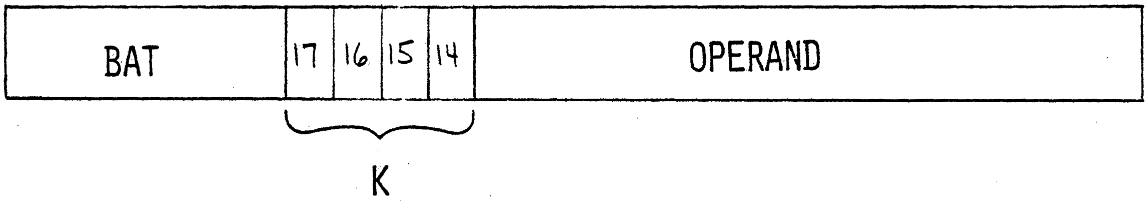
0	1	1	1	1	1	0	0	1	0	0	1	1	1	0	0	0	1	1	1	1	1	0	0

A

1	1	0	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

E

BRANCH ON ACCUMULATOR TEST



B17 = POSITIVE ACCUMULATOR (EXCLUDES ZERO CONDITION)
B16 = ZERO ACCUMULATOR
B15 = NEGATIVE ACCUMULATOR
B14 = ODD ACCUMULATOR

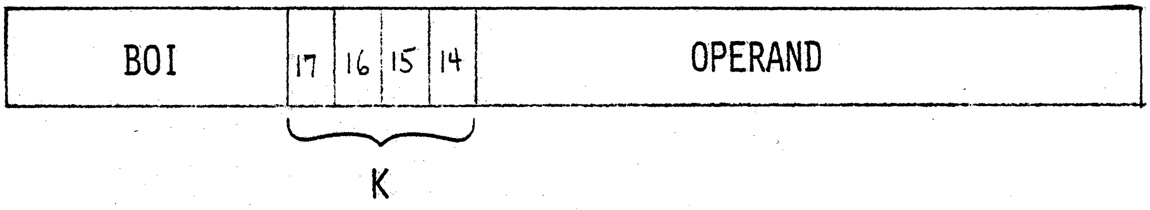
K	SPECIAL MNEMONIC	CONDITION
1	BO	ODD
2	BN	NEG
3		NEG/ODD
4	BZ	ZERO
5		ZERO/ODD
6	BNZ	ZERO/NEG
7		ZERO/NEG/ODD
10	BP	POS
11		POS/ODD
12	BNEZ	POS/NEG (NOT ZERO)
13		POS/NEG/ODD
14	BPZ	POS/ZERO
15		POS/ZERO/ODD
16		POS/ZERO/NEG
17		POS/ZERO/NEG/ODD

ASSEMBLER FORM

BAT K,M OR SPECIAL MNEMONIC

```
BAT  6,500B   BRANCH IF ACCUMULATOR ZERO OR NEG
BNZ  500B     BRANCH IF ACCUMULATOR ZERO OR NEG
```

BRANCH ON INDICATOR



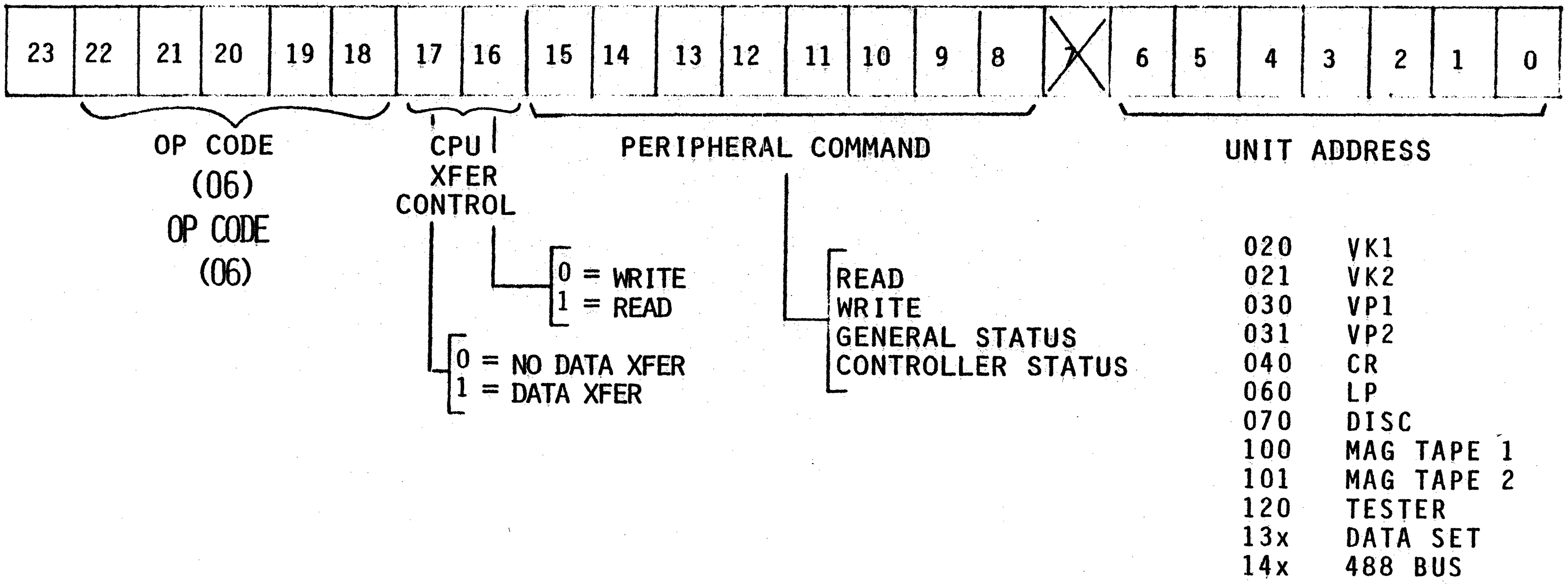
B17 = GT (GREATER THAN)
B16 = EQ (EQUAL)
B15 = LT (LESS THAN)
B14 = BE (BIT EQUAL)

K	SPECIAL MNEMONIC	CONDITION
1	BBC	BE
2	BL	LT
3		BE/LT
4	BE	EQ
5		EQ/BE
6	BLE	LT/EQ
7		EQ/LT/BE
10	BG	GT
11		GT/BE
12		GT/LT
13		GT/LT/BE
14	BGE	GT/EQ
15		GT/EQ/BE
16		GT/EQ/LT
17		GT/EQ/LT/BE

ASSEMBLER FORM

```
BOI  K,M        OR        SPECIAL MNEMONIC

BOI  6,500B   BRANCH IF COMPARE WAS LT OR EQ
BLE  500B     BRANCH IF COMPARE WAS LT OR EQ
```

PERIPHERAL I/O - OVERVIEW



020	VK1
021	VK2
030	VP1
031	VP2
040	CR
060	LP
070	DISC
100	MAG TAPE 1
101	MAG TAPE 2
120	TESTER
13x	DATA SET
14x	488 BUS

FST-2 SELECT PERIPHERAL UNIT INSTRUCTION WORD CONFIGURATION

ACCUMULATOR BUSS USES

DATA TRANSFERS

1. CPU selects a peripheral controller to perform a specific function by transmitting SPU instruction word via the 24-bit accumulator buss.

2. When the specified peripheral recognizes it's Unit Address, it then transmits it's general status back to the CPU via the four most significant bits of the accumulator buss.

3. a. If the peripheral is a non-DMA type and the general status was "not busy" then 24-bits of data are parallel transferred between the CPU Accumulator Register and the peripheral (the direction of the data transfer is determined by the individual SPU instruction).

   b. If the peripheral is a DMA type and the general status was not busy then the 14 least significant bits of the Accumulator Register are transferred to the peripheral's CPI (the 14 bits are interpreted as a Data Control Block address pointer - the DCB determines the amount of data and the direction of transfer via the memory buss).

```
                              INPUT/OUTPUT INSTRUCTIONS
                              -------------------------

        ---------------------------------------------------------------------------
        :        Ø 6         : A: R:   C (CONTROL FIELD)   :--:U (UNIT DEVICE CODE):
        ---------------------------------------------------------------------------
     BITS 23             18 17 16 15                       8  7  6               Ø
          A=1 MEANS TRANSFER BETWEEN ACCUMULATOR REGISTER AND ACC. BUS.
          R=1 MEANS DIRECTION OF TRANSFER IS FROM ACC.BUS TO ACC.REG.
          THEREFORE AR=1Ø IS "WRITE"  AND  AR=11 IS "READ",
          (EXCEPT IN DMA MODE).


                 INSTRUCTIONS FOR THE VKT (VIDEO KEYBOARD DISPLAY)

     DESCRIPTION                          OCTAL CODE    ASSEMBLY MNEMONIC

     KEYBOARD STATUS TEST.............. Ø6 ØØ ØØ 2Ø        STST 2ØB
     READ STATUS, KEYBOARD ............ Ø6 61 14 2Ø        RDS  2ØB
     READ KEYBOARD DATA ............... Ø6 6Ø 14 2Ø        RDTT 2ØB
     INTERRUPT PRIORITY ON ............ Ø6 Ø1 3Ø 2Ø        PON  2ØB
     INTERRUPT PRIORITY OFF ........... Ø6 Ø1 1Ø 2Ø        POFF 2ØB
     INTERRUPT PRIORITY COMPLETE ...... Ø6 ØØ 1Ø 2Ø        PCOMP 2ØB

     VKT SCREEN STATUS TEST ........... Ø6 ØØ ØØ 3Ø        STST  3ØB
     READ STATUS, VKT SCREEN .......... Ø6 61 14 3Ø        RDS   3ØB
     WRITE DATA TO VKT SCREEN ......... Ø6 42 14 3Ø        WRIT  3ØB
     INTERRRUPT PRIORITY ON ........... Ø6 Ø1 3Ø 3Ø        PON   3ØB
     INTERRUPT PRIORITY OFF ........... Ø6 Ø1 1Ø 3Ø        POFF  3ØB
     INTERRUPT PRIORITY COMPLETE ...... Ø6 ØØ 1Ø 3Ø        PCOMP 3ØB


                INSTRUCTIONS FOR LINE PRINTER

     LINE PRINTER STATUS TEST.......... Ø6 ØØ ØØ 6Ø        STST  6ØB
     WRITE TO LINE PRINTER ............ Ø6 42 14 5Ø        WRIT  6ØB
     INTERRUPT PRIORITY ON ............ Ø6 Ø1 3Ø 5Ø        PON   6ØB
     INTERRUPT PRIORITY OFF ........... Ø6 Ø1 1Ø 6Ø        POFF  6ØB
     INTERRUPT PRIORITY COMPLETE....... Ø6 ØØ 1Ø 5Ø        PCOMP 6ØB


                INSTRUCTIONS FOR CARD READER

     CARD READER STATUS TEST........... Ø6 ØØ ØØ 4Ø        STST  4ØB
     READ STATUS FROM C.READER......... Ø6 61 14 4Ø        RDS   4ØB
     ERROR TEST STATUS FROM C.READER.,. Ø6 Ø1 ØØ 4Ø        ETST  4ØB
     READ BINARY CARD.................. Ø6 4Ø 14 4Ø        RD    4ØB
     READ HOLLERITH CARD............... Ø6 4Ø 34 4Ø        ARD   4ØB
     INTERRUPT PRIORITY ON............. Ø6 Ø1 3Ø 4Ø        PON   4ØB
     INTERRUPT PRIORITY OFF............ Ø6 41 1Ø 4Ø        POFF  4ØB
     INTERRUPT PRIORITY COMPLETE....... Ø6 ØØ 1Ø 4Ø        PCOMP 4ØB


               INSTRUCTIONS FOR DISC MEMORY


     DISC STATUS TEST.................. Ø6 ØØ ØØ 7Ø        STST  7ØB
     READ STATUS FROM DISC............. Ø6 61 14 7Ø        RDS   7ØB
     ALTERNATE STATUS FROM DISC(TASA).  Ø6 61 34 7Ø        ARDS  7ØB
```

```
ERROR TEST STATUS FROM DISC...... 06 01 00 70        ETST  70B

READ DATA FROM DISC.............. 06 40 14 70        RD    70B
WRITE DATA TO DISC............... 06 42 14 70        WRIT  70B
READ DISC,INHIBIT DATA........... 06 40 34 70        ARD   70B

INTERRUPT PRIORITY ON............ 06 01 30 70        PON   70B
INTERRUPT PRIORITY OFF........... 06 01 10 70        POFF  70B
INTERRUPT PRIORITY COMPLETE...... 06 00 10 70        PCOMP 70B
```

INSTRUCTIONS FOR MAGNETIC TAPE UNIT

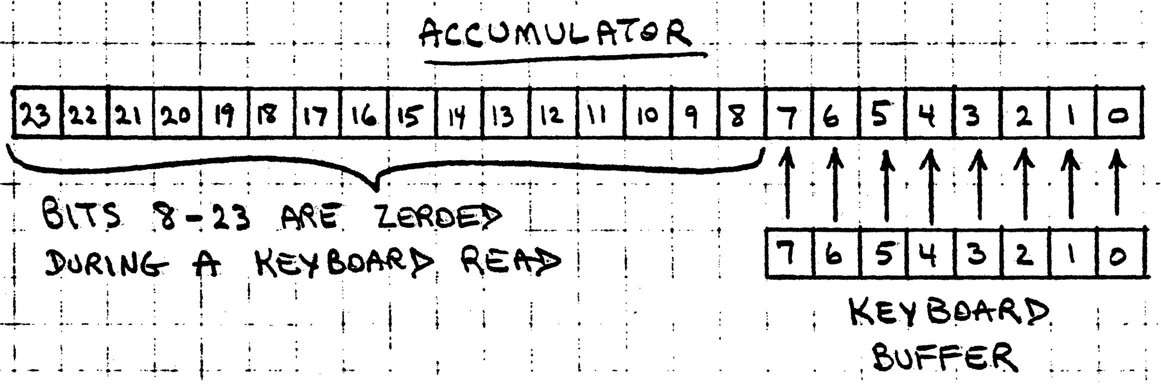
```
REWIND MAG. TAPE................. 06 00 05 00        REWIND
TAPE UNIT STATUS TEST............ 06 00 01 00        STST   100B
READ STATUS FROM TAPE UNIT....... 06 61 15 00        RDS    102B
ERROR TEST STATUS FROM TAPE UNIT. 06 01 01 00        ETST   100B
READ MAR FROM TAPE INTERFACE..... 06 61 17 00        REWC   100B

WRITE RECORD TO TAPE............. 06 42 15 00        WRIT   100B
WRITE LEADER, THEN RECORD........ 06 46 15 00        SKWR   100B
WRITE GAP AND TAPE MARK.......... 06 06 15 00        WRITM  100B
READ RECORD FROM TAPE............ 06 50 15 00        RDT    100B
READ RECORD,INHIBIT DATA......... 06 40 15 00        ARDT   100B
ADVANCE TAPE ONE RECORD.......... 06 00 15 00        RSKIPF 120B
REWIND TAPE ONE RECORD........... 06 01 15 00        RSKIPB 120B
ADVANCE TAPE TO TAPE MARK........ 06 04 15 00        FSKIPF 100B
REWIND TO TAPE MARK..............  06 05 15 00        FSKIPB

INTERRUPT PRIORITY ON............ 06 42 15 00        PON    100B
INTERRUPT PRIORITY OFF........... 06 46 15 00        POFF   100B
INTERRUPT PRIORITY COMPLETE...... 06 00 11 00        PCOMP  100B
```

DEFINITION OF PRIORITY SCHEME-LEVELS

```
PERIPHERAL UNIT    PERIPHERAL  INTERRUPT  INTERRUPT  MEMORY
                   ADDRESS.    ADDRESS.   PRIORITY.  PRIORITY

KEYBOARD              20          02      1:LOWEST    NONE
SCREEN                30          03      2           NONE
CARD READER           40          04      5           5
LINE PRINTER          60          06      3           NONE
DISC MEMORY           70          07      8           12:HIGHEST
MAGNETIC TAPE #1     100          10      7           9
MAGNETIC TAPE #2     101          11      6           8
TESTER               120       VARIABLE   9:HIGHEST   2:LOWEST
```

NON-DMA PERIPHERAL I/O

VKT/TTY KEYBOARD/TAPE READER I/O – SIMPLIFIED



GENERAL STATUS

GT	=	IDLE
EQ	=	IDLE WITH ERROR
LT	=	BUSY
BE	=	NOT AVAILABLE (OFF LINE)

INSTRUCTION REPERTOIRE SUMMARY

STST	2ØB	READ GENERAL STATUS
RDS	2ØB	READ CONTROLLER STATUS
RDTT	2ØB	DATA XFER FROM KEYBOARD BUFFER TO ACCUMULATOR
FEED	2ØB	XFER PAPER TAPE CHARACTER TO KEYBOARD BUFFER THEN ADVANCE TAPE ONE CHARACTER
PON	2ØB	PRIORITY INTERRUPTS ON
POFF	2ØB	PRIORITY INTERRUPTS OF
PCOMP	2ØB	PRIORITY INTERRUPT COMPLETE

INPUT EXAMPLES FOR VKT KEYBOARD

EXAMPLE #1

```
        RDTT   20B    READ KEYBOARD
        BL     *-1    BUSY?  BRANCH IF YES
        STA    BUF    NO - SAVE CHARACTER
```

EXAMPLE #2

```
        RDTT   20B    READ CHARACTER
        BOI    3,*-1  BUSY OR NOT AVAILABLE?
        STA    BUF    NO - SAVE CHARACTER
```

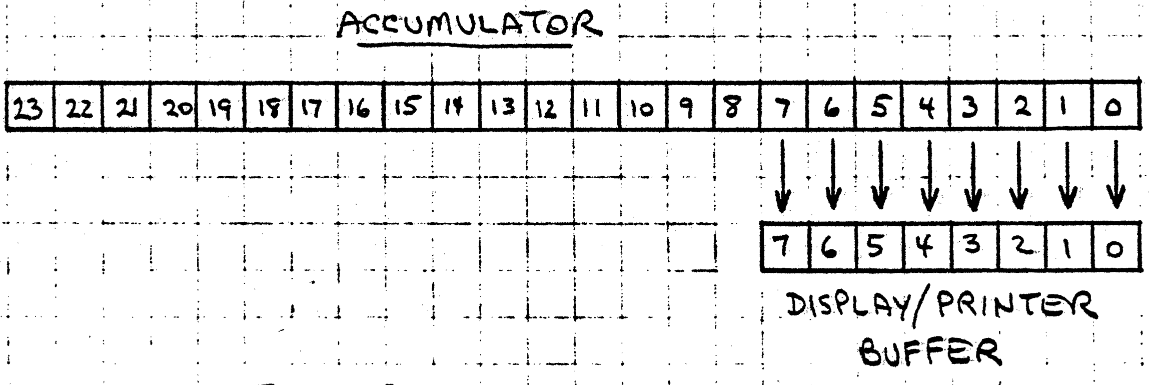
EXAMPLE #3

```
         ⌇
        BSM    INPUT  CALL INPUT SUBROUTINE
        STA    BUF    SAVE CHARACTER
         ⌇
INPUT   PZE    0      SUBROUTINE ENTRY
        RDTT   20B    READ CHARACTER
        BOI    3,*-1
        BRU*   INPUT
         ⌇
```

ROUTINE TO DEMONSTRATE CHARACTER PACKING (4 CHARACTER PER WORD) WITH NON-INTERRUPT TTY INPUT OPERATIONS

```
024Ø DATA 24ØB

     RDTT 2ØB                     Read first character
     BL *-1
     SUB 024Ø                     Convert ASCII to TRASCII
     SL 6
     STA TEMP
     RDTT 2ØB                     Read second character
     BL *-1
     SUB 024Ø
     OR TEMP
     SL 6
     STA TEMP
     RDTT 2ØB                     Read third character
     BL *-1
     SUB 024Ø
     OR TEMP
     SL 6
     STA TEMP
     RDTT 2ØB                     Read fourth character
     BL *-1
     SUB 024ØB
     OR TEMP
     STA BLOCK,7                  Store 4 character word in memory
```

VKT DISPLAY/TTY PRINTER I/O - SIMPLIFIED



FLAG BIT

1 (ON) = BUFFER EMPTY (IDLE)
ø (OFF) = BUFFER NOT EMPTY (BUSY)

GENERAL STATUS

GT = IDLE
EQ = IDLE WITH ERROR
LT = BUSY
BE = NOT AVAILABLE (OFF LINE)

INSTRUCTION REPERTOIRE SUMMARY

STST	3øB	READ GENERAL STATUS
RDS	3øB	READ CONTROLLER STATUS
WRIT	3øB	DATA XFER FROM ACCUMULATOR TO DISPLAY/PRINTER BUFFER
PON	3øB	PRIORITY INTERRUPTS ON
POFF	3øB	PRIORITY INTERRUPTS OFF
PCOMP	3øB	PRIORITY INTERRUPT COMPLETE

OUTPUT EXAMPLES FOR VKT DISPLAY

EXAMPLE #1

```
        LDA    CHARACTER
        WRIT   30B              OUTPUT CHARACTER
        BL     *-1              BUSY?  BRANCH IF YES
        LDA    NEXT             NO
```

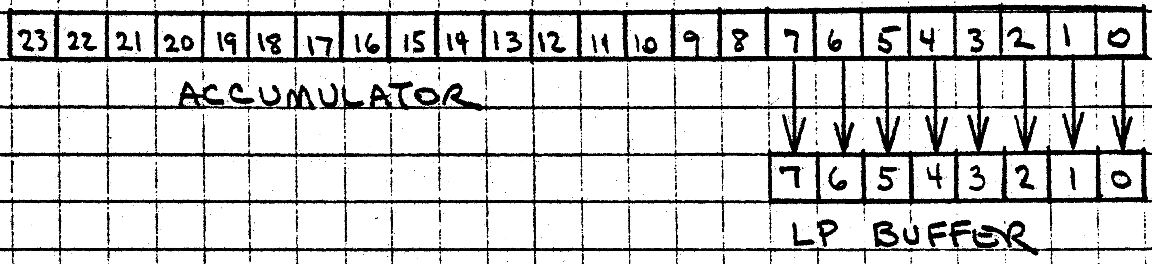
EXAMPLE #2

```
        LDA    CHARACTER
        WRIT   30B              OUTPUT CHARACTER
        BOI    3,*-1            BUSY OR NOT AVAILABLE?
        LDA    NEXT             NO
```

EXAMPLE #3

```
         }
        LDA    CHARACTER
        BSM    OUTPUT           CALL OUTPUT SUBROUTINE
         {
OUTPUT  PZE    0                SUBROUTINE ENTRY
        WRIT   30B
        BOI    3,*-1
        BRU*   OUTPUT
         }
```

LINE PRINTER I/O - SIMPLIFIED



GENERAL STATUS

GT = IDLE (LP ON & NOT BUSY)
EQ = (NOT USED)
LT = BUSY
a. LP ON AND BUSY DUMPING BUFFER
b. LP OFF (NOT READY)
BE = NOT AVAILABLE
a. LP ON & OUT OF PAPER
b. LP POWER OFF

INSTRUCTION REPERTOIRE SUMMARY

STST	6ØB	READ GENERAL STATUS
WRIT	6ØB	WRITE TO LINE PRINTER
PON	6ØB	LP PRIORITY INTERRUPT ON
POFF	60 B	LP PRIORITY INTERRUPT OFF
PCOMP	6ØB	LP PRIORITY INTERRUPT COMPLETE

LINE PRINTER I/O EXAMPLE

```
              {
              LDX  6,6             INITIALIZE INDEX REGISTERS
              LDX  7,Ø
*
              LDA  TOF
              WRIT 6ØB             WRITE TOP-OF-FORM
              BL   *-1
*
DUMP          LDA  BUF,7           LOAD WORD FROM BUFFER
              WRIT 6ØB             WRITE THE WORD
              BL   *-1
              ATX  7,1
              BL   DUMP            BUFFER EMPTY? YES-GO TO DUMP
              LDA  CR              NO - LOAD CARRIAGE RETURN
              WRIT 6ØB             WRITE CR
              BL   *-1
              LDA  LF              LOAD LINE FEED
              WRIT 6ØB             WRITE LF
              BL   *-1
              BRU  LUNCH           GO TO LUNCH
*
TOF           DATA 214B
CR            DATA 215B
LF            DATA 212B
BUF           DATA 302B            B
              DATA 325B            U
              DATA 306B            F
              DATA 306B            F
              DATA 305B            E
              DATA 322B            R
*
              {
```

DMA PERIPHERAL I/O

COMPARISON OF DATA TRANSFER METHODS
DMA vs NON-DMA

NON-DMA TRANSFER OF DATA BLOCK

```
LAST     EQU     STOP-START
         LDX     1,0
         LDX     0,LAST
XFER     LDA     START,1
         WRIT    xxB
         BL      *-1
         ATX     1,1
         BE      EXIT
         BRU     XFER
         BRU     LUNCH
*
START    DATA    1
          .
          .
          .
STOP     DATA    100
```

DMA TRANSFER OF DATA BLOCK

```
LAST     EQU     STOP-START
         LDA     DCB-ADDR-POINTER-LOCATION
         WRIT    xxB
         BL      *-1
         STST    xxB
         BL      *-1
         BRU     COFFEE
DCB      DATA    DCB-ADDR-POINTER
         DATA    BLOCK-SIZE
         DATA    MEMORY-ADDR-OF-FIRST-WORD
```

I/O TO DMA TYPE PERIPHERALS - SIMPLIFIED

```
        LDA     DCB-ADDR-POINTER-LOCATION
        WRIT    xxxB
        BL      *-1
         .
         .
         .
        LDA     DCB-ADDR-POINTER-LOCATION
        RD      xxxB
        BL      *-1
         .
         .
         .
        BRU*    BGENT
*
CRDCB   DATA    DCB-ADDR-POINTER
        DATA    WORD-COUNT
        DATA    MEMORY-START-ADDR
*
DSKDCB  DATA    DCB-ADDR-POINTER
        DATA    FILE-LENGTH
        DATA    MEMORY-START-ADDR
        DATA    TRACK-SECTOR-ADDR
*
MTDCB   DATA    DCB-ADDR-POINTER
        DATA    WORD-COUNT
        DATA    MEMORY-START-ADDR
*
        END
```

CARD READER I/O EXAMPLES

```
CARDS UNTIL // TERMINATES.
*
BEGIN       LDX     3,Ø
            LDX     2,37777B
            LDA     BLOCK1
            STA     CONSTN.
DCBADR      STX     3,CARDS
            ADD     CARDS
COPEAD      STA     STAD1
            RDS     40B
            LDA     POINT
            ARD     40B
            BOI     3,*-2
            STST    40B
            BOI     3,*-1
            BOI     7,*-6
            BRU     SLASH1
POINT       DATA    LOFAD    ADDRESS OF LENGTH OF FILE
LOFAD       DATA    20       20 24-BIT WORDS PER CARD
STAD1       DATA    Ø        START ADDRESS IN CORE FOR EACH CARD
SLASH1      LDA     BLOCK1,3
            CAM     SLASH
            BE      LAST
            ATX     3,20
            LDA     CONSTN
            BL      DCBADR
LAST        LDX     Ø,4
            LDX     1,Ø
            LAX     3
            LXA     4
            LDX     5,Ø
            LDX     6,64
            LDX     7,Ø
LOADM       LDE     BLOCK1,5
            LDA     CLEAR
SHIFT       DSL     6
LUKUP       CAM     ALPHA,7
            BE      ASCII
            ATX     7,1
            BL      LUKUP
            BRU     100B
ASCII       LDA     TABL,7
            LDX     7,Ø
            WRIT    30B
            BOI     3,*-1
            ATX     1,1
            LDA     CLEAR
            BL      SHIFT
            LDX     1,Ø
            ATX     5,1
            BL      LOADM
            LDX     5,Ø
            LDA     CONSTN
            STA     BLOCK1
FIN         BAM     BEGIN
CLEAR       DATA  Ø
SLASH        DATA    21212Ø2ØB DATA OF // CARD
ALPHA      DATA    Ø,1,2,3,4,5,6,7,10B,11B,12B,13B,14B,15B,16B
           DATA    17B,20B,21B,22B,23B,24B,25B,26B,27B,30B,31B
           DATA    32B,33B,34B,35B,36B,37B,40B,41B,42B,43B,44B,45B
           DATA    46B,47B,50B,51B,52B,53B,54B,55B,56B,57B,60B,61B
```

```
                DATA    62B,63B,64B,65B,66B,67B,70B,71B,72B,73B,74B,75B
                DATA    76B,77B
       TABL     DATA    241B,261B,262B,263B,264B,265B,266B,267B,270B,271B
                DATA    260B,243B,300B,247B,275B,242B,240B,257B,323B,324B
                DATA    325B,326B,327B,330B,331B,332B,272B,273B,245B,337B
                DATA    335B,277B,255B,312B,313B,314B,315B,316B,317B,320B
                DATA    321B,322B,276B,244B,252B,251B,273B,334B,246B,301B
                DATA    302B,303B,304B,305B,306B,307B,310B,311B,274B,256B
                DATA    333B,250B,253B,336B
       CARDS     DATA    0
       CONSTN    DATA    0
       BLOCK1    BSS    2000
       LWORD     EQU    *
                 END
```

DISC I/O EXAMPLES

```
*
*************************************************
* THIS PROGRAM ALLOWS THE USER TO TYPE  ON THE TTK AND
*  STORE IN CORE MEMORY STARTING AT LOCATION 'BLOK1'*
*       WHEN SPECIAL CHARACTER UP-ARROW IS TYPED,    *
*  THE DATA IS TRANSFERED TO THE DISC, TRACK 199     *
*  SEGMENT 00. IMMEDIATELY THE SAME DATA IS READ     *
*  BACK FROM THE DISC TO CORE MEMORY, STARTING AT    *
*  LOCATION "BLOK2" AND IT IS FINALLY TRANSFERED     *
*  FROM CORE TO THE VKT SCREEN EXACTLY AS IT WAS     *
*  ORIGINALLY ENTERED.                               *
*************************************************
*
*
SPEC        DATA   336B
BEGIN       LDX    2,0
            LDX    3,0
TTK         RDTT   20B
            BOI    3,TTK
            CAM    SPEC
            BE     WR70
            STA    BLOK1,2
            WRIT   30B
            BOI    3,*-1
            ATX    2,1
            STX    2,LOF1
            STX    2,LOF2
            BRU    TTK
WR70        RDS    70B
            LDA    PNT1
            WRIT   70B
            BOI    3,*-2
            STST   70B
            BOI    3,*-1
            BOI    7,*-6
            BRU    RD70
PNT1        DATA   PNT1+1
LOF1        DATA   0
STAD1       DATA   BLOK1
TASA1       DATA   314400B
RD70        RDS    70B
            LDA    PNT2
            RD     70B
            BOI    3,*-2
            STST   70B
            BOI    3,*-1
            BOI    7,*-6
            BRU    TTP
PNT2        DATA   PNT2+1
LOF2        DATA    0
STAD2       DATA   BLOK2
TASA2       DATA   314400B
TTP         LDA    BLOK2,3
            WRIT   30B
            BOI    3,*-1
            ATX    3,1
            BNE    TTP
            BRU    BEGIN
BLOK1       BSS    199B
BLOK2       BSS    199B
LWORD       EQU    *
            END
```

TTK -> memory -> DISC -> memory -> TTP

4-8

MAG TAPE I/O EXAMPLES

```
****************************************************
*                                                  *
*              TRAINING PROGRAM NUMBER FIVE        *
*                TO WRITE AND READ MAG-TAPE        *
*   PROGRAM NAME: WSTAPE                           *
*   JOB           : TRAIN                          *
*                                                  *
*   THIS PROGRAM ALLOWS TO TYPE ON THE TTK AND     *
*   STORE IN CONSECUTIVE MEMORY LOCATIONS, STARTING *
*   AT 'BLOK1'. GHEN SPECIAL CHARACTER 'UP-ARROW'  *
*   IS TYPED IN THE DATA IS TRANSFERED TO THE TAPE *
*   UNIT. THEN THE PROGRAM REWINDS THE TAPE, AND   *
*   READS THE DATA TO MEMORY LOCATIONS 'BLOK2' AND *
*   FINALLY THE INFORMATION IS DISPLAYED ON THE    *
*   SCREEN EXACTLY AS IT WAS ENTERED.              *
****************************************************
*
*
SPEC       DATA   336B
BEGIN      REWIND
           STST   100B
           BOI    3,*-2
           LDX    2,0
           LDX    3,0
TTK        RDTT   20B
           BOI    3,TTK
           CAM    SPEC
           BE     WRTAP
           STA    BLOK1,2
           WRIT   30B
           BOI    3,*-1
           ATX    2,1
           STX    2,LOF1
           STX    2,LOF2
           BRU    TTK
WRTAP      RDS    100B
           LDA    PNT1
           WRIT   100B
           BOI    3,*-2
           STST   100B
           BOI    3,*-1
           BOI    7,*-6
           REWIND
           STST   100B
           BOI    3,*-2
           BRU    RDTAP
PNT1       DATA   PNT1+1
LOF1       DATA   0
STAD1      DATA   BLOK1
RDTAP      RDS    100B
           LDA    PNT2
           RDT    100B
           BOI    3,*-2
           STST   100B
           BOI    3,*-1
           BOI    7,*-6
           BRU    TTP
PNT2       DATA   PNT2+1
LOF2       DATA    0
STAD2      DATA   BLOK2
TTP        LDA    BLOK2,3
           WRIT   30B
```

```
                BOI    3,*-1
                ATX    3,1
                BNE    TTP
                BRU    BEGIN
       BLOK1    BSS    1998
       BLOK2    BSS    1998
       LWORD    EQU    *
                END
```

PROGRAM DEBUGGING

USER INGENUITY WILL PROBABLY RESULT IN SEVERAL DIFFERENT APPROACHES TO DEBUGGING ASSEMBLY LANGUAGE PROGRAMS. ONE OF THE MOST EFFICIENT MEANS, HOWEVER, IS PROBABLY THROUGH THE USE OF THE "DEBUG" UTILITY ROUTINE.

"DEBUG" IS A BACKGROUND OVERLAY; HOWEVER, IT CAN ACCESS USER WRITTEN ASSEMBLY LANGUAGE PROGRAMS EXECUTING IN EITHER BACKGROUND OR FOREGROUND. WITH IT, THE USER IS ABLE TO CONTROL PROGRAM EXECUTION, EXAMINE AND ALTER MEMORY CONTENTS, AND EXAMINE AND MODIFY CPU AND TESTER REGISTERS.

BEFORE DEBUGGING AN ASSEMBLY LANGUAGE PROGRAM, "DEBUG" MAY BE LOADED USING THE MASTR "LOAD" COMMAND.

THE MASTR COMMAND TO ENVOKE "DEBUG" USES THE FOLLOWING GENERAL FORM:

```
DEBUG ('FILENAME') (INPUT) (OUTPUT)
```

WHERE:

FILENAME	NAME OF FILE TO BE DEBUGGED
INPUT	ANY STANDARD INPUT DEVICE VK1 IS DEFAULT
OUTPUT	ANY STANDARD OUTPUT DEVICE VP1 IS DEFAULT

DEBUG DIRECTIVES

DIRECTIVE	FUNCTION
(N1) (,N2)L	LIST CONTENTS OF ADDRESS N1 THRU N2
'FILENAME'(,N)L	LIST N OR ALL WORDS OF FILE NAME
(N1)(,N2)A	HALT AT ADDRESS N1 AFTER EXECUTING N TIMES
C	CONTINUE AFTER ADDRESS HALT
X	DISPLAY ALL INDEX REGISTERS
AE	DISPLAY A AND E REGISTERS
AC	DISPLAY CONTENTS OF CURRENT ADDRESS +1
DC	DISPLAY CONTENTS OF CURRENT ADDRESS -1
NG	GO TO ADDRESS N
NI	INSERT DATA AT ADDRESS N
NMA	MODIFY A REGISTER WITH VALUE N
NME	MODIFY E REGISTER WITH VALUE N
N1,N2MX	MODIFY INDEX REGISTER N1 WITH VALUE N2
N1,N2W (CT)	WRITE TESTER REGISTER N1 WITH VALUE N2
NR(CT)	READ TESTER REGISTER N
D	EXIT FROM DEBUG
SSW	DISPLAY STATE SWITCHES
(N)REL	SET RELATIVE ADDRESS
SET	SET LINE PRINTER AS OUTPUT DEVICE
RESET	RESET OUTPUT DEVICE
IND	DISPLAY INDICATORS
N1±(N2) CAL	CALCULATE N1±N2 AND DISPLAY
MON	GO TO MONITOR TO ENTER A COMMAND
CARRIAGE RETURN	SINGLE STEP (ADDRESS HALT AFTER EXECUTING NEXT INSTRUCTION)

AN OVERVIEW OF
I/O PROGRAMMING USING INTERRUPTS

PRIORITY INTERRUPTS - OVERVIEW

BEFORE THE INTERRUPT SYSTEM CAN BE USED, IT MUST BE PROPERLY "ARMED" BY ENABLING BOTH THE CPU INTERRUPT SYSTEM WITH AN "IEN" INSTRUCTION AND THE DESIRED PERIPHERAL(S) WITH A "PON xxB" INSTRUCTION.

ONCE THE INTERRUPT SYSTEM IS PROPERLY ARMED, AN " INTERRUPT" BY A PERIPHERAL WILL BE PROCESSED AT THE END OF THE CURRENT INSTRUCTION CYCLE. IF THE CURRENT INSTRUCTION IS AN SPU INSTRUCTION, HOWEVER, THE INTERRUPT PROCESSING WILL BE DELAYED UNTIL THE END OF THE NEXT NON-SPU INSTRUCTION CYCLE.

THE FOLLOWING TABLE MAY BE USEFUL IN UNDERSTANDING THE DISCUSSION OF THE INTERRUPT PROCESS, WHICH FOLLOWS:

	INTERRUPT	
PERIPHERAL	PRIORITY	ADDRESS
DATA SET (COMM LINK)	11	30 - 37
TESTER	9	12 - 21
DISC	8	07
MAG TAPE	7	10
488 BUS	6	40 - 43
CARD READER	5	04
LINE PRINTER	3	06
VKT DISPLAY	2	03
VKT KEYBOARD	1	02

THE INTERRUPT PROCESS, FROM A PROGRAMMER'S VIEWPOINT, IS AS FOLLOWS:

1. THE INTERRUPTING PERIPHERAL PRESENTS IT'S PRIORITY ON THE ACCUMULATOR BUS, USING THE 16 LEAST SIGNIFICANT BITS, WHICH IS THEN READ BY THE CPU TO RESOLVE INTERRUPT PRIORITY IN THE EVENT THERE WERE TWO OR MORE SIMULTANEOUS INTERRUPTS.

2. AFTER "PRIORITY" IS RESOLVED, THE INTERRUPTING PERIPHERAL SENDS IT'S INTERRUPT ADDRESS TO THE CPU VIA THE 6 LSBs OF THE ACCUMULATOR BUS AND IS PLACED INTO THE INTERRUPT (R) REGISTER.

3. THE INTERRUPT FLIP/FLOP IS AUTOMATICALLY RESET AND A SIGNAL IS SENT TO ALL LOWER PRIORITY PERIPHERALS WHICH INHIBITS ANY INTERRUPTS FROM THEM UNTIL THE IN-PROCESS INTERRUPT IS COMPLETE, EVEN IF THE PROGRAMMER REENABLES THE INTERRUPT F/F IN THE INTERRUPT SERVICE ROUTINE.

4. THE CPU AUTOMATICALLY LOADS THE COMMAND REGISTER WITH A BSM* (INDIRECT) IN THE OPERATION FIELD AND THE OPERAND FIELD WITH THE CONTENT OF THE INTERRUPT REGISTER.

5. UPON EXECUTION OF THE BSM* INSTRUCTION, THE OPERAND ADDRESS CONTAINS THE ENTRY POINT ADDRESS OF THE INTERRUPT SERVICE ROUTINE WRITTEN BY THE PROGRAMMER. THE "STATUS" INDICATORS AND RETURN ADDRESS (AS DEFINED BY THE PROGRAM COUNTER) ARE STORED AT THE ENTRY POINT, THEN PROGRAM CONTROLLED EXECUTION RESUMES AT THE ENTRY POINT ADDRESS +1.

6. IF YOU WISH TO PROCESS INTERRUPTS FROM HIGHER PRIORITY PERIPHERALS DURING EXECUTION OF THIS INTERRUPT SERVICE ROUTINE, THE INTERRUPT F/F MAY BE TURNED ON AGAIN AT THIS POINT WITH AN "IEN" INSTRUCTION.

7. BEFORE TERMINATING THE INTERRUPT ROUTINE, EXECUTE A "PCOMP xxB" INSTRUCTION TO INFORM LOWER PRIORITY PERIPHERALS THAT THE INTERRUPT SERVICE ROUTINE IS COMPLETE.

8. TO TERMINATE THE INTERRUPT SERVICE ROUTINE, EXECUTE A BRU* (INDIRECT) TO THE INTERRUPT SERVICE ROUTINE ENTRY POINT ADDRESS. THIS WILL RESTORE THE "STATUS" INDICATORS TO THEIR PREVIOUS CONDITION BEFORE THE INTERRUPT, AND WILL RETURN PROGRAM CONTROL TO THE MAIN PROGRAM AT THE PREVIOUSLY INTERRUPTED INSTRUCTION +1. IF FURTHER INTERRUPTS ARE TO BE PROCESSED, THE INTERRUPT F/F MUST BE TURNED ON AGAIN IF NOT ALREADY DONE SO IN THE JUST COMPLETED INTERRUPT SERVICE ROUTINE.

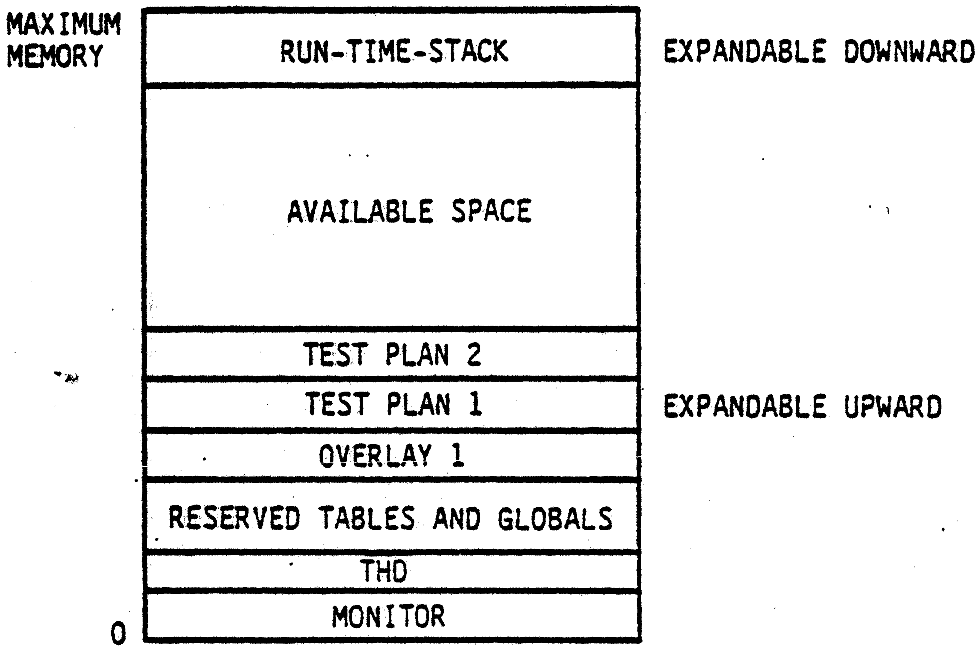
NOTE: THE PRECEEDING INFORMATION HAS BEEN PRESENTED SO THAT THE STUDENT WILL HAVE A BASIC UNDERSTANDING OF "INTERRUPT I/O PROCESSING". HOWEVER, THE USER PROGRAMMER SHOULD NOT USE INTERRUPT PROGRAMMING AS IT WILL CONFLICT WITH THE NORMAL SYSTEM INTERRUPTS AS WELL AS DEFEAT THE SYSTEM PROCEDURE OF SCHEDULING ALL I/O REQUESTS, THEREBY DEFEATING THE INTENT OF THE MULTI-USER "MASTR" OPERATING SYSTEM.

INTERRUPT PROGRAMMING EXAMPLE

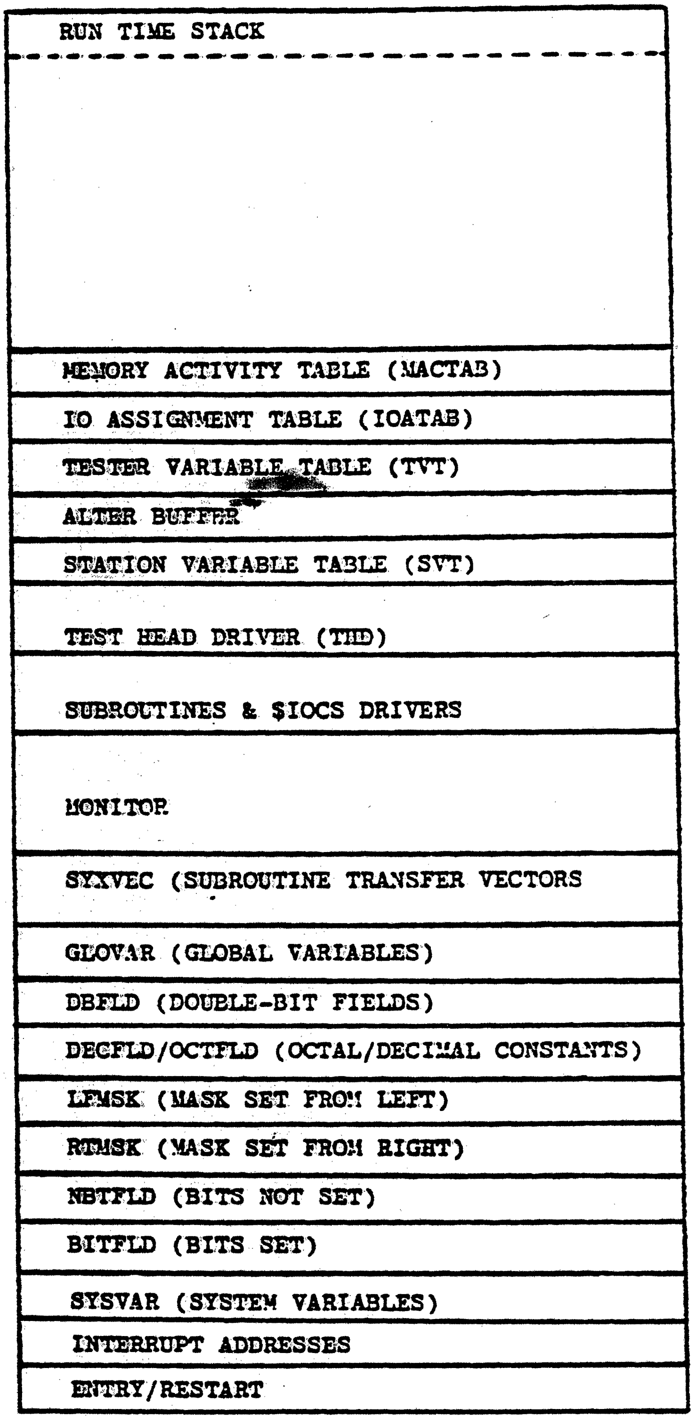
```
               .
               .
               .
               IEN     0     ENABLE CPU PRIORITY INTERRUPT SYSTEM
               PON    60B    ENABLE LINE PRINTER INTERRUPTS
               PON    xxB    ENABLE OTHER PERIPHERAL INTERRUPTS
               .
               .
               .
               BRU*    ENTRY
*
LPINT          PROC    6     ENTRY POINT TO LP INTERRUPT ROUTINE
               STA     xx    SAVE ACCUMULATOR CONTENTS
               STE     xx    SAVE EXTENSION REG CONTENTS
               STX     xx,xx SAVE INDEX REGISTER CONTENTS
               IEN     0     REENABLE INTERRUPT SYSTEM SO HIGHER
*                            PRIORITY INTERRUPTS CAN BE PROCESSED
*                            DURING THIS INTERRUPT ROUTINE -
*                            LOWER PRIORITY INTERRUPTS ARE NOT
*                            PROCESSED
               .
               .
               .
               LDA     xx    RESTORE ACCUMULATOR CONTENTS
               LDE     xx    RESTORE EXTENSION REG CONTENTS
               LDX*    xx,xx RESTORE INDEX REGISTER CONTENTS
               PCOMP   60B   NOTIFY LOWER PRIORITY PERIPHERALS
*                            THAT LP INTERRUPT ROUTINE IS COMPLETE
               BRU*    LPINT RETURN TO MAIN PROGRAM
*
               END     *     END OF USER PROGRAM
```

GENERAL REVIEW
OF
WEEK 1 COURSE MATERIAL

MASTR OPERATING SYSTEM MEMORY ORAGANIZATION



MASTR GENERAL MEMORY ORGANIZATION



RUN TIME STACK

MEMORY ACTIVITY TABLE (MACTAB)

IO ASSIGNMENT TABLE (IOATAB)

TESTER VARIABLE TABLE (TVT)

ALTER BUFFER

STATION VARIABLE TABLE (SVT)

TEST HEAD DRIVER (THD)

SUBROUTINES & $IOCS DRIVERS

MONITOR

SYXVEC (SUBROUTINE TRANSFER VECTORS

GLOVAR (GLOBAL VARIABLES)

DBFLD (DOUBLE-BIT FIELDS)

DECFLD/OCTFLD (OCTAL/DECIMAL CONSTANTS)

LFMSK (MASK SET FROM LEFT)

RTMSK (MASK SET FROM RIGHT)

NBTFLD (BITS NOT SET)

BITFLD (BITS SET)

SYSVAR (SYSTEM VARIABLES)

INTERRUPT ADDRESSES

ENTRY/RESTART

MASTR MEMORY MAP

MASTR MEMORY ORGANIZATION

ENTRY/RESTART

- SYSTEM INITIAL AND RESTART ENTRY POINTS

INTERRUPT ADDRESSES

- PERIPHERAL AND TESTER INTERRUPT ADDRESS VECTORS

SYSVAR

- GLOBAL VARIABLES USED BY SYSTEM TO DEFINE ITEMS SUCH AS LINE PRINTER TYPE, DEFAULT DEVICE FOR FILE LOAD/DUMP, SYSTEM INITIALIZED FLAG, AND DISC DIRECTORY/WORKING STORAGE/FILE AREA ADDRESSING INFORMATION.

SYSTEM CONSTANTS AND MASKS

- BITFLD/NBTFLD
- RTMSK/LFMSK
- DBFLD
- DECFLD/OCTFLD
- CAN BE REFERENCED BY USER DEFINED EQU'S

GLOVAR

- SYSTEM RELATED ITEMS WHICH ARE GLOBAL IN NATURE AND NOT UNIQUE FOR EACH TEST STATION. REFERENCED BY USER DEFINED EQU'S

SYXVEC

- ENTRY POINTS OF SYSTEM SUBROUTINES
- CAN BE REFERENCED BY USER BSM*

MONITOR

- MONITORS OVERALL SYSTEM OPERATION
- PROCESSES SYSTEM INTERRUPTS AND OPERATOR COMMANDS

SUBROUTINES AND $IOCS DRIVERS

- SYSTEM SUBROUTINES
- ACCESSABLE BY USER WITH BSM* TO SYXVEC TABLE

TEST HEAD DRIVER

- EXECUTES FACTOR TEST PROGRAM AND CONTROLS TESTER HARDWARE

STATION VARIABLE TABLE (SVT)

- CURRENT STATION VARIABLE TABLE. CONTAINS DATA UNIQUE TO STATION WHICH IS CURRENTLY ON-LINE. ACCESSED BY XR1.

ALTER BUFFER

- BUFFER FOR FACTOR PROGRAM VARIABLES TEMPORARILY ALTERED BY MASTR 'ALTER' COMMAND.

TESTER VARIABLE TABLE (TVT)

- TEST HEAD VARIABLE TABLE. CONTAINS DATA UNIQUE TO EACH OF UP TO 4 TEST HEADS (STATIONS). REFERENCED BY XR2.

I/O ACTIVITY TABLE (IOATAB)

- EACH SYSTEM OR USER INITIATED I/O OPERATION CAUSES AN ENTRY IN THIS TABLE, DEFINING INFORMATION SUCH AS: DISC FILE NAME, DISC ADDRESSES, CPU MEMORY ADDRESSES, ETC.

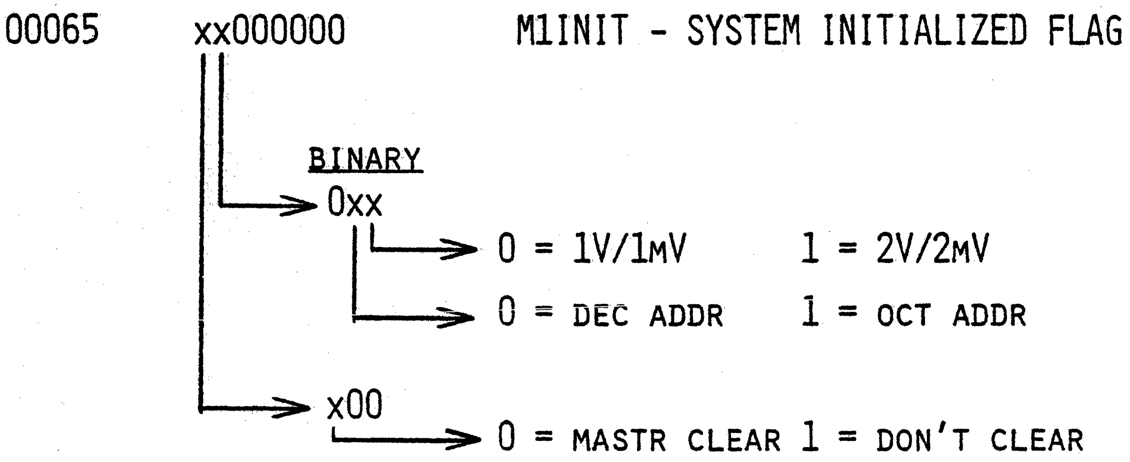
MEMORY ACTIVITY TABLE (MACTAB)

- DIRECTORY OF PROGRAMS CURRENTLY LOADED INTO CPU MEMORY, ALONG WITH STATUS INFORMATION.

```
                          *
      00000000                      ORG    0
                          *
                          *  63 ADDRESSES ARE RESERVED FOR INTERRUPTS
                          *
00000 01001753   SYSENT     BRU    SYSINT      SYSTEM INITIAL/RESTART ENTRY POINT
                          *
                          *  INTERRUPT ADDRESSES                          PRIORITY
                          *
~~00001 00000001                    BAH    1~~
00002 12011603                      BSM    TTRIN2      TTK INTERRUPT          9
00003 12012036                      BSM    TTPIN3      TTP INTERRUPT          8
~~00004 12014145                    BSM    CRINT4      CARD READER INTERRUPT  5~~
00005 00000005                      BAH    5
00006 12012771                      BSM    LPINT6      LP INTERRUPT           7
~~00007 12015000                    BSM    DSCIN7      DISC INTERRUPT         3~~
00010 12013723                      BSM    TAPIN8      MAG TAPE INTERRUPT     4
00011 00000011                      BAH    11B         FUTURE EXANSION
                          *
00012 12004434                      BSM    CLKIN10     REAL TIME CLOCK        2
00013 12005375                      BSM    TINT11      TESTER, CLOCK TIME OUT  2-12
~~00014 12005400                    BSM    TINT12      TESTER, DMA TRAP       2-8~~
00015 12005403                      BSM    TINT13      TESTER, DCT PASS/FAIL, FCT FAIL 2-5
00016 12005406                      BSM    TINT14      TFSTER, DPS TRIP       2-7
~~00017 12005411                    BSM    TINT15      TESTER, RESET          2-3~~
00020 12005414                      BSM    TINT16      TESTER, INSTR COMPARE   2-14


00021 12005417                      BSM    TINT17      TESTER, BUSY COMPLETE   2-10
~~00022 00000022                    BAH    22B         TESTER, SPARE          2~~
00023 00000023                      BAH    23B         TESTER, SPARE          2
00024 00000024                      BAH    24B         TESTER, SPARE          2
~~00025 00000025                    BAH    25B         TESTER, SPARE          2~~
00026 12005422                      BSM    TINT22      TESTER, START          2-15
00027 00000027                      BAH    27B         TESTER,                2
                          *
00030 12016507                      BSM    CLIN24      COM. LINK INTERRUPT    1
      00000052                      ORG    52B
~~00052 12012224                    BSM    VKTI42      2ND VKT KEY BOARD~~
00053 12012245                      BSM    VPTI43      2ND VKT PRINTER
```

```
                                 PAGE
         00000064                ORG    64B
                       *
                       *  GLOBAL VARIABLF FOR USE BY THE SYSTEM
                       *
         00000064   SYSVAR       EQU    *           N
00064    00000210   DFDV         DATA   210B        0  DEFAULT DEVICE FOR FILE LOAD/DUMP
00065    00000000   M1INIT       DATA   0           1  SYSTEM INITIALIZED FLAG
00066    00000002   SELP         DATA   2           2  LP TYPE, DEFAULT=PRINTRONIX
00067    00000000   M1WSWC       DATA   0           3  # OF WORD IN WS
00070    00000000   M1WSSC       DATA   0           4  # OF SECTORS IN WS
00071    00000000   M1WSDA       DATA   0           5  START SECTOR OF WS
00072    00000000   M1FDDA       DATA   0           6  START SECTOR OF DIRECTORY
00073    00000000   M1FDA        DATA   0           7  START SECTOR OF FILE AREA
00074    00000000                DATA   0           8   SPARE
00075    00001753                DATA   SYSINT      9 ENTRY POINT
00076    00000000                DATA   0           10  SPARE
                       *
         00000100                ORG    100B
00100    01002107                BRU    MPRO        RESTART FROM 100B
```

SYSVAR
(GLOBAL VARIABLES USED BY MASTR)



00065 xx000000 M1INIT - SYSTEM INITIALIZED FLAG

BINARY

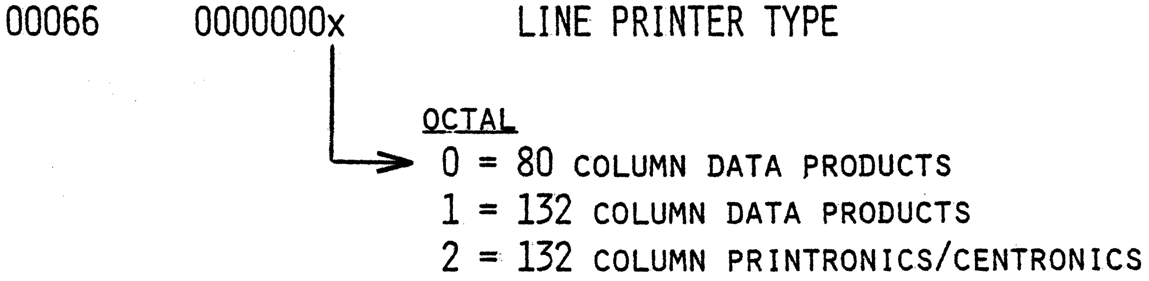
0xx

0 = 1V/1mV 1 = 2V/2mV

0 = DEC ADDR 1 = OCT ADDR

x00

0 = MASTR CLEAR 1 = DON'T CLEAR



00066 0000000x LINE PRINTER TYPE

OCTAL

0 = 80 COLUMN DATA PRODUCTS
1 = 132 COLUMN DATA PRODUCTS
2 = 132 COLUMN PRINTRONICS/CENTRONICS

```
                              PAGE
                        * BIT FIELD DEFINITIONS
                        * BX IMPLIES BIT X IS SET
                        * NBX IMPLIES NOT BIT X IS SET
                        * BX$Y IMPLIES BIT X THRU Y ARE SET
                        *
                        * SINGLE BIT FIELDS
                        *
00101 00000001                DATA   1B         FOR PMU CONVERGENCE TABLE
      00000102  BITFLD        EQU    *
00102 00000001  B0            DATA   1B
00103 00000002  B1            DATA   2B
00104 00000004  B2            DATA   4B
00105 00000010  B3            DATA   10B
00106 00000020  B4            DATA   20B
00107 00000040  B5            DATA   40B
00110 00000100  B6            DATA   100B
00111 00000200  B7            DATA   200B
00112 00000400  B8            DATA   400B
00113 00001000  B9            DATA   1000B
00114 00002000  B10           DATA   2000B
00115 00004000  B11           DATA   4000B
00116 00010000  B12           DATA   10000B
00117 00020000  B13           DATA   20000B
00120 00040000  B14           DATA   40000B
00121 00100000  B15           DATA   100000B
00122 00200000  B16           DATA   200000B
00123 00400000  B17           DATA   400000B
00124 01000000  B18           DATA   1000000B
00125 02000000  B19           DATA   2000000B
00126 04000000  B20           DATA   4000000B
00127 10000000  B21           DATA   10000000B
00130 20000000  B22           DATA   20000000B
00131 40000000  B23           DATA   40000000B
                        *
                        * NOT BIT-TABLE (INVERSE OF BIT)
                        *
      00000132  NBTFLD        EQU    *
00132 77777776  NB0           DATA   77777776B
00133 77777775  NB1           DATA   77777775B
00134 77777773  NB2           DATA   77777773B
00135 77777767  NB3           DATA   77777767B
00136 77777757  NB4           DATA   77777757B
00137 77777737  NB5           DATA   77777737B
00140 77777677  NB6           DATA   77777677B
00141 77777577  NB7           DATA   77777577B
00142 77777377  NB8           DATA   77777377B
00143 77776777  NB9           DATA   77776777B
00144 77775777  NB10          DATA   77775777B
00145 77773777  NB11          DATA   77773777B
00146 77767777  NB12          DATA   77767777B
00147 77757777  NB13          DATA   77757777B
00150 77737777  NB14          DATA   77737777B
00151 77677777  NB15          DATA   77677777B
00152 77577777  NB16          DATA   77577777B
00153 77377777  NB17          DATA   77377777B
00154 76777777  NB18          DATA   76777777B
00155 75777777  NB19          DATA   75777777B
00156 73777777  NB20          DATA   73777777B
00157 67777777  NB21          DATA   67777777B
00160 57777777  NB22          DATA   57777777B
00161 37777777  NB23          DATA   37777777B
```

```
                           *
                           * ASSORTED MASKS
                           *
            00000162  RTMSK     EQU    *           N
                           *
      00162 00000003  B1$0      DATA   3B          0
      00163 00000007  B2$0      DATA   7B          1
      00164 00000017  B3$0      DATA   17B         2
      00165 00000037  B4$0      DATA   37B         3
      00166 00000077  B5$0      DATA   77B         4
      00167 00000177  B6$0      DATA   177B        5
      00170 00000377  B7$0      DATA   377B        6
      00171 00000777  B8$0      DATA   777B        7
      00172 00001777  B9$0      DATA   1777B       8
      00173 00003777  B10$0     DATA   3777B       9
      00174 00007777  B11$0     DATA   7777B       10
      00175 00017777  B12$0     DATA   17777B      11
      00176 00037777  B13$0     DATA   37777B      12
      00177 00077777  B14$0     DATA   77777B      13
      00200 00177777  B15$0     DATA   177777B     14
      00201 00377777  B16$0     DATA   377777B     15
      00202 00777777  B17$0     DATA   777777B     16
      00203 01777777  B18$0     DATA   1777777B    17
      00204 03777777  B19$0     DATA   3777777B    18
      00205 07777777  B20$0     DATA   7777777B    19
      00206 17777777  B21$0     DATA   17777777B   20
      00207 37777777  B22$0     DATA   37777777B   21
      00210 77777777  B23$0     DATA   77777777B   22
                           *
            00000211  LFMSK     EQU    *           N
                           *                       *
      00211 77777776  B23$1     DATA   77777776B   0
      00212 77777774  B23$2     DATA   77777774B   1
      00213 77777770  B23$3     DATA   77777770B   2
      00214 77777760  B23$4     DATA   77777760B   3
      00215 77777740  B23$5     DATA   77777740B   4
      00216 77777700  B23$6     DATA   77777700B   5
      00217 77777600  B23$7     DATA   77777600B   6
      00220 77777400  B23$8     DATA   77777400B   7
      00221 77777000  B23$9     DATA   77777000B   8
      00222 77776000  B23$10    DATA   77776000B   9
      00223 77774000  B23$11    DATA   77774000B   10
      00224 77770000  B23$12    DATA   77770000B   11
      00225 77760000  B23$13    DATA   77760000B   12
      00226 77740000  B23$14    DATA   77740000B   13
      00227 77700000  B23$15    DATA   77700000B   14
      00230 77600000  B23$16    DATA   77600000B   15
      00231 77400000  B23$17    DATA   77400000B   16
      00232 77000000  B23$18    DATA   77000000B   17
      00233 76000000  B23$19    DATA   76000000B   18
      00234 74000000  B23$20    DATA   74000000B   19
      00235 70000000  B23$21    DATA   70000000B   20
      00236 60000000  B23$22    DATA   60000000B   21
```

```
                            *
                            *  DECIMAL/OCTAL NUMBER
                            *
            00000237   DECFLD    EQU    *           DECIMAL FIELD
            00000237   OCTFLD    EQU    *           OCTAL FIELD
                            *
00237 00000000   D0        DATA   0           +0
00240 00000001   D1        DATA   1           +1
00241 00000002   D2        DATA   2           +2
00242 00000003   D3        DATA   3           +3
00243 00000004   D4        DATA   4           +4
00244 00000005   D5        DATA   5           +5
00245 00000006   D6        DATA   6           +6
00246 00000007   D7        DATA   7           +7
00247 00000010   D8        DATA   8           +8
00250 00000011   D9        DATA   9           +9
00251 00000012   D10       DATA   10          +10
00252 00000013   D11       DATA   11          +11
00253 00000014   D12       DATA   12          +12
00254 00000015   D13       DATA   13          +13
00255 00000016   D14       DATA   14          +14
00256 00000017   D15       DATA   15          +15
00257 00000020   D16       DATA   16          +16
00260 00000021   D17       DATA   17          +17
00261 00000022   D18       DATA   18          +18
00262 00000023             DATA   19          +19
00263 00000024   D20       DATA   20          +20
00264 00000025   D21       DATA   21          +21
00265 00000026   D22       DATA   22          +22
00266 00000027   D23       DATA   23          +23
00267 00000030             DATA   24          +24
00270 00000031   D25       DATA   25          +25
00271 00000032             DATA   26          +26
00272 00000033             DATA   27          +27
00273 00000034   D28       DATA   28          +28
00274 00000035             DATA   29          +29
00275 00000036             DATA   30          +30
00276 00000037   D31       DATA   31          +31
00277 00000040   D32       DATA   32          +32
00300 00000041   D33       DATA   33          +33
00301 00000042   D34       DATA   34          +34
00302 00000043             DATA   35          +35
00303 00000044             DATA   36          +36
00304 00000045   D37       DATA   37          +37
00305 00000046             DATA   38          +38
00306 00000047             DATA   39          +39
00307 00000050   D40       DATA   40          +40
00310 00000051   D41       DATA   41          +41
00311 00000052   D42       DATA   42          +42
00312 00000053   D43       DATA   43          +43
00313 00000054   D44       DATA   44          +44
00314 00000055   D45       DATA   45          +45
00315 00000056             DATA   46          +46
00316 00000057   D47       DATA   47          +47
00317 00000060   D48       DATA   48          +48
00320 00000061   D49       DATA   49          +49
00321 00000062             DATA   50          +50
00322 00000063             DATA   51          +51
00323 00000064             DATA   52          +52
00324 00000065             DATA   53          +53
00325 00000066             DATA   54          +54
00326 00000067             DATA   55          +55
```

```
00327 00000070        DATA   56         +56
00330 00000071        DATA   57         +57
00331 00000072        DATA   58         +58
00332 00000073        DATA   59         +59
00333 00000074  D60   DATA   60         +60
00334 00000075        DATA   61         +61
00335 00000076  D62   DATA   62         +62
00336 00000077  D63   DATA   63         +63
00337 00000100  D64   DATA   64         +64
00340 00000101  D65   DATA   65         +65
00341 00000102        DATA   66         +66
00342 00000103  D67   DATA   67         +67
00343 00000104        DATA   68         +68
00344 00000105        DATA   69         +69
00345 00000106        DATA   70         +70
00346 00000107        DATA   71         +71
00347 00000110        DATA   72         +72
00350 00000111  D73   DATA   73         +73
00351 00000112  D74   DATA   74         +74
00352 00000113        DATA   75         +75
00353 00000114        DATA   76         +76
00354 00000115        DATA   77         +77
00355 00000116        DATA   78         +78
00356 00000117        DATA   79         +79
00357 00000120  D80   DATA   80         +80
00360 00000121        DATA   81         +81
00361 00000122        DATA   82         +82
00362 00000123        DATA   83         +83
00363 00000124        DATA   84         +84
00364 00000125  D85   DATA   85         +85
00365 00000126        DATA   86         +86
00366 00000127        DATA   87         +87
00367 00000130        DATA   88         +88
00370 00000131        DATA   89         +89
00371 00000132        DATA   90         +90
00372 00000133        DATA   91         +91
00373 00000134  D92   DATA   92         +92
00374 00000135        DATA   93         +93
00375 00000136        DATA   94         +94
00376 00000137        DATA   95         +95
00377 00000140        DATA   96         +96
00400 00000141        DATA   97         +97
00401 00000142        DATA   98         +98
00402 00000143        DATA   99         +99
00403 00000144        DATA   100        +100
                *
      00000113  D512  EQU    BITFLD+9
      00000210  DM1   EQU    RTMSK+22
      00000211  DM2   EQU    LFMSK+0
      00000212  DM4   EQU    LFMSK+1
```

```
                                 PAGE
                        *
                        *
                        * DOUBLE RIT FIELDS
                        *
            00000404    DBFLD       EQU    *               N
      00404 00006000    B11$10      DATA   00006000B       0
      00405 00014000    B12$11      DATA   00014000B       1
      00406 37000000    B22$18      DATA   37000000B       2
      00407 00000360    B7$4        DATA   00000360B       3
      00410 00000300    B7$6        DATA   00000300B       4
            00000255    B3$1        EQU    D14
      00411 00001700    D960        DATA   960         5
```

```
                         PAGE
      00000420           ORG    4208
                 *
                 * GLOBAL VARIABLES FOR USE BY OVERLAYS
                 *
      00000220   NGLOV   EQU    144          # OF WORDS IN GLOVAR
                 *
      00000420   GLOVAR  EQU    *            N
                 *
00420 00000000   ATPA    DATA   0            0  STAT1 TEST PLAN ATTACHED TO STATION
00421 00000000           DATA   0            1  STAT2 + = MACTAB POINTER FOR TP
00422 00000000           DATA   0            2  STAT3 0 = NONE ATTACHED
00423 00000000           DATA   0            3  STAT4 - = BEING EDITED
00424 21211721   RELDAT  TEXT   '11/12/78' 4,5  RELEASE DATE (SYSREL HAS RELEASE #
00425 22172730
00426 00000000   SITEQQ  DATA   0            6  STATION ON LINE (0-6)
00427 00000000   APMREV  DATA   0            REV OF APM SOFTWARE
00430 00000222   NTVT    DATA   TVTL         8  # OF VARIABLES/STATION IN TVT TABLE
00431 00000000   TVT     DATA   0            9  TVT TABLE ADDRESS
00432 00000062   NSVT    DATA   SVTL         10 # OF VARIABLES IN SVT TABLE
00433 00000000   SVT     DATA   0            11 SVT TABLE ADDRESS
00434 00000010   NMAC    DATA   MACEL        12 # OF WORDS/ENTRY IN MACTAB
00435 00000000   FWMAC   DATA   0            13 MACTAB ADDRESS
00436 00000000   LWMAC   DATA   0            14 LAST USED ADDRESS+1 MACTAB
00437 00000000   STAVKT  DATA   0            15 DEFAULT VKT FOR ALL STATIONS
00440 00000000   PIDPMF  DATA   0            16 PID ENTRY ADDRESS IN IOATAB
00441 00000000   PODPMF  DATA   0            17 POD ENTRY ADDRESS IN IOATAB
00442 00000000   DRPMF   DATA   0            18 DISC DIRECT ENTRY ADDRSS IN IOATAB
00443 00000000   FWALT   DATA   0            19 FIRST ADDRESS OF ALTER BUFFER
00444 00000000   LWALT   DATA   0            20 LAST USED ADDRESS +1 OF ALTBUF
00445 00000000   FWIOA   DATA   0            21 FIRST ADDRESS OF IOATAB
00446 00000015   NIOA    DATA   IOAEL        22 # OF WORDS/ENTRY IN IOATAB
00447 00000000   CURSYS  DATA   0            23 CURRENT SYSTEM, 0=BG, 1=FG
00450 00000000   FGBGFL  DATA   0            24 1=BG WAIT FOR FG,2=FG WAIT FOR BG
00451 00000001   REVN    DATA   REV          25 CURRENT REV NUMBER
00452 77777777   JOB     DATA   -1           26 CURRENT JOB NUMBER
00453 00000022   TPHL    DATA   THL          27 TEST PLAN HEADER LENGTH
00454 00000024   OPHL    DATA   OHL          28 OVERLAY HEADER LENGTH
00455 00000000   DATE    DATA   0,0          29 CURRENT DATE
00456 00000000
00457 00000000   TIME    DATA   0            31 CURRENT TIME IN SECONDS
00460 00000000   PGPMF   DATA   0            32 PAGE PMF POINTER
00461 00000000   LWCPU   DATA   0            33 CPU LAST AVAILABLE WORD
00462 00000000   LWSYS   DATA   0            34 LAST SYSTEM RESERVED WORD +1
00463 00000000   LWAM    DATA   0            35 LAST AVAILABLE WORD TO TP, OVERLAY
00464 00000000   FWAM    DATA   0            36 FIRST AVAILABLE WORD TO TP, OVERLAY
00465 00000000   ADJFLG  DATA   0            37 ADJMEM WAITING FOR MEMBSY
00466 00000000   THDACT  DATA   0            38 FG ON/OFF FLAG
00467 00000000   PIDFLG  DATA   0            39 COMMAND ALREADY IN BUFFER FLAG
00470 00000000   ECHFLG  DATA   0            40 ECHO FLAG FOR PROCESS
00471 00000000   COMIMG  DATA   0            41 COMMAND IMAGE FROM PROCESS
00472 00000000   CMDPMF  DATA   0,CMDBUF,18  42 PMF FOR SYSTEM COMMAND
00473 00001503
00474 00000022
00475 00000000   OCTAL   DATA   0            45 OCTAL VALUE OF INTSCN
00476 00000000   DFLERR  DATA   0            46 DECIMAL APPEARED IN INTSCN
00477 00000000   LDFLG   DATA   0            47 LOAD IS BUSY FLAG
00500 00000000   MANTISSA DATA  0            48 INTEGER VALUE FROM NUMBER
00501 00000000   CMDV    DATA   0            49 DEVICE CODE FROM PROCESS
00502 00000000   NAMEM1  DATA   0            50 1ST STRING FROM PROCESS
00503 00000000   NAMEM2  DATA   0            51
00504 00000000   BINUM   DATA   0            52 BINARY VALUE FROM INTSCN/NUMBER
00505 00000000   BINC    DATA   0            53 BINARY COUNT FORM INTSCN/NUMBER
```

```
00506 00000000  COLFLG   DATA  0          54 COLUMN FORMAT FLAG FOR PUTE
00507 00002261  RSTIO    DATA  RSTIOF     55 RESET PENDING SCHEDULER FLAG
00510 00000000  ACTFIO   DATA  0          56 FGIO IS ACTIVE OR PENDING
00511 00000000  MEMBSY   DATA  0          57 MEMORY BUSY
00512 00000000  NAMEM5   DATA  0          58 3RD NAME FROM PROCESS
    3 00000000  NAMEM6   DATA  0          59 3RD NAME FROM PROCESS
00514 00000000  NAMEM3   DATA  0          60 2ND STRING FROM PROCESS
00515 00000000  NAMEM4   DATA  0          61
00516 00000000  ONUMB1   DATA  0          62 OCTAL NUMBER 1 FROM PROCESS
00517 00000000  ONUMB2   DATA  0          63 OCTAL NUMBER 2 FROM PROCESS
00520 00000000  NUMB1    DATA  0          64 1ST DECIMAL # FROM PROCESS IN F.P.
00521 00000000  NUMB2    DATA  0          65 2ND DECIMAL # FROM PROCESS IN F.P.
00522 00000000  STATC    DATA  0          66 VALUE OF STAT N FROM PROCESS
00523 00000000  SPNUM1   DATA  0          67 SPECIAL # FROM PROCESS
00524 00000000  SPNUM2   DATA  0          68 SPECIAL # FROM PROCESS
00525 00000000  SPNUM3   DATA  0          69 SPECIAL # FROM PROCESS
00526 00000000  SPNUM4   DATA  0          70 SPECIAL # FROM PROCESS
00527 00000000  SPNUM5   DATA  0          71 SPECIAL # FROM PROCESS
00530 00000000  SPNUM6   DATA  0          72 SPECIAL # FROM PROCESS
00531 00000000  BINARY   DATA  0          73 BINARY VALUE FROM PROCESS
00532 00000000  INUMB1   DATA  0          74 1ST INTEGER FROM PROCESS
00533 00000000  INUMB2   DATA  0          75 2ND INTEGER FROM PROCESS
00534 00000000  BFLERR   DATA  0          76 NOT BINARY FLAG FROM INTSCN
00535 00000000  SPOPT    DATA  0          77 SPECIAL OPTION FLAG FROM PROCESS
00536 00000000  BINCNT   DATA  0          78 BINARY DIGIT COUNT FROM PROCESS
00537 00002263  AWATF    DATA  WATSPD     79 ADDRESS OF WAIT FLAG SCHEDULER
00540 00002264  ATHDF    DATA  THDFLG     80 ADDRESS OF THDFLAG SCHEDULER
00541 00000000  NUMFLG   DATA  0,0        81 NUMBER APPEARED FLAG FROM IDTSCN
00542 00000000
00543 00000000  NAME1    DATA  0          83 1ST NAME FROM IDTSCN
    4 00000000  NAME2    DATA  0          84 2ND NAME FROM IDTSCN
    5 00000000           DATA  0          85  SPARE
  546 00000000  RAIDRR   DATA  0          86 RAID BREAKEE RR
00547 00001561  DBUGSA   DATA  DBUGS      87 ADDR OF DEBUG ADDR HALT ROUTINE
00550 00000000  DFSTAT   DATA  0          88 DEFAULT STATION ID
00551 00000000  SMAFLG   DATA  0          89 MA STATION CONTROL
00552 00000000  SPDA     DATA  0          90 PD BUFFER BUSY FLAG
00553 00016017  RAIDER   DATA  RAIDEM     91 ADDR OF RAID PG 0 LOGIC
00554 00000100  RAIDBK   DATA  100B       92 SAVE RAID'S RR HERE
00555 00002301  AFGBGF   DATA  FGBGSC     93 FGBGH SCHEDULER FLAG
00556 22142100  SYSREL   DATA  '2,1'      94 SYSTEM REL # IN ASCII
00557 00000000  MASTAT   DATA  0          95 MA STATION FOR CR REQUEST
00560 00000000  CLIOID   DATA  0,0        96 CLIO NAME1,2
00561 00000000
00562 00000000  LOTNUM   DATA  0,0,0      98 LOT # FOR CLIO
00563 00000000
00564 00000000
00565 00000000  DEVNUM   DATA  0,0        101 DEVICE # FOR CLIO
00566 00000000
00567 00000000  CATGRY   DATA  0,0,0      103 CATEGORY FOR CLIO
00570 00000000
00571 00000000
00572 00000000  RSTTSC   DATA  0          106 STSC REG IMAGE
00573 00000000  BGID     DATA  0,0        107 BACKGROUND ID
00574 00000000
```

```
      00000640          ORG   GLOVAR+NGLOV
                *
                *
                * SYSTEM SUBROUTINE TRANSFER VECTOR (BSM* (SYXVEC+N))
                *
                *
      00000640  SYXVEC  EQU   *         N
00640 00002536          TV    WAIT      0  TESTER BUSY WAIT IN FOREGROUND
00641 00006735          TV    OUTOPN    1  OPEN OUTPUT  FOR ALPHA PRINT
00642 00006642          TV    OUTCLS    2  CLOSE OUTPUT ALPHA PRINT
00643 00001726          TV    NUMERR    3  NUMBER ERROR          (BRU*)
00644 00001741          TV    COMERR    4  COMMAND SYNTAX ERROR  (BRU*)
00645 00010322          TV    $IOCS     5  I/O DRIVER
00646 00006651          TV    MSGIN     6  INPUT A MESSAGE FROM SYSTEM PID
00647 00006666          TV    MSGOUT    7  OUTPUT A MESSAGE ON SYSTEM POD
00650 00006702          TV    UMSGW     8  OUTPUT A MESSAGE ON SYSTEM POD W/O CR
00651 00005663          TV    ADJMEM    9  ADJUST MEMORY USAGE
00652 00005655          TV    SCNFIL    10 SCAN FILES IN MEMORY
00653 00003276          TV    GTSTAT    11 DECODE STATION ID FROM COMMAND OR DEF
00654 00015763          TV    CONV      12 CONVERT BINARY TO DECIMAL
00655 00005674          TV    PUTD      13 PUT DECIMAL NUMBER IN BUFFER
00656 00015717          TV    PUTC      14 PUT CHARACTER IN BUFFER
00657 00006343          TV    MOVEDN    15 MOVE A MEMORY BLOCK ASCENDING ORDER
00660 00006352          TV    MOVEUP    16 MOVE A MEMORY BLOCK DESCENDING ORDER
00661 00005666          TV    PUTE      17 PUT ENG # IN BUFFER
00662 00005671          TV    PUTO      18 PUT OCTAL # IN BUFFER
00663 00005660          TV    PROCESS   19 PROCESS A COMMAND
00664 00005724          TV    ALTER     20 ALTER BUFFER SCAN/PROCESS
00665 00005727          TV    $PARSE    21 DEFINE NEXT INPUT RECORD FIELD
00666 00005732          TV    IDTSCN    22 SCAN IDENTIFIER
00667 00005735          TV    NUMBER    23 SCAN A NUMBER (SET IN F.P. FORM)
00670 00005740          TV    INTSCN    24 SCAN INTEGER NUMBER
00671 00015561          TV    SEARCH    25 SEARCH TABLE
00672 00015606          TV    MPZERO    26 CLEAR CORE (X6,X7)
00673 00015615          TV    GETC      27 GET A CHARACTER FROM BUFFER
00674 00015656          TV    READW     28 GET A WORD FROM BUFFER
00675 00015671          TV    WRITEW    29 PUT WORD IN BUFFER
00676 00005425          TV    IERMSG    30 TERMINAL ERROR IN THD
00677 00005710          TV    DUMP      31 DUMP TP.OVLY.MOD
00700 00003307          TV    PUTIME    32 PUT DATE, TIME IN BUFFER
00701 00004360          TV    GTTDV     33 GET IO DEVICE ADDR IN TVT
00702 00005721          TV    HEADER    34 OUTPUT HEADER (BINN,PD,DCE,MA)
00703 00003106          TV    SPIOER    35 $IOCS ERROR CHECK AND MSG
00704 00003510          TV    FGOVC     36 CALL OVERLAY FOREGROUND
00705 00003474          TV    ALLEX     37 ASSEMBLY LANGUAGE LINKAGE EXECUTION
00706 00002341          TV    COMMND    38 RETURN TO MONITOR (BRU* )
00707 00015701          TV    PUTW      39 PLACE 4 TASCII CHARACTERS IN BUFFER
00710 00003442          TV    TWAIT     40 WAIT ON TESTER ACTIVITY
00711 00003403          TV    FGBGRT    41 SCHEDULE BACKGROUND FROM FOREGROUND
00712 00003637          TV    MONINT    42 ENTER SCHEDULER(SAVE HARDWARE)
00713 00005452          TV    SCALE     43 SCALE F.P. TO TESTER VALUES
00714 00003457          TV    FGWAIT    44 FOREGROUND WAIT FOR BACKGROUND
00715 00005343          TV    ERRCNV    45 $IOCS ERROR CODE DECODER, MESSAGE OU
00716 00005472          TV    FSUB      46 FLOATING POINT SUBTRACT (A-E -> A)
00717 00005500          TV    FAND      47 FLOATING POINT AND (A AND E -> A)
00720 00005506          TV    FEOR      48 FLOATING POINT EOR (A EOR E -> A)
00721 00005522          TV    FLOG      49 FLOATING POINT LOG (LOG A -> A)
00722 00005475          TV    FADD      50 FLOATING POINT ADD (A+E -> A)
00723 00005467          TV    FDIV      51 FLOATING POINT DIVIDE (A/E -> A)
00724 00005530          TV    FFIXS     52 FIX FLOATING POINT INTO A,E
00725 00005503          TV    FOR       53 FLOATING POINT OR (A OR E -> A)
```

```
  26 00005511          TV    FNOT      54 FLOATING POINT NOT (A NOT E -> A)
~~   7 00005525          TV    FEXP      55 FLOATING POINT EXPONENT (A => A)~~
 0730 00005464          TV    FMUL      56 FLOATING POINT MULTIPLY (A * E -> A)
 0731 00005514          TV    FFIX      57 FIX FLOATING POINT INTO A (A -> A)
~~ 0732 00005517          TV    FFLT      58 FLOAT INTEGER IN A (A -> A)~~
 0733 00005533          TV    FFLTS     59 FLOAT NUMBER IN A,E (A,E -> A)
 0734 00016004          TV    FCAM      60 FLOATING POINT COMPARE (A,E)
~~ 0735 00005705          TV    LOAD      61 LOAD A FILE INTO MEMORY~~
 0736 00003052          TV    DELFIL    62 DELETE FILE BY NAME CHANGE
 0737 00004420          TV    RELOV     63 CALL OVLY FOR RELEASE
~~ 0740 00004554          TV    ATTA      64 ATTACH OVLY TO STATN~~
 0741 00004571          TV    DTTA      65 DETTACH OVLY FROM STATN
00742 00004663          TV    PAGETP    66 PAGE TEST PLAN
~~00743 00004604          TV    FGBGWT    67 FG/BG WAITING~~
00744 00004631          TV    FGBGH     68 FG/BG HALT FOR MEM MOVE
00745 00005436          TV    FINDVL    69 FIND VARIABLE ADDR IN STACK
~~00746 00006364          TV    FGIO      70 FOREGROUND IO~~
00747 00005441          TV    DMASTR    71 START TESTER DMA AND WAIT FOR DONE
00750 00006532          TV    FGOH      72 FOREGROUND HEADER OUT
~~00751 00005444          TV    ENBTST    73 ENABLE LOCAL MEMORY TEST~~
00752 00005447          TV    WWAIT     74 WAIT FOR LM TEST IN FG
00753 00005713          TV    ADRXLA    75 DISC ADDR TRANSLATE
~~00754 00005456          TV    INTERP    76 CALL INTERPRETER~~
00755 00005461          TV    ENTBSY    77 ENABLE TESTER BUSY COMP INTERRUPT
00756 00003611          TV    RSOVC     78 CALL OVLY FOR FG RESET
~~00757 00004025          TV    STALL     79 STALL BG PROCESS~~
    0 00005743          TV    UPDATE    80 CREATE/DELETE A FILE
    1 00005751          TV    PUTENG    81 PUT ENG VALUES IN BUFFER
~~00762 00005754          TV    PUTA      82 PUT OCTAL/DECIMAL ADDR IN BUFFER~~
00763 00003757          TV    BGCHK     83 CHECK IF BG ACTIVITY ON
00764 00000000          TV    0         84
~~00765 00005757          TV    CALLMOD   85 CALL OVLY MODULE~~
00766 00005677          TV    PUTB      86 PUT BINARY # IN BUFFER
00767 00005702          TV    PUTH      87 PUT HEX # IN BUFFER
~~00770 00006061          TV    SAVENV    88 SAVE ENVIRONMENT~~
00771 00006115          TV    USVENV    89 RESTORE ENVIRONMENT
                   *    TV    IBIO      90 IBUS DRIVER - RESERVED
                   *
```

```
                          PAGE
                 *
                 * CURRENT STATION VARIABLE TABLE SVT
                 *
                 *        ADDRESSED BY INDEX SP(X1)
                 *
        00000062 SVTL     EQU    50
                 *
                 *
        00000000 SITE     EQU    0               ON-LINE SITE
        00000001 STHC     EQU    1               TEST HEAD CONTROL
        00000002 SPIN     EQU    2               CURRENT PMU PIN NUMBER
        00000003 SMSR     EQU    3               CURRENT PMU MEASUREMENT
        00000004 SMF      EQU    4               MEASURE FLAG FOR DL
        00000005 SEIR     EQU    5               EIR IMAGE
        00000006 STEF     EQU    6               TERMINAL ERROR FLAG
        00000007 SVOFFS   EQU    7               VOFFSET VALUE
        00000010 SLML     EQU    8               LOCAL MEMORY LOCATION
        00000011 STRIP    EQU    9               DPS TRIP STATUS
        00000012 STPP     EQU    10              TEST PLAN ADDRESS
        00000013 SMSRH    EQU    11              SAVE MEASUREMENT IN REG FORMAT FOR MA
        00000014 SINC     EQU    12              INC ENABLE FLAG
        00000015 SFVAL    EQU    13              FORCING VALUE FOR MACRO
        00000016 SPG      EQU    14              LOCAL MEMORY PAGE SIZE
        00000017 SPMOD    EQU    15              PROGRAM MODE (PPM/SPM,ETC)
        00000020 SDLAF    EQU    16              DL ADDITIONAL FAIL FLAG
        00000021 SIFC     EQU    17              IF COUNT FOR DL BY COUNT
        00000022 SIFV     EQU    18              IF COUNT FOR MA SYNC, SET IF
        00000023 SMR      EQU    19              MODE OF MEASUREMENT -SET PMU SENSE
        00000024 SFR      EQU    20              MODE OF FORCE - SET PMU FORCE
        00000025 SSAMC    EQU    21              CLAMP BIT (28 VOLT SWING) FOR SAMC, ANY
                 *                               TO SAMC SHOULD OR IN THIS GLOBAL, EXCEPT
                 *                               ANALYSIS
        00000026 SQ       EQU    22              ORIGINAL VALUE OF Q REG
        00000027 SQL      EQU    23              ORIGINAL VALUE OF QL
        00000030 SLIM0    EQU    24              LIMIT 0 OR LIMIT IF ONLY ONE
        00000031 SLIM1    EQU    25              LIMIT 1 WHEN THERE ARE 2 LIMITS
        00000032 SDCT0E   EQU    26              ENABLE DCT0 FLAG WORD
        00000033 SDCT0    EQU    27              ENABLE DCT0 VALUE
        00000034 SDCT1F   EQU    28              ENABLE DCT1 FLAG WORD
        00000035 SDCT1    EQU    29              ENABLE DCT1 VALUE
        00000036 SILOE    EQU    30
        00000037 SILO     EQU    31              ENABLED LIMIT VALUE
        00000040 SIHIE    EQU    32              ENABLE IHI
        00000041 SIHI     EQU    33
        00000042 SVLOE    EQU    34              ENABLE VLO
        00000043 SVLO     EQU    35
        00000044 SVHIE    EQU    36              ENABLE VHI
        00000045 SVHI     EQU    37
                 *
        00000046 S488CT   EQU    38              488 BUS CONTROL WORD
        00000047 SAPMCT   EQU    39              APM CONTROL WORD
```

ACCESSING THE SVT TABLE

THERE ARE 2 METHODS:

1) F.G.

XR1 POINTS TO SVT TABLE ON ENTRY TO FOREGROUND. XR1 SHOULD BE SAVED FOR LATER REFERENCE:

```
          STX        XR1, SVTSAV
```

2) F.G. AND B.G.

THE ABSOLUTE ADDRESS OF THE SVT TABLE IS DEFINED IN GLOVAR CELL #11 (433B).

```
    SVT       EQU        433B
    SVOFFS    EQU        7  (MUST BE DECIMAL!)
    XR1       EQU        1
              .
              .
              LDX*       XR1, SVTSAV
              LDA        SVOFFS, XR1        GET FLOATING POINT
              .                             VOFFSET VALUE
              .
              .
              .
SVTSAV        DATA       Ø
```

```
                   *
                   *
                   * TEST HEAD VARIABLE TABLE EQU'S (TVT)
                   *
                   *        ACCESSED BY INDEX TP(X2)
                   *
         00000222  TVTL     EQU   146
                   *
         00001110  TVTLT    EQU   TVTL+TVTL+TVTL+TVTL  4 STATIONS
                   *
         00000000  TSWITCH  EQU   0          GLOBAL VARIABLE SWITCH
         00000001  TVALUE   EQU   1          GLOBAL VARIABLE VALUE
         00000002  TSN      EQU   2          GLOBAL VARIABLE SN
         00000003  TTT      EQU   3          GLOBAL VARIABLE TT
         00000004  TDATAL   EQU   4          DATALOG REQUEST
         00000005  TRTD     EQU   5          ROUND TRIP DELAY
                   *        EQU   6               SPARE
                   *        EQU   7               SPARE
                   *        EQU   8               SPARE
                   *        EQU   9               SPARE
                   *        EQU   10              SPARE
         00000013  TGLOB1   EQU   11         GLOBAL VARIABLE GLOB1
                   *TGLOB2  EQU   12         GLOBAL VARIABLE GLOB2
                   *TGLOB3  EQU   13         GLOBAL VARIABLE GLOB3
                   *TGLOB4  EQU   14         GLOBAL VARIABLE GLOB4
                   *TGLOB5  EQU   15         GLOBAL VARIABLE GLOB5
                   *TGLOB6  EQU   16         GLOBAL VARIABLE GLOB6
                   *TGLOB7  EQU   17         GLOBAL VARIABLE GLOB7
                   *TGLOB8  EQU   18         GLOBAL VARIABLE GLOB8
                   *TGLOB9  EQU   19         GLOBAL VARIABLE GLOB9
                   *TGL010  EQU   20         GLOBAL VARIABLE GLOB10
                   *TGL011  EQU   21         GLOBAL VARIABLE GLOB11
                   *TGL012  EQU   22         GLOBAL VARIABLE GLOB12
                   *TGL013  EQU   23         GLOBAL VARIABLE GLOB13
                   *TGL014  EQU   24         GLOBAL VARIABLE GLOB14
                   *TGL015  EQU   25         GLOBAL VARIABLE GLOB15
                   *TGL016  EQU   26         GLOBAL VARIABLE GLOB16
                   *TGL017  EQU   27         GLOBAL VARIABLE GLOB17
                   *TGL018  EQU   28         GLOBAL VARIABLE GLOB18
                   *TGL019  EQU   29         GLOBAL VARIABLE GLOB19
                   *TGL020  EQU   30         GLOBAL VARIABLE GLOB20
                   *TGL021  EQU   31         GLOBAL VARIABLE GLOB21
                   *TGL022  EQU   32         GLOBAL VARIABLE GLOB22
                   *TGL023  EQU   33         GLOBAL VARIABLE GLOB23
                   *TGL024  EQU   34         GLOBAL VARIABLE GLOB24
                   *TGL025  EQU   35         GLOBAL VARIABLE GLOB25
                   *TGL026  EQU   36         GLOBAL VARIABLE GLOB26
                   *TGL027  EQU   37         GLOBAL VARIABLE GLOB27
                   *TGL028  EQU   38         GLOBAL VARIABLE GLOB28
                   *TGL029  EQU   39         GLOBAL VARIABLE GLOB29
                   *TGL030  EQU   40         GLOBAL VARIABLE GLOB30
                   *TGL031  EQU   41         GLOBAL VARIABLE GLOB31
                   *TGL032  EQU   42         GLOBAL VARIABLE GLOB32
                   *TGL033  EQU   43         GLOBAL VARIABLE GLOB33
                   *TGL034  EQU   44         GLOBAL VARIABLE GLOB34
                   *TGL035  EQU   45         GLOBAL VARIABLE GLOB35
                   *TGL036  EQU   46         GLOBAL VARIABLE GLOB36
                   *TGL037  EQU   47         GLOBAL VARIABLE GLOB37
                   *TGL038  EQU   48         GLOBAL VARIABLE GLOB38
                   *TGL039  EQU   49         GLOBAL VARIABLE GLOB39
         00000062  TGL040   EQU   50         GLOBAL VARIABLE GLOB40
                   *        EQU   51              SPARE
```

```
          *        EQU    52              SPARE
          *        EQU    53              SPARE
          *        EQU    54              SPARE
          *        EQU    55              SPARE
          *        EQU    56              SPARE
          *        EQU    57              SPARE
          *        EQU    58              SPARE
          *        EQU    59              SPARE
          *        EQU    60              SPARE
          *
00000075  TINDEX   EQU    61          BINNIG INDEX
00000076  TCPC     EQU    62          PROCESS CONTROL,PAUSE,SYNC
00000077  TDCDLY   EQU    63          DC TIME DELAY
00000100  TODLY    EQU    64          LM TIME OUT DELAY
00000101  TOVER    EQU    65          MA OVERRIDE
          *        EQU    66              SPARE
00000103  TDLO     EQU    67          DL OPTION & DEVICE
00000104  TDLF     EQU    68          DATALOG FREQUENCY COUNT
00000105  TDLS     EQU    69          DATALOG SKIP CONTROL
00000106  TDLR     EQU    70          DATALOG REQUEST
00000107  TDLC     EQU    71          DATALOG CONTROL & STATUS
00000110  TLMFC    EQU    72          DATALOG ADDITIONAL FAIL COUNT
00000111  TPDF     EQU    73          PD FREQUENCY COUNT
00000112  TPDS     EQU    74          PD SKIP CONTROL
00000113  TPDR     EQU    75          PD REQUEST
00000114  TDFR     EQU    76          DCF REQUEST
00000115  TMACTL   EQU    77          MA CONTROL
00000116  TPPO     EQU    78          MA PPM REQUEST
00000117  TSYNC    EQU    79          MA SYNC COUNT
          *        EQU    80              SPARE
00000121  TMOD     EQU    81          PPM,LM MODULE NUMBER
          *
00000122  TAPMP1   EQU    82          APM PROCEDURES #1
00000123  TAPMP2   EQU    83          APM PROCEDURES #2
00000124  TAPMF1   EQU    84          APM FILE NAME #1
00000125  TAPMF2   EQU    85          APM FILE NAME #2
          *
          *        EQU    86              SPARE
          *        EQU    87              SPARE
          *        EQU    88              SPARE
          *        EQU    89              SPARE
          *        EQU    90              SPARE
          *        EQU    91              SPARE
          *        EQU    92              SPARE
00000135  TPDD     EQU    93          DIST COUNT
00000135  TVTLL    EQU    93          END OF LOCAL CLEAR (// CLEAR STAT)
00000136  TSTEP    EQU    94          PROGRAM STEP COUNT
00000137  TPAUSE   EQU    95          PROGRAM PAUSE COUNT
00000140  TIP      EQU    96          INSTRUCTION POINTER
00000141  TBINT    EQU    97          BIN INITIALIZED
00000142  TBINS    EQU    98          BIN STATUS
00000143  TMPIN    EQU    99          MAX PIN (DEFAULT = 60)
          *        EQU    100             SPARE
          *        EQU    101             SPARE
          *        EQU    102             SPARE
          *        EQU    103             SPARE
          *        EQU    104             SPARE
00000150  TVTLG    EQU    104         END OF GLOBAL CLEAR (// LOAD STAT)
00000151  TPID     EQU    105         PID ADDRESS IN IOATAB
00000152  TTTK     EQU    106         TTK  ADDRESS IN IOATAB
00000153  TMTR1    EQU    107         MTR1 ADDRESS IN IOATAB
00000154  TMTR2    EQU    108         MTR2 ADDRESS IN IOATAB
00000155  TCR      EQU    109         CR   ADDRESS IN IOATAB
```

```
00000156  TDIF      EQU   110       DIF  ADDRESS IN IOATAB
          *         EQU   111            SPARE
00000160  TVK2      EQU   112       VK2 ADDRESS IN IOATAB
00000161  TMIF      EQU   113       MIF ADDRESS IN IOATAB
00000162  TPOD      EQU   114       POD  ADDRESS IN IOATAB
00000163  TTTP      EQU   115       TTP  ADDRESS IN IOATAB
00000164  TMTW1     EQU   116       MTW1 ADDRESS IN IOATAB
00000165  TMTW2     EQU   117       MTW2 ADDRESS IN IOATAB
00000166  TLP       EQU   118       LP   ADDRESS IN IOATAB
00000167  TDOF      EQU   119       DOF  ADDRESS IN IOATAB
00000170  TCLO      EQU   120       CLO  ADDRESS IN IOATAB
00000171  TVP2      EQU   121       VP2 ADDRESS IN IOATAB
00000172  TMOF      EQU   122       MOF ADDRESS IN IOATAB
00000173  TOPT      EQU   123       TESTER OPTION CONTROL
00000174  TATTA     EQU   124       ATTACH FLAG
00000175  TJOB      EQU   125       STATION'S JOB NUMBER
00000176  TMSTK     EQU   126       MAX STACK SIZE USED
00000177  TOMSTK    EQU   127       MAX STACK SI7F SPECIFIED
00000200  TRTDS     EQU   128       SAVE ROUND TRIP DELAY SO IT WONT BE CLE
          *         EQU   129            SPARE
00000202  TTITLE    EQU   130 THRU 145 STATION TITLE
```

ACCESSING THE TVT TABLE

REFERENCED VIA XR2

F.G.: XR2 POINTS TO TVT FOR CURRENT STATION ON LINE ON ENTRY TO FOREGROUND. XR2 SHOULD BE SAVED FOR LATER REFERENCE:

```
STX    XR2, TVTSAV
```

B.G.: MUST USE SYSTEM ROUTINE 'GTSATS' TO OBTAIN POINTER:

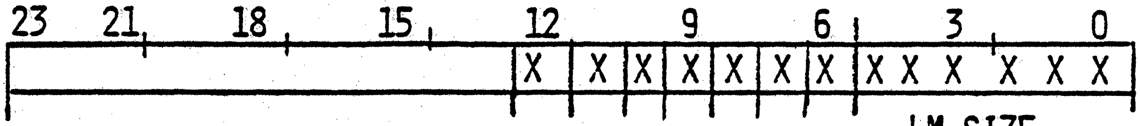
```
LDA    SITEQQ         CURRENT STAT ON LINE (426B)
ADD    D1             NEED NO. 1-4
STA    STATC          (522B)
BSM*   GTSTAT         (653B)
NOP    Ø
STX    XR2, TVTSAV    SAVE FOR LATER
```

EXAMPLE:

```
TOPT   EQU    123            (123 DECIMAL!)
B11    EQU    115B
       .
       .
       LDX*   XR2, TVTSAV
       LDA    TOPT, XR2      GET HARDWARE CONFIGURATION
       CAM    B11            LOW VOLTAGE HEAD?
       BBC    ISLVTH
       .
       .
       .
TVTSAV DATA   Ø
D1     DATA   1
```

TOPT (TVT 123)

IT IS USED TO SAVE HARDWARE OPTION. IT IS SET DURING SYSTEM INITIALIZATION (AFTER BOOT FROM MAG TAPE OR ENTERED FROM DOPSY) AND NEVER CLEARED.



LM SIZE

1K = 001
2K = 011
4K = 111

B12	28 VOLT SWING	B18 MODE REGISTER
B10	NEW REF/MUX MODULE	B15 MODE
B11	LOW VOLTAGE	B17 MODE
BIT 9 = 1	2V/2MV OPTION	RVS RANGE BIT
8 = 1	SPM	
7 = 1	PPM	SAMC BITS 6-8
6 = 1	10 MHZ HEAD	
BITS 5-0 =	LM SIZE	

MACTAB (MEMORY ACTIVITY TABLE)

23 21 18 15 12 9 6 3 0

MAJOB	0	JOB NUMBER					
MANAM	1	NAME 1					
MATYP	2	NAME 2	ACTIVE IND	TYPE			
MAMADR	3	0	MEMORY ADDRESS				
MAFSIZ	4	M	K	R	P		FILE SIZE
MAMSIZ	5	0	MEMORY SIZE				
MAIOPT	6	0	IOATAB POINTER				
MAWSTR	7	0	WINDOW START				

8 words/entry, 32 entries in table
Total 256 words. The table is expandable by ASSIGN command.

IOATAB (INPUT/OUTPUT ACTIVITY TABLE)

13 WORDS/ENTRY, 20 ENTRIES/TABLE
260 WORDS
First 6 Entries are reserved by the system

Bits: 23 21 18 15 12 9 6 3 0

FDSTAT	0	BF DM EF BK CM LM BM STAT WS IF DEVICE CODE
FNAME1	1	FIRST 4 CHARACTERS OF FILE NAME
FNAME2	2	LAST 2 CHAR OF FILE 0 FILE TYPE
FJOB	3	JOB NUMBER
FDADDR	4	STARTING DISC ADDRESS
FSIZE	5	FILE SIZE
FCBBP	6	0 CURRENT BLOCK ADDRESS
FBBADR	7	0 BLOCK BUFFER ADDRESS
FBBSIZ	8	BLOCK BUFFER SIZE
FCDADR	9	ERROR CODE
FCSIZE	10	CURRENT FILE SIZE
FRSIZE	11	0 RECORD SIZE
	12	0

REVIEW OF ASSEMBLY LANGUAGE PROGRAM LAYOUT

GENERAL PROGRAM LAYOUT

```
        REL
*EQU TABLE


*PROGRAM HEADER


*MAIN PROGRAM
     CONTAINS FOREGROUND & BACKGROUND SECTIONS
     IF BOTH USED
*OPTIONAL RESET ROUTINE


*OPTIONAL RELEASE ROUTINE


*SUBROUTINES - IF USED


*DATA DIRECTIVES/FIXED BUFFERS



*
LWORD   EQU   *
        END
```

PROGRAM HEADER

ALWAYS 22 WORDS THOUGH CONTENTS MAY VARY

```
        REL
FWORD   DATA   Ø
        TEXT   `ANAME '        PROGRAM NAME

        DATA   77B OR 71B      PROGRAM TYPE[1]
        DATA   '1.2'           OPTIONAL RELEASE #
        DATA   LWORD-FWORD     PROGRAM SIZE
        BSS    6
RSENT   PZE    Ø               RESET      ENTRY[1]
        BRU    RESET
        BSS    2
RLENT   PZE    Ø
        BRU    RELESE          RELEASE ENTRY[1]
BGENT   PZE    Ø               BACKGROUND ENTRY[1]
        BRU*   BGENT
FGENT   PZE    Ø               FOREGROUND ENTRY[1]
        BRU    BEGIN
        DATA   2               OPTIONAL:  DATE OF
        TEXT   `12/9/77'       LAST PROGRAM RELEASE
```

----

1) SEE DETAILED DESCRIPTION ON FOLLOWING PAGES.

PROGRAM TYPE DEFINITION - 77B AND 71B

TYPE 77B: MOVABLE PROGRAMS. ALL FOREGROUND PROGRAMS AND ANY BACKGROUND PROGRAMS WHICH DO NOT QUALIFY FOR TYPE 71B. THEY ARE LOADED FOLLOWING LAST FILE AT END OF MEMORY.

TYPE 71B: FIXED PROGRAMS. ANY BACKGROUND PROGRAM WHICH:

- RUNS FOR MORE THAN 1 SECOND
- LOADS FILES TO MEMORY WITH SYSTEM ROUTINE 'LOAD' OR 'PAGETP' (E.G. LMLOAD)
- WILL EXPAND ITS SIZE IN MEMORY DURING EXECUTION WITH SYSTEM ROUTINE 'ADJMEM' (E.G. COMPILER)

ARE LOADED ONLY DURING A FOREGROUND BREAKPOINT IN TESTING AND ARE INSERTED JUST FOLLOWING LAST ACTIVE TYPE 71 PROGRAM NEAR END OF OPERATING SYSTEM. WILL SLOW TESTING THRU-PUT IF ASSIGNED TYPE 77B INSTEAD.

- ARE AUTOMATICALLY RELEASED FROM MEMORY WHEN COMPLETED

EXAMPLES:

TYPE 71B	TYPE 77B
LMLOAD	GLOBS
PSCAN	LABEL
LMIO	XMIT
XGRAPH	PPLOG
COMPILE	

AUTOMATIC PROGRAM EXPANSION

A BUFFER AT THE END OF THE PROGRAM MAY BE EXPANDED BY ANY AMOUNT AT LOAD TIME.

```
        REL
FWORD   DATA    Ø
        TEXT    'ANAME'

        DATA    77B OR 71B
        DATA    `1.2'
        DATA    LWORD-FWORD
BUFS1Z  DATA    3072            BUFFER EXPANSION SIZE
        BSS     5
RSENT   PZE     Ø
        .
        .
        .                       3072 WORDS ARE APPENDED TO
                                END OF PROGRAM AT 'BUF'

        .
        .
        .
LWORD   EQU     *
BUF     EQU     *
        END
```

BUF EQU * BECOMES EQUIVALENT TO
BUF BSS 3072

RESET ENTRY POINT DEFINITION

THE RESET ENTRY POINT IS ENTERED AUTOMATICALLY AS THE RESULT OF AN OPERATOR PRESSING THE TESTER RESET PUSH-BUTTON AT THE TEST STATION, WHICH IN TURN, CAUSES A TESTER INTERRUPT. AFTER THE INTERRUPT OCCURS, THE ASSEMBLY LANGUAGE PROGRAM EXECUTION IS HALTED AUTOMATICALLY AND CONTROL IS PASSED BACK TO THE RESET ENTRY POINT OF THE PROGRAM. THE RESET ROUTINE WITHIN THE PROGRAM, IF ONE IS USED, SHOULD CLOSE ANY OPENED FILES, AND RESTORE REGISTERS AS REQUIRED BEFORE THE PROGRAM IS ABORTED.

EXAMPLE OF RESET ENTRY USAGE

```
                       REL
             FWORD     DATA   0
                       .
                       .
             RSENT     PZE    0
                       BRU    RESET
HEADER                 .
                       .
             BGENT     PZE    0
                       BRU    BGSTRT
                       .
                       .
             BGSTRT    ---
BACKGROUND             .
PROGRAM                .
                       BRU*   BGENT
             RESET     ---
RESET                     CLOSE I/O DEVICES & FILES & MISCELLANEOUS
ROUTINE                BRU*   RSENT
```

RELEASE ENTRY POINT DEFINITION

THE RELEASE ENTRY POINT IS ENTERED AUTOMATICALLY AS THE RESULT OF AN OPERATOR TYPING THE MASTR "RELEASE" COMMAND TO RELEASE THE OVERLAY. THE RELEASE ROUTINE WITHIN THE PROGRAM, IF ONE IS USED, SHOULD CLOSE ANY OPENED FILES, AND RESTORE REGISTERS AS REQUIRED BEFORE THE PROGRAM IS RELEASED FROM MEMORY.

EXAMPLE OF RELEASE ENTRY USAGE

```
                      REL
           FWORD      DATA  0
                      .
                      .
           RLENT      PZE   0
                      BRU   RELESE
HEADER                .
                      .
           BGENT      PZE   0
                      BRU   BGSTRT
                      .
                      .
           BGSTRT     ---
                      .
BACKGROUND            .
PROGRAM               BRU*  BGENT
           RELESE     ---
RELEASE                  CLOSE I/O DEVICES & FILES & MISCELLANEOUS
ROUTINE               BRU*  RLENT
```

FOREGROUND/BACKGROUND ENTRY POINT DEFINITION

FOREGROUND ENTRY POINT

USED AS THE RESULT OF A FACTOR EXEC STATEMENT SPECIFYING THE COREIMAGE NAME OF THE PROGRAM.

BACKGROUND ENTRY POINT

USED AS THE RESULT OF A MASTR KEYBOARD COMMAND SPECIFYING THE COREIMAGE NAME OF THE PROGRAM.

NORMALLY, A PROGRAM IS WRITTEN TO BE ENTERED THROUGH ONLY ONE OF THE TWO ENTRY POINTS. HOWEVER, ENTRY THROUGH EITHER ENTRY POINT AT DIFFERENT TIMES IS ALLOWED AND MUST BE PROGRAMMED AS SUCH.

EXAMPLES OF SYSTEM OVERLAYS WITH DIFFERENT ENTRY POINTS ARE AS FOLLOWS:

F.G.	B.G.	F.G./B.G.
TTIME	LMIO	SPLOT
LPLF	PPLOG	PSCAN
GLOBS	PXLOG	LABEL
DATAIO		XMIT
LMLOAD		
XGRAPH		

ENTRY POINT CODING

A PROGRAM MAY USE ONE OR MORE OF THE FOUR ENTRY POINTS, ACCORDING TO THE FUNCTION OF THE PROGRAM. THE ENTRY POINT IS CODED WITH A "PZE" AT THE ENTRY POINT LOCATION AND, IF IT HAS A FUNCTION WITHIN THE PROGRAM, IS FOLLOWED BY A BRU TO THE SECTION OF THE PROGRAM WHICH IMPLEMENTS THE FUNCTION.

E.G.

```
FGENT   PZE   0
        BRU   FGSTRT
```

ANY ENTRY POINT WHICH WILL NOT HAVE A PROGRAMMED FUNCTION WITHIN THE PROGRAM MUST BE PROGRAMMED WITH A BRU* BACK TO THE ENTRY POINT LOCATION.

E.G.

```
FGENT   PZE   0
        BRU*  FGENT

RSENT   PZE   0
        BRU*  RSENT

RLENT   PZE   0
        BRU*  RLENT

BGENT   PZE   0
        BRU*  BGENT
```

BASIC ASSEMBLY LANGUAGE PROGRAM LAYOUT
(BACKGROUND OVERLAY)

```
        REL
*  EQUATE TABLE - IF USED
CS1     EQU    12B
          .
*  22-WORD HEADER
FWORD   DATA   0
          .

          .
BGENT   PZE    0              BACKGROUND ENTRY POINT
        BRU    BGSTRT
          .

*         .
   END OF HEADER
*  START OF PROGRAM MAIN BODY
BGSTRT  LDA    N
          .
          .
          .
          .

          .
        BRU*   BGENT          BACKGROUND EXIT
*  END OF PROGRAM MAIN BODY
*  SUBROUTINES - IF USED
NAME    PZE    0              SUBROUTINE ENTRY
          .

          .
        BRU*   NAME           SUBROUTINE EXIT
*  END OF SUBROUTINES
*  DATA TABLE - IF USED
D100    DATA   100
θ100    DATA   100B
*
        END                   END OF PROGRAM
```

PASSING PARAMETERS IN BACKGROUND MODE

WHEN CALLING A PROGRAM FOR EXECUTION IN THE BACKGROUND MODE AS A RESULT OF A KEYBOARD COMMAND, THE OPERATOR MAY SPECIFY VARIOUS PARAMETERS ALONG WITH THE FILE NAME TO BE EXECUTED.

E.G. *FILL LP 3

THE PARAMETERS WILL BE SAVED IN GLOVAR BY MASTR BEFORE CONTROL IS PASSED TO THE PROGRAM ("FILL" IN THIS EXAMPLE) WHERE THE PROGRAM CAN THEN FETCH THE PARAMETERS AND PROCESS THEM AS DESIRED.

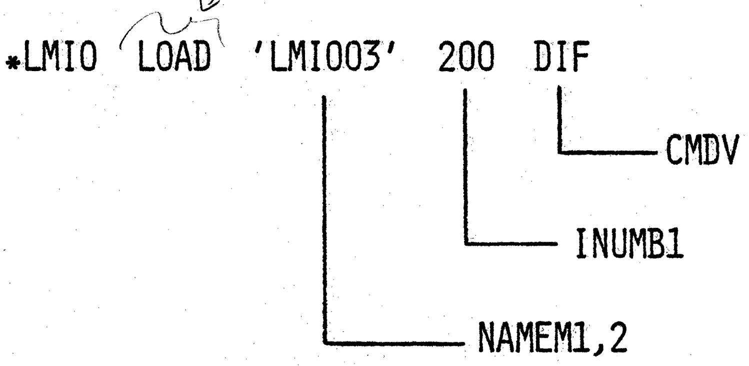
THE TYPES OF PARAMETERS WHICH CAN BE SAVED ARE AS FOLLOWS:

- 2 NUMBER FIELDS
  OCTAL NUMBERS IN ONUMB1 AND ONUMB2
  DECIMAL NUMBERS IN INUMB1 AND INUMB2
  FLOATING PT. NUMBERS IN NUMB1 AND NUMB2
- 1 BINARY NUMBER (I.E. 101101*) IN BINARY
- 3 STRING NAMES (E.G. 'LMI003') IN NAMEM1-6
- 6 IDENTIFIER-NUMBER FIELDS (E.G. TG4) IN SPNUM1-6
- PERIPHERAL DEVICE TYPE CODE (E.G. LP) IN CMDV
- STATION NUMBER IN STATC

BACKGROUND PARAMETER PASSING EXAMPLES

7. stored were
is not name of LMIO program

5

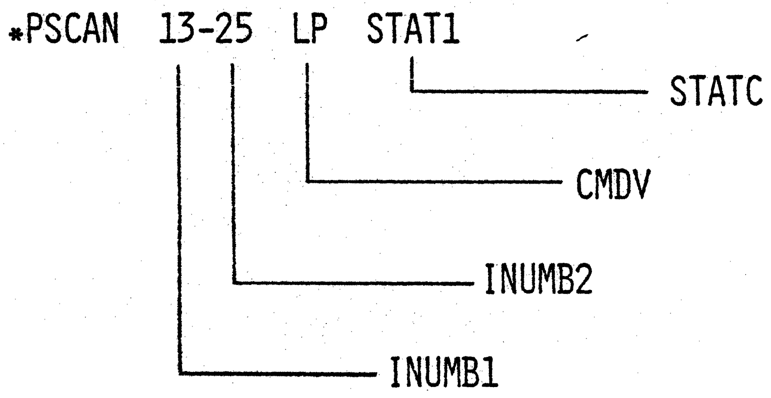


*LMIO LOAD 'LMI003' 200 DIF

CMDV

INUMB1

NAMEM1,2



*PSCAN 13-25 LP STAT1

STATC

CMDV

INUMB2

INUMB1

PERIPHERAL I/O USING IOCS

OVERVIEW OF IOCS

SINCE MASTR IS DESIGNED AS A MULTI-USER OPERATING SYSTEM, CARE SOULD BE TAKEN BY THE ASSEMBLY LANGUAGE PROGRAMMER TO AVOID AFFECTING OTHER USER PROGRAMS OR THE OPERATING SYSTEM DURING THE TIME THE ASSEMBLY LANGUAGE PROGRAM IS EXECUTING. THE MOST COMMON AREA TO BE AFFECTED BY IMPROPER PROGRAMMING PRACTICES IS THAT OF PERIPHERAL I/O.

MASTR ACCEPTS ALL I/O REQUESTS FROM THE OPERATOR COMMANDS AND FACTOR PROGRAMS AND SCHEDULES THEM FOR EXECUTION IN A MANNER WHICH WILL NOT CREATE CONFLICTS. ASSEMBLY LANGUAGE PROGRAMS, ON THE OTHER HAND, HAVE DIRECT ACCESS TO THE CPU THEREBY ALLOWING THEM TO BYPASS NORMAL MASTR PROTOCOL AND COURTESIES. THIS PRACTICE, HOWEVER, CAN LEAD TO GREAT DISTRESS ON THE PART OF OTHER SYSTEM USERS AND SHOULD BE DISCOURAGED.

MASTR USES THE ROUTINE $IOCS TO PERFORM ALL I/O IN THE FOLLOWING GENERAL MANNER:

- o OPEN DEVICE

  CREATE 13 WORD ENTRY FOR DEVICE IN "IOATAB". RETURN "IOATAB" ADDRESS IN XR6. WILL FLAG ERROR IF DEVICE NOT AVAILABLE.

- o PERFORM I/O

  USE VALUE IN XR6 AS POINTER TO "IOATAB" AND DRIVE DEVICE AS REQUESTED. FLAGS ERROR IF DEVICE GOES OFF LINE.

- o CLOSE DEVICE

  CLEARS ENTRY IN "IOATAB" POINTED TO BY XR6.

IOCS OPEN PROCEDURE

```
$IOCS    EQU    645B
ERRCNV   EQU    715B
         .
         .

         LDX    XR1,OPNDCB
RETRY    BSM*   $IOCS
         BRU    IOERR            ERROR RETURN
         STX    XR6,IOADR        NORMAL RETURN
         .
         .

IOERR    EQU    *                PROCESS DEVICE OPEN ERROR
         CAM    D7               BUSY?
         BE     RETRY
         BSM*   ERRCNV           DISPLAY ERROR MSG
         NOP    Ø
         BRU    ABORT
         .
         .
         .

OPNDCB   DATA   (CODE)
         DATA   Ø                OPTIONAL BLOCK BUFFER ADDR
         DATA   Ø                OPTIONAL BLOCK BUFFER SIZE
         DATA   Ø                OPTIONAL JOB ID
         DATA   Ø,Ø              OPTIONAL FILE NAME
```

```
CODE:   MT1 RD   40100000B     DISC RD   40240000B
        MT1 WR   00100000B     DISC WR   00240000B
        VKT RD   40040000B     MEM  RD   40400000B
        VKT WR   00040000B     MEM  WR   00400000B
        LP       00200000B     CR        40200000B
```

IOATAB (INPUT/OUTPUT ACTIVITY TABLE)

13 WORDS/ENTRY, 20 ENTRIES/TABLE
260 WORDS
First 6 Entries are reserved by the system

23 21 18 15 12 9 6 3 0

FDSTAT	0	BF	DM	EF	BK			CM	LM	BM	STAT	WS		IF	DEVICE CODE
FNAME1	1	FIRST 4 CHARACTERS OF FILE NAME													
FNAME2	2	LAST 2 CHAR OF FILE	0	FILE TYPE											
FJOB	3	JOB NUMBER													
FDADDR	4	STARTING DISC ADDRESS													
FSIZE	5	FILE SIZE													
FCBBP	6	0	CURRENT BLOCK ADDRESS												
FBBADR	7	0	BLOCK BUFFER ADDRESS												
FBBSIZ	8		BLOCK BUFFER SIZE												
FCDADR	9		ERROR CODE												
FCSIZE	10		CURRENT FILE SIZE												
FRSIZE	11	0	RECORD SIZE												
	12	0													

ERROR CODE RETURNED FROM IOCS OPEN

ERROR CODE	DESCRIPTION
2	An invalid device code specified or device not available (no interface)
3	An invalid function code specified
4	Parity error during disc directory Read or OPEN call to DP system
5	File not found for DIF, DOF, MIF, MOF
7	Device busy
9	Insufficient block size or memory space
10	No write ring on mag tape
11	CLIO error
12	I/O table overflow
14	Working storage already in use
15	MACTAB overflow for MOF

TO DISPLAY MESSAGE:

```
LDA     ERRCOD
BSM*    ERRCNV
NOP     Ø
```

IOCS READ/WRITE OPERATION

```
         .
         .
         LDX*   XR6,IOADR          IOATAB ADDR FROM OPEN
         LDX    XR1,IODCB          3 WORD DCB POINTER
         BSM*   $IOCS
         BRU    IOERR              SAME AS FOR OPEN
         .                         NORMAL RETURN
         .
         .



         .
         .
         .
IODCB    DATA  {41000000B          READ
               {01000000B          WRITE
         DATA   BUFADR
         DATA   BUFSIZ
```

OPTIONS FOR DCB WORD Ø:

PROMPTING CHAR (ASCII)	BITS 0-7
I/O WITH IMMEDIATE RTRN	BIT 13
ASCII CONTROL MODE	BIT 12

ERROR CODES RETURNED FROM IOCS READ/WRITE OPERATION

Error Code

1 - end of input file
2 - device off-line
4 - parity error
6 - unrecoverable hardware error
7 - device busy
8 - invalid I/O assignment address
9 - file overflow for DOF or MOF
13 - excess word count on read

For error 13, E register contains the actual number of words.read. The error 13 takes normal return rather than error return.

TO DISPLAY MESSAGE:

```
LDA     ERRCOD
BSM*    ERRCNV
NOP     Ø
```

IOCS CLOSE OPERATION

CLOSES OUT A DEVICE OR FILE BY REMOVING THE `IOATAB' ENTRY SPECIFIED BY XR6.

```
       .
       .
       .
       LDX*   XR6,IOADR
       LDX    XR1,CLDCB
       BSM*   $IOCS
       BRU    IOERR          SAME AS FOR OPEN
       .                     NORMAL RETURN
       .
CLDCB  DATA  {52000000B      READ
             {12000000B      WRITE
```

DO NOT CONFUSE WITH IOCS `TERMINATE I/O' CALL (ITEM 3.4)

A CLOSE MAY ALSO BE ACCOMPLISHED BY CLEARING IOATAB ENTRY WORD Ø TO A Ø:

```
        CLA
        STA   Ø,XR6
```

ERROR CODES RETURNED FROM CLOSE

ERROR CODE

4 = PARITY ERROR
7 = DEVICE BUSY
8 = INVALID I/O ASSIGNMENT TABLE ADDRESS

OTHER IOCS FUNCTIONS

TERMINATE I/O - IMMEDIATELY STOP I/O, OR CLEAR THE TTP SCREEN

```
DCB = (12000000B  READ
      (02000000B  WRITE
```

(SEE SECTION 3.4)

LP TOP OF FORM - TREATED AS NOP FOR OTHER DEVICES

```
DCB =  03000000B
```

UNFORMATTED WRITE - A TTP WRITE WITH NO CR/LF

```
       LDX*   XR6,IOADR
       LDX    XR1,DCB
       BSM*   $IOCS
       BRU    IOERR          SAME AS OPEN
       .
       .
       .
DCB    DATA   04000000B
       DATA   BUFADR
       DATA   BUFSIZ         BUFFER SIZE IN WORDS
```

ADDITIONAL AVAILABLE FUNCTIONS

- VARIOUS COM LINK FUNCTIONS
- DEVICE STATUS CHECK
- MAG TAPE FILE/RECORD SKIP & REWIND
- WRITE MAG TAPE 'TAPE MARK'
- TRANSMIT TTP SCREEN TO A BUFFER IN ASCII

I/O TO STATION OR SYSTEM PID/POD

IF I/O IS DESIRED TO STATION OR SYSTEM PID/POD, THERE IS NO NEED TO DO AN OPEN, AS DEVICE IS ALREADY OPENED.

SYSTEM PID/POD:

set VK1, VP1

```
        LDX*    XR6,PIDPMF/PODPMF      (440B/441B)
        LDX     XR1,DCB
        BSM*    $IOCS                  DO READ OR WRITE
```

STATION PID/POD:

use VK1, VP2

```
        LDX*    XR2,TVT                PREVIOUSLY OBTAINED TVT POINTER
        LDA     TPID/TPOD,XR2          (105/114)
        LXA     XR6
        LDX     XR1,DCB
        BSM*    $IOCS                  DO READ OR WRITE
```

PERFORM I/O TO USER'S VKT

A PROGRAM SHOULD NOT BE 'HARD WIRED' TO VKT1 OR VKT2, BUT SHOULD BE SMART ENOUGH TO DETECT TERMINAL WHICH USER IS USING FOR ERROR MESSAGES OR OTHER USER MESSAGES.

RULE: PIDPMF (GLOVAR) ALWAYS POINTS TO LAST VKT TO RECEIVE A 'CR'.

SINCE A GIVEN TTP AND TTK HAVE THE SAME DEVICE CODE, PIDPMF MAY BE USED FOR OUTPUT.

METHOD #1 - VERY EASY BUT NOT FOOL-PROOF. DO NOT CHANGE PID WHILE PROGRAM IS RUNNING; PID MUST BE VKT.

```
LDX*   XR6,PIDPMF        (440B)
LDX    XR1,DCB
BSM*   $IOCS             DO READ OR WRITE
```

PERFORM I/O TO USER'S VKT (CONT)

METHOD #2 - HARDER METHOD BUT FOOL-PROOF. PROGRAM CAN BE EXECUTED FROM DIF OR CR AND OUTPUT WILL STILL GO TO USER'S TTP. INADVERTANT PID CHANGE DURING EXECUTION WILL NOT REDIRECT OUTPUT FROM TTP.

```
LDA*   PIDPMF
AND    D17          17B
CAM    D1           IF PID IS NOT VK1
BE     *+4          OR VK2, THEN DEFAULT
CAM    D7           TO VK1, AND OPEN THE
BE     *+2          DEVICE
LDA    D1
SL     14           POSITION FOR OPEN DEVICE
OR     OPNDCB
STA    OPNDCB
LDX    XR1,OPNDCB
BSM*   $IOCS        OPEN THE TTP
.
.
.
```

ALTERNATIVE I/O METHODS

ALTERNATIVE METHODS FOR PERFORMING I/O WHICH IS SCHEDULED FOR EXECUTION BY MASTR ARE AVAILABLE IN ADDITION TO IOCS.

THESE METHODS USE THE FOLLOWING SYSTEM SUPPLIED ROUTINES WHICH THE USER ALSO HAS ACCESS TO AND ARE DESCRIBED ON FOLLOWING PAGES:

- MSGIN - READ A RECORD FROM SYSTEM PID
- MSGOUT - WRITES A RECORD TO SYSTEM VKT
- $DOIO - SIMILAR TO $IOCS PLUS ALLOWS ACCESS TO SYSTEM/STATION PID/POD AND USER'S VKT
- $RDWR - PERFORMS SAME FUNCTIONS AS $DOIO AS WELL AS AUXILLIARY DEVICE FUNCTIONS SUCH AS LP TOP-OF-FORM AND TAPE REWIND. IN ADDITION, A DEVICE OPEN IS NOT REQUIRED AS IS FOR $IOCS.

ALTERNATIVE I/O METHODS - MSGIN/MSGOUT

MSGIN - READS A RECORD FROM THE SYSTEM PID. USE IS LIMITED SINCE PID MAY CHANGE FROM ONE TTK TO THE OTHER DURING YOUR PROGRAM EXECUTION.

```
              LDX    X1,INBUF
              LDA    ZERO OR PROMPTING CHAR
              BSM*   MSGIN        (646B)
              BRU    ERROR
                                  NORMAL RETURN

     INBUF    BSS    2Ø
```

MSGOUT - WRITES A RECORD TO SYSTEM TTP. HAS SAME LIMITATIONS AS MSGIN.

```
              LDX    X2,5
              LDX    X1,OUTBUF
              BSM*   MSGOUT       (647B)
              NOP    *
                                  NORMAL RETURN
```

ALTERNATIVE I/O METHODS - $DOIO

ADVANTAGES:

- SIMPLIFIED CALLING SEQUENCE
- CONVENTIONAL DOPSY TYPE DCB
- ACCESSES ALL DEVICES PLUS SYSTEM/STATION PID/POD AND USER'S VKT

DISADVANTAGES:

- MUST STILL OPEN EACH DEVICE EXPLICITLY
- LARGE BLOCK OF CODE MUST BE INSERTED INTO USER'S PROGRAM

ALTERNATIVE I/O METHODS - $DOIO (CONT)

Calling Sequence:

To open a given I/O device or file for usage:

```
LDA   Device Code
OR    $RD/$WR
BSM   $OPNIO
---   error return
```

Device Codes

```
   0 - users VK/VP     1 - VK1/VP1
   2 - MTR1/MTW1       3 - MTR2/MTW2
   4 - CR/LP           5 - DIF/DOF
   6 - CLI/CLO         7 - VK2/VP2
   8 - MIF/MOF        11 - SYS PID/POD
  12 - STAT PID/POD  (13B)
(14B)
```

For disc or memory file I/O the file name must be placed where indicated following the label "$OPDCB".

The labels $RD (read) and $WR (write) are predefined in $DOIO.

To perform I/O to the last device opened by a call to $OPNIO:

```
LDX   X1,DCB
LDA   $RD/$WR
BSM   $DOIO
```

DCB is a conventional 2 word DCB:

```
DCB   DATA   word count, buf addr
```

To close the last device opened by a call to $OPNIO:

```
BSM   $CLSIO
```

ALTERNATIVE I/O METHODS - $RDWR

ADVANTAGES:

- NO DEVICE OPEN REQUIRED. AUTOMATICALLY OPENS AND RETAINS RECORD OF ALL OPENED DEVICES ALLOWING MANY TO BE OPENED CONCURRENTLY.
- SIMPLIFIED CALLING SEQUENCE
- CONVENTIONAL DOPSY TAPE DCB
- ACCESSES ALL DEVICES PLUS SYSTEM/STATION PID/POD AND USER'S VKT
- PERFORMS MOST AUXILLIARY DEVICE FUNCTIONS LIKE LP TOP OF FORM, TAPE REWIND

DISADVANTAGES:

- A LARGE BLOCK OF CODE MUST BE INSERTED INTO USER'S PROGRAM

ALTERNATIVE I/O METHODS - $RDWR (CONT)

Calling Sequence:

```
LDA    Device Code
BSM    $READ/$WRITE
DATA   Control Code
DATA   DCB Address
----   Error/EOF Return
----   Normal Return
```

To close all previously used devices at program termination:

```
BSM    $CLOSE
```

Device Codes

```
  0 - users VK/VP        1 - VK1/VP1
  2 - MTR1/MTW1          3 - MTR2/MTW2
  4 - CR/LP              5 - DIF/DOF
  6 - CLI/CLO            7 - VK2/VP2
  8 - MIF/MOF           11 - SYS PID/POD
 12 - STAT PID/POD    (13B)
(14B)
```

For disc or memory file I/O, the file name must be placed where indicated following the label "$ODCB".

Control Codes

```
bits 0-3     Ø  Normal I/O mode          1  LP top of form
             2  Tape rewind              3  Write tapemark
             4  Tape file skip           5  Tape record skip
             6  Kill I/O, or clear       7  VP write with no
                VP screen                   CR/LF
             8  Dev status check,
                return with A =
                status word

bits 6-8     1  ASCII control mode
             2  Immediate return, no wait
             4  Blocked disc I/O

bits 16-24      Prompting character in ASCII
                for VK read
```

DCB Address

For normal I/O:

word 1 - buffer size in words
word 2 - buffer address

PROGRAM EXECUTION IN FOREGROUND MODE

FACTOR PROGRAM CALLING FOREGROUND OVERLAY

```
SET PAGE 4096;
A = 1;
B = 2;
P1 = 123;
DCL P2[5]/.21,3.5E-3,1.23,15B,-27/;
.
.
.
EXEC NAME1 (P1,P2,A+B,1492,RSLT);
WRITE (LP) RSLT:
.
.
EXEC NAME2;
.
.
EXEC TTIME (P1, - - -);
.
.
EXEC NAME3 (P1, - - - - -,P63);
.
.
.
END:
```

FOREGROUND OVERLAY PROGRAM LAYOUT

```
        REL
*EQUATE TABLE
GLOVAR  EQU   420B
SYXVEC  EQU   640B
IERMSG  EQU   SYXVEC+30
*PROGRAM HEADER
FWORD   DATA  0
        TEXT  'NAME1'      PROGRAM NAME
        .
        .
FGENT   PZE   0            FOREGROUND ENTRY POINT
        BRU   FGSTRT       GO TO FOREGROUND START
        .
*END OF HEADER
FGSTRT  LDA*  1,4          FETCH FIRST PARAMETER
        .
        .
        .
        .
        .
        BRU*  FGENT        NORMAL FOREGROUND EXIT
*END OF FOREGROUND PROGRAM
*TERMINAL ERROR EXIT ROUTINE
TEEXIT  LDA   TENUM        LOAD TERMINAL ERROR NUMBER
        BSM*  IERMSG       ERROR EXIT THRU TERMINAL
                           ERROR ROUTINE - NO RETURN
TENUM   DATA  N            TERMINAL ERROR NUMBER
*
LWORD   EQU   *
        END
```

INDEX REGISTER USES BY FOREGROUND PROGRAM

UPON ENTRY INTO ASSEMBLY LANGUAGE OVERLAY THRU FOREGROUND ENTRY POINT (FACTOR 'EXEC' STATEMENT), THE CPU INDEX REGISTERS CONTAIN THE FOLLOWING DATA:

X0 - NOT USED

X1 - ADDRESS OF SVT TABLE FOR CURRENT TEST STATION

X2 - ADDRESS OF TVT TABLE FOR CURRENT TEST STATION

X3 - NUMBER OF PARAMETERS PASSED FROM FACTOR PROGRAM

X4 - ADDRESS (MINUS ONE) OF POINTER TO FIRST PARAMETER IN RUN TIME STACK PASSED FROM FACTOR PROGRAM

X5 - NOT USED

X6 - STATION CODE FOR CURRENT TEST STATION

0 = STAT1
1 = STAT2
2 = STAT3
3 = STAT4

X7 - NOT USED

CAUTION: THE ABOVE REGISTER CONTENTS SHOULD BE SAVED UPON ENTRY TO THE PROGRAM IN ORDER TO AVOID ACCIDENTAL LOSS WHEN USING OTHER SYSTEM ROUTINES WHICH MAKE USE OF INDEX REGISTERS. E.G. $IOCS USES:

X1 - PROGRAMMER DEFINES ADDRESS OF DCB FOR ALL $IOCS OPERATIONS

X6 - $IOCS PASSES ADDRESS OF IOATAB ENTRY DURING OPEN BACK TO PROGRAMMER IN THIS INDEX REGISTER, WHICH IS THEN USED IN ALL SUBSEQUENT EXECUTE AND CLOSE OPERATIONS

FOREGROUND PARAMETER PASSING EXAMPLE

```
ONE = 123;
DCL TWO [9]/11,12,13,14,15,16,17,18,19/;
THREE = 456;
EXEC X (ONE, TWO, THREE);
```

```
LDA* 1,4    LOADS ACCUMULATOR WITH FIRST PARAMETER FROM
            LOCATION 4000B

LDA* 2,4    LOADS ACCUMULATOR WITH ADDR OF ARRAY FROM
            LOCATION 5500B


LXA  1      LOAD ADDRESS INTO INDEX REGISTER 1

LDA  2,1    LOADS ACCUMULATOR WITH 2ND ELEMENT OF ARRAY
            FROM LOCATION 6002B

LDA* 3,4    LOADS ACCUMULATOR WITH THIRD PARAMETER FROM
            LOCATION 7000B
```

PARAMETER ADDRESS POINTERS

ADDR	CONTENTS	PARAMETER #
5500	00006000	
5003	00007000	3
5002	20005500	2
5001	0004000	1
5000		

NOTE: ALL PARAMETERS ARE
IN FLOATING POINT
FORMAT

LOCATION	CONTENTS
4000	DATA 123
6000	DATA 9 (FIXED PT)
6001	DATA 11
6002	DATA 12
6003	DATA 13
6004	DATA 14
6005	DATA 15
6006	DATA 16
6007	DATA 17
6010	DATA 18
6011	DATA 19
7000	DATA 456
X_4	5000B
X_5	6000B

CAUTION!

ALL PARAMETERS ARE IN FLOATING POINT

ALL PARAMETERS PASSED TO THE FOREGROUND OVERLAY FROM A FACTOR PROGRAM ARE IN FLOATING POINT FORMAT.

THE PARAMETERS MAY BE OPERATED ON IN FLOATING POINT BY A VARIETY OF FLOATING POINT LIBRARY ROUTINES OR THEY MAY BE CONVERTED TO FIXED POINT (BINARY) FOR CONVENTIONAL MANIPULATION.

FLOATING POINT ROUTINES

FSUB	SYXVEC+46	SUBTRACT
FAND	47	AND
FEOR	48	EXCLUSIVE OR
FLOG	49	F.P. TO LOGARITHMIC FORMAT
FADD	50	ADD
FDIV	51	DIVIDE
FFIXS	52	F.P. TO SIGNED OCTAL FORMAT
FOR	53	INCLUSIVE OR
FNOT	54	NOT
FEXP	55	CALCULATES 2^N OF SIGNED F.P.EXPONENT
FMUL	56	MULTIPLY
FFIX	57	F.P. TO OCTAL INTEGER
FFLT	58	OCTAL TO F.P.
FFLTS	59	OCTAL*10^N TO F.P.
FCAM	60	COMPARE ACCUMULATOR WITH EXTENSION

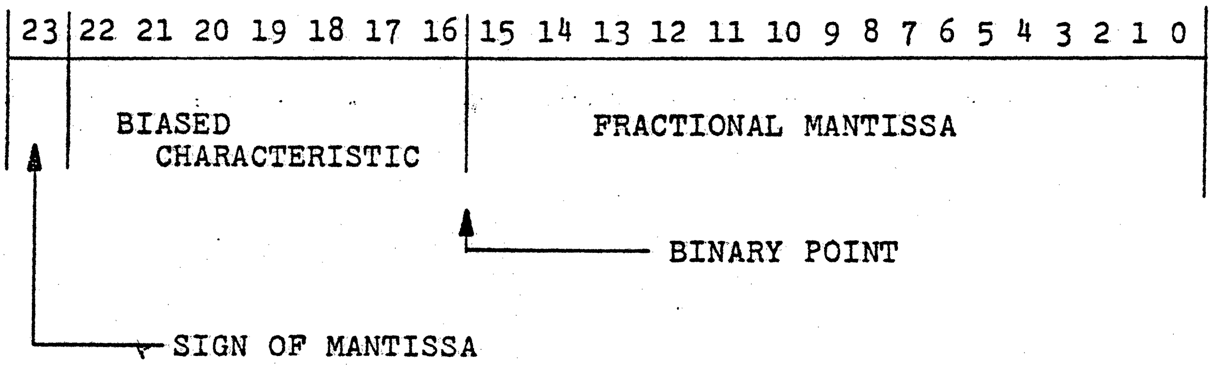
FOREGROUND OVERLAY FETCHING PARAMETERS FROM FACTOR PROGRAM

```
          REL
SYXVEC    EQU    64ØB
FFIX      EQU    SYXVEC+57
FFLT      EQU    SYXVEC+58
FWORD     DATA   Ø
          {

FGSTRT    LDA    1,4       LOAD PARAMETER ADDRESS POINTER
          CAM    B22       CHECK FOR ARRAY FLAG
          BBC    ARRAY     BRANCH IF IT WAS ARRAY
          LDA*   1,4       WAS A SINGLE VARIABLE SO
                             FETCH PARAMETER
          BSM*   FFIX      CONVERT TO FIXED POINT
          STA    TEMP1
          BRU    MAIN
ARRAY     LDA*   1,4       LOAD ADDRESS OF ARRAY
          LXA    1
          LDA    Ø,1       LOAD ARRAY ELEMENT 'Ø' (SIZ
          STA    SIZE        AND STORE AWAY
          LAX    1
          ADD    SIZE
          LXA    Ø         ADDRESS OF LAST ELEMENT
FETCH     LDA    1,1       FETCH PARAMETER
          BSM*   FFIX      CONVERT TO FIXED POINT
          STA    ~~~~
          ATX    1,1       CONTINUE IN LOOP UNTIL LAST
          BLE    FETCH       ELEMENT IS FETCHED
MAIN      {

          BRU*   FGENT     EXIT THRU FOREGROUND ENTRY
SIZE      DATA   Ø
          {
LWORD     EQU    *
          END
```

RETURNING PARAMETERS TO RUN TIME STACK

```
           REL
SYXVEC     EQU    64ØB
FFIX       EQU    SYXVEC+57
FFLT       EQU    SYXVEC+58
FWORD      DATA   Ø
           }

GETIT      LDA*   1,4       FETCH FIRST PARAMETER
           BSM*   FFIX    - CONVERT TO FIXED POINT
           STA    P1
           }


           LDA    P1
           ADD    PIE       CHANGE VALUE OF P1
           BSM*   FFLT      CONVERT TO FLOATING POINT
           STA*   1,4       STICK IT BACK IN THE STACK
           }

           BRU*   FGENT     EXIT THRU FOREGROUND ENTRY
P1         DATA   Ø
PIE        DATA   31414
           }
LWORD      EQU    *
           END
```

24-BIT FLOATING POINT WORD FORMAT FOR THE FST-2 COMPUTER



BIT 23 SIGN OF MANTISSA

0 = positive mantissa
1 = negative mantissa (i.e. two's compliment of positive)

BITS 22 - 16 BIASED CHARACTERISTIC (biased by 100_8)

$100_8 = 0$ characteristic

$177_8 = 63_{10}$ characteristic

$000_8 = -64_{10}$ characteristic

BITS 15 - 0 FRACTIONAL MANTISSA

Bit 15 = .50 Bit 14 = .25 Bit 13 = .125 etc.

EXAMPLES

The floating point representation of $+23.5_{10} = 21336000_8$

$= 0\ 1000101\ 1011110000000000_2$

Conversion process:

$23.5_{10} = 27.4_8 = 10111.1_2 = 0.101111_2 * 2^5$

Sign = 0

Characteristic = $105_8 = 1000101_2$ (i.e. $100_8 + 5$ from 2^5 above)

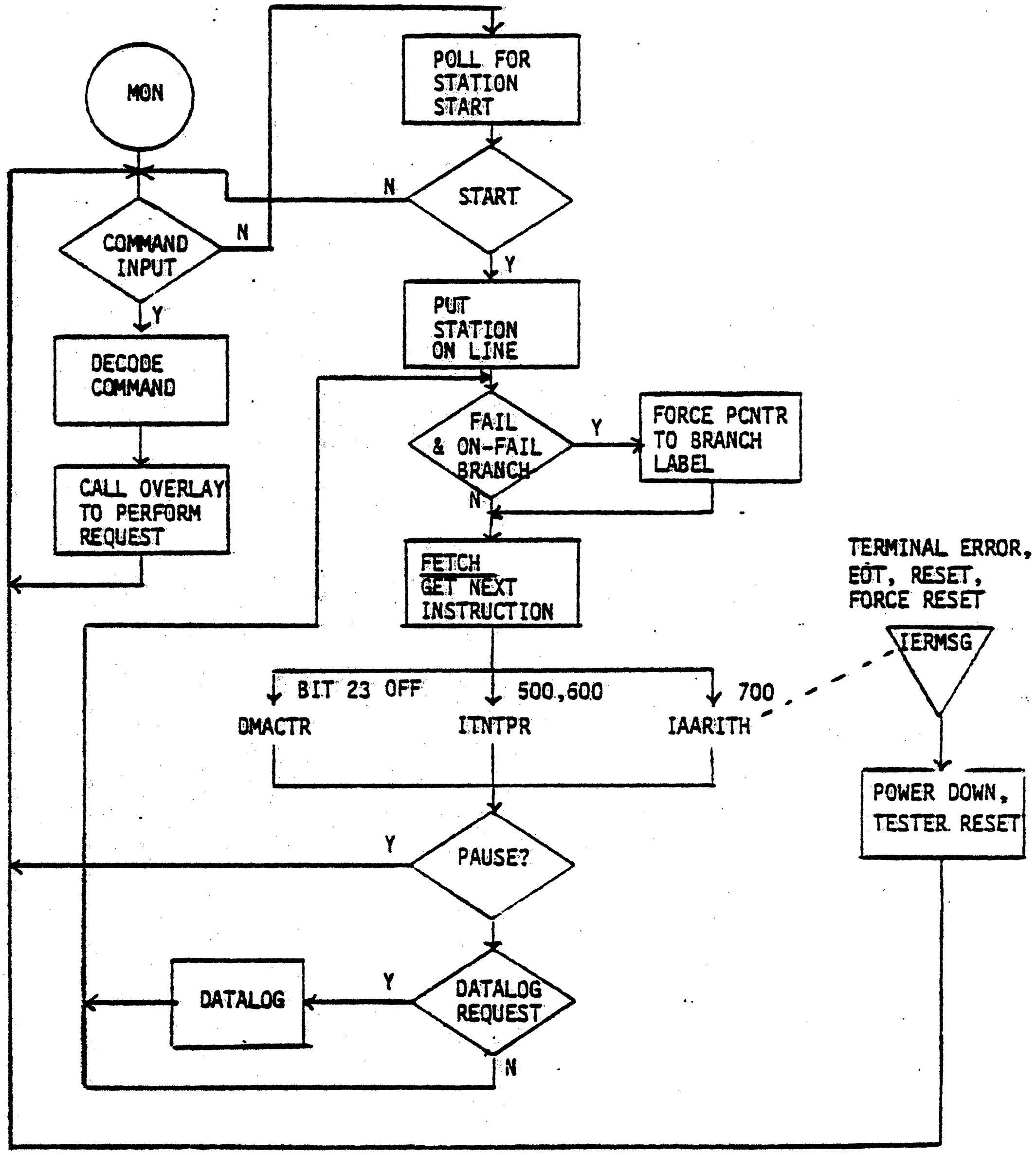
Mantissa = 1011110000000000_2 (i.e. $.101111_2$ from above left justified at bit 15)

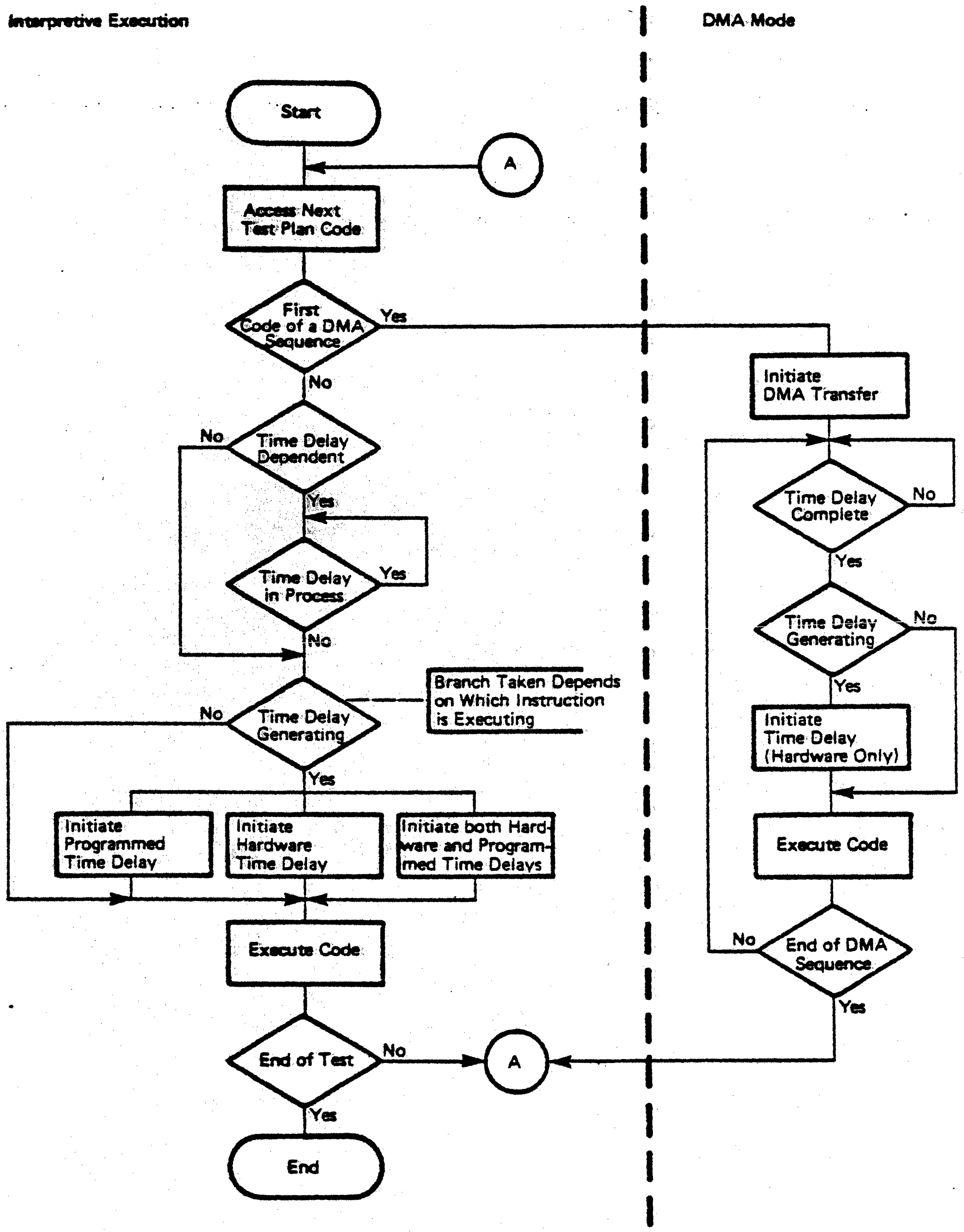
The floating point representation of $-23.5_{10} = 56442000_8$

i.e. The 2's compliment of $21336000_8 = 56442000_8$

TESTER I/O

```
****          FAIRCHILD FACTOR COMPILER                          REL:    2.1          ***
**   SOURCE FILE:      *DEMO                                     DATE:     17 JUL79   **
**   DATA   FILE:      DEMO                                      TIME:       07:50    **
**   SYSTEM CONF:      S8         MV2                            COND:     00000000   **
  *000000        00000000
  *000001        44455557
  *000002        00000000
  *000003        00000076
  *000004        00000000
  *000005        00000000
  *000006        00000764
  *000007        00000000
  *000010        00000000
  *000011        00000000
  *000012        00000000
  *000013        00000000
  *000014        00000000
  *000015        00000000
  *000016        00000000
  *000017        00000000
  *000020        00000000
  *000021        00000000
  *000022        50000000
  *000023        00000000
  *000024        00000000
  *000025        72000075
        1           SET PAGE 1024;
        2           SET DA (16:1);
  *000026        02077777
  *000027        22100001
        3           SET MA [17](14:1);
  *000030        24177776
        4           VCC = 5;
  *000031        71300102
  *000032        70200004
  *000033        20720000
  *000034        70200000
        5           FORCE VF1 VCC, RNG2;
  *000035        71000102
  *000036        62550000
        6           FORCE VF1 5, RNG2;
  *000037        62510764
        7        0  SET F (15:10);
  *000040        06052525
  *000041        06125252
  *000042        06200000
  *000043        06300000
  *000044        06400000
  *000045        06500000
  *000046        06600000
  *000047        26700000
        8        1  SET F (15:01);
  *000050        06025252
  *000051        26152525
        9           ENABLE TEST;
  *000052        17000000
  *000053        17066001
  *000054        17030000
  *000055        37300101
       10           END;
  *000056        71600000
   0  COMPILATION ERRORS,        10  STATEMENTS
```


MON
COMMAND INPUT
N
Y
DECODE COMMAND
CALL OVERLAY TO PERFORM REQUEST
POLL FOR STATION START
START
N
Y
PUT STATION ON LINE
FAIL & ON-FAIL BRANCH
Y
N
FORCE PCNTR TO BRANCH LABEL
FETCH
GET NEXT INSTRUCTION
BIT 23 OFF
500,600
700
DMACTR
ITNTPR
IAARITH
TERMINAL ERROR, EOT, RESET, FORCE RESET
IERMSG
POWER DOWN, TESTER RESET
PAUSE?
Y
DATALOG REQUEST
Y
N
DATALOG



Test Plan Processing, Simplified Diagram

Compiler Generated Tester Opcodes-DMA and Interpretive

DMA Opcodes

```
00000000                 SFT R7
00000000,A               SFT XOR
01000000                 SFT STROBE/ST
02000000                 SFT DA
02000000,A               SFT CRO
03000000                 SET DB
04000000                 SET MA
05000000                 SFT MB
06000000                 SFT F

06710004      S          LSFT IX
06730001      S          LSET STROBE/ST
06730002      S          LSFT RZ
06730004      S          LSET INVERT/I
06730005      S          LCGEN TG1
06730006      S          LCGEN TG2
06730007      S          LCGEN TG3
06730010      S          LSET DB
06730011      S          LSET DA
06730012      S          LSET MB
06730013      S          LSET MA
06740000      S          SFT F-LCALL
06750000      S          LSET IX-LCALL
06760000      S          SET F-LGOTO
06770000      S          SET F-LEND
07700000      S          LSUBR NORMAL
07710000      S          LSUBR MATCH
07720000      S          LSUBR CONTINUOUS
07740000      S          SET FC NORMAL
07750000      S          SET FC MATCH
07760000      S          SET FC CONTINUOUS

10000000                 SET S
11000000                 CGEN TGX
12000000                 SET INVERT
13000000                 CGEN TGX
14000000                 SFT R
15000000                 CGEN TGX
16000000                 CPMU/XPMU PIN
16040000                 DISABLE RELAY
16040400                 ENABLE RELAY
16400000                 FORCF CURRENT
16420000                 FORCE VOLTAGE
16500000                 SET CLAMP
16502000                 SFT DCT (PSL REGISTER)
17000000                 SFT START/ENABLE TEST
17010000                 SET MINOR (WORD 1, LOOP COUNT - M)
17020000                 SET MAJOR (WORD 1, LOOP COUNT - N)
17030000                 AT/ENABLE TEST
17040000                 SET MINOR (WORD 2, LOOP START ADDRESS - J)
17050000                 SET MINOR (WORD 3, LOOP END ADDRESS - K)
17060000                 SET MAJOR (WORD 2, END ADDRESS - L)/ENABLE TEST
17100000                 SFT DCT
17140000                 MEASURE PIN (DCT REGISTER)
17200000                 SET IOMODF/IOM3 (PIN LIST)
17300000                 ENABLE TEST, ENABLE PPM
17300200                 MFASURE (2ND WORD) (SAMA REGISTER)
17360000                 ENABLE/DISABLE SPLIT/RTO/MUX/IMASK
17374000                 (ENABLE ALTERNATE BANK)
17400000      P          RD/WR ONE/ZERO
17400100      P          RD/WR CHECK/NCHECK
17400200      P          BRANCH UNLESS
17400300      P          BRANCH TO/RESFT
```

```
17410000        P          PGEN LOAD/START/ENABLE/DISABLE PGEN
17420000       .P          SET PGEN1 (WORD2)
17430000        P          SET PGEN1 (WORD 3)
17440000        P          SET PGENA
17443200        P          SET PGEN1
17443400        P          SET PGEND
17443600        P          SET PGENDN
17444000        P          PIN CONFIGURE (WORD1)
17444400        P          PIN CONFIGURE (WORD2)
17443600        P          SET PGENCN
17447600        P          SET PGENC
17700000                   XCON PIN
17700400                   CONN TCOM
17701000                   CONN CLK
17701400                   CONN DPS2
17702400                   CONN DPS1
17703400                   CONN DPS3


S         SPM
P         HARDWARE PATTERN GENERATOR
```

Interpretive Opcodes

```
50000000                NO SET PAGE
50100000                SET PAGE
50240000                SET START
50340000                AT
50440000                SET MAJOR (WORD 2, END ADDRESS - L)
50540000                SET MINOR (WORD 2, LOOP START ADDRESS - J)
50560000                SET MINOR (WORD 3, LOOP END ADDRESS - K)
50640000                SET MAJOR (WORD 1, LOOP COUNT N)
50660000                SET MINOR (WORD 1, LOOP COUNT M)
50740000                SET PERIOD
50760000                SET APERIOD
51040000                SET TG1 DELAY/SET TG1 WIDTH
51140000                SET TG2
51240000                SET TG3
51340000                SET TG4
51440000                SET TG5
51540000                SET TG6
51640000                SET TG7
51740000                SET TG8

52000000                N/A
52100000                N/A
52200000                N/A
52340000                SET ATG4
52400000                N/A
52500000                N/A
52600000                N/A
52700000                N/A

53014000                SET FI
53020000                SET SI
53100000                ENABLE TEST
   04000                    AMATCH
   00004                    MOMENTARY
   00002                    MATCH
   00010                    LOOP
   20000                    CONTINUOUS
   00020                    EXT
   02020                    AEXT
   00400                    IFAIL
53200000                ENABLE PPM
53300000                SET Q
53400000                SET PPM ON
53420000                SET PPM OFF
53504000                DISABLE DOUBLE STROBE
53504020                ENABLE DOUBLE STROBE
53500000                SET IOMODE
53520000                SET IOM3
53600000                SET MPIN
53700000                SET IFAIL
54000000                SET CHAIN OFF
     040                    TWO
     100                    FOUR
```

```
60100000                    ENABLE TRIP1/TRIPI1
60200000                    ENABLE TRIP2/TRIPI2
60300000                    ENABLE TRIP3/TRIPI3
60400000                    N/A
60500000                    FORCE CURRENT
60520000                    FORCE VOLTAGE
60600000                    N/A
60700000                    FORCE PMU
61040000                    SET DELAY (TIME OUT)
61100000                    SET DELAY, DC
61200000                    CPMU PIN/XPMU PIN
61202000                    XCON VF1/VF2/VF3
61300000                    ENABLE TRIPV1
61400000                    ENABLE TRIPV2
61500000                    ENABLE TRIPV3
61600001                    MEASURE PIN - NO SET DCT ELSE DMA
61602000                    MEASURE PIN #
61600000                    MEASURE VALUE/NODE
61644000                    MEASURE VARIABLE
61702000                    SET PMU SENSE
61720002                    SET PMU FORCEV
61700002                    SET PMU FORCEI
   00001                        AUTO, ELSE RANGE IN RANGE POSITION
62000000                    FORCE E0
62020000                    FORCE E1
62100000                    FORCE EA0
62120000                    FORCE EA1
62200000                    FORCE EB0
62220000                    FORCE EB1
62300000                    FORCE EC0
62320000                    FORCE EC1
62460000                    SET VOFFSET
62500000                    FORCE VF1
62600000                    FORCE VF2
62700000                    FORCE VF3
63000000                    SET S0
63020000                    SET S1
63100000                    DISABLE DCT0
63100001                    ENABLE DCT0 LT
63120001                    ENABLE DCT0 GT
63200000                    DISABLE DCT1
63200001                    ENABLE DCT1 LT
63220001                    ENABLE DCT1 GT
63300001                    ENABLE ILO LT
63320001                    ENABLE ILO GT
63400001                    ENABLE IHI LT
63420001                    ENABLE IHI GT
63500001                    ENABLE VLO LT
63520001                    ENABLE VLO GT
63600001                    ENABLE VHI LT
63620001                    ENABLE VHI GT
63700000                    N/A
64000003                    FORCE RESET
64000020                    SET LOGIC NEGATIVE
64000040                    ENABLE LATCHES
64100020                    SET LOGIC POSITIVE
64100040                    DISABLE LATCHES
64200000                    FORCE DELAY
64200001                    FORCE WAIT
64300000                    N/A
64400000                    ON DCT
```

```
64400001                        ON FCT
64400002                        ON TRIP
64400003                        ON RESET
64400000                        CLEAR DCT
64400001                        CLEAR FCT
64400002                        CLEAR TRIP
64400003                        CLEAR RESET
64500000                        LIN
64600400                        DISABLE TRIP
64700000                        FORCE IF1
65000000                        FORCE IF2
65100000                        FORCE IF3
65200000                        SET DCT
65300000                        CLEAR FAIL
65400000                        SET SA0
65420000                        SET SA1
65500000                        SET TEST #
```

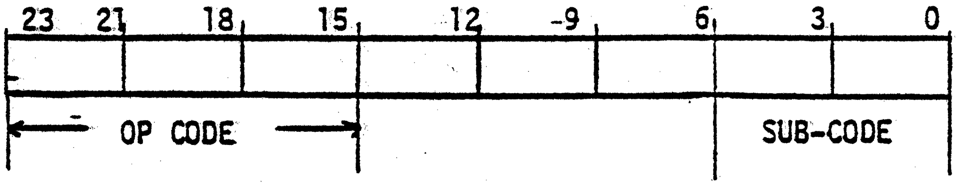
J.3 Arithmetic Opcodes

70000000	ADD	
70000001	SUB	
70000002	MUL	
70000003	DIV	
70000004	EXP	
70000005	AND	
70000006	OR	
70000007	EOR	
70000010	NOT	
70000011	NEG	
70100002	LT	
70100004	EQ	
70100006	LEQ	
70100010	GT	
70100012	NEQ	
70100014	GE	
70200000	STR	STORE
70200001	CF	CALL PROCEDURE WITH PARAMETERS
70200002	PAE	PARAMETER ARRAY ELEMENTS
70200003	PEX	PARAMETER EXPRESSION STACK TOP
70200004	STKC	STACK FLOATING POINT CONSTANT
70200005	STKCI	STACK INTEGER CONSTANT
70200006	INDR	CALCULATE INDEX RESULT
70200007	INDA	CALCULATE INDEX ADDRESS
70200010	PAUSE	PAUSE STATEMENT
70200011	CIO	COMPLETE IO STATEMENT (END OF READ/WRITE)
70200012	FOR	FOR STATEMENT
70200013	DO	DO STATEMENT
70200014	EAS	END OF DCL STATEMENT
70200015	COL	COLUMN FORMAT FOR WRITE
70200016	LIT	LITERAL VARIABLE FOR READ/WRITE
70300000	DCLV	DECLARE ARRAY
70400000	CPN	CHECK PARAMETER COUNT
70500000	SIO	START IO
70600000	INV	INITIALIZE ARRAY
70700000	INS	INITIALIZE SCALAR
71000000	STKV	STACK VALUE
71100000	STKFV	STACK PARAMETER VALUE
71200000	STKAA	STACK ARRAY ADDRESS
71300000	STKA	STACK VARIABLE ADDRESS
71400000	STKFA	STACK PARAMETER ADDRESS
71500000	GOTO	GOTO STATEMENT
71600000	BE	BLOCK END
71700000	TEXT	TEXT FOR WRITE
72000000	BB	BLOCK BEGIN
72100000	PB	PROCEDURE BEGIN
72200000	PE	PROCEDURE END
72300000	PS	PARAMETER IS SCALAR
72400000	PA	PARAMETER IS ARRAY
72500000	PFP	PARAMETER IS PARAMETER
72600000	PFE	FUNCT PROCEDURE END
72700000	EXPEX	EXEC PARAMETER ON STACK TOP
73000000	ONDIF	ON DIFEOF
73100000	RDIF	RESET FDIF
73200000	GOTOI	INDEXED GOTO STATEMENT
73300000	EXEC	EXEC ALLINK PROGRAM
73400000	REXEC	EXEC MICROPROGRAM
74000000	IFJ	IF FALSE, JUMP
75000000	JMP	UNCONDITIONAL JUMP
76000000	TRAF7	TRANSFER TO SUBROUTINE (NO PARAMETERS)
77000000	TRAF	TRANSFER TO FUNCTION/SUBROUTINE
77777776	PAGE	DISC ACCESS

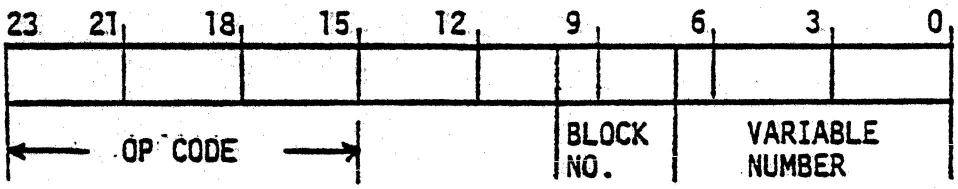
Arithmetic & Control

Arithmetic and Control opcodes are 700 and above. There are three basic formats:

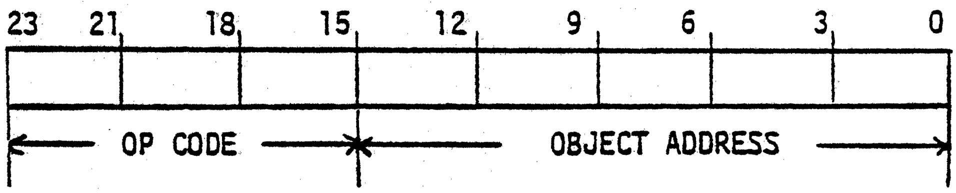
Arithmetic, Logical, Stack reference



Variable reference

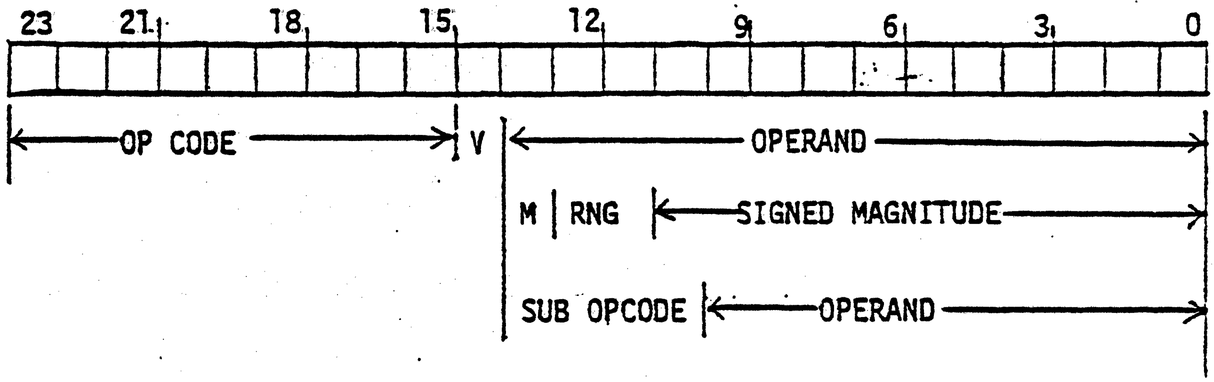


Control



Interpreter

FACTOR opcodes 700 and above are used by the arithmetic interpreter. Opcodes 500-677 are used for interpretive control of tester functions. One interpretive FACTOR statement may generate one or more words of data. The basic format of at least the first word is:



V is the stack indicator.

V = 1 - the operand required is in floating point format at the
        top of the working stack.

V = 0 - the operand required is in the low order bits.

M is the mode indicator.

M = 0 -	Voltage force	or	GT.
M = 1 -	Current force	or	LT.

RNG   0 - 4 depending on hardware.

EXAMPLE OF FACTOR LANGUAGE COMPILER OPERATION

THE FOLLOWING GENERAL STEPS ARE PERFORMED WHEN COMPILING THE FACTOR STATEMENT:

```
ENABLE TRIP1 GT X, RNG3;
```

1. GET RECORD FROM DISC BUFFER
2. SCAN 'ENABLE'
3. GO TO TESTER STATEMENT ANALYSIS SECTION OF COMPILER
4. SCAN 'TRIP1' AND SET SKELETON OPCODE FOR TOPSY INTERPRETER
5. SCAN 'GT' AND INSERT 'GREATER THAN' FLAG BIT IN SKELETON CODE
6. SCAN 'X' THEN GO TO COMPILER AND GENERATE ONE OR MORE WORDS OF CODE TO DEFINE VALUE OF 'X' AT EXECUTION TIME
7. SET FLAG IN OPCODE TO TELL INTERPRETER TO SCALE FLOATED VALUE OF 'X'
8. SCAN ',RNG3' AND SET RANGE BITS IN OPCODE
9. GENERATE COMPOSITE CODE FOR STATEMENT

EXAMPLE OF INTERPRETER OPERATION WHEN EXECUTING A COMPILED FACTOR LANGUAGE STATEMENT

THE FOLLOWING GENERAL STEPS ARE PERFORMED WHEN EXECUTING THE INTERPRETATIVE CODE GENERATED BY THE FACTOR COMPILER FOR THE FACTOR STATEMENT:

```
ENABLE TRIP1 GT X, RNG3;
```

1. FETCH THE INSTRUCTION "DEFINE X" FROM THE DISC BUFFER (WHICH RESIDES IN CORE MEMORY).

2. CALL IAARITH TO SET THE VALUE OF "X" ON THE IAARITH WORK STACK (THE VALUE IS IN FLOATING POINT FORMAT).

3. FETCH THE INSTRUCTION "ENABLE TRIP1 GT X, RNG3;" FROM THE DISC BUFFER THEN GO TO THE HARDWARE INTERPRETER.

4. CONVERT THE FLOATING POINT VALUE OF "X" (WHICH IS STORED ON THE IAARITH WORK STACK) TO HARDWARE FORMAT FOR DPS1..

5. WRITE THE HARDWARE VALUE INTO DPS1.

6. INCREMENT THE PROGRAM STATEMENT NUMBER.

FACTOR Tester Object Codes

FACTOR statements which control the tester can be either interpretive or directly executable by the hardware ie. DMA. The codes generated to control short register functions are always interpretive. The codes generated to control long register functions can be either interpretive or DMA. Interpretive codes can be distinguished from DMA codes by the fact that B23 is always a 0 in DMA code words and B23 is always a 1 in the first word of every interpretive statement sequence.

Long Registers

DMA object words generated by the compiler are in a format which is directly loadable into the hardware registers. Interpretive codes are not directly loadable into hardware registers.

NOTE: WRITE IN DMA OR NON-DMA
READ IN NON-DMA ONLY

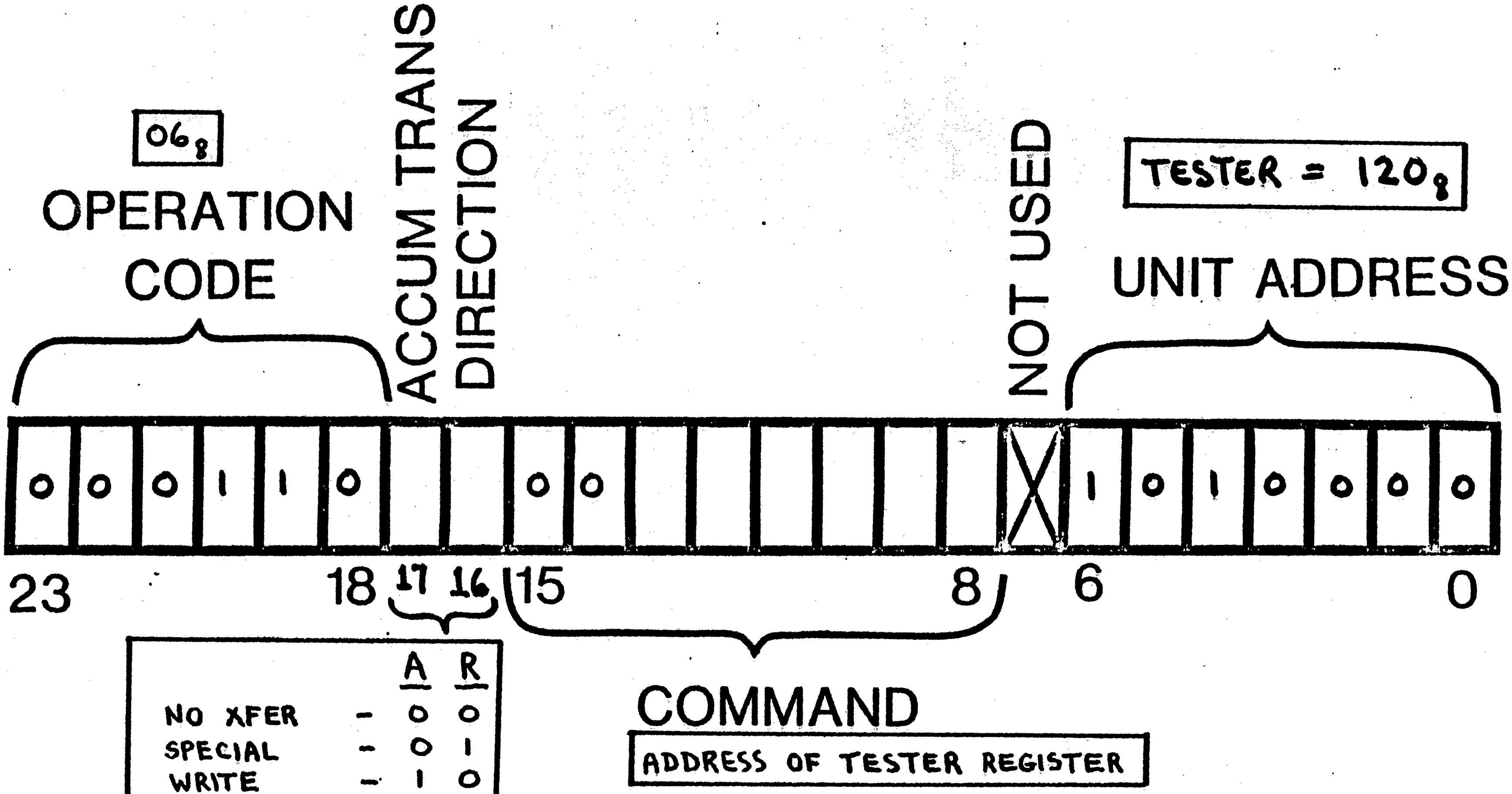
Short Registers

All short registers are read or written via the I/O SPU instruction (the I register is a special case). The word to be written to the short register is loaded in the accumulator and the appropriate SPU instruction is executed.

NOTE: WRITE AND READ IN NON-DMA MODE ONLY

24 BIT WORD CONFIGURATION

SPU INSTRUCTIONS



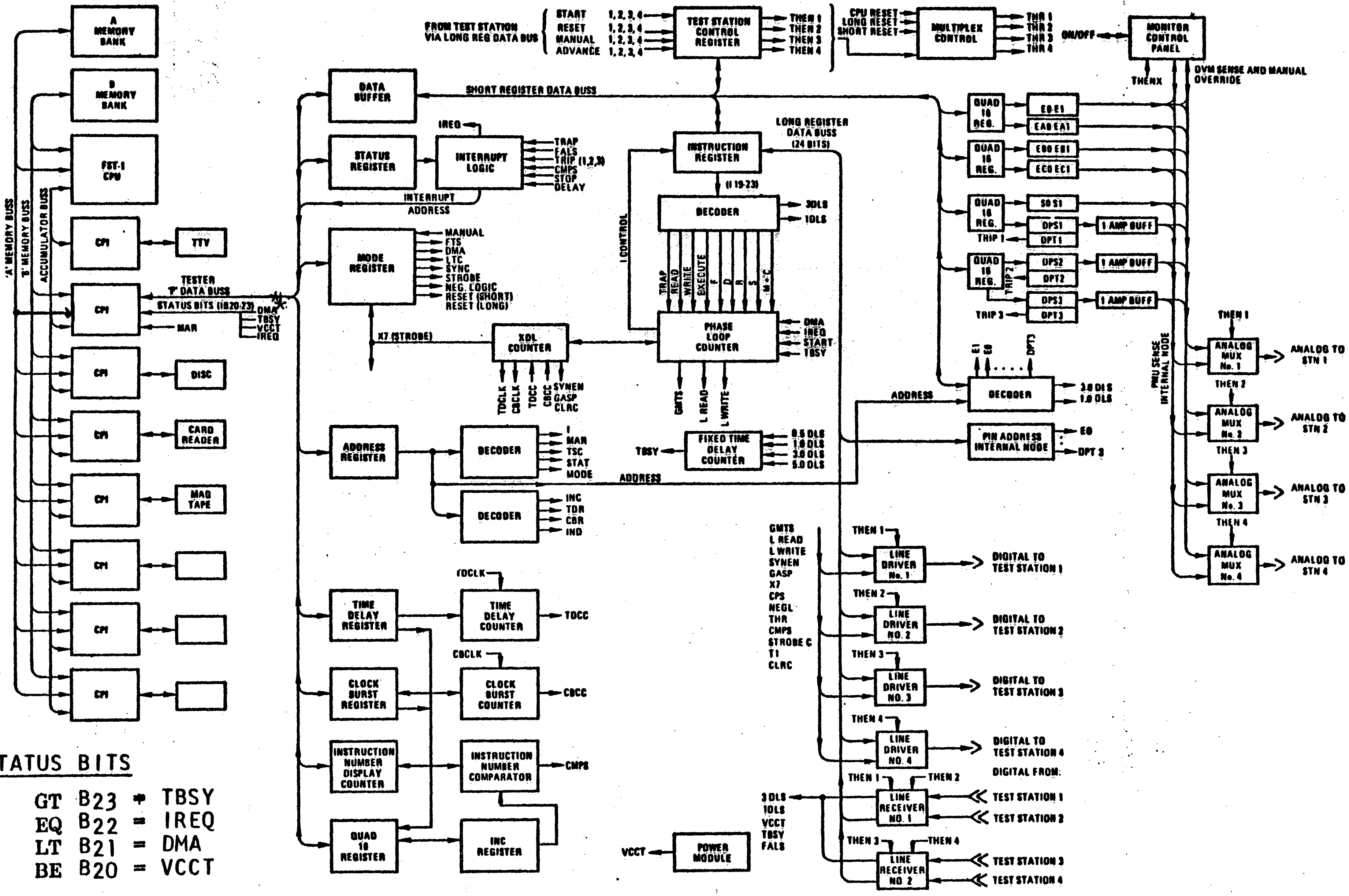
		A	R
NO XFER	–	0	0
SPECIAL	–	0	1
WRITE	–	1	0
READ	–	1	1

```
               11   11
               32   109  8
000  110  000  0XX  XXX  X01  010  000
```

REGISTER
ADDRESS

000	110	000	0XX	XXX	X01	010	000	XXX
MODE			00	000	1	=		06*00520
STATUS			00	001	0	=		06*01120
INSTRUCTION			00	001	1	=		06*01520
MAR			00	010	0	=		06*02120
TSC			00	010	1	=		06*02520
TDC			00	100	1	=		06*04520
INC			00	110	0	=		06*06120
DPS1			01	000	1	=		06*10520
DPS2			01	001	0	=		06*11120
DPT3			01	001	1	=		06*11520
DPS3			01	010	0	=		06*12120
DPT2			01	010	1	=		06*12520
DPT1			01	011	0	=		06*13120
E1			01	101	0	=		06*15120
EØ			01	101	1	=		06*15520
S1			01	110	0	=		06*16120
SØ			01	110	1	=		06*16520
EA1			01	111	0	=		06*17120
EAØ			01	111	1	=		06*17520
EB1			10	001	0	=		06*21120
EBØ			10	001	1	=		06*21520
EC1			10	010	0	=		06*22120
ECØ			10	010	1	=		06*22520

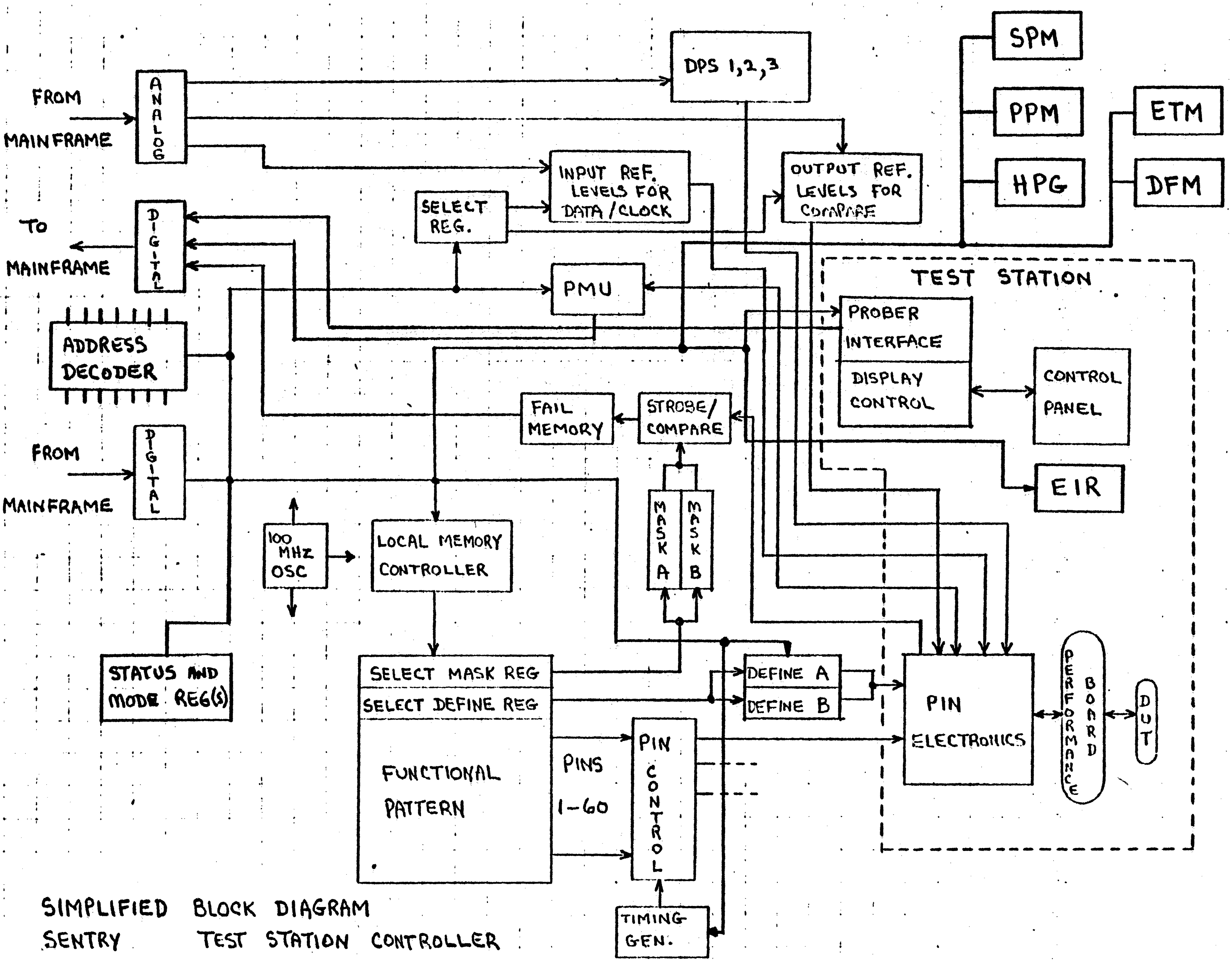
* 0 = NO XFER
2 = SPCL
4 = WRITE
6 = READ



*STATUS BITS

GT	B_{23}	=	TBSY
EQ	B_{22}	=	IREQ
LT	B_{21}	=	DMA
BE	B_{20}	=	VCCT

SIMPLIFIED BLOCK DIAGRAM
SENTRY MAINFRAME
TRAINING DEPT



SIMPLIFIED BLOCK DIAGRAM
SENTRY TEST STATION CONTROLLER

CATEGORIES OF TEST SYSTEM HARDWARE REGISTERS

The Sentry test system hardware registers are divided into two general categories: "short" registers, which are physically located in the system mainframe and are accessed via the peripheral data bus or short register data bus, and "long" registers, which are normally located in either the system mainframe or one of the test stations and are accessed via one of the long register data busses. Simplified block diagrams on the following pages should be helpful in showing the relationship between the various registers in the system; to a limited extent, they should also be helpful in illustrating the basic data flow paths between the computer and the test head.

TESTER I/O METHODS - OVERVIEW

Two different I/O methods, DMA and non-DMA, are used to transmit digital data between the computer and the test system hardware registers. The actual method used largely depends upon whether the addressed register is a "short" or "long" register and whether the data transfer is a "read" or a "write" operation.

All "short" registers are read or written using "non-DMA" I/O, where one 24-bit data word is read or written using one computer SPU instruction. When in the "write" mode, the 24-bit data word to be written must have been previously loaded into the computer Accumulator using a computer LDA instruction before the desired SPU instruction is executed. When in the "read" mode, the contents of the addressed register is transferred to the computer Accumulator as a result of executing the SPU instruction to "read". A more detailed discussion of addressing short registers along with a diagram showing the general format of the SPU instruction can be found elsewhere in this description.

All "long" registers are accessed through the Instruction Register which is itself a short register. The method of data transfer, however, can be either DMA or non-DMA depending upon whether the register addressed is being "read" or "written"; that is, a "read" is always performed using non-DMA while a "write" can be non-DMA or DMA, according to the programmer's preference at the time. In a DMA transfer, a "block" of data consisting of one or more 24-bit words is transferred from the computer memory to any number of registers on the long register data bus as the result of a single I/O instruction. A more detailed discussion of addressing long registers along with a diagram showing the general formats of the data words can be found elsewhere in this description.

ADDRESSING SHORT REGISTERS

Each short register can be addressed by a unique computer SPU instruction which will accomplish one of three possible operations: READ, WRITE, or SPECIAL. The READ and WRITE operations involve the transfer of data between the computer Accumulator and a specified test system "short" register. The data transfer is accomplished using a non-DMA I/O mode where one 24-bit data word is read or written as the result of executing one SPU instruction. The SPECIAL operation does not involve a data transfer but does cause a single specific hardware operation to be performed which is unique to the register being addressed by the SPU instruction. This operation is also performed using the non-DMA I/O mode.

The SPU instruction word format used when addressing a short register is shown on page 8-32. A description of the three short register operations is as follows:

READ

Up to 24 bits of data are "read" from the addressed register and transferred to the computer Accumulator for further processing as defined by the programmer. To accomplish the "read" operation, simply execute an SPU instruction which is coded according to the instruction word format shown on page 8-32.

For example, to read the "status" register:

```
RDSTAT  DATA  Ø66Ø112ØB     READ STATUS REGISTER
```

This I/O operation will leave the Status register contents in the CPU accumulator.

WRITE

Up to 24 bits of data are transferred from the computer Accumulator and written into the addressed "short" register. To accomplish the "write" operation, the programmer must first load the correctly formatted 24-bit data word into the computer Accumulator from CPU memory using an LDA instruction. The data transfer is then accomplished by executing an SPU instruction which is coded according to the instruction word format shown on page 8-32.

For example, to write an address of 377ØØB to the Memory Address (MAR) register;

```
LDADR  LDA   ADDR           LOAD ADDRESS INTO ACCUMULATOR
WRMAR  DATA  Ø64Ø212ØB      WRITE TO MAR
        :
        :
ADDR   DATA  377ØØB
```

SPECIAL

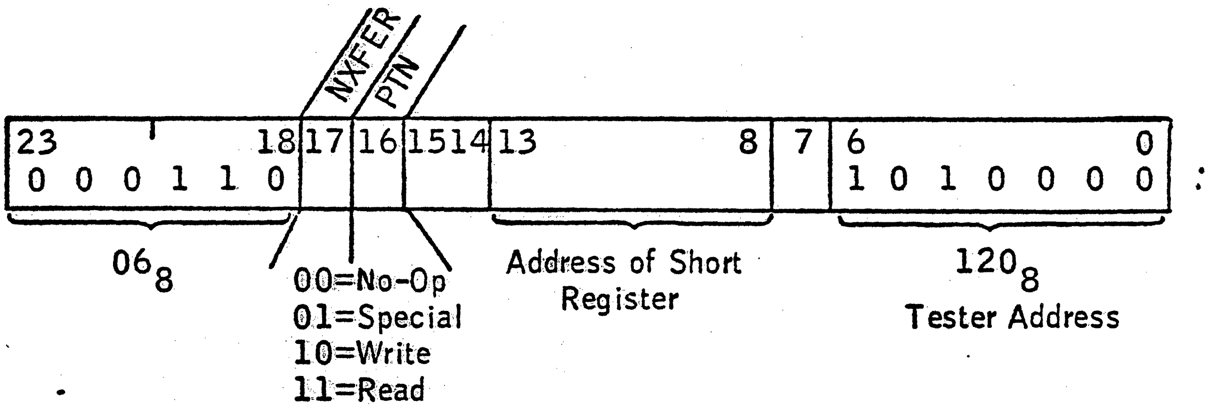
This operation causes a short register hardware operation, unique to the register being addressed, to be performed with no data transferred between the short register and the Accumulator. For example, a "special" to a specific DPS register will disconnect that DPS supply from the test system pin electronics. The "special" operation is accomplished by executing an SPU instruction which is coded according to the instruction word format shown on page 8-32.

For example, to "disconnect" DPS1 with a "special":

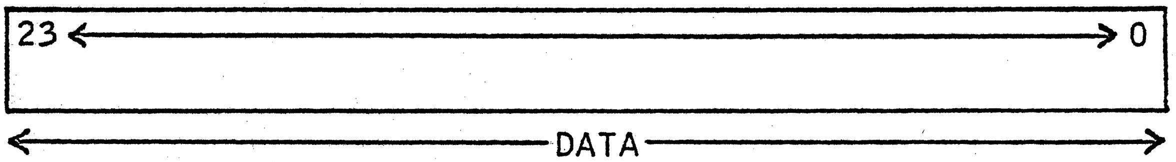
```
DISCON  DATA  Ø621Ø52ØB     DISCONNECT DPS1
```

SPU INSTRUCTION FORMAT

PART 1 - Issued at CPU T4, valid to the Tester Registers at T5.



PART 2 - Issued at CPU T1.



SHORT REGISTER ADDRESSES

	Bits								Bits					
Reg. Name	13	12	11	10	9	8	Octal	Reg. Name	13	12	11	10	9	8	Octal
MODE	0	0	0	0	0	1	01	DPT1	0	1	0	1	1	0	26
STATUS	0	0	0	0	1	0	02	E1	0	1	1	0	1	0	32
INSTRUCTION	0	0	0	0	1	1	03	E0	0	1	1	0	1	1	33
MAR	0	0	0	1	0	0	04	S1	0	1	1	1	0	0	34
TSC	0	0	0	1	0	1	05	S0	0	1	1	1	0		35
TDC	0	0	1	0	0	1	11	EA1	0	1	1	1	1	0	36
INC	0	0	1	1	0	0	14	EA0	0	1	1	1	1	1	37
DPS1	0	1	0	0	0	1	21	EB1	1	0	0	0	1	0	42
DPS2	0	1	0	0	1	0	22	EB0	1	0	0	0	1	1	43
DPT3	0	1	0	0	1	1	23	EC1	1	0	0	1	0	0	44
DPS3	0	1	0	1	0	0	24	EC0	1	0	0	1	0	1	45
DPT2	0	1	0	1	0	1	25								

ADDRESSING LONG REGISTERS

All "long" register I/O, whether "read" or "write" and whether DMA or non-DMA uses the Instruction Register which is accessed via the Short Register Data Bus as the portal through which the data is transferred. As indicated earlier, a long register "read" is always performed using non-DMA I/O while a "write" can be either DMA or non-DMA.

WRITING TO ONE OR MORE LONG REGISTERS:

The DMA method is normally used to address any register considered to be a "functional data register" (i.e. S, R, F, DA, DB, MA, MB, C, RZ, STROBE, INVERT, etc.). These registers require up to 60 bits of functional data to define the appropriate conditions for the 60 tester pins. Up to four data words, with 15 bits of data per word, are required to completely load one 60-bit register. The data word format for these registers is shown in the table on page 8-37 for the Group I data words. To accomplish a DMA data transfer to one of these registers, the assembly language programmer must perform the following general steps:

1. Insure that the data words to be transferred to the specific "long" register are sequentially arranged in contiguous CPU memory locations. One additional data word with bit 23 set to a binary "1" (representing a TRAP, or end of DMA) must be present as the last data word.

   One additional note concerning the data words set up in this step might be appropriate at this time. If it is desired, the programmer may include data words for more than one specific register before encountering the TRAP word. The data words should be arranged, however, such that one specific register is completely loaded with the last data word to that register being a "write and execute" before going on to the next register.

2. Using a non-DMA "write", load the Memory address Register (MAR) with the memory address of the first DMA data word (i.e. the memory address where it is actually located).

3. Using a non-DMA "write", load the MODE Register with the DMA mode bit (bit 9) set to a binary "1".

Upon executing step 3, the test system will begin transmitting the data words defined in step 1 to the appropriate "long" register, via the mainframe Instruction Register, in the DMA mode. Upon encountering the last data word with the TRAP condition defined in the "control" field, DMA will be terminated and the programmer may then proceed on to the next step in the program.

Example of DMA "write" to a "long" register:

```
            :
*  INITIATE DMA TRANSFER TO DA REGISTER
        STST  12ØB                        CHECK TESTER GENERAL STATUS
        BG    *-1                         TESTER BUSY?
        BL    *-2                         TESTER DMA ACTIVE?
        LDA   BUFADR
        DATA  Ø64Ø212ØB                   WRITE TO MAR
        DATA  Ø66ØØ52ØB                   READ MODE
        OR    DMAON                       PRESERVE CURRENT CONTENTS OF MODE REG.
        DATA  Ø64ØØ52ØB                   WRITE TO MODE REGISTER
*  DMA NOW ACTIVE - NOW WAIT FOR DMA COMPLETE
        STST  12ØB                        CHECK TESTER GENERAL STATUS
        BG    *-1                         TESTER BUSY?
        BL    *-2                         TESTER DMA STILL ACTIVE?
        BRU   EXIT                        DMA COMPLETE
*  DATA BUFFER
DMAON   DATA  1ØØØB
BUFADR  DATA  *+1
WDA1    DATA  Ø2Ø1Ø1Ø1B                   W&H   TO DA REG RANK 1
WDA2    DATA  Ø21Ø1Ø1ØB                   W&H   TO DA REG RANK 2
WDA3    DATA  Ø221Ø1Ø1B                   W&H   TO DA REG RANK 3
WDA4    DATA  223Ø1Ø1ØB                   W&E   TO DA REG RANK 4
TRAP    DATA  4ØØØØØØØB                   TRAP STOPS DMA WHEN ENCOUNTERED
*
            :
```

does get not absolute address of BUFADR [illegible] LDX LDA

BSM * DMA[illegible]

The non-DMA method of addressing long registers is normally used to transfer data to any register which is not a "functional data register". Those registers are also variable in bit length, but any one register never requires more than one data word to completely load it. The data word format for those registers (Group II and Group III) is shown on pages 8-38 and page 8-39.

To accomplish a non-DMA data transfer to one of those registers, the assembly language programmer must perform the following steps:

1. Load the data word, formatted for the desired register, into the computer Accumulator with the appropriate LDA instruction. Since this data is to be "written" into a register, the "control" field bits must define a "write and execute" condition (i.e. bit 23 set to a binary "0" and bit 22 set to a "1").

2. Execute an SPU instruction which is coded to cause a "write" to the Instruction Register on the Short Register Data Bus.

Upon executing step 2, the test system will transmit the data word from the computer Accumulator to the specified register, via the Instruction Register.

The two methods just previously described are the "normal" methods for writing to the "long" registers. In actual practice, non-DMA I/O could be used to write a single data word to a specific "rank" of a Group I register provided the "control" field of the data word defines a "write and execute" operation. Also, for writing a contiguous set of data words to the Group II and III registers, DMA I/O could be used, provided the last data word in the contiguous data word string is a TRAP word.

Example of non-DMA "write" to a "long" register:

```
          .
          .
          .
* NON-DMA WRITE TO MAJOR LOOP START ADDRESS REGISTER
          STST   120B        CHECK TESTER GENERAL STATUS
          BG     *-1         TESTER BUSY?
          BL     *-1         TESTER DMA ACTIVE?
          LDA    LRWRD
          DATA   Ø64Ø152ØB   WRITE TO INSTRUCTION REG.
          BRU    EXIT
* DATA    BUFFER
LRWRD     DATA   37ØØØØØØB   FACTOR "SET START Ø";
*         .
          .
          .
          .
          .
```

READING A LONG REGISTER:

Long registers may only be read using non-DMA I/O. The procedure to accomplish a long register "read" is similar to that of a long register "write" using non-DMA as previously discussed; one additional step is required, however, along with a change to the command field of the 24-bit long register data word.

To accomplish a non-DMA read of a "long" register, the assembly language programmer must perform the following steps:

1. Load a 24-bit data word, formatted to read a specific long register, into the computer Accumulator using the appropriate LDA instruction. (Refer to the Long Register Group I, II, and III word format charts previously used for the correct data word format.)

2. Execute an SPU instruction to the tester which is coded to "write" to the Instruction Register on the Short Register Data Bus. The data word instructing the specified long register to be read is now transferred through the Instruction Register and onto the Long Register Data Bus where it is decoded and executed. This returns the specified register's contents to the Instruction Register.

3. Execute an SPU instruction to the Tester which is coded to "read" the Instruction Register on the Short Register Data Bus. The Instruction Register contents, which is actually the contents of the long register specified in step 1, is now transferred to the computer Accumulator and is available for processing in accordance with the programmer's wishes.

4. Repeat steps 1 through 3 for each additional "register" or "rank" which contains data to be read.

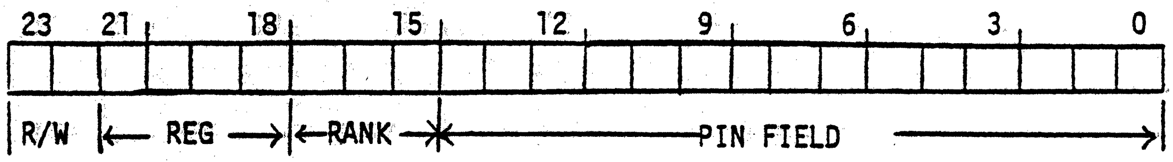
Example of non-DMA "read" of a long register:

```
                :
* READ 'C' REGISTER RANK 1 ONLY
            STST    120B            CHECK TESTER GENERAL STATUS
            BG      *-1             TESTER BUSY?
            BL      *-2             TESTER DMA ACTIVE?
            LDA     LRWRD
            DATA    Ø64Ø152ØB       WRITE TO INSTRUCTION REGISTER
            DATA    Ø66Ø152ØB       READ THE INSTRUCTION REGISTER
            STA     TEMP1           SAVE REGISTER DATA IN TEMP STORAGE
            BRU     EXIT
*  DATA     BUFFER
LRWORD      DATA    52ØØØØØØB       READ 'C' REG RANK 1
TEMP1       DATA    Ø
                :
```

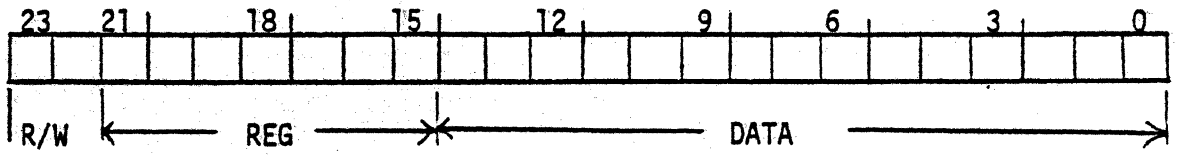
LONG REGISTER FORMATS

All the long registers on the Sentry fall under one of the following formats.

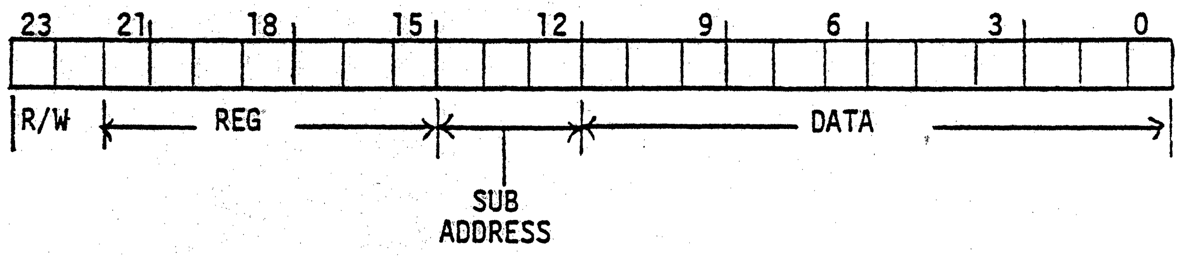
LRF 1



LRF 2



LRF 3



R/W

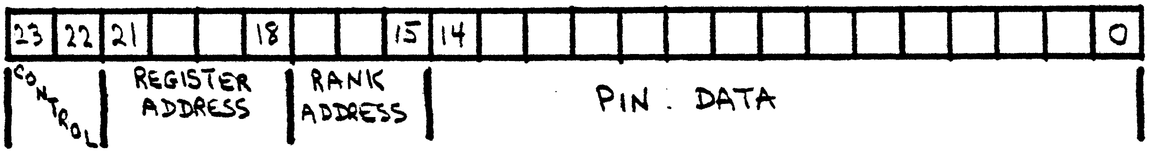
0 0 = Write & hold.

0 1 = Write & execute (increments IND reg. & initiates delay if applicable).

1 0 = Read (never a DMA instruction).

1 1 = DMA trap.

GROUP I - FUNCTIONAL DATA REGISTERS WORD FORMAT



CONTROL FIELD

23	22	Meaning
0	0	WRITE AND HOLD
0	1	WRITE AND EXECUTE
1	0	READ
1	1	TRAP

REGISTER ADDRESS FIELD

21	20	19	18	Meaning
0	0	0	0	RZ RETURN TO ZERO
0	0	0	1	ST STROBE SELECT
0	0	1	0	D/DA DEFINE I/O PINS
0	0	1	1	DB DEFINE I/O PINS (ALTERNATE)
0	1	0	0	M/MA MASK
0	1	0	1	MB MASK (ALTERNATE)
0	1	1	0	F FUNCTIONAL PATTERN
1	0	0	0	S SELECT ALTERNATE REFERENCE
1	0	0	1	TGA0 TIMING GENERATOR PIN ADDR (1)
1	0	1	0	C COMPARE (FAIL PATTERN)/INVERT
1	0	1	1	TGA1 TIMING GENERATOR PIN ADDR (2)
1	1	0	0	R UTILITY RELAY
1	1	0	1	TGA2 TIMING GENERATOR PIN ADDR (4)
1	1	1	X	SPECIAL REGISTERS

RANK AND PIN DATA FIELDS

RANK BITS				PIN DATA
17	16	15	Rank #	Bit 0 - - - - - - - - Bit 14
0	0	0	1	Data for pins 1 thru 15
0	0	1	2	Data for pins 16 thru 30
0	1	0	3	Data for pins 31 thru 45
0	1	1	4	Data for pins 46 thru 60
1	0	0	5	Data for pins 61 thru 75
1	0	1	6	Data for pins 76 thru 90
1	1	0	7	Data for pins 91 thru 105
1	1	1	8	Data for pins 106 thru 120

GROUP II - LONG REGISTER DATA WORD FORMAT

23	22	21						15	14														0
CONTROL		REGISTER ADDRESS							DATA														

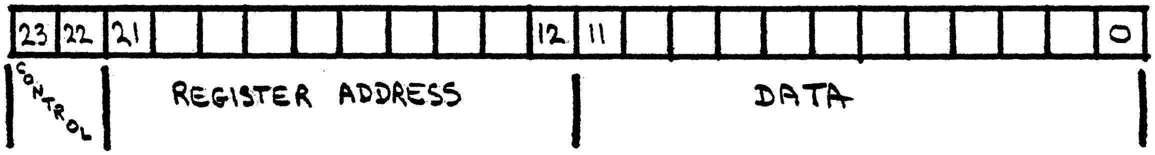
SPECIAL REGISTER ADDRESSES

21	20	19	18	17	16	15	Register
1	1	1	0	0	0	0	PA(160)
1	1	1	0	0	1	1	TRG(163)
1	1	1	0	1	0	0	PPS(164)
1	1	1	0	1	0	1	PSL(165)
1	1	1	0	1	1	0	EIR(166)
1	1	1	0	1	1	1	STSC(167)
1	1	1	1	0	0	1	DCT(171)
1	1	1	1	0	1	0	CHAINING(172)
1	1	1	1	1	0	1	PULSE WIDTH(175)
1	1	1	1	1	1	0	PULSE DELAY(176)
1	1	1	1	1	1	1	POWER PIN ADD(177)

CONTROL FIELD - Same as bit configuration for Group I

DATA FIELD - Varies for individual register

GROUP III - LONG REGISTER DATA WORD FORMAT



SPECIAL REGISTER ADDRESSES

21	20	19	18	17	16	15	14	13	12	Register
1	1	1	1	0	0	0	0	0	0	Major Loop Start(1700)
1	1	1	1	0	0	0	0	0	1	Minor Loop Count(1701)
1	1	1	1	0	0	0	0	1	0	Major Loop Count(1702)
1	1	1	1	0	0	0	0	1	1	Mem. Cycle Steal(1703)
1	1	1	1	0	0	0	1	0	0	Minor Loop Start(1704)
1	1	1	1	0	0	0	1	0	1	Minor Loop End(1705)
1	1	1	1	0	0	0	1	1	0	Major Loop End(1706)
1	1	1	1	0	0	0	1	1	1	Ignore Fail (1707)
1	1	1	1	0	0	1	0	0	0	Status and Mode A (1730)
1	1	1	1	0	1	1	0	0	0	Status and Mode B (1734)

CONTROL FIELD - Same bit configuration as for Group I

DATA FIELD - Varies for individual register

TESTER I/O INSTRUCTIONS

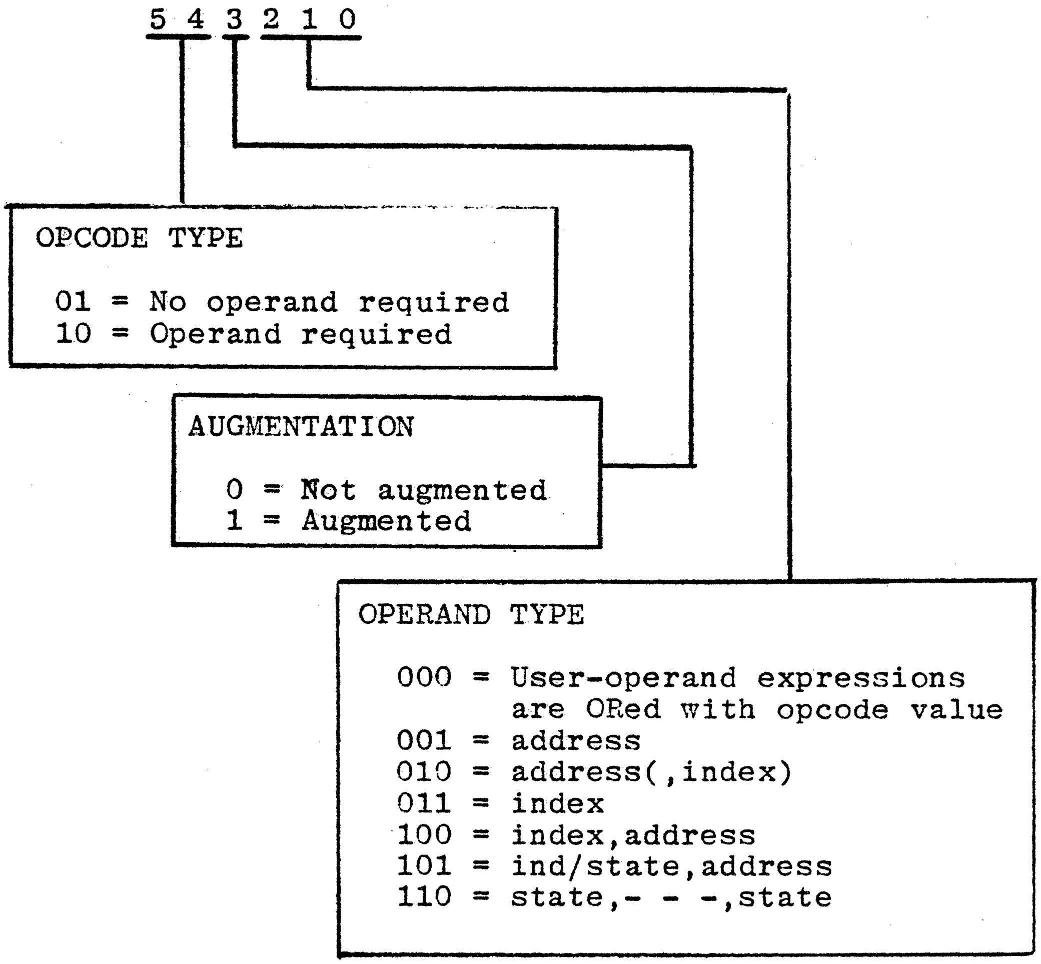
```
STST    120B                    (TESTER GENERAL STATUS)

      GT  -  TESTER BUSY
      EQ  -  TESTER INTERRUPT REQUEST
      LT  -  TESTER DMA IN PROGRESS
      BE  -  TESTER POWER SUPPLY FAILURE


PON     120B                    (ENABLE TESTER INTERRUPTS)
POFF    120B                    (DISABLE TESTER INTERRUPTS)
PCOMP   120B                    (TESTER INTERRUPT COMPLETE)
```

OTHER TESTER INSTRUCTION MNEMONICS DO NOT EXIST BUT MAY BE GENERATED BY THE PROGRAMMER USING THE EQU DIRECTIVE IE - LATCH EQU EXP-1, EXP-2

```
label    EQU    expression1,expression2
```

expression1 is actual value assigned to label

expression2 means label is to be treated as an opcode mnemonic. The expression can be a decimal or octal (octal is preferable) integer which represents a six-bit binary value whose bits have the following meaning:



OPCODE TYPE

01 = No operand required
10 = Operand required

AUGMENTATION

0 = Not augmented
1 = Augmented

OPERAND TYPE

000 = User-operand expressions are ORed with opcode value
001 = address
010 = address(,index)
011 = index
100 = index,address
101 = ind/state,address
110 = state,- - -,state

TESTER INSTRUCTION MNEMONIC GENERATION

```
ASSEMBLER
GENERATED
CODE            ASSEMBLER INSTRUCTION/DIRECTIVE

                *
06400120        WRITE    EQU     06400120B, 40B
06600120        READ     EQU     06600120B, 40B
00001400        INST     EQU     1400B
                *
                *
06401520                 WRITE   INST
06601520                 READ    INST
                *
```

TEST EXECUTION METHODS - DMA AND ENABLE TEST

DMASTR ENABLE DMA MODE AND WAIT FOR NOT BUSY. ALLOWS BACKGROUND OPERATIONS CONCURRENTLY.

```
BSM*   DMASTR              (747B)
```

SEE SYXVEC +71

ENBTST DO AN ENABLE TEST WHILE ALLOWING BACKGROUND ACTIVITY TO OCCUR CONCURRENTLY.

```
BSM*   ENTBSY              (755B) ENABLE INTERRUPTS
LDA    SAMA DATA
BSM*   ENBTST              (751B)
BSM*   WWAIT               (752B)
```

SEE SYXVEC +73

INTERP DO AN ENABLE TEST THE SAME AS INTERPRETER. ALL OVERHEAD FUNCTIONS ARE AUTOMATICALLY DONE.

```
LDA    OPCODE
BSM*   INTERP              (754B)
NOP    Ø
BSM*   WWAIT
```

OPCODE = 53100101B + ANY ADDITIONAL DESIRED ENABLE TEST MODES PER SAMA.

REVIEW OF TESTER I/O

PROGRAM EXECUTION WITH DUAL ENTRY POINTS (FOREGROUND/BACKGROUND)

COMBINATION F.G./B.G. PROGRAMS

PROGRAMS WHICH MAY BE EXECUTED FROM EITHER AN 'EXEC' STATEMENT OR THE KEYBOARD.

EXAMPLE: PSCAN, SPLOT, LABEL

- F.G. AND B.G. ENTRY POINTS ARE BOTH USED
- EACH USUALLY PASS CONTROL TO A SECTION OF CODE UNIQUE TO EACH ENTRY, THEN CONVERGE ON COMMON CODE
- B.G. ENTRY MUST WAIT FOR F.G. NOT BUSY (SUBROUTINE `TWAIT')
- F.G. ENTRY SHOULD SET FLAG THAT ENTRY WAS VIA F.G. NOT B.G.

F.G./B.G. EXAMPLE

```
FWORD   DATA   Ø          1ST WORD HEADER
        .
        .
BGENT   PZE    Ø
        BRU    BGSTR
FGENT   PZE    Ø
        BRU    FGSTR
        .
        .
        .
BGSTR   EQU    *          START B.G. ENTRY SET UP
        BSM*   TWAIT      WAIT FOR F.G. NOT BUSY
        AOM    THDACT     LOCK OUT OTHER STATIONS
        .
        .
        .
        BRU    MAIN
*
FGSTR   EQU    *
        AOM    FGBSY      SET F.G. BUSY FLAG
        .
        .
        .
        BRU    MAIN
*
MAIN    EQU    *
        .
        .
        .
      DO MAIN TASK
        .
        .
        .
QUIT     EQU    *         ALL DONE
         LDA    FGBSY     F.G. OR B.G. ENTRY
         BZ     BGOUT
*      F.G. EXIT
         CLA
         STA    FGBSY
         BRU*   FGENT
*      B.G. EXIT
BGOUT    EQU    *
         CLA              DISABLE 'OTHER STATION'
         STA    THDACT    LOCK OUT FLAG
         IDA
         STAT   R
         OR     B18       SET START ENABLE BIT
         STAT   W
         STAT   W
         IEN
         BRU*   BGENT
```

MISCELLANEOUS ROUTINES, PROCEDURES AND SUGGESTIONS

TO LOCK OUT OTHER STATION WHILE YOUR B.G. PROGRAM IS RUNNING

THE FLAG 'THDACT' (TEST HEAD ACTIVE) IS ALWAYS SET BY THE TEST HEAD DRIVER WHEN A FACTOR PROGRAM IS RUNNING.

THEREFORE, SETTING THIS FLAG IN YOUR BACKGROUND PROGRAM MAKES THE SYSTEM THINK THE TEST HEAD DRIVER IS ALREADY BUSY SO NO OTHER USER CAN COME ON LINE.

```
BGSTR   EQU     *
        AOM     THDACT    IST EXECUTABLE INSTRUCTION IN
        .                 B.G. PROGRAM
        .
        .
      DO SOME TASK
        .
        .
        .
        SOM     THDACT
        IDA
        STAT    R
        OR      B18       SET START ENABLE BIT
        STAT    W
        STAT    W
        IEN
        BRU*    BGENT     RETURN THRU FOREGROUND
```

ACCESSING THE TVT TABLE

REFERENCED VIA XR2

F.G.: XR2 POINTS TO TVT FOR CURRENT STATION ON LINE ON ENTRY TO FOREGROUND. XR2 SHOULD BE SAVED FOR LATER REFERENCE:

```
        STX     XR2,TVT
```

B.G.: MUST USE SYSTEM ROUTINE 'GTSATS' TO OBTAIN POINTER:

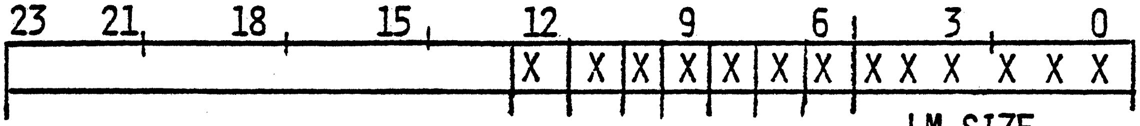
```
        LDA     SITEQQ   CURRENT STAT ON LINE (426B)
        ADD     D1       NEED NO. 1-4
        STA     STATC    (522B)
        BSM*    GTSTAT   (653B)
        NOP     Ø
        STX     XR2,TVT  SAVE FOR LATER
```

EXAMPLE:

```
TOPT   EQU   123         (123 DECIMAL!)
       .
       .
       .
       LDX*  XR2,TVT
       LDA   TOPT,XR2  GET HARDWARE CONFIGURATION
       CAM   $B11      LOW VOLTAGE HEAD?
       BBC   ISLVTH
       .
       .
       .
```

TOPT (TVT 123)

IT IS USED TO SAVE HARDWARE OPTION. IT IS SET DURING SYSTEM INITIALIZATION (AFTER BOOT FROM MAG TAPE OR ENTERED FROM DOPSY) AND NEVER CLEARED.



LM SIZE

1K = 001
2K = 011
4K = 111

B12	28 VOLT SWING	B18 MODE REGISTER
B10	NEW REF/MUX MODULE	B15 MODE
B11	LOW VOLTAGE	B17 MODE
BIT 9 = 1	2V/2MV OPTION	RVS RANGE BIT
8 = 1	SPM	
7 = 1	PPM	SAMC BITS 6-8
6 = 1	10 MHZ HEAD	
BITS 5-0 =	LM SIZE	

ACCESSING THE SVT TABLE

THERE ARE 2 METHODS:

1) F.G.

XR1 POINTS TO SVT TABLE ON ENTRY TO FOREGROUND. XR1 SHOULD BE SAVED FOR LATER REFERENCE:

```
          STX        XR1,SVT
```

2) F.G. AND B.G.

THE ABSOLUTE ADDRESS OF THE SVT TABLE IS DEFINED IN GLOVAR CELL #11 (433B).

```
SVT       EQU        433B
SVOFFS    EQU        7  (MUST BE DECIMAL!)
          .
          .
          LDX*       XR1,SVT
          LDA        SVOFFS,XR1     GET FLOATING POINT
          .                         VOFFSET VALUE
          .
          .
          .
```

MISCELLANEOUS - $ASIGN

Description:

$ASIGN is a general purpose, self-contained subroutine which may be used to assign either disc or memory files of user specified name, size, and type. If a file of the specified name is found to exist, a normal return will occur with no action taken.

The source code for $ASIGN must be inserted into the calling program, however, no further additions are required other than the BSM statement which references $ASIGN.

Calling Sequence:

```
DLD    NAME
BSM    $ASIGN
DATA   dev
DATA   type
DATA   size
---                    error return
---                    normal return
```

Where: NAME = references first 4 characters of file name

NAME + 1 = references last 2 characters of file name

dev = 5 for disc file

= 8 for memory file

type = 74B for string file

72B for data file

size = desired size of file in words

In case of an error, a standard ERRCNV error message will be displayed and the error return taken with the A register containing the standard $IOCS error code.

Registers A, E, X1, and X6 are <u>not</u> restored by $ASIGN.

SYSTEM SUBROUTINE LIBRARY - MASTR *vs* DOPSY

	DOPSY	MASTR	DIFFERENCE
GET A CHARACTER - SEQUENTIAL	GET	GETC	NO DISC
PUT A CHARACTER - SEQUENTIAL	PUT	PUTC	NO DISC
GET SYNTACTICAL ENTITY	SCAN	$PARSE	NO DISC
GET A WORD - SEQUENTIAL	GETW		
PUT A WORD - SEQUENTIAL	PUTW	PUTW	NO DISC
GET A WORD - RANDOM ACCESS	READ	READW	NO DISC
PUT A WORD - RANDOM ACCESS	WRITE	WRITEW	NO DISC
GET A RECORD - SEQUENTIAL	GFREC		
PUT A RECORD - SEQUENTIAL	PFREC		
CONVERT BINARY TO BCD	BINDEC	CONV	
FIND A FILE ON DISC	FIND/SRCH	SCNFIL	FOR MEMORY
		$IOCS OPEN	FOR DISC/MEMORY
PUT A RECORD TO POD	OUTREC	MSGOUT	NO DISC
GET A RECORD FROM PID	INREC	MSGIN	NO DISC
F.P. # TO BUFFER IN DECIMAL TRASCII		PUTD	
F.P. # TO BUFFER IN SCI FORMAT (TRASCII)		PUTE	
F.P. # TO BUFFER IN OCTAL TRASCII		PUTO	
F.P. # TO BUFFER IN OCTAL OR DECIMAL TRASCII		PUTA	
PUTS TIME OF DAY TO BUFFER IN TRASCII		PUTIME	
F.P. # TO BUFFER IN ENGR OR SCI FORMAT (TRASCII)		PUTENG	
AUTO MEMORY/DISC FILE PAGING ROUTINE		ADRXLA	

----

1) OCTAL OR DECIMAL ACCORDING TO BIT 19 OF ALL M1INIT (65B)

0 = DECIMAL 1 = OCTAL

GETC

GETS 1 CHARACTER FROM BUFFER SPECIFIED BY XR7. THE CHARACTER IS DETECTED AS EITHER ALPHA, NUMERIC, OR A SPECIAL CHARACTER. SAME AS GET IN DOPSY EXCEPT FOR 3 WORD DCB.

```
        LDX     XR7,DCB
        BSM*    GETC            (673B)
        ---                     ALPHA RETURN
        ---                     NUMERIC RETURN
        ---                     SPECIAL CHARACTER (177B = EOF)
```

A STANDARD 3 WORD DCB IS USED UNIVERSALLY:

```
DCB     DATA    CHAR POSITION (0-N)
        DATA    BUFADR
        DATA    BUFSIZ
```

THE VALUE AT DCB IS AUTOMATICALLY INCREMENTED TO THE NEXT POSITION.

SEE SYXVEC +27

READW

READS 1 WORD FROM BUFFER SPECIFIED BY XR7. SAME AS READ IN DOPSY EXCEPT FOR 3 WORD DCB.

```
        LDX     XR7,DCB
        LDA     WDCNT           # OF DESIRED WORD (O-N)
        BSM*    READW           (674B)
        ---                     EOF RETURN
                                NORMAL RETURN
```

STANDARD 3 WORD DCB:

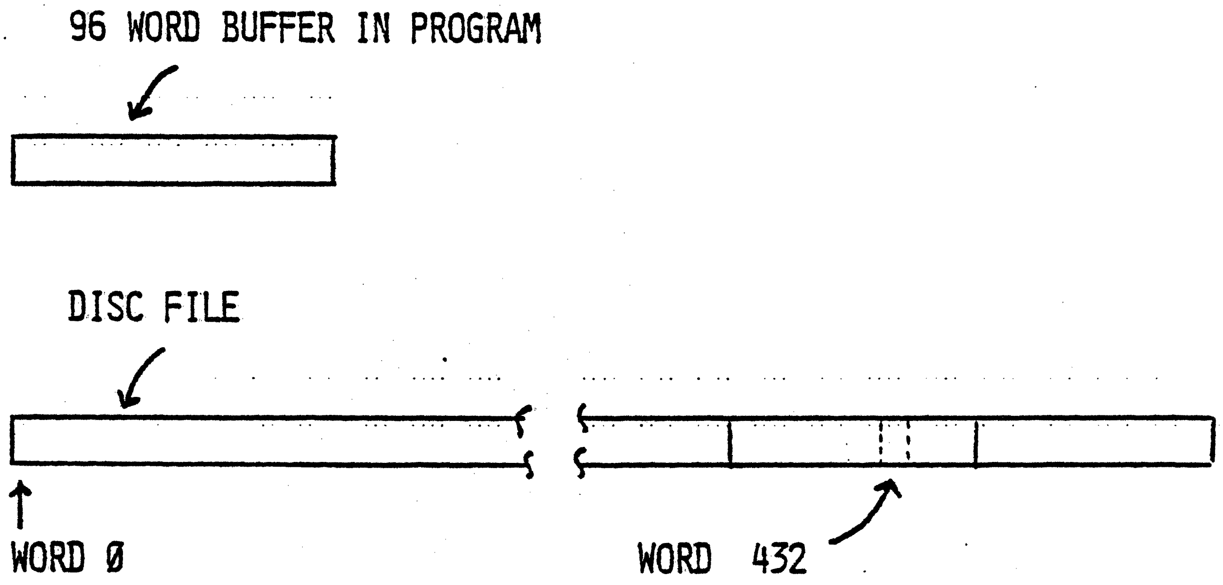
```
DCB     DATA    Ø               (NOT USED)
        DATA    BUFADR
        DATA    BUFSIZ
```

SEE SYXVEC +28

ADRXLA

PERFORMS AUTO PAGING OF BUFFER BETWEEN MEMORY AND DISC OR MEMORY FILE.

FETCH WORD 432 FROM A DISC FILE:



RETURNS ABSOLUTE MEMORY ADDRESS OF REQUESTED WORD IN A REGISTER.

```
LXA   X1
LDA   Ø,XR1
```

A REGISTER NOW CONTAINS DESIRED DATA.

ADRXLA (CONT)

OPEN DISC OR MEMORY FILE AND SAVE XR6 IN CELL IOADR.

```
        LDX*    XR6,IOADR
        LDX     XR7,DCB
        LDA     WORD ADDR           (Ø RELATIVE)
        BSM*    ADRXLA
        BRU     ERROR
        LXA     X1                  NORMAL RETURN
        LDA     Ø,X1                DESIRED DATA FROM FILE
        STA     ---

DCB     DATA    Ø                   NOT USED
        DATA    BUFADR
        DATA    BUFSIZ
IOADR   DATA    Ø                   XR6 VALUE SAVED FROM FILE OPEN
```

SEE SYXVEC +75

$PARSE

SCANS SPECIFIED BUFFER FOR 1 ALPHA IDENTIFIER, 1 NUMBER, OR A SPECIAL CHARACTER OTHER THAN A SPACE OR PERIOD. MUST BE CALLED ONCE FOR EACH FIELD.

IDENTIFIER: UP TO 8 ALPHA CHARACTERS BRACKETED BY BLANKS OR SPECIAL CHARACTERS. FOUND IDENTIFIER IS PACKED LEFT JUSTIFIED INTO 'NAME1' AND 'NAME2' IN 'GLOVAR'. ADDRESS OF NAME1 IS RETURNED IN XR6.

NUMBER: A STRING OF NUMERIC CHARACTERS IN TRASCII. THEY ARE CONVERTED TO FLOATING POINT AND RETURNED IN THE E REGISTER. THE OCTAL EQUIVALENT IS STORED IN 'OCTAL' IN 'GLOVAR'.

SPECIAL CHAR: A SPECIAL CHARACTER OTHER THAN A BLANK OR PERIOD. FOUND CHARACTER IS STORED IN A REGISTER ON RETURN.

```
        LDX     XR7,DCB          3 WORD DCB
        BSM*    $PARSE           (665B)
                                 NORMAL RETURN
```

ON RETURN, THE A REGISTER IS SET TO:

0 IF ERROR IN NUMBER SCAN OCCURRED.

1-77B IF SPECIAL CHAR FOUND.

100B IF AN IDENTIFIER WAS PACKED INTO NAME1 AND NAME2.

101B IF A NUMBER WAS FOUND. F.P. VALUE IN ENG.
OCTAL EQUIVALENT IN 'OCTAL' (GLOVAR).

177B END OF RECORD DETECTED.

SEE SYXVEC +21

NUMBER CONVERSION FROM INTERNAL FORMAT TO EXTERNAL TRASCII PRINTING FORMAT

FIXED POINT BINARY TO TRASCII DECIMAL	-	PUTD
FIXED POINT BINARY TO TRASCII OCTAL	-	PUTO
FLOATING POINT TO TRASCII INTEGER OR SCIENTIFIC NOTATION	-	PUTE
FIXED POINT BINARY TO TRASCII DECIMAL OR OCTAL BASED ON CONTENTS OF CELL 65B	-	PUTA
FLOATING POINT TO TRASCII ENGINEERING FORMAT	-	PUTENG

PUTD 11010 ⟶ 26

```
LDX     XR1,2        # OF DIGITS DESIRED
LDA     NUM          (32B)
LDX     XR7,DCB      3 WORD DCB
BSM*    PUTD
```

SEE SYXVEC +13

PUTO 11010 ⟶ 32B

```
LDX     XR1,2        # DIGITS DESIRED
LDA     NUM          (32B)
LDX     XR7,DCB      3 WORD DCB
BSM*    PUTO
```

SEE SYXVEC +18

NUMBER CONVERSION... (CONT)

```
PUTE     F.P. 26  ──→ 26
         F.P. .026  ──→ +2.6E-2

         LDA    F.P. VALUE
         LDX    XR7 DCB            3 WORD DCB
         BSM*   PUTE               (661 B)
```

SEE SYXVEC +17

```
PUTA     11010  ──→ 26  IF BIT 19 OF 65B = Ø, OR
         11010  ──→ 32B IF BIT 19 OF 65B = 1

         LDX    XR1,2             # DIGITS DESIRED
         LDA    NUM               (32B)
         LDX    XR7,DCB           3 WORD DCB
         BSM*   PUTA              (762B)
```

SEE SYXVEC +82

```
PUTENG   F.P. 26  ──→ 26V
         F.P. .026  ──→ 26MV

         LDE    ZERO OR V,S,A
         LDA    NUM              (F.P. VALUE)
         LDX    XR7,DCB          3 WORD DCB
         BSM*   PUTENG           (761B)
```

SEE SYXVEC +81

MISCELLANEOUS ROUTINES

CONV CONVERT BINARY ⟶ BCD

```
        LDA    NUM                    BINARY VALUE
        LDE    BIT WIDTH OF EACH DECIMAL DIGIT
        BSM*   CONV                   (654B)

     NOTE:  LDE D4 FOR BINDEC EQUIVALENT
```

SEE SYXVEC +12

SCNFIL SEE IF A GIVEN FILE IS IN MEMORY + OTHER FUNCTIONS

```
        DLD    FILNAM
        LDX    XR1,FUNCT CODE         (USUALLY Ø)
        BSM*   SCNFIL                 (652B)
        DATA   JOB#                   JOB TO BE SEARCHED
        BRU    NFOUND
        BRU    FOUND
```

SEE SYXVEC +10

MPZERO CLEAR AREA OF MEMORY TO ZEROES

```
        LDX    XR6,BUFEND
        LDX    XR7,BUFSTR
        BSM*   MPZERO                 (672B)
```

SEE SYXVEC +26

MISCELLANEOUS ROUTINES (CONT)

SEARCH   SEARCH A TABLE FOR A GIVEN ENTRY AND RETURN THE LOCATION.

```
DLD     ITEM                  ITEM TO BE FOUND
DST     NAME1                 (543B IN 'GLOVAR')
LDX     XR6,TABEND
LDX     XR7,TABSTR
LDA     ITMSIZ                # OF WORDS/ENTRY
BSM*    SEARCH                (671B)
BRU     FOUND
BRU     NFOUND
```

RETURNS WITH XR7 POINTING TO FOUND ENTRY.

SEE SYXVEC +25

IERMSG   DISPLAY A TERMINAL ERROR MESSAGE TO THE SYSTEM POD.

```
LDA     NUM                   ANY TERMINAL ERROR #
BSM*    IERMSG
```

THERE IS NO RETURN TO CALLING PROGRAM.

SEE SYXVEC +30

HEADER   OUTPUT A STANDARD DATALOGGER HEADER.

```
LDA     TSN,XR2               SERIAL #
LDE     SITEQQ                CURRENT STATION (426B)
LDX*    XR6,IOADR             PREVIOUSLY OPENED DEVICE
BSM*    HEADER                (702B)
```

SEE SYXVEC +64

MISCELLANEOUS ROUTINES (CONT)

ERRCNV DISPLAY A STANDARD $IOCS ERROR MESSAGE.

```
LDA     CODE                ERROR CODE 1-15
BSM*    ERRCNV              (715B)
NOP     Ø
```

SEE SYXVEC +45 FOR ERROR CODE DEFINITIONS AND THE CORRESPONDING MESSAGES.

ATTA ATTACH PROGRAM TO STATION; STATION ID WILL APPEAR IN NAME ALL 'STAT' COLUMN. PROGRAM WILL NOT BE AUTOMATICALLY PURGED FROM MEMORY.

```
LDX*    XR5,SAVX5           SAVED ON ENTRY
LDA     SITEQQ              (426B)
BSM*    ATTA                (740B)
```

SEE SYXVEC +64

INTERP INVOKE ANY TEST HEAD DRIVER INTERPRETIVE FUNCTION WITH AN OP CODE IN THE 500-600 SERIES.

SEE SOURCE LISTING OF *THD FOR OP CODES

```
LDA     OPCODE
BSM*    INTERP              (754B)
NOP     Ø
BSM*    WWAIT
```

SEE SYXVEC +76

MISCELLANEOUS PROCEDURES

1) FIND THE ABSOLUTE ADDRESS OF A FILE IN MEMORY:

- OPEN FILE (IOCS OPEN PROCEDURE)

- GET 'MACTAB' ENTRY ADDRESS FROM 'IOATAB' WORD 1 (FNAME1 CELL)

```
LDA   1,XR6
LXA   XR5            ADDR 1ST WORD MACTAB ENTRY
```

- GET ADDRESS OF 1ST WORD OF MEMORY FILE

```
LDA   3,XR5          ADDR
LXA   XR5            ADDR 1ST WORD OF FILE
.
.
.
LDA   ____
STA   Ø,XR5          ACCESS FILE
.
.
```

MACTAB (MEMORY ACTIVITY TABLE)

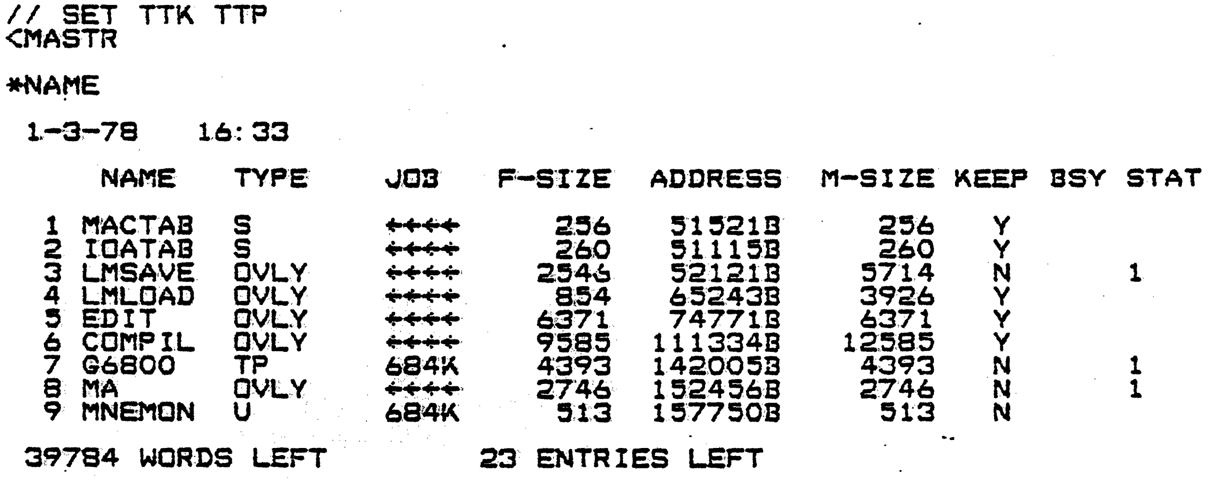
23 21 18 15 12 9 6 3 0

MAJOB	0	JOB NUMBER					
MANAM	1	NAME 1					
MATYP	2	NAME 2	ACTIVE IND	TYPE			
MAMADR	3	0	MEMORY ADDRESS				
MAFSIZ	4	M	K	R	P		FILE SIZE
MAMSIZ	5	0	MEMORY SIZE				
MAIOPT	6	0	IOATAB POINTER				
MAWSTR	7	0	WINDOW START				

8 words/entry, 32 entries in table
Total 256 words. The table is expandable by ASSIGN command.

MISCELLANEOUS PROCEDURES (CONT)

3) FORCE AN 'ATTACH' AND/OR 'KEEP' STATE FOR A MEMORY
FILE (PROGRAM).



```
// SET TTK TTP
<MASTR

*NAME

 1-3-78    16:33

     NAME    TYPE     JOB    F-SIZE   ADDRESS   M-SIZE KEEP BSY STAT

   1 MACTAB  S        ←←←←      256    51521B      256   Y
   2 IOATAB  S        ←←←←      260    51115B      260   Y
   3 LMSAVE  OVLY     ←←←←     2546    52121B     5714   N        1
   4 LMLOAD  OVLY     ←←←←      854    65243B     3926   Y
   5 EDIT    OVLY     ←←←←     6371    74771B     6371   Y
   6 COMPIL  OVLY     ←←←←     9585   111334B    12585   Y
   7 G6800   TP       684K     4393   142005B     4393   N        1
   8 MA      OVLY     ←←←←     2746   152456B     2746   N        1
   9 MNEMON  U        684K      513   157750B      513   N
 39784 WORDS LEFT          23 ENTRIES LEFT
```

THE FOLLOWING DEFINITIONS APPLY TO ACTION AUTOMATICALLY TAKEN AGAINST FILES IN MEMORY BY THE SYSTEM WHEN INSUFFICIENT MEMORY SIZE EXISTS TO LOAD A NEW FILE.

RELEASE - REMOVE FROM MEMORY AND PURGE MACTAB ENTRY

BUMP - REMOVE FROM MEMORY BUT KEEP MACTAB ENTRY TO ALLOW FASTER RELOAD

KEEP - NEVER RELEASE OR BUMP EXCEPT WITH RELEASE COMMAND

STAT - ATTACHED TO GIVEN STATION. DON'T BUMP UNLESS TEST PLAN OR MOD FILE.

ATTACH/KEEP (CONT)

KEEP: SET FLAG IN WORD 4 OF MACTAB ENTRY FOR FILE (PROGRAM)

```
LDX*   XR5,SAVX5
LDA    4,XR5
OR     B22
STA    4,XR5
```

ATTACH:

```
LDX*   XR5,SAVX5          XR5 SAVED ON ENTRY
LDA    SITEQQ             (426B)
BSM*   ATTA               (740B)
```

ATTACH/KEEP RELATIONSHIP

	ATTACHED[1]		NOT ATTACHED	
KEEP (Y)	NEVER RELEASE[2] NEVER BUMP		NEVER RELEASE NEVER BUMP	
DON'T KEEP (N)	OVLY	TP/MOD[3]	OVLY	TP/MOD
	NEVER RELEASE NEVER BUMP	BUMP ONLY NEVER RELEASE	RELEASE	RELEASE

1) A STATION NUMBER IN THE 'STAT' COLUMN INDICATES THAT THE GIVEN FILE IS ATTACHED TO THE SPECIFIED STATION.

2) RELEASED ONLY VIA USER RELEASE COMMAND.

3) OVLY = ASSEMBLY LANGUAGE F.G. OR B.G. PROGRAM
TP = FACTOR PROGRAM
MOD = LMLOAD 'LMI' FILE OR FACTOR DMA SUBROUTINE FILE (NOT CURRENTLY AVAILABLE)

MISCELLANEOUS PROCEDURES (CONT)

4) INTERRUPT SHOULD BE DISABLED WHILE DOING I/O OF STAT REGISTER.

```
IDA
STAT   R
STA    STATSV
CLA
OR B12 ENABLE RESET
STAT   W
STAT   W
IEN
```

5) DETERMINE LINE PRINTER TYPE.

```
LPFLG  EQU    66B          SEE PATCH CELLS

LPFLG = Ø   80 COL DATA PRODUCTS
        1   132 COL DATA PRODUCTS
        2   132 COL PRINTRONICS
```

MISCELLANEOUS PROCEDURES (CONT)

6) GET THE NAME OF THE TEST PLAN CURRENTLY EXECUTING.

```
SITEQQ  EQU   426B
ATPA    EQU   420B
MANAM   EQU   2

        LDX*  XR5,SITEQQ        CURRENT STAT #
        LDA   ATPA,XR5          T.P. MACTAB ADDR
        LXA   XR5
        DLD   MANAM,XR5         NAME1,2
```

7) DETECT A STATION RESET FROM A B.G. PROGRAM
(SEE STSC REG FORMAT).

```
RSTTSC  EQU   572B
SITEQQ  EQU   426B

        LDA   SITEQQ            CURRENT STAT
        AND   D3
        LXA   XR1
        LDA   RSTTSC
        LS    IO
        LS    Ø,XR1
        AND   D1
        BP    ABORT             FOUND RESET OR KILL
```

MISCELLANEOUS PROCEDURES (CONT)

8) DETERMINE IF THE SYSTEM HAS A DISC AVAILABLE (DISTINGUISH A SVII FROM SV).

```
DFDV   EQU   65B            SEE PATCH CELLS

DFDV = 125B IF DISC AVAILABLE
```

9) DETERMINE IF THE LP HAS BEEN SPECIFIED BY ANY OF THE FOLLOWING:

```
DATALOG    LP              (SET DATALOG DEVICE)
USE        LP              (SET STATION POD)
WRITE (LP)

LDX*    XR2,TVT            CHECK FOR 'DATA LP'
LDA     TDLO,XR2           ($67_{10}$)
CAM     D4
BE      USELP
LDA     TLP,XR2            ($118_{10}$)  CHECK FOR 'USE' OR 'WRITE'
BP      USELP
```

SUGGESTIONS

1) THE CONTENTS OF TRANSFER VECTORS THROUGH WHICH A BSM* WILL OCCUR MUST NOT USE BIT 23 AS A DATA FLAG, SINCE THE ASSEMBLER WILL ALWAYS SET IT TO A 1'.

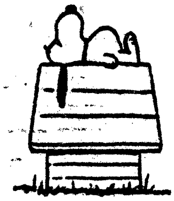
   EXAMPLE:

```
  BIT20     DATA   04000000B
  BIT23     DATA   40000000B
  TAB       DATA   LABL1+BIT23+BIT20
            DATA   LABL2+BIT20
            .
            .
            .
            .
            BSM*   TAB
            .
            .
            .
```

   AT TAB AND TAB+1 BIT 23 SHOULD NOT BE USED AS A FLAG AS IT WILL ALWAYS BE SET TO 1 BY THE ASSEMBLER.

2) ABSOLUTE MEMORY ADDRESSES MUST NOT BE SAVED IN A MEMORY CELL TO BE USED ON A SUBSEQUENT EXECUTION OF THE PROGRAM SINCE THE FILE MAY BE MOVED.

3) ALL CELLS MUST BE INITIALIZED TO THEIR DESIRED INITIAL STATE ON ENTRY TO PROGRAM.



FST-2 COMPUTER
ASSEMBLY LANGUAGE PROGRAMMING COURSE

STUDENT ASSIGNMENTS

FST-2 COMPUTER ASSEMBLY LANGUAGE PROGRAMMING COURSE

HOMEWORK ASSIGNMENT - DAY 1

1. IN THE FST-2 COMPUTER MANUAL, REVIEW PAGES 1-1 THRU 1-21.

2. COMPLETE QUIZ #1 - A TRUE/FALSE QUIZ.

3. COMPLETE QUIZ #2, COVERING NUMBER SYSTEMS AND BINARY ARITHMETIC.

4. TO READ AHEAD FOR TOMORROW'S LECTURE, AS TIME PERMITS, READ PAGES 3-1 THRU 3-27 OF THE FST-2 COMPUTER MANUAL.

FST-2 ASSEMBLY LANGUAGE PROGRAMMING COURSE - QUIZ #1

Instruction: This is a True/False quiz. Please answer the following questions with either T or F.

__T__ 1. The index registers can be used to count within a program.

__F__ 2. The name CPI stands for Common Program Interface.

__F__ 3. The assembler allows the programmer to write a program in the "native" language of the computer.

__F__ 4. The R Register is used for remainders left over from division instructions.

__T__ 5. The Program Counter can be loaded from the CPU control panel.

__T__ 6. The A, B, and N busses are bi-directional data busses.

__T__ 7. Indirect addressing uses the contents of an address as an address.

__F__ 8. DMA allows direct data transfers between peripheral units.

__T__ 9. In DMA, data is transferred over the A, B, and N busses.

✓__T__ 10. The E Register can be loaded manually from the CPU control panel.

__T__ 11. The Program Counter usually points to the next memory location.

__F__ 12. The Relocation Register contains the effective operand address for instructions involving indexing operations.

__F__ 13. An arithmetic overflow causes a carry into the Extension Register.

__F__ 14. The results of an arithmetic operation are stored in the Arithmetic Logic Unit (ALU).

__F__ 15. The Command Register contents are sent directly to the Accumulator Bus during an I/O operation.

__F__ 16. An overflow occurs when an arithmetic operation changes the sign bit in the Accumulator.

__F__ 17. An Index Register can store up to 24 bits of data.

__F__ 18. Console switches are used to control conditional branching.

__F__ 19. The TV indicator is lit only between 7 and 9 pm.

__T__ 20. The Reset pushbutton clears the Program State indicators.

FST-2 COMPUTER ASSEMBLY LANGUAGE PROGRAMMING COURSE

QUIZ #2 - Number Systems and Binary Arithmetic

1. Convert the following numbers to the base system as indicated:

68_{10} = 1000100$_{2}$

1010011_{2} = 83$_{10}$

1372_{8} = 762$_{10}$

156_{10} = 234$_{8}$

142273_{8} = 1100010010111011$_{2}$

010011011_{2} = 233$_{8}$

2. What is the difference between 1's compliment and 2's compliment? Please detail your answer.

3. What is the octal 8's compliment of 5555_{10}?

65115_8

12663_8

65114_5 -?

76225

4. Perform the following:

$1011010_{2} + 1011011_{2}$ = 10110101$_{2}$

$11011_{2} - 10010_{2}$ = 1001$_{2}$

FST-2 ASSEMBLY LANGUAGE PROGRAMMING COURSE

DAY 2

CLASSROOM ASSIGNMENTS:

1. COMPLETE PROGRAM WRITING ASSIGNMENT #1 AND #2

HOMEWORK ASSIGNMENTS:

1. COMPLETE HOMEWORK PROGRAM WHICH REQUIRES ASSEMBLY LANGUAGE/MACHINE LANGUAGE ENCODING AND DECODING. (PAGE 5)

2. COMPLETE PROGRAMMING WRITING ASSIGNMENTS #3, #4, AND #5.

HOMEWORK PROBLEM

Encode or decode the following instructions as required:

Address (octal)	Content	Instruction Mnemonic
1000	25001014	LDA 1014
1001	02036030	DSL 30
1002	26001015	AND 1015
1003	22001016	SUB 1016B
1004	02401006	BP 1006
1005	12001010	BSM 1010B
1006	07700000	LXA 7
1007	01001000	BRU 1000B
1010	00000000	PZE 0
1011	36001010	AOM 1010B
1012	10000000	NOP
1013	01041010	BRU 1010
1014	77760000	DATA 77760000B
1015	00037777	DATA 37777B
1016	77767774	DATA -5000

77766170

11110

NOTE: This program is not using the relocation flag bit. In an actual assembled program bit 23 would be set to allow relocation.

PROGRAM WRITING ASSIGNMENT #1

Problem: Add 3 numbers together, store the results in CPU memory, then halt at the beginning of the program.

Assume the answer is to be stored at location 400B and the three numbers are found at the following locations:

Location	Content
300	200B
301	150B
302	460B
.	
.	
.	
400	Store Answer Here

[illegible] [illegible]
[illegible] [illegible]
[illegible] [illegible]
STA ANS
[illegible] [illegible]

PROGRAM WRITING ASSIGNMENT #2

Problem: Repeat program writing assignment #1, except store the answer in 100 consecutive locations beginning with location 400B before halting.

DO NOT USE indexing, indirect addressing, or operand arithmetic.

PROGRAM WRITING ASSIGNMENT #3

Problem: Repeat program writing assignment #2. This time operand arithmetic using AOM or SOM is allowed.

PROGRAM WRITING ASSIGNMENT #4

Problem: Repeat program writing assignment #3. This time indirect addressing is also allowed.

PROGRAM WRITING ASSIGNMENT #5

Problem: Repeat program writing assignment #2. This time use indexing only.

```
LDX   X6,100
[illegible]DX  X[illegible]
LDA   [illegible]
```

PROGRAM WRITING ASSIGNMENT #6

Write a short program which will allow you (the operator) to enter your name at the VKT keyboard, displaying each character as it is entered.

Upon entering a carriage return, your name should be repeated at the VKT screen on the next line.

After your name is repeated, exit the program and return to MASTR.

The student may use either of two I/O methods:

1. Leave the characters input from the keyboard in standard ASCII format and store one character per CPU word in memory.

2. Convert the characters input from the VKT from ASCII to TRASCII then pack four TRASCII characters per word into memory.

Turn in an assembler listing of your successfully operating program to your instructor at the end of lab on Day 3.

PROGRAM WRITING ASSIGNMENT #7

1. Write an assembly language (background overlay) program for the FST-2 computer which will operate as defined by the Echo Program Definition on the next page.

2. Using the appropriate MASTR commands, assemble, execute, and debug the Echo program. 7

3. As part of the requirements for successful completion of this course, the student must turn in the following to the instructor:

   a. Copy of the first assembly listing (the one with all the mistakes on it!).

   b. Copy of final debugged assembly listing for program which actually executes according to the Echo program definition.

ECHO PROGRAM

DEFINITION OF PROGRAM:

The "Echo Program" requires that random data be entered from the VKT keyboard and then stored in programmer defined internal CPU memory buffers. At the operator's decision, the content of the memory buffers are to be dumped to an output peripheral, also selected by the operator, in a format which is identical to the original input data.

Specifically, the program must perform as follows:

1. Input a variable number of characters in any order from the VKT keyboard, repeat the character on the VKT display, and store it away in a CPU memory buffer which you have defined in the program.

2. After at least one character has been stored away, use Console Switch #6 to determine whether to remain in the "input mode" or to go to the "output mode".

3. Use Console Switch #5 to determine whether the data output is to go to the VKT display or the line printer. In either case, the data output must be "echoed", or output, in exactly the same format as it was input.

4. Use Console Switch #1 to abort the "Echo Program" and return control back to MASTR from within any point in the Echo Program - ie when lifting CS1, the program should abort immediately without waiting for any further operator interaction.

5. Console Switch usage is to be as follows:

CS1	UP	- unconditionally return to MASTR
	DOWN	- remain in Echo Program
CS5	UP	- output to line printer
	DOWN	- output to VKT display
CS6	UP	- output mode of Echo Program
	DOWN	- input mode of Echo Program

PROGRAM WRITING ASSIGNMENT #8

This is a supplemental assignment to the ECHO Program.

1. Modify your existing ECHO Program such that you are required to pass parameters to the program from the keyboard execution command which will:

   a. Define the peripheral which the output of the ECHO program is directed to (VKT or LP, with a default to the VKT if LP is not specified).

      e.g. *ECHO LP

   b. Define TOF to be performed if LP was specified, along with the number of TOF's desired.

      e.g. *ECHO LP 3

2. Delete the usage of Console Switch #5 to determine the output device used as it is now determined by a parameter passed from the background keyboard command (step la above).

3. Output a short message to the VKT screen at the beginning of your Echo program so you know the program execution has started. When your program enters the "input" mode, output a single character prompt so the operator knows when to input characters.

PROGRAM WRITING ASSIGNMENT #9

1. Rewrite your ECHO program to input and output using IOCS only; i.e. do not perform direct input/output operations with the peripherals using SPU instructions.

2. When executing the ECHO program, include the following IOCS functions:

   a. Upon entering the ECHO program, clear the VKT screen.

   b. Output operator messages to the VKT screen giving a brief definition of the ECHO program and instructions for use.

   c. When output is to line printer, perform TOF using IOCS control rather than direct line printer control as previously done.

PROGRAM WRITING ASSIGNMENT #10

1. Write a short FACTOR program which will input an integer number between one and ten (inclusive). Use WRITE statements which will instruct the operator when to input, the range of the numbers, etc. Write the message at the VKT screen.

   Pass the number just input to an assembly language program in the foreground mode which will in turn, turn on the corresponding numbered EIR register bit and lamp at the tester (i.e. the number 7 turns on EIR lamp 7, not 1, 2, & 3 which would be the binary equivalent of 7 decimal).

   Return to the FACTOR program and read the EIR register, then compare the value read to the original number input at the VKT keyboard. Write a message at the line printer which will indicate the results of the comparison.

   Call the same assembly language program, or another one according to your choice (as long as it is written by you and is not a system overlay such as LPLF or XGRAPH or SPLOT, etc) which will execute two top-of-forms (passes as a parameter).

   Return to the FACTOR program and end the program.

2. Turn in a copy of the first and last assembly and compiler listings, and the printout of the line printer messages during the program execution.

NOTE: The EIR register lamp which is lit during the program execution must remain on after the tester EOT.