IBM

TRADEMARK

Technical Newsletter No. 10

APPLIED SCIENCE DIVISION

IBM

APPLIED SCIENCE DIVISION

Technical Newsletter No. 10

October 1955

A Computation Seminar, sponsored by the International Business Machines Corporation, was held in the IBM Department of Education, Endicott, New York, from August 1 through August 4, 1955. Participating in this seminar were 67 research engineers representing computing facilities which employ IBM 650 Magnetic Drum Data Processing Machines. The formal papers of the seminar are published in this Technical Newsletter so that the authors' valuable information and experience may be shared as widely as possible.

The papers may be grouped into three general classifications. The first few papers describe different systems which have been developed and used for programming and operating the 650, including a wide variety of subroutines. The second group of papers describes methods and programs for solving various general classes of mathematical problems. The third group describes the use of the 650 in many different fields of engineering and science. Also included are descriptions of recently developed special attachments to the 650, which provide for even greater flexibility, a partial listing of subroutines used by 650 customers, and a listing of typical 650 customer applications.

The authors of these papers have very generously agreed to make available program card decks and other necessary information (flow charts, wiring diagrams, etc.) so that other 650 customers may be able to use the same procedures.

The International Business Machines Corporation wishes to express its appreciation to all those who participated in this seminar.

CONTENTS

SYMBOLIC CODING AND ASSEMBLY FOR THE IBM TYPE 650

R. E. Ruthrauff
Douglas Aircraft Company

The following describes a method of coding for the IBM Model 650 Computer which employs symbolic rather than actual locations and addresses. The ideas presented here are not original and represent merely a modification of symbolic coding techniques developed and used by Douglas Aircraft Company, Inc., Santa Monica Division, for the IBM Model 701. A logic is developed for coding and a program is described which performs the assembly into actual machine language.

CODING IN ACTUAL MACHINE LANGUAGE

The product of any coding technique for a computer must be the series of machine language instructions which, when executed, perform the desired series of computations. The means of attaining this "actual" as it is called, are as diverse as the functions of the companies currently using the 650. However, the problems of these organizations may be grouped into two major categories:

1) Those organizations whose principal computing requirements are the solution of a few extremely large problems which are modified infrequently. The majority in this category are accounting type problems.

2) Those companies engaged in the solution of many technical problems subject to a variety of changes.

Coding in actual machine language presents many serious objections for organizations of the second type above. Among them are the following:

1) Changes are difficult to make.

2) Portions of the coding cannot be easily relocated in memory.

3) It <u>is</u> actual coding and as such represents a compromise between feasible machine design and programming requirements. Since, of course, machine design gets more than programming does from the compromise, actual coding is not an efficient means of programming.

Symbolic coding is an attempt to remedy these serious objections to actual coding and has been used successfully by many users of the EDPM 700 series machines.

SYMBOLIC CODING

In coding any problem for a stored program computer, the operations to be performed consist of a sequence of subordinate computations, tests, data read-in, and data read-out. These pieces or blocks are called regions and constitute logical stages in the execution of a program. They may be the evaluation of some specific arithmetic function, such as sin x. In addition, they may also constitute the storage required by other regions.

For the purpose of coding these regions, a group of numbers commencing at 10 and terminating at 99 is set aside. Numbers are then selected from this group by the programmer and assigned to these regions: all subsequent references, of course, must consistently employ these numbers. Since data storage regions are required by all programs, these have been assigned the special region numbers 1, 2, 5, and 6 and will be discussed later.

Each actual instruction is completely determined by its actual location, actual operation, and actual data and instruction address. Symbolically, it is determined by location region, symbolic operation code, data address region, and instruction address region. Obviously, some sequence must be specified within each of these regions and, therefore, location sequence, data address sequence and instruction address sequence are introduced. The symbolic operation code is a three position alphabetic abbreviation.

All machine instructions may be grouped into one of the following categories:

1) Instructions which refer to data storage locations.

2) Instructions which refer to the location of other instructions.

3) Instructions whose addresses are a special function of the operation performed.

4) Instructional constants: i.e. numerical constants which are entered as part of the program deck.

Special region numbers are assigned to these categories as follows:

0 - The use of 0 as either a data address region or instruction address region indicates that the corresponding sequence is the actual machine address (type 3 and 4 mentioned before). For example, the instruction SR (shift right) 4 has a data address region of 0 and a data address sequence of 4. Instructional constants are always entered with data address region and instruction address region equal to zero. Since 44 of a possible 100 operation codes are provided on the 650, the operation bits of an instructional constant are handled in a different manner. For these constants the symbolic code abbreviation is numeric rather than alphabetic and is indicated by the presence of a zero in the first position of the field. This also serves to call attention to the fact that this symbolic instruction is such a constant. For example, pi at nine decimals is written as

Symbolic Operation - 031
Data Address Sequence - 4159
Instruction Address Sequence - 2654

with both address regions 0.

1 - Region 1 refers to the location of all temporary storage which may be used by the coder, but is primarily reserved for the execution of subroutines (type 1 mentioned previously). The sequence within this region designates the word of the region to be used. The first word in this region has address sequence 0. For example, the instruction RAL (Reset Add Lower) with a data address region of 1 and a data address sequence of 5 causes the contents of the sixth word in region 1 to be placed into the lower accumulator.

2 - This region is reserved for the storage of all data of a permanent nature required by the coder in the execution of a given problem (type 1 mentioned previously). The sequence designation within this region is identical to that of region 1. However, the first two words of region 2 are reserved for the storage of two important constants. Region 2 sequence 0 contains a 1 in the instruction address units position and region 2 sequence 1 contains a 1 in the data address units position.

3 - The use of 3 as either a data address region or instruction address region indicates that this instruction refers to the location of another instruction somewhere within the same region (type 2 above). The sequence used with this address region designates the specific instruction in that region. Example:

> The instruction LDS (Load distributor) with a data address region of 3 and data address sequence of 17 causes the instruction in this same region having a location sequence number of 17 to be placed in the distributor.

This data address region could be the actual region number rather than three. Three is used merely to facilitate a change in the region number without changing all references to other instructions within the same region.

5 - The use of 5 refers to the read locations peculiar to the 650, and is used for purposes of input. The sequencing is identical with that of region 1.

6 - This address region refers to the punch locations peculiar to the 650, used by output routines. The sequencing is identical with that of region 1.

In addition to these special regions a data address region or instruction address region of 10 or larger is permissible if a reference is made to an instruction in another region. However, by its very nature, a region is independent and references beyond linkage instructions to these regions are kept to a minimum.

The following example illustrates the use of symbolic coding in a simple parameter study.

The problem consists of the following:

(Figure 1) Tables of the E parameters are given with the constants F, G, and μ, which are to be used with the E's. All possible combinations of these values are to be formed in the equations given and the corresponding results with the specific choice of E's which gave those results are to be recorded on the output card. The flow chart (Figure 2) gives the step-by-step procedure followed in the execution of the program.

The problem begins by reading all necessary input data which consists of the tables of E values, F, G, and μ. Since these tables may not be of the same length, three additional pieces of information are furnished in the form of $E_{a_{final}}$, $E_{b_{final}}$, and $E_{c_{final}}$. As the notation implies these are the last entries in the E_a, E_b, E_c tables.

Once the input has been accomplished some preliminary computations are performed and the locations of the first entries for each of these three tables are reset. Then the first set of parameters is selected and the quantities σ_1, σ_2, and Υ are computed using the fixed point square root subroutine. The result of this computation along with the E's used is then punched in an output card. A test is performed to determine whether the E_a value just used was equal to the $E_{a_{final}}$ value. If it was not, the location of the E_a value is advanced by one and control is returned to that portion of the program which selects the next set of E values. However, if the two quantities are equal, the E_a value is then reset to the location of the first E_a and a test is performed on E_b to determine if it may be equal to $E_{b_{final}}$. This procedure is repeated on E_c thus generating all combinations of the E values. When all such computations have been performed, the machine is instructed either to stop or to return to read more input cards. The number appearing in the lower right-hand corner of certain blocks in this diagram is the region number which was arbitrarily assigned from the numbers 10 through 99. The flow chart constitutes the master control region normally called region 10. Portions of the flow diagram having no region number indicated are part of this region.

Prior to the coding, a layout of region 2, the permanent storage region, is made and sequences are assigned to the quantities needed. The regions may then be coded independently, the only information required being the location of the quantities which are needed in region 2. These regions are then coded by the programmer using the sheet shown in Figure 3. These coding sheets are then submitted to keypunch where they are punched one instruction per card with descriptive notes. Certain of the regions needed in the problem are standard subroutines which are available in a permanent file. These are reproduced with the appropriate region number for this problem. The entire file of cards is now sorted to location region major, location sequence minor, and listed on the 407 (Figure 4). At this point the final checking is performed prior to the assembly.

INPUT F, G, μ

E_{a_i} $i = 1, 2, 3 \ldots N_a$ $E_{a_{FINAL}}$

E_{b_j} $j = 1, 2, 3 \ldots N_b$ $E_{b_{FINAL}}$

E_{c_k} $K = 1, 2, 3 \ldots N_c$ $E_{c_{FINAL}}$

EQUATIONS

$$A_{ijk} = \frac{E_{a_i} + E_{b_j} + E_{c_k}}{1 - \mu}$$

$$B_{ijk} = \frac{\sqrt{\left(E_{a_i} - 1/3\left(E_{a_i} + E_{b_j} + E_{c_k}\right)\right)^2 + \left(E_{c_k} - E_{b_j}\right)^2}}{1 + \mu}$$

$$\sigma_{1ijk} = FAijk + GBijk$$

$$\sigma_{2ijk} = FAijk - GFijk$$

$$\tau_{ijk} = GBijk$$

OUTPUT

E_{a_i} E_{b_j} E_{c_k} σ_{1ijk} σ_{2ijk} τ_{ijk}

Figure 1

FLOW DIAGRAM

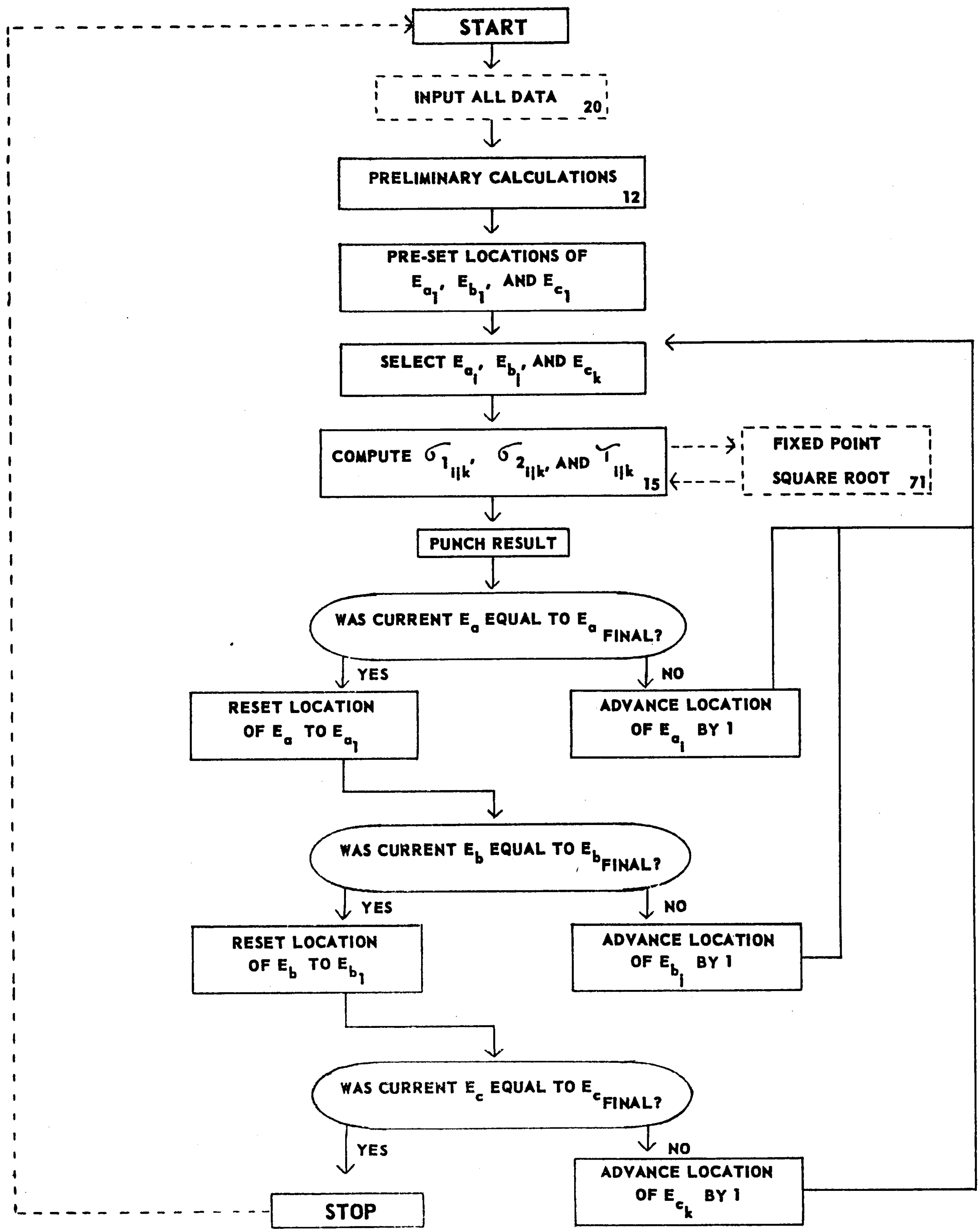


Figure 2

CODING SHEET – TYPE 650
FORM 25-5-121 (1-55)

PROGRAM NUMBER	PROGRAM TITLE	JOB NUMBER
57.0	PARAMETER STUDY	

REGION NUMBER	REGION TITLE	
10	MASTER CONTROL	PAGE 1 OF 9

PREPARED BY	DATE	CHECKED BY	DATE

SYMBOLIC CODING / **REMARKS**

LOC. SEQ.		OPER.	DATA ADD. REG.	DATA ADD. SEQ.		INST. ADD. REG.	INST. ADD. SEQ.		+ –	REMARKS
0		HLT	3	0		3	1		+	
1		LDS	3	2	2	20	0		+	INPUT ALL DATA
2	2	LDS	3	3		12	0		+	SET-UP FOR RUN
3		LDS	3	900		3	4		+	
4		SDS	3	9		3	5		+	PRE-SET E A ADDRESS
5		LDS	3	901		3	6		+	
6		SDS	3	11		3	7		+	PRE-SET E B ADDRESS
7		LDS	3	902		3	8		+	
8		SDS	3	13		3	9		+	PRE-SET E C ADDRESS
9		LDS	[0	0]		3	10		+	
10		SDS	6		2	3	11		+	WORKING E A
11		LDS	[0	0]		3	12		+	
12		SDS	6		3	3	13		+	WORKING E B
13		LDS	[0	0]		3	14		+	
14		SDS	6		4	3	15		+	WORKING E C
15		LDS	3	15	5	15	0		+	COMPUTATION
17		RAL	6		2	3	18		+	
18		SL	2		5	3	19		+	
19		TS	3	20		3	23		+	TEST E A
20		RAL	3	9		3	21		+	
21		AL	2		1	3	22		+	
22		STL	3	9		0	800	2	+	
23		LDS	3	900		3	24		+	
24		SDS	3	9		3	25		+	
25		RAL	6		3	3	26		+	
26		SL	2		6	3	27		+	
27		TS	3	28		3	31		+	TEST E B
28		RAL	3	11		3	29		+	
29		AL	2		1	3	30		+	
30		STL	3	11		3	9		+	
15	5	PCH	6		0	3	17		+	PUNCH RESULT

Figure 3

57.0		10	0.	HLT	3	0.	3	1.	
57.0		10	1.	LDS	3	2.2	20	0.	INPUT ALL DATA
57.0		10	2.2	LDS	3	3.	12	0.	SET-UP FOR RUN
57.0		10	3.	LDS	3	900.	3	4.	
57.0		10	4.	LDS	3	9.	3	5.	PRE-SET E A ADDRESS
57.0		10	5.	LDS	3	901.	3	6.	
57.0		10	6.	SDS	3	11.	3	7.	PRE-SET E B ADDRESS
57.0	7	10	7.	LDS	3	902.	3	8.	
57.0		10	8.	SDS	3	13.	3	9.	PRE-SET E C ADDRESS
57.0		10	9.	LDS	0	0	3	10.	
57.0		10	10.	SDS	6	2	3	11.	WORKING E A
57.0		10	11.	LDS	0	0	3	12.	
57.0		10	12.	SDS	6	3	3	13.	WORKING E B
57.0		10	13.	LDS	0	0	3	14.	
57.0	14	10	14.	SDS	6	4	3	15.	WORKING E C
57.0		10	15.	LDS	3	15.5	15	0.	COMPUTE
57.0		10	15.5	PCH	6	0	3	17.	
57.0		10	17.	RAL	6	2	3	18.	
57.0		10	18.	SL	2	5	3	19.	
57.0		10	19.	TS	3	20.	3	23.	TEST E A
57.0		10	20.	RAL	3	9.	3	21.	
57.0	21	10	21.	AL	2	1	3	22.	
57.0		10	22.	STL	3	9.	0	8002	
57.0		10	23.	LDS	3	900.	3	24.	
57.0		10	24.	SDS	3	9.	3	25.	
57.0		10	25.	RAL	6	3	3	26.	
57.0		10	26.	SL	2	6	3	27.	
57.0		10	27.	TS	3	28.	3	31.	TEST E B
57.0	28	10	28.	RAL	3	11.	3	29.	
57.0		10	29.	AL	2	1	3	30.	
57.0		10	30.	STL	3	11.	3	9.	
57.0		10	31.	LDS	3	901.	3	32.	
57.0		10	32.	SDS	3	11.	3	33.	
57.0		10	33.	RAL	6	4	3	34.	
57.0		10	34.	SL	2	7	3	35.	
57.0	35	10	35.	TS	3	36.	3	1.	TEST E C
57.0		10	36.	RAL	3	13.	3	37.	
57.0		10	37.	AL	2	1	3	38.	
57.0		10	38.	STL	3	13.	3	9.	
57.0		10	900.	LDS	2	10	3	10.	
57.0		10	901.	LDS	2	30	3	12.	
57.0		10	902.	LDS	2	50	3	14.	
57.0	42	12	0.	SDS	3	12.	3	7.	
57.0		12	7.	LDS	3	901.	3	8.	
57.0		12	8.	SDS	6	9	3	9.	
57.0		12	9.	RAU	2	3	3	10.	
57.0		12	10.	AU	2	4	3	11.	
57.0		12	11.	STU	6	0	3	11.2	
57.0		12	11.2	STL	6	7	3	12.	
57.0	49	12	12.	NOP	0	0	0	0	
57.0		12	900.	010	0	0	0	0	
57.0		12	901.	000	0	0	0	80	
57.0		15	0.	SDS	3	36.	3	1.	COMPUTE SIGMA 1, 2, & TAU
57.0		15	1.	RAU	6	2	3	2.	
57.0		15	2.	AU	6	3	3	3.	
57.0		15	3.	AU	6	4	3	4.	SUM E

Figure 4

ASSEMBLY

The assembly program performs a translation from symbolic instructions into actual machine instructions. The resulting program deck is then punched by the computer seven instructions per card with a three digit card sequence number and appropriate identification.

To perform an assembly, the assembly program cards are placed in the card hopper. Immediately following this deck the symbolic instruction cards are entered. These cards are keypunched one symbolic instruction per card and must be in sort on location region major, location sequence minor. The computer performs a sequence check during the read-in and stops if the cards are out of sort. In addition, the symbolic code abbreviation is converted into the actual numeric code on the read-in using the table look-up feature of the 650. As the cards are read, each symbolic instruction is stored in two words of memory. These words are constructed so as to facilitate table look-up in the case of a data address, or an instruction address which refers to the location of another instruction. The last symbolic card is followed by a control card which contains the following information.

1) A 7 digit identification number.

2) The card number which is to be punched on the first card of the resulting program deck.

3) The actual location of the first symbolic instruction.

4) The origin of region 1.

5) The origin of region 2.

6) The origin of region 5.

7) The origin of region 6.

Items 4, 5, 6, and 7 may be assigned by the programmer or the assembly program.

The actual instructions which comprise the coding may begin in any location but usually commence at actual location 0000. The origin of region 2 will then be computed by the machine as the location of the first word beyond the last instruction in the coding. The origin of region 1 is chosen such that the highest region 1 sequence number will be the actual location 1999.

After the control card is read, the origins of the regions are assigned or computed as is necessary and the actual assembly begins. An error may be sensed at this time if a reference is made to an instruction which cannot be found. If no errors are detected, the machine begins punching the finished program deck after the completion of the entire assembly. A maximum of 720 instructions may be assembled at one time.

ASSEMBLY TIME

The time required to perform an assembly having 720 instructions is given below:

Read-in of Assembly Program	.4 minutes
Read-in of Symbolics	3.6 minutes
Assembly	6.0 minutes
Punch-out of Program Deck	1.3 minutes
TOTAL	11.3 minutes

Subsequent assemblies, of course, do not require that the assembly program be re-entered.

SPECIAL DEVICES

One-half time emitter for the read feed.

RELATIVE PROGRAMMING FOR THE IBM TYPE 650

John T. Horner
General Motors Corporation

A method of relative programming has been devised for the IBM Type 650 Computer which facilitates programming of engineering problems. All instructions and data are assigned relative locations and commands are given by mnemonic alphabetical character sets. A 650 Relative Program is used to read relative instructions and punch a card set with actual instructions which is the final problem program, ready for running and checking. Sub-routines may also be referred to by the mnemonic commands and have actual locations assigned by the Relative Program. Present relative sub-routines include floating point arithmetic, matrix floating point arithmetic, formation of elementary functions in fixed point or floating point form, interpolate, differentiate and quadrature sub-routines, and indexing and plotting sub-routines. Additional sub-routines may be added as required.

General Features of System

The Allison Relative Programming System has been designed to permit rapid and efficient preparation of engineering and scientific problems for the Type 650 Computer. The specific requirements for a system to accomplish these purposes are that the system must be easy to learn, it must have wide application and have sufficient flexibility to handle unusual problems and it must use machine time economically. We consider that the last two requirements are adequately satisfied. The system is not, however, readily learned by personnel untrained in computer techniques but is easily mastered by experienced personnel.

The Relative Program for a problem is prepared by using 650 basic instructions in relative form. This is not a new technique, it has been used for programming of IBM 701, IBM 702, Remington Rand Univac and other large and medium size computers. An instruction, itself, has the following form:

Op	Data Address	Transfer Address
XX	XXXX	XXXX

but, to be completely defined, it must be assigned a location. Since the location, data address and transfer address are identical in form, referring ordinarily to storage locations within the computer, these may be treated uniformly in assigning relative locations. A relative location is defined by the following form:

Aa.a XXXX

where Aa.a called the deck number, consists of an alphabetical character (or number) A followed by two numerical digits (or alphabetical characters and XXXX is the relative location within the deck. The starting relative deck location is 0000.

Figure 1 shows the Allison Relative Program Sheet for 650 Computer. Note that the relative instruction, consisting of an instruction location, basic operation, data address and transfer address has the same form as a basic computer instruction. As an aid to memory, the operation codes which are considered to be fixed and not relative are defined by their mnemonic equivalents. Thus, (65) Reset Add to Lower Accumulator becomes: R ADD. Table 1 defines these basic instructions for programming use. Programming using basic relative instructions is exactly the same as using basic absolute instructions but is somewhat slower because there are more numbers and characters to be written.

The power and flexibility of relative programming are due to the use of functional operations or commands. The functional command in the relative program automatically inserts a relative transfer address. The location of the transfer address is the start of a sub-routine which is entered and executed. Then the program normally returns to the next sequential instruction as determined by a program count. The functional command SETPC (Set program count) is given at the beginning of a series of functional commands. This command sets the program count to the address of this instruction location. Each functional sub-routine always exits through a program count sub-routine. This sub-routine increases the program count by 1 and then transfers to the corresponding location. The Allison library of functional sub-routines has been designed such that many engineering problems can be programmed most efficiently using only functional commands. For extensive logic, however, the use of basic instructions can create a much more rapid and efficient program. The programmer can elect to use basic instructions at any time in the program and may return to the sequence of functional commands by giving any functional command.

Operation of Relative Program

Relative program cards are punched from the relative program sheet in the order of the script. Notice that the "Remarks" are also punched. "Remarks" are entered for the following purposes:

a. To serve as an aid to memory. Appropriate comments point out unusual features of the program or program devices, the beginning and end of loops, and, in general, <u>when</u> actions are taken.

b. To serve as a guide to other programmers or engineers, enabling them to use the program, if suitable, in another problem. Thus <u>each</u> program has potential use as a library routine in another problem.

Remarks can be written such that a problem is completely defined by then. Each step in a program can be clearly related to the corresponding step in an equation written in standard notation. A listing of the relative program cards after the program is checked enables programmers to modify or use the program without alteration if suitable in other programs.

After preparing the relative program sheets, the programmer decides on the final storage allocations of the program decks. He enters the absolute starting location for each deck used in the program on the relative program sheets. This information is stored as a table in the relative program. Two other tables are also stored in the relative program, a table of Basic Commands and a table of Functional Commands. The relative program and tables are read into the 650 followed by any number of relative program cards for the problem. The relative program computes an absolute instruction, an absolute location of the instruction and punches a program deck sequentially numbered and containing the correct problem number. Punch speed is 100 cards a minute.

As mentioned previously, functional instructions are actually basic instructions and the functional command normally establishes only a transfer location. Each functional command must use a basic operation command which is established by the programmer. Most functional commands use, however, only a limited number of basic commands and to enable the programmer to think in terms of the operations to be performed instead of program details, additional entries are added to the basic table. These are:

Added Basic Op.	Equivalent Basic Op.
Blank	R ADD
MIN	R SUB
AB	RADDA
MINAB	RSUBA

Thus the command MIN MULT means "floating point multiply minus." The command AB SQRT means "take the square root of the absolute value", etc. Normally, any basic operation may be given as part of a functional command. Certain functional commands require however, particular basic operations which are not suggested as logically pertaining to the functional command. For these cases short sub-routines are introduced which supercede the previously established basic command (usually R ADD) and set the proper one. For example, the functional table look-up sub-routine requires the basic operation TLU. Instead of writing TLU TLU, we write Blank (= R ADD) TLU and the basic operation TLU is then inserted.

To facilitate problem tracing, each functional instruction is signed minus by the relative program.

Features of Functional Sub-Routines

Operands for all sub-routines are stored in the lower accumulator by use of the basic commands R ADD, R SUB, RADDA or RSUBA. The result after each functional command is executed is stored both in the lower accumulator and in a floating point accumulator. There is one binary operation, the functional command R ADD which stores the contents of the lower accumulator in the floating point accumulator. If the operand of any operation is the result of the previous step, the data address in the relative program may be left blank and the relative program automatically inserts the correct address (that of the lower accumulator).

The logical handling of operands as described permits wide use of indexing for indexing operands. The operand of the functional command INDEX is entered into the lower accumulator as in any other functional command. The Index command then adds the lower accumulator to the next sequential instruction as determined by the program count and executes the altered instruction, without changing the instruction to be indexed. By the insertion of one basic step in the program, it is possible to create a command which could be titled "Increase (or Decrease) Index (Temporarily) and Index." A second basic instruction could permanently modify the index before indexing. It is also possible to examine the index and correct it before indexing. Other procedures may also be devised depending on the need and imagination of the programmer. Indexing is widely used in some of the functional commands and is a powerful tool for programming a wide variety of problems.

Table 2 lists the functional command programs which have been prepared at the present time. These include, of course, floating point arithmetic, floating point and fixed point elementary function commands. It is probable that other more uncommon functions such as Bessel's functions will be added if these are required in a problem. Functional commands which are unusual are the commands for floating point matrix arithmetic, the interpolate, differentiate and quadrature commands and the fixed and floating point plot commands.

Matrix arithmetic is handled logically in the same manner as floating point arithmetic. The functional R ADD command is given addressing the first matrix or A matrix. A binary command follows addressing the B matrix and a store command is then given locating the C matrix. This command may immediately be followed by another binary matrix command. All matrix operands are indexable. The commands perform the following matrix operations:

ADDM   Add Matrix

$A + B = C$
Note: (C may replace A or B)

MULTM Post Multiply B by A

$B \cdot A = C$

MULTS Multiply A by Scalar

$B \cdot A = A \cdot B = C$

where B is a scalar. (C may replace A).

INVRT Invert matrix A

The invert command forms A^{-1} replacing the elements of A by the inverted matrix. During the inversion, the determinant of A is also formed for use if required. One or more sets of simultaneous equations may be solved at the same time by the following commands:

Command	Data Address
INVRT	Loc (Acw)
MULTM	Loc (Bcw)
STORM	Loc (Xcw)

where: $X = A^{-1} \cdot B$

Each control word has the form, e.g., R Loc a_{11} C where R is the number of rows, Loc a_{11} is the location of the first element of the a matrix and C is the number of columns of the A matrix. The largest matrix which may be inverted is 40 by 40 with these standard instructions.

Interpolate, differentiate and quadrature command operate on the columns of a matrix. Two commands are given, a STORC (Store Control Word) command followed by the appropriate functional command addressing the argument. The control word has the following form:

C_A Loc_{CW} C_F

C_A - Column number of argument

C_F - Column number of function

Loc_{CW} Location of matrix control word

After the command STORC is given any number of interpolate, differentiate or quadrature commands may follow which use the same argument and function columns for the same matrix. All functions use the interpolation formula:

$$Y = a_i + b_i(X - X_i) + c_i(X - X_i)^2 + d_i(X - X_i)^3$$

$$X_{i+1} < X \leq X_{i+2}$$

$$i = 1, 2 \quad R - 3$$

(If $X > X_{R-1}$, coefficients with the subscript R-3 are chosen which causes extrapolation).

When a functional command is given, comparisons determine:

a. Whether or not the control word has changed. If so, new controls are set.

b. If the control word has not changed, whether or not the next argument X lies in the same interval. If so, previously computed coefficients are used. If not, a search is made to find the correct interval and new coefficients are computed. These functional commands perform the following calculations:

INTPL Interpolate

$$Y = a_i + b_i(X - X_i) + c_i(X - X_i)^2 + d_i(X - X_i)^3$$

DIFF Differentiate

$$\frac{dY}{dX} = b_i + 2c_i(X - X_i) + 3d_i(X - X_i)^2$$

STLOL Set Lower Limit

$$\int_{X_0}^{X_i} YdX = -(X_i - X_o)\Big[a_i + \frac{b_i}{2}(X_o - X_i) + \frac{c_i}{3}(X_o - X_i)^2 + \frac{d_i}{4}(X_o - X_i)^3\Big]$$

STUPL Set Upper Limit

This command causes several integrals to be summed as follows:

$$\int_{X_0}^{X} YdX = \int_{X_o}^{X_i} YdX + \int_{X_{i+1}}^{X_{i+2}} + \cdots + \int_{X_{i+K-1}}^{X_{i+K}} YdX - \int_{X}^{X_{i+K}} YdX$$

$$(X \leq X_{i+K})$$

This sum is taken as the result and also used as the lower limit. If another argument greater than the previous argument is used in a STUPL command, the total value of the integral $\int_{X_0}^{X} YdX$ is obtained.

These functional operations as described are used primarily for rearranging data, either at the beginning of calculation or at the end. For high speed running, where tables need to be consulted many times in the course of calculation, the same type of interpolation formula is used but coefficients are pre-computed and the 650 Table Look-Up Instruction is used for high speed searching.

The command PLOT enables a programmer to plot computed results on an IBM 407 Accounting Machine. It is possible to prepare plots which may be accurately scaled for interpolation or rough plots showing data trends. In starting a plot, the following information must be stored: Y_0, $Y_1 - Y_0$, S, Plot Format

where:

Y_0	- ordinate origin
$Y_1 - Y_0$	height of ordinate
S	scale factor or number of type bars selected minus 1.

Plot Format -

8	X	X	XXXXXXX
P	S	SYM	BIN CON

The P symbol 8 or 0 distinguishes the plot format from alphabetical or numerical information.

S - Space Control { 0 Space 1; 8 Suppress Space }

SYM - Symbol Control

1	.
2	X
3	*
4	□

BIN CON - Binary digits to determine Y print wheel location.

The argument X is read and stored in a selected location for printing during the plot cycle. Each space on the 407 is then as an increment of X. The function Y is now computed or selected and addressed in the command PLOT. This command operates as follows:

$$P = \left(\frac{Y - Y_0}{Y_1 - Y_0}\right) \cdot S$$

P is rounded to an integer. The proper type bar is now selected by:

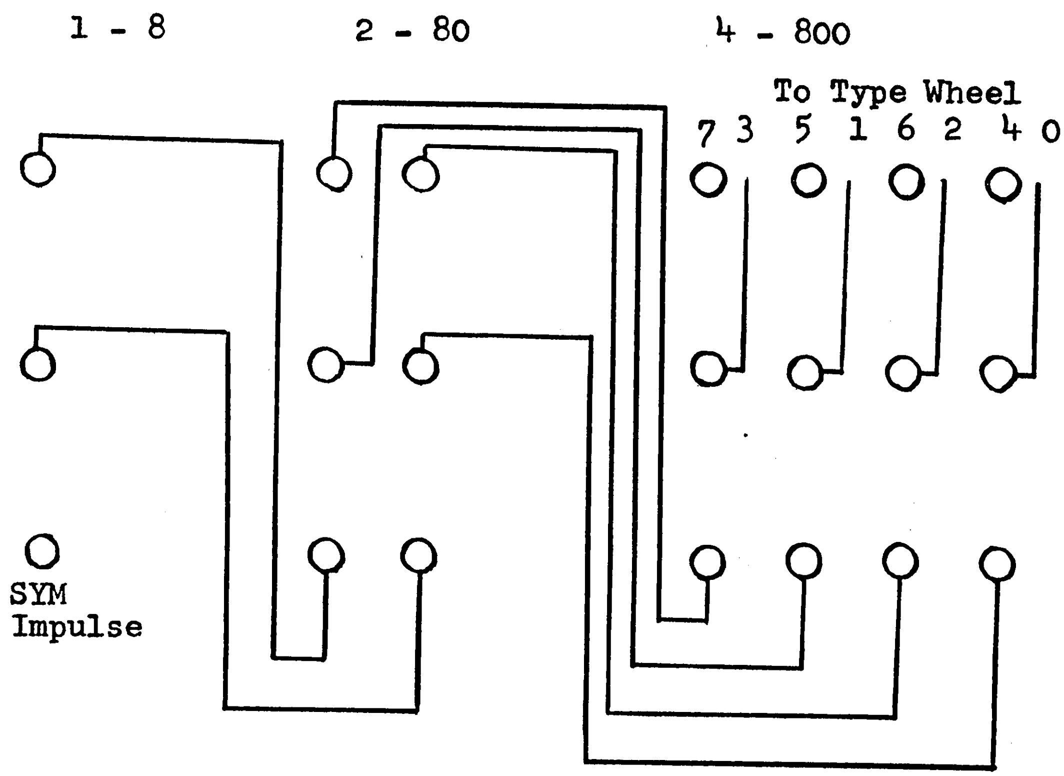
$T = P-(n-1)S$, an integral n being chosen to satisfy the condition: $0 \leqq P-(n-1)S \leqq S$

We define n as the band number. If n is 1, the first band is plotted and:

$$Y_0 \leq Y \leq Y_1$$

If Y is outside the first band, the proper band is automatically selected.

The maximum number of type bars controlled by the present plot program is 101. After T is selected, it is converted to a binary number which is used for selector pick up. The wiring for an 8 point plotter is shown below. An extension is made for the 64 point plotter.



Operational Features of Absolute Program

As mentioned previously, the relative program produces an absolute program which consists of a card deck with absolute locations and instructions and in addition, the relative step. The read program will automatically load any number of these cards until interrupted by a control (Load) Card. A Check Punch program is used for tracing. Check Punching is initiated by a set of Sequence Control cards which cause the program to start in any selected location, to either check punch or run at high speed and then stop the sequence at any selected location. For basic instructions, the Check Punch routine lists the contents of the distributor, upper and lower accumulators together with the instruction and its location. For functional instructions, contents of the data address and contents of the floating point accumulator are shown after the functional instruction is executed. Tracing is controlled by signing instruction, + for basic, - for functional. Any number of sequences may be scheduled, e.g., the program could run at high speed to the beginning of a loop, the loop could be check punched twice, high speed running could complete the loop and then check punching can be resumed.

When checking is complete for a problem, instructions are punched seven per card by the 650 into a sequenced deck. Title cards and column heading cards may be read and punched by the 650. The programmer may also control 407 format including page skips and spacing selection enabling neat and readable presentation of results.

BASIC 650 INSTRUCTIONS

Basic Op.		Operation	Optimum Data Addr	Optimum Trans Addr
R ADD	(65)	Reset-Add to Lower	P+3	P+8
R SUB	(66)	Reset-Sub from Lower	↓	↓
RADDA	(67)	Reset-Add Abs. Value to Lower	↓	↓
RSUBA	(68)	Reset-Sub Abs. Value from Lower	↓	↓
ADD	(15)	Add to Lower	↓	↓
SUB	(16)	Sub from Lower	↓	↓
ADDA	(17)	Add Abs Value to Lower	↓	↓
SUBA	(18)	Sub Abs Value from Lower	↓	↓
RADDU	(60)	Reset-Add to Upper	↓	↓
RSUBU	(61)	Reset-Sub from Upper	↓	↓
ADDU	(10)	Add to Upper	↓	↓
SUBU	(11)	Sub from Upper	↓	↓
MULT	(19)	Multiply	↓	---
DIV	(14)	Divide	↓	---
DIV R	(64)	Divide-Reset Remainder	↓	---
SH RT	(30)	Shift Right	---	P+7+2(S-1)
SH RD	(31)	Shift and Round	---	↓
SH LT	(35)	Shift Left	---	↓
SH CT	(36)	Shift Left and Count	---	↓
STORE	(20)	Store Lower	P+5	P+8
STORU	(21)	Store Upper	P+5	P+8
LOAD	(69)	Load Distributor	P+3	P+6
STORD	(24)	Store Distributor	P+3	P+6
STORA	(22)	Store Lower Data Addr	P+4	P+7
STORI	(23)	Store Lower Instr Addr	P+4	P+7
TLU	(84)	Table Lookup	P+3	P+3+n n(No. of tab arg)
NO OP	(00)	No Operation	---	P+4
STOP	(01)	Stop	---	P+4
BRNZU	(44)	Branch on Non-Zero in Upper	P+4	P+5
BRNZ	(45)	Branch on Non-Zero	P+4	P+5
BRMIN	(46)	Branch on Minus	P+4	P+5
BROV	(47)	Branch on Overflow	P+4	P+5
BR01	(91)			
BR02	(92)			
⋮				
BR09	(99)			
BR10	(90)			
READ	(70)		---	---
PUNCH	(71)		---	---

TABLE I

F-1725

ALLISON RELATIVE PROGRAM SHEET
FOR 650 COMPUTER

PROBLEM NO. ______ PAGE NO. ______

LOCATION		INSTRUCTION						CONSTANT FIELD	REMARKS
DECK	LOC.	OPERATION		DATA ADDR		TRANS ADDR			(Columns 45 to 80)
		BASIC	FUNCTIONAL	DECK	LOC.	DECK	LOC.		

FIGURE 1

FUNCTIONAL COMMAND LIST

Command	Operation	Deck No.	No.Wrds in Deck	Remarks
READ	Read next card	01.0	150	Start location should be 1850. Always required
PUNCH	Punch card			
MOVE	Transfer data set			
INDEX	Index next step			
SETPC	Set program count			
INCPC	Increase program count			
TR	Unconditional transfer			
TRM	Transfer if minus			
TRP	Transfer if plus			
TRO	Transfer if zero			
RADD	Read No. into floating point accumulator			
EQUAL	Store result			
CPOFF	Turn off check punch Store program count			
CPON	Turn on check punch Restore program count			
ADD	Floating point add	02.0	100	Should start in locations XX00 or XX50
MULT	Floating point multiply			
DIV	Floating point divide			
RDIV	Reverse floating point divide			
SQ	Floating point square root			
EXP	Floating point exponential	03.0	175	02.0 required Should start in locations XX00 or XX50
LOG	Floating point logarithm			
SIN	Floating point sine			
COS	Floating point cosine			
ARCTN	Floating point arc tangent			
ADDM	Add matrix	04.0	110	02.0 required Should start in locations XX00 or XX50
MULTM	Matrix multiplication			
MULTS	Scalar multiplication			
STORM	Store matrix			
INVRT	Invert matrix	04.3	125	02.0 required Should start in locations XX00 or XX50
STORC	Store control word	05.0	230	02.0 required
INTPL	Interpolate			
DIFF	Differentiate			
SETLO	Set lower limit for forward integration	05.1	89	02.0 05.0 required
SETUP	Set upper limit			
TLU	Floating point table lookup	06.0	31	02.0 required
CHKPN	Check Punch	07.0	60	

TABLE II

Functional Command List

FSQ	Fixed point square root	08.0	175	Should start in locations XX00 or XX50
FEXP	Fixed point exponential	↓		
FLOG	Fixed point logarithm	↓		
FSIN	Fixed point sine	↓		
FCOS	Fixed point cosine	↓		
FARCT	Fixed point arc tangent	↓		
FTLU	Fixed point table lookup	09.0	37	
PLOT	Floating point plot	10.0	67	02.0 required
FPLOT	Fixed point plot	11.0	50	
ZTLU	Floating point double TLU	15.0	71	02.0 required 06.0

DEVELOPMENT OF A FLOATING DECIMAL ABSTRACT CODING SYSTEM (FACS)

Robert Bosak
Lockheed Aircraft Corporation

Before going into the methods and organization of the work that went into the development of our Abstract Coding System, it might be well to first explain what we mean by such a system. An Abstract Coding System, or as it is sometimes called, a psuedo-code is the use of one machine to simulate another. In some cases this is done in order to check out programs coded for one machine on some other machine. For example; using the 701 to check out problems coded for the 704 before the 704 becomes available. In our case the purpose was to make a fixed decimal machine appear to the programmer as a floating decimal one.

Our planning for the 650 began in April of 1954 with the formation of a steering committee whose purpose was to make basic decisions in regard to the optimum use of the machine. One of the earliest decisions was to use floating decimal point operation whenever possible. Our earlier experience had shown that this was by far the biggest step we could take to increase the accuracy and ease of coding. It was apparent at this stage of the development that our most important consideration would be one of actual machine time. To speed up the subroutines as much as possible it was decided that we would restrict the size and complexity of the list of instructions in our Abstract Coding System. The arithmetic operations finally decided upon were what we thought a minimal list, namely: add, subtract, multiply, divide and negative divide as well as two branch instructions -- branch on minus and branch on relative zero. By June we had a number of outlines of different systems to consider. The variation among these systems was in regard to the word breakdown rather than to the actual operations performed. To determine which of these systems was the best, portions of several problems were actually coded up in each system and a qualitative study made as to which one was easiest to code and to use and which one accomplished the most in the least number of instructions. After these factors were weighed, we decided on a three address system.

10	9	8	7	6	5	4	3	2	1	0
OP	γ	α				β				S

FACS COMMAND (ARITHMETIC)

10	9	8	7	6	5	4	3	2	1	0
Power		NUMBER								S

FLOATING DECIMAL NUMBER

Figure 1

Two of these addresses are full four digit numbers and generally indicate the addresses of the operands. The third address is a one digit number indicating one of ten "result" storages. This digit is used as part of the operation code in the instructions that do not require a third address.

At this point in the development the question arose as to the advisability of including a multiply-add instruction in the list of commands. Some of the members of the committee maintained that the frequency of its usage did not warrant that it be included. Other members of the committee, however, felt just as strongly that it did merit inclusion. To resolve this argument a study was made to determine the frequency of occurrence within a program of various arithmetic operations. This study was based on problems that had been run on our Card Programmed Calculators. The results of this study are shown in Figure 2.

FREQUENCY OF OPERATIONS

OPERATION	Job 1	Job 2	Job 3	Average
Add (not including multiply-add)	.14	.14	.30	.19
Subtract	.14	.15	.02	.10
Multiply (not including multiply-add)	.30	.32	.19	.27
Negative Multiply	.10	.01	.15	.09
Divide	.05	.12	.01	.06
Negative Divide	.00	.00	.00	.00
Multiply-add	.28	.26	.34	.29

Figure 2

The study conclusively bore out the contention that the multiply-add operation was frequent enough to be one of the operations in our command list and, in fact, was so frequent that two different versions of it were finally incorporated. Two other unexpected benefits were derived from this study -- the first was that the negative-divide operation was dropped from the command list since it appeared that it was rarely, if ever used and secondly, the relative frequency of the various operations was used to determine which of the operations were the most important to optimize.

In August, the final list of commands was frozen and coding was begun on the subroutines making up the system. In doing the actual coding of these subroutines the committee, as such, did very little of the work. Instead, the coding was used as training for the people in the department who had no previous experience with stored program machines. Individual members of the committee, however, did spend time in guiding the less experienced personnel and in actually doing a considerable amount of trimming when it came time to pack the subroutines together into a minimum amount of storage. In doing the detail coding, speed of the operation was again our primary concern. Subroutines were reworked in whole or in part several times in an attempt to cut down the time. We went so far as to try to determine the optimum placement of the machine language instruction after a multiplication. Several CPC problems were investigated and a table showing the frequency of occurrence of sums of multiplier digits was constructed. The conclusions that were drawn from this study were that the optimum placement of the next instruction varied considerably from one problem to another and that the maximum difference between an optimum and a non-optimum placement of the next machine language instruction was less than 2 milliseconds. Considering these results it was decided to make multiplications and divisions convenient breaking points; that is, the subroutine would be optimized up to a multiplication and the next instruction placed so as to be optimum with respect to some other portion of the program rather than to the multiplication itself. Another device that was used to speed up the system is based on the fact that there are locations for placement of data which are optimum as far as each of the abstract instructions is concerned. Realizing this, all of the subroutines were constructed so that the optimum placement of data for one instruction was the same or nearly so as for another instruction. We call these groups of storages that are placed optimumly with respect to the system, preferred storages. They vary somewhat from one instruction to another but always include our ten result storages so that the result of one operation will be placed optimumly for any future instruction referring to it.

By the end of November the system was complete and when we came to Endicott the last of November we succeeded in checking out almost all of the program. This was early enough so that we could transfer all of our routine work to the 650 before the first machine was delivered in March.

While we were developing our system, our associates in the Missiles System Division of the company, were preparing other subroutines that would tie in with the FACS system. These subroutines were square root, logarithm, antilogarithm, sine and cosine and arc tangent. These latter subroutines were arranged so that they could be added to or left off of the rest of the system as desired.

In using the FACS system for the last five months we have reached the following conclusions:

1. The command list is a well balanced one and has proven to be easy to use.

2. In training new personnel no difficulty was encountered in teaching the system except for one of the branch instructions (the branch on relative zero instruction in which ease of coding was sacrificed for speed)

3. 80 to 90% of our work now utilizes the FACS system with no compulsion placed upon our programmers to do so.

4. We do not intend to do any further development work for the 650 because of the imminence of our 704. However, if this were not so the only further development that we would consider in regard to the FACS system is to recode portions of it in order to increase speed and/or compactness.

A GENERAL UTILITY SYSTEM FOR THE IBM TYPE 650

The Mathematical Analysis Section
Missile Systems Division
Lockheed Aircraft Corporation

THE MATERIAL CONTAINED HEREIN SHOULD ALLOW EFFECTIVE AND EFFICIENT USAGE OF THE TYPE 650 WITHOUT DUPLICATION OF DEVELOPMENT OR MISDIRECTION OF PRINCIPLES. THIS COLLECTION OF ROUTINES AND METHODS REPRESENTS AN OVERALL PHILOSOPHY OF OPERATION WHICH HAS HAD GOOD SUCCESS IN ACTUAL OPERATION IN AN ENGINEERING AND SCIENTIFIC APPLICATION. THESE ROUTINES HAVE BEEN USED IN MUCH THEIR PRESENT FORM ON 650S NUMBER 10 AND 37 AND WILL BE USED ON A THIRD MACHINE DELIVERED AT THE END OF JULY 1955.

ALL OF THESE ROUTINES ARE OF THE TYPE COMMONLY KNOWN AS UTILITY . THIS MEANS THAT THEY ARE APPLICABLE TO MOST PHASES OF ENGINEERING OR SCIENTIFIC COMPUTING. MANY ARE EQUALLY SUITABLE FOR BUSINESS APPLICATIONS. THE STANDARD CARD FORM[1] AND CONTROL PANELS DESCRIBED ARE VITAL TO INTEGRATED OPERATION OF THIS SYSTEM. INITIAL ADOPTION OF THIS SYSTEM FOR LATER MODIFICATION SHOULD PROVE TO BE A GREAT HELP TO NEW INSTALLATIONS.

THE DEVELOPMENT OF THE FLOATING DECIMAL ABSTRACTION WAS DONE JOINTLY BY THE MATHEMATICAL ANALYSIS DEPARTMENTS OF BOTH THE GEORGIA DIVISION AND THE MISSILE SYSTEMS DIVISION OF LOCKHEED. THE ARITHMETIC PORTION IS DUE TO GEORGIA AND THE SUBROUTINE PORTION TO MSD. LATER DEVELOPMENTS WERE MADE AT MSD IN PACKAGING THE SYSTEM AND PUTTING TRACING UNDER CONTROL OF THE CONSOLE. THEREFORE FACS AT GEORGIA AND FLAIR AT MSD ARE SOMEWHAT DIFFERENT IN OPERATION. FOR THIS REASON THE ENTIRE SYSTEM IS PRESENTED HERE AS MSD USES IT - DESPITE POSSIBLE DUPLICATION IN CERTAIN RESPECTS OF THE WORK OF THE GEORGIA PEOPLE.

IT MAY BE NOTICED THAT THE MAJORITY OF THESE ROUTINES ARE NOT WHAT ARE COMMONLY TERMED ELEGANT. EXCESSIVE POLISHING WOULD NOT GAIN US VERY MUCH IN MACHINE SPEED AND WOULD CERTAINLY LOSE EFFORT THAT HAD BETTER BE PUT TO DOING USEFUL COMPUTING WORK. THESE ROUTINES WORK AND THEY WORK SUCCESSFULLY. THE MOST IMPORTANT THING IS THAT THEY ARE AVAILABLE TO ANYONE FOR IMMEDIATE USE. CREDITS FOR THE VARIOUS ITEMS ARE AS FOLLOWS

ARITHMETIC FLAIR-FACS INCLUDING TRACE	GEORGIA MATH ANALYSIS DEPT.
FLAIR COMPILATION AND EDITING	ED DODGE
FLAIR SUB-ROUTINE SQUARE ROOT	ROBERT BEMER
FLAIR SUB-ROUTINE LOG-ANTILOG	IRENE BROWN AND JACK ANTCHAGNO
FLAIR SUB-ROUTINE SINE-COSINE	ALBERT PODVIN
FLAIR SUB-ROUTINE ARCTANGENT	CHARLES WIMBERLEY
MACHINE LANGUAGE TRACE USABLE WITH FLAIR	RAY CIANCI
REGIONAL ASSEMBLY ROUTINE	RAY CIANCI
PUNCH DRUM FROM α TO β	DON JACKSON
PUNCH β EIGHTHS OF THE DRUM	DON JACKSON
TYPE 407 UTILITY PANEL	RICHARD MIDDLETON
TYPE 533 UTILITY PANEL	RICHARD MIDDLETON
FIVE-FIELD LOAD ROUTINE AND CARD FORM	ROBERT BEMER
FLAIR TO FIXED DECIMAL ROUTINE	ROBERT BEMER AND ELAINE GATTEN

1. Other companies may temporarily order the card form from IBM in San Jose, California, if they so desire.

FIVE-FIELD LOAD ROUTINE

THIS TYPE 650 LOADING ROUTINE IS DESIGNED TO LOAD FIVE WORDS PER CARD IN RANDOM ADDRESSES. THE FORMAT IS THAT LABELED NUMBER 1 ON THE STANDARD 650 CARD FORM. A FIVE-WORD CARD WAS CHOSEN ARBITRARILY TO EFFECT THE MOST EFFICIENT LOADING WITH A MINIMUM OF RESTRICTIONS. THIS ROUTINE IS BELIEVED TO BE THE SIMPLEST IN OPERATION AND CAN LOAD THE ENTIRE MEMORY IN 2 MINUTES.

A LOAD-IDENTIFICATION CARD CONTAINING THE SIX INSTRUCTIONS OF THE LOADING ROUTINE MUST PREFACE ANY ROUTINE. 8000 IS SET TO 70 1901 XXXX. DEPRESS THE COMPUTOR RESET AND PROGRAM START BUTTONS. PLACE THE ROUTINE IN THE READ HOPPER OF THE TYPE 533 AND DEPRESS THE READ START BUTTON. THE LOAD-IDENTIFICATION CARD IS READ UNDER THE CONTROL OF 8000 AND THE NEXT INSTRUCTION WILL BE TAKEN FROM 1901. THIS INSTRUCTION IS ONE OF THOSE READ IN FROM THE LOAD-HUB CARD AND CALLS FOR THE READING OF THE FIRST FIVE-FIELD LOADING CARD. THE NEXT INSTRUCTION IS TAKEN FROM 1902 AND RANDOM LOADING PROCEEDS BY SUCCESSIVE LOAD AND STORE DISTRIBUTOR COMMANDS. THE CYCLICAL PATTERN OF LOADING IS EVIDENT BY TRACING THE INSTRUCTIONS. THE O AND I PARTS OF THE STORE DISTRIBUTOR COMMANDS ARE EMITTED ON THE TYPE 533 PANEL. THE DIAGRAM OF THE TYPE 533 UTILITY PANEL SHOWS THIS WIRING IN THE C READ POSITION.

THE ONLY RESTRICTION OF THIS SYSTEM IS THAT THE LAST INSTRUCTION LOADED IN MEMORY IS THE FIRST TO BE OBEYED IN THE ROUTINE. THIS IS ACCOMPLISHED BY A 12 PUNCH IN THE UNITS POSITION OF THE A PART OF ANY OF THE FIVE FIELDS. THIS PUNCH TRANSFERS A CO-SELECTOR WHICH REPLACES THE I PART OF THE STORE DISTRIBUTOR COMMAND BY THE D PART. THUS THE LAST INSTRUCTION IS LOADED INTO ITS ADDRESS AND THE LOAD ROUTINE IS DISRUPTED SO THAT THIS INSTRUCTION IS THE NEXT TO BE OBEYED. THIS AUTOMATICALLY STARTS THE PROGRAM UPON COMPLETION OF LOADING. TO RESTART THE PROGRAM ONCE IT HAS BEEN LOADED IT IS NECESSARY TO USE ONLY THE LOAD-IDENTIFICATION CARD AND THE CARD CONTAINING THAT FIRST INSTRUCTION TO BE OBEYED.

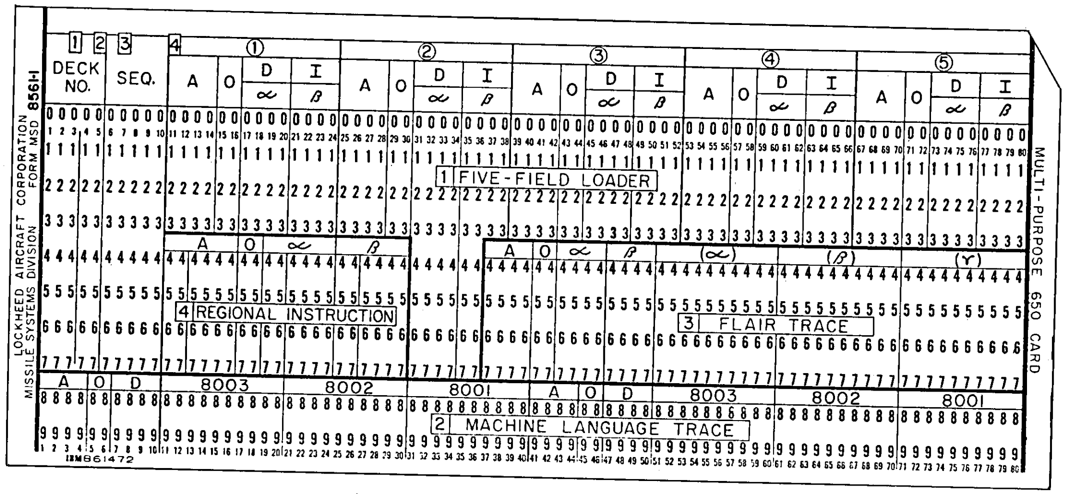
LOAD-IDENTIFICATION CARD		12-PUNCH IN COLUMN 1	
WORD 1	70 1951 1902+	WORD 5	69 1958 1957+
WORD 2	69 1952 1951+	WORD 6	69 1960 1959+
WORD 3	69 1954 1953+	WORD 7	10 8001 1965+
WORD 4	69 1956 1955+	WORD 8	35 0001 1966+

NOTE --- WORDS 1 THRU 8 ENTER ADDRESSES 1901 TO 1908 RESPECTIVELY. WORDS 7 AND 8 IN STORAGES 1907 AND 1908 ARE USED IN FLAIR. THEY MUST BE ON THE LOAD-IDENTIFICATION CARD TO PRESERVE THEM IN CASE THE LOAD-IDENTIFICATION CARD IS USED AFTER FLAIR IS ALREADY ON THE DRUM.

A O D AND I ARE READ FROM EACH FIELD OF THE FIVE-FIELD LOAD CARD SO THAT STORAGES 1951 THRU 1960 ARE FILLED AS FOLLOWS

A	O	D	I	A	O	D	I	A	O	D	I
1951	24	A_1	1903	1955	24	A_3	1905	1959	24	A_5	1901
1952	O_1	D_1	I_1	1956	O_3	D_3	I_3	1960	O_5	D_5	I_5
1953	24	A_2	1904	1957	24	A_4	1906				
1954	O_2	D_2	I_2	1958	O_4	D_4	I_4				

SINCE THE I PART OF 8000 IS NOT USED IN THIS ROUTINE THESE FOUR POSITIONS MAY BE USED AS EFFECTIVE SENSE SWITCHES BY SETTING THEM AT 8 OR 9 AND INTERROGATING 8000 DURING THE ROUTINE. 8000 MAY ALSO BE SET EITHER + OR - AND INTERROGATED FOR DECISION. DO NOT ALTER THE SETTING OF 8000 SWITCHES WITHOUT FIRST DEPRESSING THE PROGRAM STOP BUTTON.



DECK 033.01 PUNCH β EIGHTHS OF THE DRUM

THIS ROUTINE PUNCHES β EIGHTHS OF THE DRUM IN SUCH A FASHION THAT THE LIST IS REPRESENTATIVE OF DRUM LAYOUT. β MAY VARY FROM 1 TO 8. PUNCHING STARTS WITH THE CONTENTS OF α. TO OPERATE THIS ROUTINE

1. SET 8000 TO 70 1901 XXXX
2. PUT LOAD-IDENTIFICATION CARD IN FRONT AND LOAD DECK 033.01
3. BEFORE DEPRESSING END-OF-FILE CHANGE CONSOLE TO 00 α β
4. DEPRESS END-OF-FILE BUTTON

A	O	D	I	A	O	D	I	A	O	D	I
0997	01	8000	8000	0912	69	8003	0913	0913	23	0991	0914
0914	16	8001	0915	0915	15	0990	0916	0916	10	0992	8002
0917	24	0978	0918	0918	22	0977	0919	0919	15	8003	8002
0920	24	0980	0921	0921	22	0979	0919	0923	24	0982	0924
0924	22	0981	0919	0926	24	0984	0927	0927	22	0983	0919
0929	24	0986	0930	0930	22	0985	0931	0937	10	8001	0938
0938	35	0008	0939	0939	44	0946	0940	0940	65	0991	0941
0941	16	0994	0942	0942	45	0943	0997	0943	20	0991	0944
0944	65	0996	0945	0945	15	0995	0916	0946	65	0995	0916
0990	69	0000	0917	0992	00	0050	0003	0993	00	0199	0012
0994	00	0000	0001	0996	00	0200	0000	0931	71	0977	0932
0932	16	0993	0933	0933	20	0995	0934	0934	30	0004	0936
0936	60	8002	0937	0911Y	65	8000	0912				

PUNCHING IS IN THE FIVE-FIELD LOADER FORM FROM THE C PUNCH OF THE TYPE 533 UTILITY PANEL.

DECK 033.02 PUNCH DRUM FROM α TO β

OPERATION INSTRUCTIONS FOR THIS ROUTINE ARE THE SAME AS FOR DECK 033.01. PUNCHING IS ALSO ON THE FIVE-FIELD LOADER FORM BUT SEQUENTIAL ON EACH CARD.

A	O	D	I	A	O	D	I	A	O	D	I
1911	69	1946	1912	1920	24	1930	1921	1939	46	1940	1941
1912	35	0004	1913	1921	21	1929	1945	1940	10	8001	1942
1913	22	1910	1914	1922	24	1932	1923	1942	10	1949	8003
1914	35	0002	1943	1923	21	1931	1945	1941	01	0000	0000
1943	60	8003	1915	1924	24	1934	1925	1945	10	8002	8003
1915	35	0004	1916	1925	21	1933	1945	1946	69	0000	0000
1916	15	1947	1917	1926	24	1936	1944	1947	00	0001	0002
1917	10	1948	8003	1944	21	1935	1937	1948	69	0000	1918
1918	24	1928	1919	1937	71	1927	1938	1949	00	0000	9992
1919	21	1927	1945	1938	11	1910	1939	1900Y	67	8000	1911

DECK 033.05 - MACHINE LANGUAGE TRACING ROUTINE

ALL MACHINE LANGUAGE COMMANDS ARE ANALYZED IN OPERATIONAL ORDER. THE LOCATION ADDRESS - OPERATION CODE - DATA ADDRESS AND THE THEORETICAL CONTENTS OF 8003-8002 AND 8001 ARE PUNCHED. TWO SUCH INSTRUCTIONS ARE PUNCHED PER CARD. LISTING OF THESE CARDS ENABLES STEP-WISE FOLLOWING OF THE RESULTS OF AN ACTUAL PROGRAM. THE CARD FORMAT IS THAT LABELED NUMBER 2 ON THE STANDARD 650 CARD FORM.

THE TRACING ROUTINE MAY BE STORED IN ANY TWO ADJACENT DRUM BANDS. THE ATTACHED CODING IS LOCATED FROM 1200 TO 1299. THE ROUTINE MAY BE EITHER PLACED ON THE DRUM PREVIOUSLY OR ACCOMPANY THE PROGRAM TO BE TRACED. IN EITHER CASE A TRACING CONTROL CARD MUST BE INSERTED IN THE PROGRAM DECK BEYOND THE LOADING OF THAT INSTRUCTION WITH WHICH TRACING BEGINS. IF THE TRACING CONTROL CARD IS LOADED SEPARATELY A_1 CANNOT BE 800X NOR CAN THE ORIGINAL INSTRUCTION IN A_1 CONTAIN 800X. TRACING MAY START AT ANY PLACE ALONG THE PROGRAM. THE PROGRAM CONTINUES AT MACHINE SPEED WITHOUT TRACING AFTER THE LAST ADDRESS TRACED IS REACHED. SYMBOLS FOR THIS ROUTINE ARE

- A_1 - ADDRESS OF FIRST INSTRUCTION TO BE TRACED.
- I_1 - THE INSTRUCTION AT ADDRESS A_1.
- α_1 - I_1 IS SENT TO ADDRESS α_1. USUALLY $\alpha_1 = A_1$ HOWEVER IF $\alpha_1 \neq A_1$ TRACING WILL BEGIN WHENEVER THE ADDRESS A_1 IS AGAIN INSTRUCTED. THIS FEATURE FACILITATES LOOP TRACING.
- A_n - ADDRESS OF LAST INSTRUCTION TO BE TRACED.

THE TRACING CONTROL CARD IS A FIVE-FIELD LOADER. IT SHOULD CONTAIN THE FOLLOWING THREE WORDS FOR USING ONLY MACHINE LANGUAGE TRACE.

A	O	D	I	A	O	D	I	A	O	D	I
1298	←	I_1	→	1299	65	α_1	A_n	A_1	24	1284	1265

IT SHOULD CONTAIN THE FOLLOWING TWO WORDS WHEN TRACING IS TO BEGIN IN FLAIR AND CONTINUE ALTERNATELY IN MACHINE LANGUAGE AND FLAIR.

A	O	D	I	A	O	D	I
1290	69	8000	1243	1677	16	1834	1239

IF TRACING IS TO BEGIN WITH MACHINE LANGUAGE AND ALTERNATE WITH FLAIR ALL FIVE OF THESE WORDS MUST BE ON THE TRACING CONTROL CARD WITH A_n = 1735. COMPOSITE TRACING OF BOTH MACHINE LANGUAGE AND FLAIR COMMANDS IS UNDER THE CONTROL OF THE HUNDREDS POSITION OF 8000D. WHEN 8000 READS 70 1901 XXXX TRACING WILL BE OPERATIVE IN MACHINE LANGUAGE UNTIL THE PROGRAM GOES TO FLAIR. TRACING WILL NOT RESUME UPON RETURN TO MACHINE LANGUAGE. WHEN 8000 READS 70 1801 XXXX TRACING WILL CONTINUE THRU BOTH M. L. AND FLAIR.

A	O	D	I	A	O	D	I	A	O	D	I
1200	69	1249	1201	1229	22	1233	1218	1259	00	1227	1215
1201	23	1285	1202	1230	60	1282	1231	1260	71	1277	1237
1202	60	1282	1242	1231	15	1283	1232	1261	65	1254	8002
1203	19	1283	1204	1232	69	1284	1233	1262	99	9999	9999-
1204	46	1207	1205	1233	65	0000	1234	1265	20	1283	1266
1205	60	1282	1206	1234	24	1249	1235	1266	21	1282	1267
1206	15	1283	1211	1235	65	1233	1236	1267	65	1299	1268
1207	60	1283	1208	1236	30	0004	1287	1268	69	1258	1269
1208	19	1262	1209	1237	69	1221	1238	1269	23	1258	1270
1209	15	1282	1210	1238	24	1218	1240	1270	69	1273	1271
1210	14	1262	1211	1239	20	1249	1248	1271	22	1273	1272
1211	21	1278	1212	1240	69	1249	1241	1272	69	1298	1273
1212	20	1279	1213	1241	24	1285	1202	1273	24	0000	1274

MACHINE LANGUAGE TRACING ROUTINE -- CONTINUED

A	O	D	I	A	O	D	I	A	O	D	I
1213	69	1284	1214	1242	45	1203	1205	1274	24	1249	1275
1214	24	1280	1285	1243	97	1244	1245	1275	30	0004	1276
1215	24	1284	1216	1244	45	1293	1291	1276	20	1277	1287
1216	20	1283	1217	1245	65	1247	1246	1286	88	8080	0000
1217	21	1282	1225	1246	20	1218	1293	1287	69	1249	1288
1218	71	1277	1219	1247	00	0000	1237	1288	23	1281	1289
1219	69	1220	1222	1248	24	1283	1236	1289	16	1258	1290
1220	69	1221	1222	1250	39	9000	0000	1290	45	1293	1291
1221	71	1277	1219	1251	49	9000	0000	1291	65	1260	1292
1222	24	1218	1223	1252	89	9000	0000	1292	20	1218	1293
1223	69	1281	1224	1253	99	9000	0000	1293	65	1261	1294
1224	24	1277	1230	1254	65	1259	1200	1294	69	1249	1295
1225	65	1249	1226	1255	65	1259	1296	1295	84	1254	8002
1226	35	0004	1228	1256	65	1259	1200	1296	69	1249	1297
1227	65	1249	1228	1257	65	1259	1296	1297	22	1285	1201
1228	69	1233	1229	1258	00	0065	1735				

DECK 033.06 - REGIONAL ASSEMBLY ROUTINE

REGIONAL CODING IS DESIRABLE FOR ABSTRACT SYSTEMS. INDEXED REGIONAL ADDRESSES ARE ASSIGNED WHICH CAN BE CONVENIENTLY CONVERTED TO MACHINE ADDRESSES. LONG PROGRAMS MAY BE BROKEN INTO SECTIONS WHICH MAY BE CODED CONCURRENTLY AND SEQUENTIALLY AS IF STARTING AT ADDRESS 0000. EACH SECTION IS ASSIGNED TO TRUE DRUM ADDRESSES WITH THE ASSEMBLY ROUTINE WHEN THE PROGRAMMING IS COMPLETED. C2 0352 IS AN EXAMPLE OF A REGIONALLY CODED ADDRESS. C2 IS THE ADDRESS INDEX AND 0352 IS THE ADDRESS WITHIN THE C2 REGION. ALPHA-NUMERIC INDICES FROM A0-A9 TO H0-H9 ARE ALLOWABLE.

ONE REGIONAL INSTRUCTION IS PUNCHED PER CARD. THE FORMAT IS NUMBER 4 OF THE STANDARD 650 CARD FORM. THE LOCATIONS OF REGIONAL INSTRUCTIONS AND THE REGIONS THEMSELVES DO NOT HAVE TO BE SEQUENTIALLY ORDERED. A DUMMY INSTRUCTION WITH THE INDEX ADDRESS I0 MUST FOLLOW THE LAST INSTRUCTION OF THE LAST REGION TO BE ASSEMBLED.

RELOCATION OF ANY INDEXED ADDRESS TO THE TRUE DRUM ADDRESS IS ACCOMPLISHED BY SPECIFYING THE INCREMENT BY WHICH THE ADDRESS PART IS TO BE ADJUSTED AND THE LAST INDEXED INSTRUCTION TO BE SO ADJUSTED. THE ASSEMBLY ROUTINE WILL PUNCH THE DESIRED ASSEMBLED PROGRAM FROM THE C POSITION OF THE TYPE 533 UTILITY PANEL ONTO THE STANDARD FIVE-FIELD LOAD CARD.

INSERT ADDITIONAL REGIONAL INSTRUCTIONS INTO A COMPLETED REGION BY ADDRESSING AS MANY AS ARE NEEDED WITH THE SAME ADDRESS AS THE INSTRUCTION THEY FOLLOW. PLACE THEM IN THE PROGRAM DECK IN THIS ORDER. CONTROL CARD INFORMATION MUST BE ADJUSTED ACCORDINGLY. DELETION IS COMPARABLE TO INSERTION EXCEPT THAT THE UNDESIRED INSTRUCTION CARDS ARE REMOVED. THESE ALTERATIONS AND EACH REGIONAL INDEX USED MUST BE REPRESENTED WITH CONTROL INFORMATION. CONTROL WORDS ARE LOADED ON FIVE-FIELD LOADERS IN SEQUENTIAL ADDRESSES STARTING WITH 1000. AN EXAMPLE OF AN ASSEMBLY CONTROL CARD IS

A	O	D	I	A	O	D	I	A	O	D	I	
1000	B2	0315	0100	1001	B5	0106	0500	1002	D3	0021	0620	ETC.

O IS THE ALPHA-NUMERIC ADDRESS INDEX OF THE REGION
D IS THE LAST REGIONALLY INDEXED ADDRESS OF THAT REGION
I IS THE INCREMENT TO BE ADDED TO ALL ADDRESSES IN THAT REGION

REGIONAL ASSEMBLY ROUTINE -- CONTINUED

CARDS ARE PLACED IN THE TYPE 533 IN THE FOLLOWING ORDER

1. LOAD-IDENTIFICATION CARD
2. DECK 033.06 - REGIONAL ASSEMBLY ROUTINE
3. ASSEMBLY CONTROL CARDS AS NEEDED
4. STARTER CARD - 0500Y 65 0807 0501 IN FIELD 1.
5. REGIONALLY-CODED PROGRAM - ONE INSTRUCTION PER CARD

A	O	D	I
0500	65	0807	0501
0501	35	0001	0502
0502	20	0817	0503
0503	20	0818	0504
0504	21	0819	0505
0505	65	0803	0506
0506	20	0559	0508
0508	70	0401	0509
0509	65	0401	0510
0510	35	0002	0511
0511	21	0816	0512
0512	30	0001	0513
0513	11	0807	0514
0514	46	0515	0615
0515	65	0817	0516
0516	16	0816	0517
0517	45	0580	0518
0518	65	0818	0519
0519	16	0401	0520
0520	45	0521	0525
0521	46	0522	0641
0522	24	0818	0523
0523	65	0802	0524
0524	21	0819	0528
0525	65	0819	0526
0526	15	0806	0527
0527	20	0819	0528
0528	65	0808	0529
0529	69	0401	0530
0530	84	1000	8002
0531	69	8003	0532
0532	23	0811	0533
0533	65	0809	0534
0534	69	0402	0530
0600	20	0782	0603
0536	69	8003	0537
0537	23	0812	0538
0538	65	0810	0539
0539	69	0403	0530
0810	65	1000	0541
0541	69	8003	0542
0542	23	0813	0543
0543	65	0812	0544
0544	35	0004	0545
0545	15	0402	0546
0546	69	8003	0547
0547	22	0821	0548
0548	65	0403	0549
0549	30	0004	0550
0550	15	0813	0551
0551	69	8003	0552
0552	23	0822	0553
0553	65	0811	0554
0554	35	0004	0555
0555	15	0401	0556
0556	15	0819	0557
0557	69	8003	0558
0558	22	0820	0559
0559	24	0777	0560
0560	65	0559	0561
0561	15	0806	0562
0562	69	0570	0563
0563	22	0570	0564
0564	65	0405	0565
0565	46	0566	0568
0566	66	0821	0567
0567	16	0822	0642
0568	65	0821	0569
0569	15	0822	0642
0570	20	0778	0571
0571	65	0570	0572
0572	16	0804	0573
0573	45	0574	0577
0574	15	0805	0575
0575	69	0559	0576
0576	22	0559	0508
0577	71	0777	0578
0578	65	0803	0579
0579	20	0559	0508
0580	24	0817	0581
0581	69	0401	0582
0582	24	0818	0583
0583	65	0559	0584
0584	21	0819	0585
0585	16	0803	0586
0586	45	0587	0528
0587	16	0802	0588
0588	45	0589	0593
0589	16	0802	0590
0590	45	0591	0597
0591	16	0802	0592
0592	45	0605	0601
0593	65	0801	0594
0594	10	0800	0595
0595	21	0779	0596
0596	20	0780	0599
0597	65	0801	0598
0598	10	0800	0599
0599	21	0781	0600
0601	65	0801	0602
0602	10	0800	0603
0603	21	0783	0604
0604	20	0784	0607
0605	65	0801	0606
0606	10	0800	0607
0607	21	0785	0608
0608	20	0786	0609
0609	71	0777	0610
0610	65	0803	0611
0611	20	0559	0528
0615	65	0559	0616
0616	16	0803	0617
0617	45	0618	0641
0618	16	0802	0619
0619	45	0620	0624
0620	16	0802	0621
0621	45	0622	0628
0622	16	0802	0623
0623	45	0636	0632
0624	65	0801	0625
0625	10	0800	0626
0626	21	0779	0627
0627	20	0780	0630
0628	65	0801	0629
0629	10	0800	0630
0630	21	0781	0631
0631	20	0782	0634
0632	65	0801	0633
0633	10	0800	0634
0634	21	0783	0635
0635	20	0784	0638
0636	65	0801	0637
0637	10	0800	0638
0638	21	0785	0639
0639	20	0786	0640
0640	71	0777	0641
0641	01	0000	0500
0642	15	0404	0570
0800	00	1960	0000
0801	99	9999	9999
0802	00	0002	0000
0803	24	0777	0560
0804	20	0786	0571
0805	20	0787	0571
0806	00	0001	0000
0807	00	0000	0009
0808	65	1000	0531
0809	65	1000	0536

DECK 033.18 FLAIR TO FIXED DECIMAL ROUTINE

THIS ROUTINE TAKES A DECK OF LOAD HUB CARDS CONTAINING EIGHT FLAIR NUMBERS OF THE FORM PP .XXXXXXXX AND CONVERTS THEM TO NINE-DIGIT FIXED DECIMAL NUMBERS. THE POSITIONS OF THE DECIMALS ARE DETERMINED BY A LOAD HUB CONTROL CARD WHICH ALSO CONTAINS THE DECK NUMBER.

THE FIRST FIELD AAAAA 000 5B
THE SECOND THROUGH EIGHTH FIELDS 0000 0000 5B
WHERE
AAAAA IS THE DECK NUMBER
B IS THE NUMBER OF WHOLE NUMBERS IN A NINE-DIGIT FIELD

THE DECK IS PLACED INTO THE TYPE 533 IN THE FOLLOWING ORDER.

1. LOAD-IDENTIFICATION CARD
2. DECK 033.18
3. LOAD HUB CONTROL CARD
4. LOAD HUB DETAIL CARDS

THE DECK NUMBER IS SPLIT OFF FROM THE FIRST FIELD AND STORED IN 0077

A	O	D	I	A	O	D	I	A	O	D	I
0039	70	0042	0042	0017	11	8003	0025	0095	21	0001	0015
0042	60	0001	0011	0025	24	0077	0089				
0011	30	0002	0017	0089	35	0002	0095				

FIELDS ONE THROUGH EIGHT ON THE DETAIL CARDS ARE CONVERTED TO FIXED DECIMAL AS SPECIFIED AND STORED IN 0078 THROUGH 0085 RESPECTIVELY.

A	O	D	I	A	O	D	I	A	O	D	I
0015	70	0064	0064	0062	20	0067	0070	0021	16	0024	0030
0064	65	0067	0071	0070	16	0073	8002	0030	20	0033	0036
0071	15	0074	0029	0073	00	0050	0050	0036	65	0040	0009
0067	65	0050	0063	0013	18	0072	0027	0088	65	0091	0096
0061	65	0050	0063	0027	35	0004	0037	0096	16	0033	0038
0074	00	0001	0000	0037	46	0073	0041	0038	45	0064	0050
0029	20	0047	8002	0044	31	0001	0033	0050	69	0061	0014
0063	35	0002	0069	0041	15	0044	0049	0014	24	0067	0020
0069	21	0072	0075	0049	20	0009	0012	0020	71	0077	0015—PCH
0075	20	0040	0094	0012	65	0067	0021	0091	20	0085	0088
0094	65	0047	0062	0024	44	9972	9975				

PUNCHING FOR THIS ROUTINE IS NOT ON THE TYPE 533 UTILITY PANEL

TYPE 407 UTILITY PANEL

THIS PANEL WILL LIST THE FOUR TYPES OF CARDS WHOSE FORMATS ARE ON THE STANDARD 650 CARD FORM. PRINTING OF THE SELECTED FORM AND APPROPRIATE HEADING IS AUTOMATIC WITH THE 12 PUNCH IN COLUMN 3 5 7 OR 11. ALWAYS TAKE A FINAL TOTAL BEFORE PRINTING. PREFACE LIST DECKS BY A BLANK CARD. THIS AUTOMATICALLY CAUSES A SKIP TO THE NEXT PAGE AND HEADS BEFORE PRINTING. PRINTING IS BASICALLY 50-10. THIS IS CONVENIENT FOR PRINTING DRUM PUNCH-OUT IN DRUM FORMAT. THE FIVE-FIELD LOADERS LIST WITH α AND β IN THE HEADING IF ALTERATION SWITCH 1 IS NORMAL. THE HEADING CONTAINS THE NORMAL D AND I IF THIS SWITCH IS TRANSFERRED.

THE LOCKHEED 407S FOR MATHEMATICAL WORK HAVE SPECIAL TYPE WHEELS AS FOLLOWS

4-8	α	3-8	+	0-1	∫
0-4-8	β	0-3-8	ω	12	γ
11-4-8	Σ	11-3-8	ρ		
12-4-8	Δ	12-3-8	•		

WIRING FOR THIS PANEL IS SHOWN ON A 407 BOARD DIAGRAM WHERE CONVENIENT. OTHER WIRING IS LISTED BELOW BY TERMINALS ACCORDING TO THE DIAGRAM INDEX.

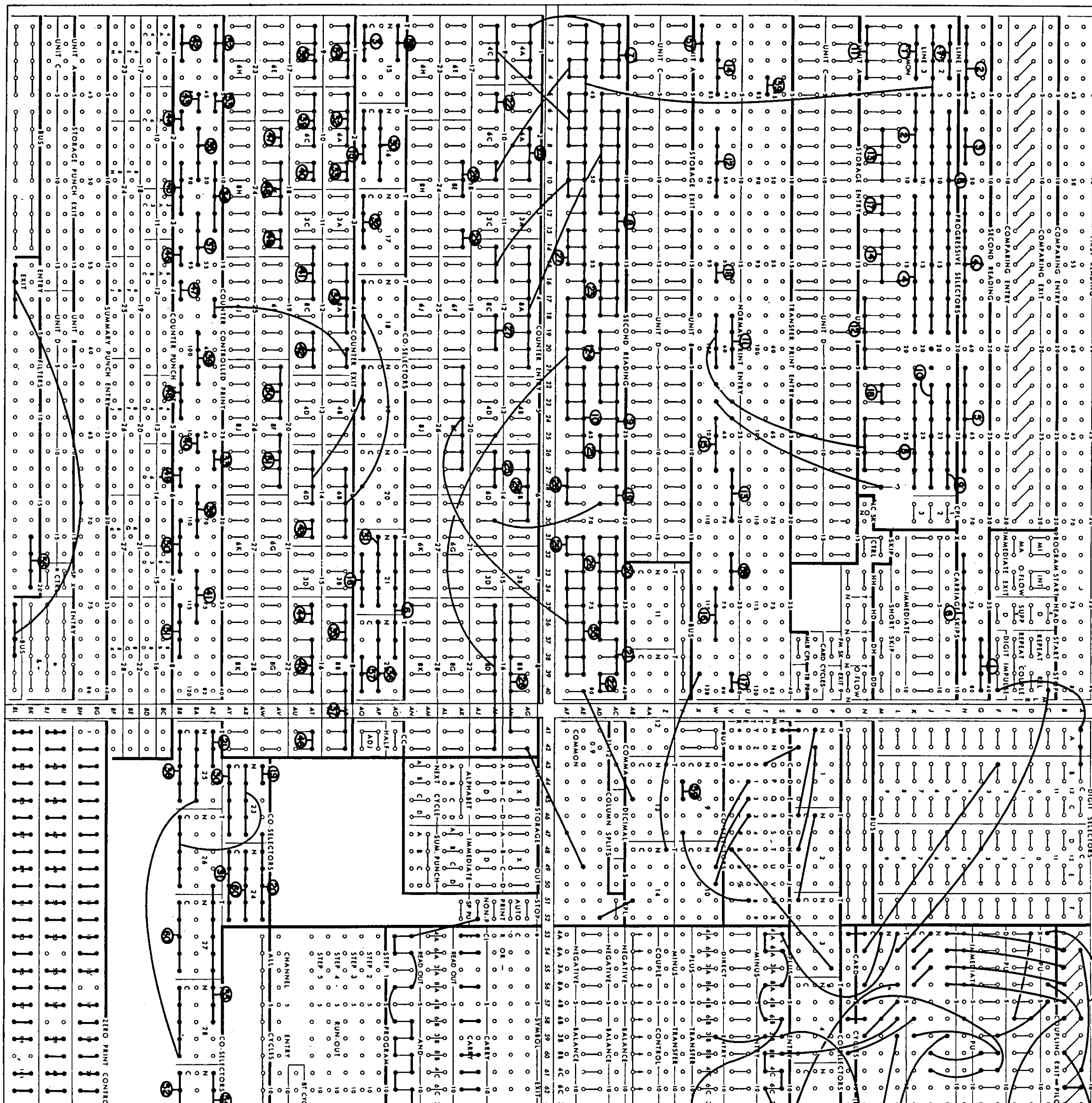
A59- I30	Q54- R58	Z49- X25	AD13-AG05	AO06-AF28	AQ36- W39
A63-AC52	R37- G64	Z50- X18	AD41- O41	AO10- A31	AQ37- W40
A67- C55	R43- R61	Z51- X04	AD42- O45	AO11-AZ19	AQ79- R39
A78- J30	R44- R62	Z52-BG78	AD43- O49	AO14-AZ24	AT18-BA40
D57- C10	R66-AD48	AA33-AK08	AD44- N41	AO15-AZ28	AZ46-AE28
E33- N64	R67- V17	AA34-AK12	AE45-AG06	AO16-AZ29	AZ47-AE29
E34- J55	R68-AZ76	AA35-AK28	AE46-AG31	AO18-AZ32	AZ65-AR14
F41-AG32	R69-AZ77	AA36- I20	AE47-AI29	AO21-AZ38	AZ66-BI54
G65-AP79	R71-AW66	AA37- J20	AE48-AI31	AO24-BA01	AZ67-BI70
G73-AW68	R72-AW67	AA38-AO04	AE50-AI08	AO25- L04	AZ68-BL60
I58- S68	S20- P56	AA39- K20	AE51-AI17	AO26-AD47	AZ69-BI38
K30- B75	S26- P58	AA40-AO08	AE52-AI21	AO27-AD49	AZ71-BI39
K53-AQ67	S28- R47	AA41-AO39	AF01-AI07	AO28- M28	AZ72- V23
K57- E11	S33- P59	AA42-AW45	AF25-AK07	AO29-AD51	AZ73- M15
L72- X27	S49- P62	AA43-AZ44	AF41- A03	AO30-AD46	AZ74-BA47
M05- V24	T02- P57	AA44-AZ48	AF42- A05	AO37- A29	AZ75- X06
M06- V25	T06- Y50	AA45-AZ59	AF43- A07	AO41-AJ32	AZ78-AC15
N42-BK18	T20- Y43	AA46- I05	AF44- A11	AP04-AQ30	AZ79-AC29
N43- E56	T21- Y46	AA52-BI39	AF45-AD14	AP12-AZ21	AZ80-AH09
O42-BK15	T22- Y49	AB33-AF26	AF46-AD24	AP13-AZ22	BA69-BI79
O43- E53	T37- R49	AB34-AF30	AF47-AD28	AP15-AZ30	BA70-BI58
O46-BK16	T38- Y48	AB35-AF40	AF48-AD30	AP19-AZ35	BB66-BJ54
O47- C38	Z33-AB50	AB36-AE20	AF49-AD38	AP20-AZ36	BB67-BJ70
O48- E54	Z34-AB49	AB37-AF20	AF50-AF02	AP23-BA02	BB69-BJ79
O50-BK17	Z35-AB48	AB38- H10	AF51-AF12	AP24-BA03	BB70-BJ58
O51- E55	Z36- I21	AB39- H20	AF52-AF16	AP67- L31	BB71-BH77
P66- W06	Z37- J21	AB40- H30	AF59- K68	AQ04- V35	BB75- L70
P67- Z47	Z38-AQ05	AB41- H40	AF63- K61	AQ10- W31	BB76- G19
P68- L10	Z39- K21	AB42- G30	AF67- H66	AQ15-AR26	BB77- G25
P69- L11	Z40-AO09	AB43- G40	AF71- K60	AQ25-AD45	BI39-BH56
Q48- R54	Z41-AO40	AB44-AE30	AI12- F42	AQ26- L13	BL31- X71
Q49- R55	Z42-AW46	AB46-AC05	AJ35- N70	AQ27- L23	BL32- X67
Q51- R56	Z43-AZ45	AB47- G11	AJ35- K70	AQ28-AD50	BL33- X63
Q52- R59	Z44-AZ49	AB52-BH78	AK16- H41	AQ29-BA55	BL34- X59
Q53- R57	Z45-AZ60	AC51- H70	AO05- V36	AQ31- W52	BL40- E57

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION

ACCOUNTING MACHINE, TYPE 407, CONTROL PANEL DIAGRAM

Form 22-8275-5
Printed in U. S. A.



REPORT R. V. MIDDLETON DIAGRAM NO. 650 UTILITY PANEL DATE

TYPE 407 UTILITY PANEL -- CONTINUED

S19- P51- P46	U43- Y51- Y52	AP17-AO19-AZ33
S24- P48- P54	V49- R52- Q61	AP18-AO20-AZ34
S34- P53- X46	V50- R53- Q62	AP21-AO22-AZ39
T36- P49- W52	AP11-AO12-AZ20	AP22-AO23-AZ40
Q47- P52- X45	AP14-AO13-AZ23	BA66-BI35-BA67
S44- P47- P61	AP16-AO17-AZ31	BB68-BI36-AZ70

T03- T19- T35- W47	W43- W44- W45- W46- W51
S18- S39- U01- U22- X43- P41	S27- T08- T29- U10- U31- P43
S23- T25- U06- U27- X44- P42	S32- T13- T34- U15- U36- P44

TYPE 533 UTILITY PANEL

THIS PANEL WILL READ

12 IN COL 1	LOAD HUB CARDS
A READ	OPEN FOR TEMPORARY WIRING
B READ	REGIONAL INSTRUCTION CARDS
C READ	FIVE-FIELD LOADER CARDS

THIS PANEL WILL PUNCH

A PUNCH	OPEN FOR TEMPORARY WIRING	
B PUNCH	MACHINE LANGUAGE TRACE 10TH WORD IS	88808 00000
	FLAIR TRACE	88808 08000
	LOAD HUB CARDS FROM PUNCH WORDS 1 TO 8	88808 88000
C PUNCH	FIVE-FIELD LOADER CARDS	

WIRING FOR THIS PANEL IS SHOWN ON A 533 BOARD DIAGRAM WHERE CONVENIENT. OTHER WIRING IS LISTED BELOW BY TERMINALS ACCORDING TO THE DIAGRAM INDEX.

Z33- R10	Z39-AA10	W59-AL45	AK44- L42	S28- R41	S23- X35
Z34- L21	Z40- A33	W60- Q50	AK45- M41	AA21-AC15	D22- X40
Z35- X14	Z21-AC10	W61-AL46	AK46-AD60	AN14- L09	
Z36- Y08	H42- W42	AM44- Q45	AP59- D43	AL44- W55	
Z37- Z02	K42- V42	AM45- Q49	AP61- R30	Y33- L06	
Z38- Z16	AM43- X47	AM46- Q51	AQ61- S38	X21- V05	

Y35- U05-AE07	Y37-AB10-AG07	Y39-AB20-AI07	T42-AB60- N41
Y36-AB05-AF07	Y38-AB15-AH07	S41-AP60-AR61	V41- W41-AR55-AR60

AC03-AC08-AC13-AC18-AD21	W31- Y34- L17- L18- L19
W21-AE01-AF01-AG01-AH01-AJ01	W30- Y04-AC04-AC09-AC14-AC19
AN13- K07- K08- K09- L07- L08	V21- V02-AC02-AC07-AC12-AC17-AC20
V31- K18- K19- K20- K21- L20	Y21-AE02-AF02-AG02-AH02-AJ02-AC05
V30- L14- L15- L16- K10- L10	

CORRECT TYPE 533 PANEL SO WORD 5 OF B READ HAS A WORD LENGTH OF 2.

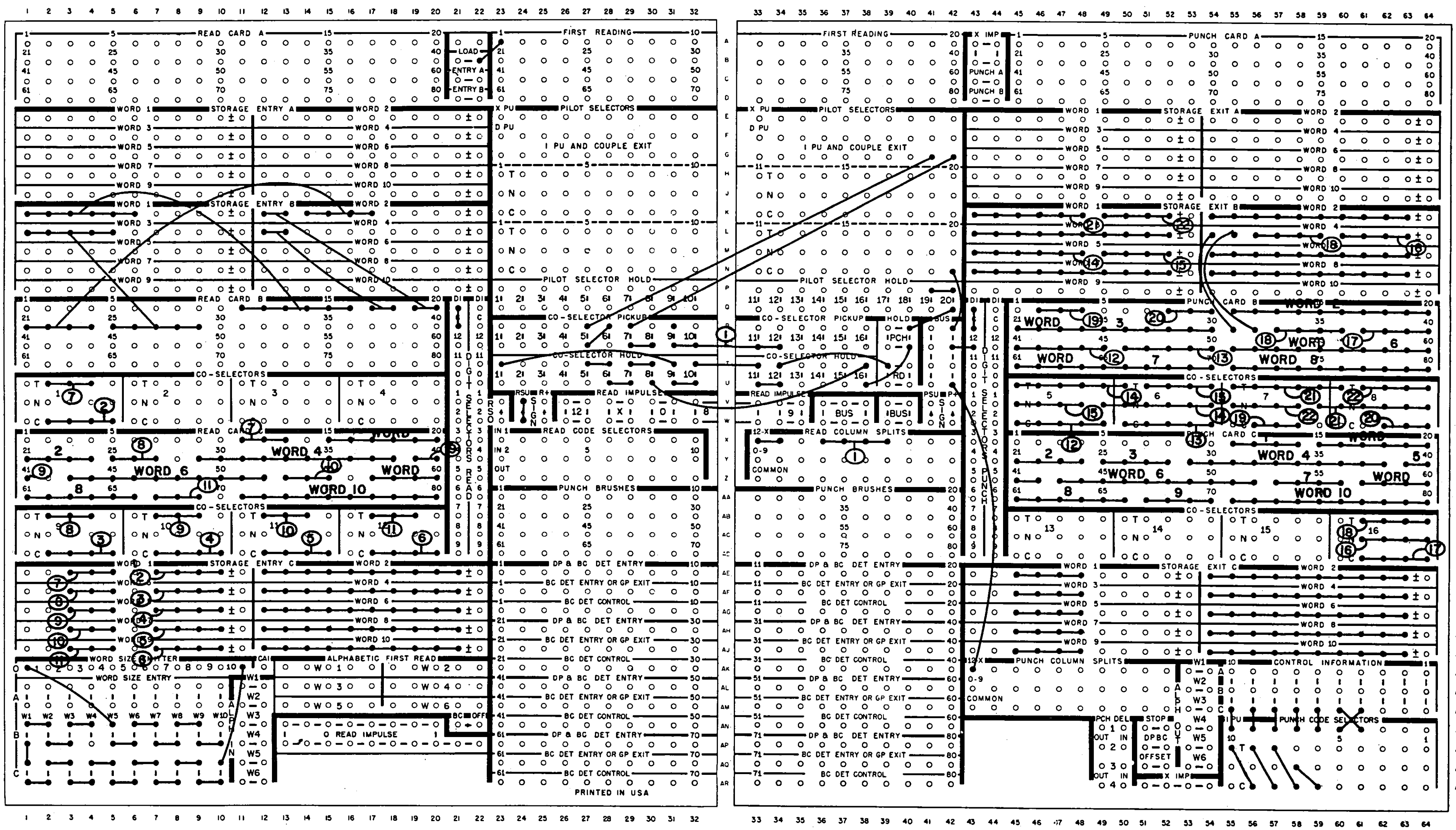
IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION

READ-PUNCH UNIT, TYPE 533 CONTROL PANEL

(USED WITH TYPE 650 MAGNETIC DRUM DATA-PROCESSING MACHINE)

Form No. 22-6207-1
Printed in U.S.A.



FLAIR - FLOATING ABSTRACT INTERPRETATIVE ROUTINE

ONLY A BRIEF SUMMARY OF THIS SYSTEM IS GIVEN HERE. IT IS INTENDED TO SHOW DEVIATIONS FROM THE ORIGINAL SYSTEM AS PUBLISHED ELSEWHERE. A LISTING OF THE INSTRUCTIONS AND CONSTANTS IS FURNISHED TOGETHER WITH ENOUGH DESCRIPTIVE MATERIAL TO OPERATE THE SYSTEM WITHOUT GOING INTO SPECIFIC DETAIL. DETAILED BREAKDOWNS OF THE INDIVIDUAL ROUTINES ARE AVAILABLE IN THIS SAME FORMAT FOR THOSE INTERESTED OR HAVING A NEED TO ALTER. ADDRESS A REQUEST TO THE MATHEMATICAL ANALYSIS SECTION - MISSILE SYSTEMS DIVISION LOCKHEED AIRCRAFT CORPORATION - 7701 WOODLEY AVENUE - VAN NUYS CALIFORNIA.

FLAIR IS A PSEUDO-THREE-ADDRESS FLOATING POINT COMPUTING SYSTEM FOR USE ON THE TYPE 650. NUMBERS ARE OF THE FORM

PP .XXXXXXXX   WHERE PP IS 50 + THE ASSOCIATED POWER OF 10

ARITHMETIC COMMANDS ARE OF THE FORM

OP α β WHERE α AND β ARE FOUR DIGIT ADDRESSES

LOGICAL AND SUB-ROUTINE COMMANDS ARE OF THE FORM

Oγ α β WHERE α AND β ARE FOUR DIGIT ADDRESSES

THE γ IN THE ARITHMETIC COMMANDS REPRESENTS THE UNITS DIGIT OF THE 10 RESULT STORAGES 0000 TO 0009. THESE ADDRESSES ARE A PART OF FLAIR AND MAY BE USED ONLY FOR THIS TEMPORARY PURPOSE. THE BLOCK TRANSFER COMMAND 8γ FREES THEM FOR FURTHER USE.

THE FLAIR SYSTEM IS UNDER THE CONTROL OF THE WORD IN 1615. THE D PART OF THIS WORD IS THE ADDRESS OF THE FLAIR COMMAND TO BE OBEYED. COMMANDS ARE OBEYED IN SEQUENTIAL ORDER EXCEPT AFTER TRANSFER COMMANDS. ENTER FLAIR BY RESET ADDING A WORD 17 J 1735 TO 8003 AND OBEYING 8003. J IS THE ADDRESS OF THE FIRST FLAIR COMMAND TO BE OBEYED.

THE ENTIRE SYSTEM OCCUPIES THE ADDRESSES FROM 1300 TO 1999. THE ARITHMETIC PORTION PLUS SQUARE ROOT AND ABSOLUTE VALUE OCCUPY THE ADDRESSES FROM 1600 TO 1999 AND MAY BE USED IN THIS ABBREVIATED FORM. IF LESS THAN THE FULL COMPLEMENT OF SUBROUTINES IS NEEDED - USE ARITHMETIC FLAIR PLUS GROUPS OF STORAGES AS INDICATED.

LOG	1500 - 1599
ANTILOG	1450 - 1599
SINE AND COSINE	1350 - 1449
ARCTANGENT	1300 - 1499

TRACING IS UNDER THE CONTROL OF THE CONSOLE. SETTING THE HUNDREDS SWITCH OF 8000D TO AN 8 CAUSES TRACING. A 9 IN THIS POSITION CAUSES THE TRACING TO BE IGNORED AND FLAIR WILL RUN NORMALLY. THE MACHINE WILL STOP IF A DIGIT OTHER THAN AN 8 OR 9 IS INADVERTENTLY SET IN THIS SWITCH. IF THE PROGRAM IS IN FLAIR IT MAY BE RESTARTED BY

1. DEPRESSING PROGRAM RESET BUTTON
2. SETTING THE SWITCH PROPERLY
3. TRANSFER TO 1735 FOR NEXT COMMAND

FLAIR OPERATION SUMMARY - LOGICAL COMMANDS

00 - 06	NO OPERATION. THE NEXT COMMAND OBEYED IS IN β
01	MACHINE STOP. IF PROGRAM START BUTTON IS DEPRESSED THE NEXT COMMAND OBEYED IS IN β
02	NO OPERATION. NEXT SEQUENTIAL COMMAND IS OBEYED.
03	CONDITIONAL TRANSFER ON THE SIGN OF THE CONTENTS OF α THE NEXT COMMAND OBEYED IS IN β IF THE SIGN IS - THE NEXT COMMAND OBEYED IS SEQUENTIAL IF THE SIGN IS +
04	CONDITIONAL TRANSFER ON RELATIVE ZERO - SEE DETAILED ITEM.
05	UNCONDITIONAL TRANSFER OUT OF FLAIR. THE NEXT COMMAND OBEYED IS THE MACHINE LANGUAGE COMMAND IN α. IF THE RETURN TO FLAIR IS AT 1612 THE NEXT FLAIR COMMAND IS SEQUENTIAL TO THE 05 COMMAND. IF THE RETURN IS AT 1792 IT IS FOUND IN β OF THE 05 COMMAND.
07 TO 09	NO OPERATION. MACHINE STOP - THEN SEQUENTIAL COMMAND.

FLAIR OPERATION SUMMARY - ARITHMETIC COMMANDS

1γ	$(\alpha) \cdot (\beta) + (\gamma) \longrightarrow \gamma$	
2γ	$(\alpha) \cdot (\gamma) + (\beta) \longrightarrow \gamma$	
3γ	$(\alpha) + (\beta) \longrightarrow \gamma$	
4γ	$(\alpha) - (\beta) \longrightarrow \gamma$	
5γ	$(\alpha) \cdot (\beta) \longrightarrow \gamma$	
6γ	$-(\alpha) \cdot (\beta) \longrightarrow \gamma$	
7γ	$(\alpha) \div (\beta) \longrightarrow \gamma$	
8γ	$(\alpha+K) \longrightarrow (\beta+K)$	K MAY VARY FROM 0 TO γ

FLAIR OPERATION SUMMARY - SUBROUTINE COMMANDS

90	$\sqrt{(\alpha)} \longrightarrow \beta$	
91	SIN $(\alpha) \longrightarrow \beta$	ARGUMENT IN RADIANS
92	COS $(\alpha) \longrightarrow \beta$	ARGUMENT IN RADIANS
93	ARCTAN $(\alpha) \longrightarrow \beta$	
94	LOG $(\alpha) \longrightarrow \beta$	
95	ANTILOG $(\alpha) \longrightarrow \beta$	
96	$\lvert(\alpha)\rvert \longrightarrow \beta$	
97 TO 99	NO OPERATION. NEXT SEQUENTIAL COMMAND IS OBEYED.	

MACHINE STOPS

9000	SQUARE ROOT OF A NEGATIVE NUMBER
9001	SINE OR COSINE OF AN ANGLE GREATER THAN 100 RADIANS
9004	LOG OF ZERO OR A NEGATIVE NUMBER
9005	POWER OF 10 INDEX OUT OF RANGE

A	O	D	I	A	O	D	I	A	O	D	I
1300	66	1416	1374	1350	00	8334	4000	1400	46	1403	1362
1301	46	1304	1401	1351	46	1365	1358	1401	66	1861	1843
1302	20	1867	1322	1352	22	1805	1408	1402	20	1861	1414
1303	30	0000	1328	1353	60	1836	1391	1403	66	8002	1362
1304	15	1357	1361	1354	00	0000	0007	1404	10	1407	1412
1305	10	1308	1316	1355	61	8003	1363	1405	65	1836	1368
1306	22	1909	1312	1356	46	1410	1411	1406	19	1409	1380
1307	65	1867	1475	1357	00	0000	0003	1407	16	6666	8080-
1308	99	9999	3329	1358	65	1417	1843	1408	66	1861	1415
1309	19	1313	1341	1359	46	1365	1401	1409	40	0000	0000-
1310	60	8003	1486	1360	61	1890	1377	1410	11	1744	1974
1311	11	1890	1419	1361	46	1366	1339	1411	10	1744	1974
1312	61	1876	1909	1362	16	1416	1422	1412	60	8003	1372
1313	00	4054	0580-	1363	19	1875	1404	1413	16	1416	1371
1314	64	1876	1320	1364	24	1867	1420	1414	67	8002	1423
1315	35	0002	1321	1365	15	8001	1373	1415	35	0002	1421
1316	60	8003	1333	1366	15	1369	1323	1416	15	7079	6327
1317	10	1324	1484	1367	10	1325	1334	1417	51	1000	0000
1318	10	1326	1336	1368	45	1446	1843	1418	10	8001	1376
1319	10	1476	1386	1369	00	0000	0001	1419	60	8003	1377
1320	20	1876	1331	1370	46	1835	9001	1420	21	1875	1378
1321	20	1876	1329	1371	14	1424	1434	1421	65	8002	1379
1322	61	1375	1314	1372	19	1875	1426	1422	67	8002	1381
1323	20	1835	1338	1373	35	0004	1385	1423	35	0002	1429
1324	33	3298	5605-	1374	20	1890	1307	1424	62	8318	5307
1325	19	9465	3599	1375	09	9999	9999	1425	35	0001	1382
1326	13	9085	3351-	1376	60	8003	1384	1426	60	8003	1433
1327	10	1330	1335	1377	36	0000	1390	1427	67	8003	1388
1328	21	1836	1340	1378	10	8003	1435	1428	67	8003	1443
1329	68	8003	1337	1379	35	0002	1805	1429	60	8003	1432
1330	05	5909	8861-	1380	10	1383	1387	1430	24	1835	1439
1331	46	1485	1300	1381	24	1836	1389	1431	24	1835	1438
1332	67	1835	1339	1382	60	8003	1440	1432	11	1437	1370
1333	19	1836	1349	1383	30	0000	0000	1433	19	1867	1418
1334	60	8003	1342	1384	19	8001	1436	1434	67	8003	1392
1335	60	8003	1345	1385	69	1899	1352	1435	31	0004	1447
1336	60	8003	1343	1386	60	8003	1494	1436	24	1890	1397
1337	21	1890	1393	1387	60	8003	1398	1437	00	0000	0053
1338	15	1396	1302	1388	16	1354	1359	1438	69	1441	1445
1339	35	0004	1399	1389	16	1444	1449	1439	69	1442	1445
1340	60	8003	1347	1390	20	1893	1448	1440	19	8001	1364
1341	10	1348	1310	1391	19	1394	1425	1441	31	0000	1371
1342	19	1899	1317	1392	16	1395	1400	1442	31	0000	1413
1343	19	1899	1367	1393	15	1346	1301	1443	16	1396	1351
1344	21	1899	1452	1394	33	3333	3333	1444	00	0300	0000
1345	19	1899	1319	1395	31	4159	2654	1445	24	1899	1402
1346	00	0000	0047	1396	00	0000	0008	1446	60	8002	1377
1347	19	8001	1344	1397	60	8003	1406	1447	11	1350	1355
1348	02	1861	2288	1398	19	1890	1419	1448	65	8003	1356
1349	30	0001	1311	1399	69	1303	1306	1449	46	1405	1353

A	O	D	I	A	O	D	I	A	O	D	I
1450	00	7300	0000	1500	07	9432	8234	1550	52	4900	0000
1451	46	1554	1459	1501	08	1283	0516	1551	60	8003	1558
1452	60	8003	1309	1502	08	3176	3771	1552	64	8001	1496
1453	10	1856	1948	1503	08	5113	8038	1553	10	1456	1470
1454	19	1836	1474	1504	08	7096	3589	1554	16	1463	1467
1455	60	8003	1873	1505	08	9125	0938	1555	20	1861	1564
1456	11	5129	2770	1506	09	1201	0839	1556	84	1500	1577
1457	16	1714	1469	1507	09	3325	4300	1557	46	1561	1525
1458	35	0001	1465	1508	09	5499	2586	1558	19	1861	1563
1459	15	1462	1467	1509	09	7723	7220	1559	67	8003	1457
1460	10	1464	1471	1510	10	0000	0000	1560	10	1936	1591
1461	10	1466	1472	1511	12	5892	5410	1561	15	1567	1523
1462	19	1510	1488	1512	15	8489	3190	1562	60	8003	1570
1463	19	1510	1498	1513	19	9526	2310	1563	21	1867	1521
1464	06	6273	1000	1514	25	1188	6430	1564	30	0001	1571
1465	16	8002	1524	1515	31	6227	7660	1565	69	1568	1522
1466	02	5439	0000	1516	39	8107	1700	1566	64	8002	1555
1467	20	1873	1526	1517	50	1187	2330	1567	09	3900	0000
1468	65	8003	1477	1518	63	0957	3440	1568	64	0000	1560
1469	35	0004	1529	1519	79	4328	2340	1569	65	8003	1576
1470	60	8003	1528	1520	99	9999	9990	1570	19	1573	1551
1471	60	8003	1479	1521	60	1875	1575	1571	60	8002	1579
1472	60	8003	1480	1522	22	1875	8001	1572	60	8002	1581
1473	19	8001	1495	1523	46	1534	1527	1573	02	8952	9655
1474	60	8003	1482	1524	30	0005	1487	1574	60	8002	1533
1475	46	1360	1332	1525	66	1835	1542	1575	30	0001	1531
1476	09	6420	0441	1526	67	8003	1481	1576	46	1584	1590
1477	10	1586	1592	1527	66	1835	1540	1577	69	1530	1583
1478	10	1936	1492	1528	19	1836	1478	1578	11	1744	1548
1479	19	1836	1553	1529	46	1483	1532	1579	15	1582	1588
1480	19	1836	1460	1530	64	0000	1572	1580	35	0002	1587
1481	20	1836	1490	1531	10	1836	1541	1581	30	0001	1537
1482	19	1836	1461	1532	69	1535	1489	1582	30	0000	0000
1483	69	1536	1489	1533	44	1538	1843	1583	22	1836	1543
1484	60	8003	1990	1534	16	8002	1491	1584	11	1593	1974
1485	65	1416	1374	1535	35	0000	1491	1585	11	1893	1589
1486	19	1899	1327	1536	31	0000	1491	1586	01	0000	0000
1487	10	8001	1545	1537	84	1500	1565	1587	21	1893	1596
1488	65	8003	1496	1538	36	0000	1598	1588	19	1861	1562
1489	22	1893	1546	1539	67	8002	1547	1589	30	0002	1595
1490	60	1493	1497	1540	35	0002	1549	1590	10	1593	1974
1491	10	1744	1499	1541	11	1594	1599	1591	11	8002	1597
1492	60	8003	1473	1542	46	1544	9005	1592	16	8002	1552
1493	00	1750	0000	1543	30	0001	8001	1593	00	0000	0052
1494	19	1899	1318	1544	16	8002	1843	1594	70	6171	1728
1495	35	0001	1455	1545	30	0001	1451	1595	16	1867	1574
1496	35	0001	1453	1546	65	1805	1893	1596	61	8002	1556
1497	15	1450	1454	1547	16	1550	1557	1597	15	1510	1566
1498	35	0001	1468	1548	65	8002	1458	1598	20	1893	1569
1499	21	1856	1578	1549	20	1805	1559	1599	30	0005	1585

FLAIR (Con't.)

A	O	D	I	A	O	D	I	A	O	D	I
1600	61	8001	1608	1650	65	0000	1659	1700	69	1753	1707
1601	36	0000	1622	1651	15	1708	1664	1701	69	1704	1707
1602	60	8002	1761	1652	65	0000	1665	1702	35	0003	1812
1603	11	8003	1611	1653	15	1656	1662	1703	35	0001	1810
1604	60	8003	1761	1654	35	0003	1663	1704	65	0000	1621
1605	44	1609	1610	1655	30	0004	1732	1705	46	1738	1710
1606	20	1861	1867	1656	65	0000	1689	1706	20	1867	1870
1607	11	1861	1616	1657	11	1861	1628	1707	35	0003	1716
1608	10	1861	1666	1658	10	1861	1627	1708	65	0000	1667
1609	10	1613	8003	1659	35	0002	1715	1709	46	1762	1763
1610	60	8001	8001	1660	35	0002	1668	1710	65	1869	1873
1611	24	1615	1618	1661	35	0002	1669	1711	18	1714	1719
1612	60	1615	8001	1662	22	1867	1720	1712	69	1865	1768
1613	30	0002	1672	1663	22	1867	1672	1713	69	1766	1770
1614	66	0000	1923	1664	22	1867	1670	1714	00	0000	0050
1615	17	0000	1735	1665	35	0002	1671	1715	20	1869	1722
1616	11	1619	1623	1666	10	1619	1623	1716	22	1870	1626
1617	35	0004	1732	1667	35	0002	1724	1717	60	1890	1796
1618	35	0001	1625	1668	15	1721	1682	1718	10	1836	1631
1619	00	0000	0010	1669	15	1723	1682	1719	17	1878	1833
1620	11	8003	1728	1670	16	8001	1827	1720	16	8001	1877
1621	35	0002	1777	1671	20	1875	1779	1721	65	0000	1725
1622	21	1876	1632	1672	69	1780	1783	1722	67	8003	1782
1623	15	1876	1631	1673	69	1615	1787	1723	65	0000	1667
1624	71	1877	1824	1674	00	0000	1612	1724	21	1878	1685
1625	44	1629	1630	1675	69	1778	1781	1725	35	0002	1731
1626	69	1729	1783	1676	11	8003	1786	1726	16	1879	1734
1627	46	1634	1631	1677	16	1834	8002	1727	64	1881	1602
1628	46	1631	1634	1678	00	0000	1792	1728	30	0002	1635
1629	10	1633	8003	1679	00	0000	1643	1729	65	0000	1641
1630	35	0003	1839	1680	00	0000	1643	1730	65	0000	1689
1631	30	0002	1637	1681	00	0000	1643	1731	21	1836	1739
1632	69	1687	1640	1682	69	1636	1740	1732	69	1636	1790
1633	11	8003	1694	1683	35	0004	1644	1733	35	0004	1693
1634	60	1687	1843	1684	68	8003	1691	1734	46	1737	1742
1635	22	1897	1617	1685	20	1890	1897	1735	69	8000	1891
1636	20	0000	1612	1686	16	1692	1755	1736	65	1885	1855
1637	44	1643	1843	1687	00	0000	0000	1737	60	1890	1795
1638	01	0000	1612	1688	18	1642	1798	1738	65	1893	1718
1639	22	1895	1648	1689	35	0002	1745	1739	20	1893	1748
1640	23	1893	1746	1690	15	1893	1647	1740	22	1843	8002
1641	35	0002	1747	1691	17	1744	1799	1741	35	0004	1801
1642	00	0000	0050	1692	00	0008	0000	1742	65	1805	1760
1643	69	1615	1638	1693	22	1897	1752	1743	45	1792	1612
1644	22	1897	1750	1694	22	1897	1655	1744	00	0000	0051
1645	46	1848	1849	1695	30	0001	1651	1745	20	1899	1754
1646	10	1649	1603	1696	30	0001	1653	1746	46	1749	1600
1647	44	1601	1602	1697	69	1650	1654	1747	21	1853	1706
1648	30	0002	1605	1698	69	1751	1654	1748	67	8003	1606
1649	00	0001	0000	1699	69	1652	1707	1749	60	8001	1607

A	O	D	I	A	O	D	I	A	O	D	I
1750	65	8003	1660	1800	10	1756	1967	1850	69	1892	1854
1751	66	0000	1659	1801	20	1805	1758	1851	20	1805	1709
1752	65	8003	1661	1802	15	1805	1809	1852	22	1856	1859
1753	66	0000	1665	1803	10	1856	1811	1853	00	0000	0000
1754	67	8003	1711	1804	22	1837	1806	1854	22	1892	1736
1755	46	1710	1759	1805	00	0000	0000	1855	10	1880	1646
1756	00	0000	0025	1806	30	0008	1850	1856	00	0000	0000
1757	20	1861	1764	1807	20	1861	1814	1857	20	1861	1717
1758	46	1712	1713	1808	11	8003	1816	1858	44	1612	1832
1759	60	1867	1821	1809	46	1612	1813	1859	30	0004	1872
1760	30	0004	1773	1810	10	1614	1620	1860			
1761	45	1864	1843	1811	16	8002	1819	1861	00	0000	0000
1762	69	1865	1868	1812	69	1815	1818	1862	24	1884	1837
1763	69	1766	1820	1813	65	8001	1871	1863	69	1885	1817
1764	60	1867	1772	1814	60	1867	1771	1864	36	0000	1838
1765	65	8003	1873	1815	24	0000	1769	1865	30	0000	1690
1766	35	0000	1690	1816	35	0003	1874	1866			
1767				1817	24	1881	1624	1867	00	0000	0000
1768	22	1873	1776	1818	22	1881	1784	1868	22	1873	1785
1769	11	1873	1828	1819	24	1873	1826	1869	00	0000	0000
1770	22	1873	1726	1820	22	1873	1686	1870	00	0000	0000
1771	30	0001	1727	1821	10	1774	8003	1871	30	0004	1831
1772	19	1875	1604	1822	46	1612	1792	1872	15	8001	1830
1773	17	1836	1857	1823	20	1885	1841	1873	00	0000	0000
1774	00	0000	0184	1824	21	1880	1823	1874	69	1837	1804
1775	24	1882	1846	1825	24	1883	1863	1875	00	0000	0000
1776	15	1829	1933	1826	15	1881	8003	1876	00	0000	0000
1777	20	1881	1684	1827	69	1730	1683	1877	69	1780	1733
1778	65	0000	1789	1828	46	1832	1858	1878	00	0000	0000
1779	67	8003	1688	1829	00	0007	0000	1879	00	0008	0000
1780	65	0000	1725	1830	22	1835	1888	1880	00	0000	0000
1781	22	1835	8001	1831	17	1836	1743	1881	00	0000	0000
1782	18	1836	1791	1832	10	1835	1840	1882	00	0000	0000
1783	30	0004	1694	1833	18	1836	1741	1883	00	0000	0000
1784	30	0004	1845	1834	32	0000	0000	1884	00	0000	0000
1785	15	1788	1705	1835	00	0000	0000	1885	00	0000	0000
1786	22	1890	1794	1836	00	0000	0000	1886	88	8080	8000
1787	01	0000	1792	1837	69	0000	1825	1887			
1788	00	0007	0000	1838	15	8003	1645	1888	65	8001	1896
1789	46	1792	1612	1839	22	1895	1648	1889			
1790	22	1843	1897	1840	15	1793	8003	1890	00	0000	0000
1791	35	0004	1851	1841	69	1844	1847	1891	97	1892	1646
1792	60	1895	8001	1842				1892	69	0000	1862
1793	00	0001	0000	1843	00	0000	0000	1893	00	0000	0000
1794	65	8001	1802	1844	69	0000	1775	1894			
1795	19	1899	1765	1845	69	1898	1852	1895	00	0000	0000
1796	19	1899	1604	1846	35	0001	1808	1896	16	1649	1803
1797				1847	22	1853	8001	1897	00	0000	0000
1798	17	1853	1757	1848	11	8002	1657	1898	69	0000	8002
1799	17	1853	1807	1849	11	8002	1658	1899	00	0000	0000

FLAIR (Con't.)

A	O	D	I	A	O	D	I
1900	15	1856	1922	1950	24	1856	1968
1901				1951			
1902				1952			
1903				1953			
1904				1954			
1905				1955			
1906				1956			
1907	10	8001	1965	1957			
1908	35	0001	1966	1958			
1909				1959			
1910				1960			
1911	90	1999	9999	1961	15	1964	1969
1912	90	2999	9999	1962	11	8001	1920
1913	90	4999	9999	1963	69	8002	1971
1914	90	9999	9999	1964	90	0000	0000
1915	91	9999	9999	1965	10	8001	1973
1916	93	9999	9999	1966	44	1978	1970
1917	95	9999	9999	1967	16	8002	1976
1918	97	9999	9999	1968	65	8003	1934
1919	99	9999	9999	1969	16	8002	1996
1920	15	8001	1983	1970	30	0001	1978
1921	16	8002	1935	1971	30	0006	1985
1922	15	1744	1949	1972	35	0003	1992
1923	45	1977	1843	1973	10	8001	1981
1924	69	1427	1430	1974	11	1893	1948
1925	69	1428	1431	1975	15	1979	8002
1926	20	1861	1315	1976	19	8001	1986
1927	46	1580	9004	1977	46	1980	9000
1928	20	1835	1539	1978	66	8002	1989
1929	67	8002	1843	1979	60	0026	1998
1930	00	0000	1612	1980	35	0002	1987
1931	00	0000	1612	1981	30	0001	1988
1932	00	0000	1612	1982			
1933	46	1738	1737	1983	35	0004	1993
1934	10	1899	1921	1984	65	8003	1991
1935	30	0001	1946	1985	64	8001	1972
1936	10	0000	0000	1986	10	8001	1994
1937	00	0040	0610	1987	11	1744	1999
1938	00	0030	0830	1988	21	1893	1997
1939	00	0025	0980	1989	30	0001	1995
1940	00	0020	1210	1990	19	1899	1305
1941	00	0013	1890	1991	10	1899	1963
1942	00	0010	2420	1992	15	8001	1800
1943	00	0007	3530	1993	19	1899	1984
1944	00	0006	4150	1994	16	8002	1950
1945	00	0005	5000	1995	20	1899	1961
1946	64	8001	1900	1996	84	1900	1975
1947	15	1936	1974	1997	11	8001	1908
1948	30	0002	1843	1998	22	1805	1962
1949	44	1947	1974	1999	10	8003	1907

A SELECTIVE AUTOMONITORING TRACING ROUTINE CALLED SAM

A. R. Mandelin and K. D. Weaver
Chance Vought Aircraft, Incorporated

RESUME

In order to reduce substantially the elapsed time and high cost of checking out programs, a routine has been developed at Chance Vought Aircraft which will automatically simulate manual check out procedures on the IBM 650. This routine, which we call SAM (Selective Automonitoring Routine), uses a control table furnished by the programmer to auto-interpret specified single instructions and sequences of instructions of a program executed by the 650. For each instruction monitored SAM punches out on a card the location of the instruction, the instruction itself, and other items of information depending on the instruction, such as the contents of the accumulator and distributor.

SAM has the following features:

a. It will monitor one or more instructions a specified number of times in accordance with console settings or information in a prestored table. Thus if a program loops many times, a sequence of instructions in the loop may be monitored only once, or twice, or as many times as desired.

b. Between monitored sequences, control is returned to the routine being checked out, so that the instructions are executed at normal speed. Execution of the program is slowed only for those instructions which are monitored.

c. SAM will automatically cease to monitor for closed subroutines which are entered by a negative load distributor instruction. The subroutine is executed at normal machine speed.

There are two versions of SAM, i.e., SAM-I and SAM-II. SAM-I will monitor routines coded entirely in 650 machine language. SAM-II will monitor routines coded in both machine language and the 2 and 8 interpretive system described in IBM Technical Newsletter No. 8. SAM-I requires 221 storage locations exclusive of the control table and SAM-II requires 291.

SAM CONTROL TABLES

A SAM control table is a series of words prestored sequentially in memory.

	650 Word		
Location	Op	DA	IA
T_1	N_1	F_1	L_1
T_2	N_2	F_2	L_2
.	.	.	.
.	.	.	.
.	.	.	.
T_n	N_n	F_n	L_n
T_{n+1}	00	0000	0000

Each word in the table specifies a sequence of instructions to be monitored. The location of each first instruction is an F and that of each last instruction an L. If only one instruction is to be monitored then L = F. The number of times the sequence is to be monitored is specified by N. The specified ranges must be in operational sequence and a sequence to be monitored N times must be monitored N times before the next specified sequence can be monitored. The control table is loaded from a deck of control table cards which have one entry per card. As the routine is checked out on successive runs, cards may be removed from or added to the control table deck. If a sequence is to be monitored first six times and then only once, two cards with N = 5 and N = 1 may be used, or an N = 6 card may be replaced with an N = 1 card. Monitoring may also be controlled from the 650 console either without a loaded control table or in conjunction with a control table.

PRINCIPLE OF OPERATION

To check out a coded program, the routine to be checked, the 2 and 8 interpretive routine if required, the control table, and SAM are loaded into memory. The machine is instructed to begin in SAM and the following occurs:

a. The instruction at location F_1 is temporarily removed and a SAM instruction is put in its place.

b. The address L_1 is planted in SAM.

c. Control is then passed to the location of the first instruction of the routine being checked. If this instruction is not at location F_1 itself, execution of the routine begins at normal machine speed and proceeds without monitoring.

d. The first time the program being checked reaches the location F_1 SAM is re-entered and the original instruction at F_1 is replaced.

e. Monitoring is then started at location F_1.

f. Monitoring ceases when location L_1 is reached.

g. After the execution of the instruction at L_1 arrangements are made for monitoring the next sequence in the same manner as in (a) and (b) above.

h. Control is returned to the instruction in the routine which follows L_1.

i. The routine is executed at normal machine speed without monitoring until the end of the program or until an instruction at location F_2 is reached.

The restrictions to be observed in the use of SAM are for the most part obvious consequences of the principles of its operation. Once monitoring begins with an instruction located at F it will continue until L is reached. Thus there must not be a branch out of the sequence. There should be no load card instruction although there may be normal non-branch read card instructions. Also between the execution of an instruction at L_i and the arrival at F_{i+1} there must be no modification of the instruction located at F_{i+1}. Monitored sequences must be separated by at least one unmonitored instruction. F and L instructions may not be interpretive instructions. On instructions having the operation code 14 (i.e., divide with remainder) the accumulator will be reset on the following two operations with the results of the division. No arithmetic operation should occur for two instructions and no reset operation for one.

FORMAT OF RESULTS

For each instruction monitored, SAM-I punches out eight 10-digit words with sign over the units position as indicated on Slide 1.

Word 1. Address of instruction monitored. This is in the DA position for the first instruction of a monitored sequence or for a branch instruction. Otherwise, it is in the IA position. Other positions are zeros.

Word 2. The instruction itself.

Word 3. Contents of upper accumulator after execution of the instruction.

Word 4. Contents of lower accumulator (after).

Word 5. Contents of distributor (after).

Word 6. Zero.

Word 7. Zero.

Word 8. Identification word. This word contains the routine number, the table entry count number, loop count number, and card count number.

For each interpretive instruction monitored, SAM-II punches out on a card eight 10-digit words with sign over the units position as follows:

Word 1. Address of the interpretive instruction monitored. This will be in the DA position. Other positions zeros.

Word 2. The instruction itself or the first word of a two-word instruction.

Word 3. Zero or the second word of a two word instruction.

Word 4. Contents of A address before execution of the interpretive instruction.

Word 5. Contents of B address before execution or zero if no B address.

Word 6. Contents of the C address after execution of the instruction (for two-word interpretive instructions only.)

Word 7. Contents of the floating point accumulator K after execution of an interpretive instruction.

Word 8. Identification word. This word contains the routine number, the table entry count number, loop count number and card count number.

For machine language instructions, SAM-II will monitor in the same manner as SAM-I.

SUBROUTINES

SAM will not slow down the check out of a routine with unnecessary monitoring of previously checked out subroutines. This is avoided by specifying that entry to a subroutine must be made by using a negative load distributor instruction, i.e., (-69 NI SR) where SR is the entry location of the subroutine and NI is the location of the next instruction in the main routine. Entry to the subroutine must be made from machine language and the negative load distributor instruction must not immediately follow a (negative) interpretive instruction.

OPERATIONAL EXPERIENCE

By monitoring single instructions a programmer can rapidly determine and narrow down regions of error. Having determined these, he can then examine his coding to find his mistakes. If he cannot locate the source of trouble, he can have SAM trace all instructions within the suspected region of error. With the proper preparation of the control tables the amount of check out time on the 650 for each check out run should usually be no more than 10 minutes. The experience with the 650 at Chance Vought during the first month of operation (650 was installed May 27) showed that the check out period varied between 5 and 20 minutes with a 10 minute average. As the programmers have had to consider and plan their work more thoroughly than they would have had to using console monitoring, and as the information furnished by SAM was better both in quantity and quality, the total elapsed time on check out has been reduced to approximately 25% of the time that would have been necessary using manual methods. At CVA we have planned our control panels so that in no case is it necessary for us to change our control panel in a check out of a problem in order to be able to use SAM.

EXAMPLE 1

Slide 3 illustrates a simple hypothetical problem with its check points and control table for the first check out run. R_1 is the location of the first instruction of the routine, and F_1, F_2, and F_3 key check point locations. From R_1 to F_1 the routine is run unmonitored. At F_1 a single card is punched out with all needed information. From this point, the routine is run unmonitored until F_2 is reached. At F_2 the instruction is monitored. From F_2 the routine proceeds unmonitored to point b where it branches to point a along path 1 and continues unmonitored to point F_2. The instruction at F_2 is monitored again. The above process is repeated, going to point b to point a via path 2, and continuing until location F_2 is again reached. The third F_2 value is punched and the routine now proceeds unmonitored to point F_3. F_3 is monitored and the routine continues unmonitored to out.

The control table starts at location 1901,i (the i represents the regional SAM translation tag) and ends at 1904,i. The operation part of the first word is 01, the data address F_1 and the instruction address $L_1 = F_1$. The next two table values are fashioned in the same way except that the operation part of 1902,i is 03 in order to get the three check point values located at F_2. The last word in the table, which is all zeros, is a tag for SAM to indicate that no further monitoring is required after F_3.

EXAMPLE 2

Slide 4 illustrates a problem wherein it is desired to monitor completely all instructions in the 15th loop. It should be noted immediately, that in our illustration, the instructions in the loops 1 through 15 are all the same, but are shown separate for purposes of illustration. The program starts at R_1. Then it proceeds through point a to point b where it branches around a loop ending at b fifteen times. After this, it proceeds through a series of instructions to out. The fourteen loops preceding the fifteenth are counted off by taking a single instruction within the loop and monitoring it fourteen times. This is done by the first word in our control table which has 14 in the operation part and F_1 equal to L_1. The fifteenth loop, which is to be monitored completely, is then set up in the control table as the second word with 01 as the operation part.

SAM-II LISTING

A listing of the instructions in the SAM-II routine is furnished. The routine is in absolute address form with regional tags to facilitate its translation or modification. Entry is at 1600 for control table operation and 1653 for console operation. For console operation set the storage entry switches to N F L . For control table operations the storage entry switches must be set to 69 R 1600 where R is the location of the first instruction of the routine being checked.

SAM RESULT FORM

	WORD 1	WORD 2	WORD 3	WORD 4	WORD 5	WORD 6	WORD 7	WORD 8
SAM I or SAM II Machine Language Instruction	Location	Instruction	Result in Upper Accum.	Result in Lower Accum.	Result in Distributor	Zero	Zero	Identification
SAM II Interpretive Instruction	Location	Instruction	Zero or C Instruction	A Result	Zero or B Result	Zero or C Result	Value in K Register	Identification

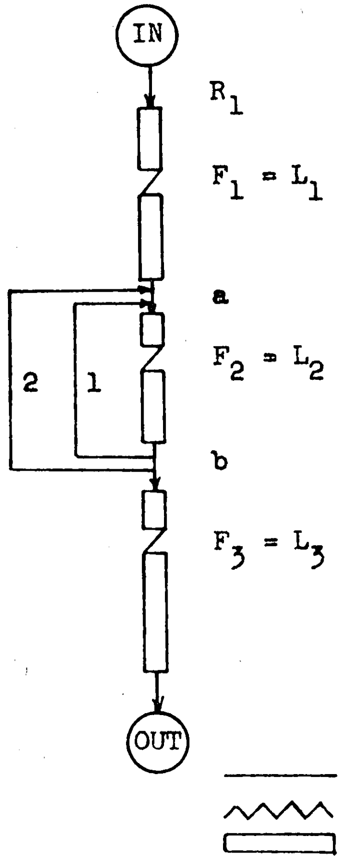
SLIDE 1

SAM CONTROL TABLE

LOC.	OP	DA	NI
T_1	N_1	F_1	L_1
T_2	N_2	F_2	L_2
	•		
	•		
	•		
T_n	N_n	F_n	L_n
T_{n+1}	00	0000	0000

SLIDE 2

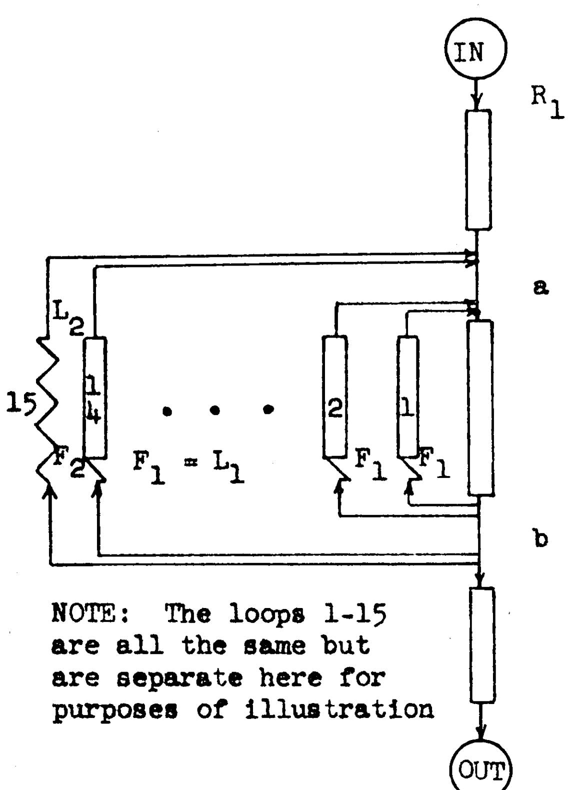
SAM SAMPLE WITH TABLE CONTROL
For First Check Out Run



LOC	OP	D.A.	I.A.
1901,i	01	F_1	L_1
1902,i	03	F_2	L_2
1903,i	01	F_3	L_3
1904,i	00	0000	0000

SLIDE 3

SAM SAMPLE WITH TABLE CONTROL
Illustrating How to Monitor
The 15th Loop



LOC	OP	D.A.	I.A.
1901,i	14	F_1	L_1
1902,i	01	F_2	L_2
1903,i	00	0000	0000

NOTE: The loops 1-15 are all the same but are separate here for purposes of illustration

SLIDE 4

SAM 2 LISTING

CD NO	M	LOC	ABBR	OP	DA	IA	REMARKS
001	0	1600 01	STD	24+	1603 01	1641 08	T1T2 STORE INST
002	9	1601 01	TEMP	+			SF FIRST MONITORED INST
003	9	1602 01	TEMP	+			SL LAST MONITORED INST
004	9	1603 01	TEMP	+			SM MODIFIED INST OR
005	9	01	TEMP	+			FIRST UNMONITD INST
006	9	1604 01	TEMP	+			ST TEMPORARY STORAGE1/2
007	9	1605 01	TEMP	+			SA NEXT INST ADDRESS
008	9	1606 01	TEMP	+			SP STORE CONSOLE ENTRY
009	0	1607 01	CONST	00+	0000 00	0000 00	K0 00 0000 0000 1/2
010	0	1608 01	CONST	65+	0000 00	1667 07	K1 65 0000 C8/1 1/2
011	0	1609 01	CONST	00+	0001 00	0000 00	K2 00 0001 0000 1/2
012	0	1610 01	CONST	00+	0000 00	0001 00	K3 00 0000 0001 1/2
013	0	1611 01	CONST	00+	0000 00	0004 00	K4 00 0000 0004 1/2
014	0	1612 01	CONST	00+	0000 00	0005 00	K5 00 0000 0005 1/2
015	0	1613 01	CONST	65+	8001 00	1689 12	K8 65 8001 A13/1 1/2
016	0	1614 01	CONST	65+	1631 02	1689 12	K9 65 P5 A13/1 1/2
017	0	1615 01	CONST	71+	1627 02	1777 14	K10 71 P1 B18 1/2
018	0	1616 01	CONST	71+	1627 02	1793 14	K11 71 P1 B25 1/2
019	0	1617 01	CONST	00+	1768 14	1765 14	K12 00 B14 B13 1/2
020	0	1618 01	CONST	65+	0000 00	1830 09	K13 65 0000 SB4/2 2
021	0	1619 01	CONST	65+	1901 06	1646 08	K14 65 T1 T3/2 1/2
022	0	1620 01	CONST	69+	1765 14	1600 01	K15 69 B13 T1 1/2
023	0	1621 01	CONST	00+	0000 00	0069 00	K16 00 0000 0069 1/2
024	0	1622 01	CONST	69+	0000 00	1782 14	K17 69 0000 B19/4 1/2
025	0	1623 01	CONST	00+	0000 00	1000 00	K27 00 0000 1000 1/2
026	0	1624 01	CONST	69+	1616 01	1683 12	K28 69 K11 A6/2 1/2
027	0	1625 01	CONST	69+	1637 01	1683 12	K29 69 K31 A6/2 1/2
028	0	1626 01	CONST	71+	1627 02	1779 14	K30 71 P1 B19 1/2
029	0	1637 17	STD	24+	1635 02	1882 16	T11 STORE RETURN
030	0	1638 17	CONST	01+	0220 00	1000 00	K6 01 0220 1000 2
031	0	1639 17	CONST	64+	1680 06	1646 08	K7 64 T1 MINUS 0221 2
033	0	1837 03	CONST	65+	0026 05	1689 12	K18 65 0026 A13 2
034	0	1838 03	CONST	68+	0000 00	1855 15	K19 68 0000 A5/2
035	0	1839 03	CONST	69+	0000 00	1858 15	K20 69 0000 Q6/3 2
036	0	1840 03	CONST	65+	0000 00	1864 15	K21 65 0000 Q10 2
037	0	1841 03	CONST	65+	0001 00	1868 15	K22 65 0001 G11/4
038	0	1842 03	CONST	60+	0029 05	0092 05	K23 60 0029 0092 2
039	0	1843 03	CONST	46+	0025 05	8002 00	K24 46 0025 8002 2
040	0	1844 03	CONST	65+	0029 05	1872 15	K25 65 0029 Q15 2
041	0	1845 03	CONST	46+	0025 05	1673 12	K26 46 0025 A1
042	9	1627 02	PUNCH	+			P1 INST LOCATION
043	9	1628 02	PUNCH	+			P2 SAM 2 FIRST INST
045	9	1629 02	PUNCH	+			P3 SAM 2 2ND INST OR 0
047	9	1630 02	PUNCH	+			P4 SAM 2 A FACTOR
049	9	1631 02	PUNCH	+			P5 SAM 2 B FACTOR OR 0
051	0	1632 02	PUNCH	00+	0000 00	0000 00	P6 SAM 2 C FACTOR OR 0
053	0	1633 02	PUNCH	00+	0000 00	0000 00	P7 SAM 2 CONTENTS OF K
055	9	1634 02	PUNCH	+			P8 IDENTIFICATION
056	9	1635 02	PUNCH	+			P9 WASTEBASKET
057	9	1636 02	PUNCH	00+	0800 00	0000 00	P10 PUNCH CONTROL INFORM
058	9	8000 00	CONSL	69+	R1	1600 01	T3T1 TABLE ENTRY
059	9	1640 08	TEMP	+			SN LOOP COUNT
060	0	1641 08	RAL	65+	1645 08	1642 08	T2 FORM AND STORE
061	0	1642 08	SL	16+	1639 17	1643 08	T2 IDENTIFICATION
062	0	1643 08	AL	15+	1623 01	1644 08	T2 IN P8

CD NO	M	LOC		ABBR	OP	DA		IA		REMARKS
063	0	1644	08	STL	20+	1634	02	1645	08	T2T3
064	0	1645	08	RAL	65+	1901	06	1646	08	T3 STORE TABLE ENTRY
065	0	1646	08	STL	20+	1606	01	1647	08	T3T4
066	0	1647	08	BRNZ	45+	1650	08	1648	08	T4T9T5 END OF TABLE TEST
067	0	1648	08	LD	69+	1619	01	1649	08	T9 RESET TI TO T1
068	0	1649	08	STD	24+	1645	08	1878	16	T9T8
069	0	1650	08	LD	69-	1651	08	1803	10	T5SA GO TO SUBROUTINE
070	0	1651	08	SRT	30+	0002	00	1652	08	T6 STORE N FOR
071	0	1652	08	STU	21+	1640	08	1821	09	T6T7 LOOP COUNT
072	9	8000	00	CONSL	NI+	FI		LI		COC1 CONSOLE ENTRY
073	0	1653	07	STD	24+	1631	02	1654	07	C1 STORE PREVIOUS
074	0	1654	07	STL	20+	1630	02	1655	07	C1 RESULTS
075	0	1655	07	STU	21+	1629	02	1656	07	C1C2
076	0	1656	07	LD	69+	1638	17	1657	07	C2 SET IDENT
077	0	1657	07	STD	24+	1634	02	1658	07	C2C3 IN P8
078	0	1658	07	LD	69+	1625	01	1659	07	C3 SET OUT PUT
079	0	1659	07	STD	24+	1682	12	1660	07	C3C4 CONNECTOR
080	0	1660	07	RAL	65+	8000	00	1661	07	C4 STORE CONSOLE
081	0	1661	07	STL	20+	1606	01	1662	07	C4C5 IN SP
082	0	1662	07	LD	69-	1663	07	1803	10	C5SA GO TO SUBROUTINE
083	0	1663	07	SRT	30+	0002	00	1664	07	C6 STORE LOOP COUNT
084	0	1664	07	STU	21+	1640	08	1665	07	C6C7 IN SN
085	0	1665	07	RAL	65+	1645	08	1666	07	C7
086	0	1666	07	LD	69+	1608	01	1781	14	C7
088	8	8001	00	RAL	65+	0000	00	1667	07	C7C8
089	0	1667	07	BRNZ	45+	1671	07	1764	13	C8T10/4T10/6
090	0	1668	07	STD	24+	1631	02	1669	07	T10 STORE PREVIOUS
091	0	1669	07	STL	20+	1630	02	1670	07	T10 RESULTS
092	0	1670	07	STU	21+	1629	02	1671	07	T10
093	0	1671	07	LD	69+	1601	01	1672	07	T10 RESTORE F INST
094	1	1672	07	STD	24+	0000	00	1764	13	T10T11
095	0	1673	12	RAL	65+	1688	12	1674	12	A1 TEST FOR LAST
096	0	1674	12	SL	16+	1602	01	1675	12	A1A2 INSTRUCTION
097	0	1675	12	BRNZ	45+	1886	16	1676	12	A2A7A3
098	0	1676	12	RSABL	68+	1640	08	1677	12	A3
099	0	1677	12	AL	15+	1610	01	1678	12	A3
100	0	1678	12	STL	20+	1640	08	1679	12	A3A4 CHECK LOOP COUNT
101	0	1679	12	BRMIN	46+	1680	12	1682	12	A4A5A6
102	7	1680	12	LD	69+	1626	01	1681	12	A5 SET PUNCH CONN
103	0	1681	12	STD	24+	1776	14	1886	16	A5A7 FOR LOOPING
104	7	1682	12	LD	69+	1616	01	1683	12	A6 SET OUTPUT CONN
105	0	1683	12	STD	24+	1776	14	1886	16	A6A7
106	0	1684	12	RAL	65+	1688	12	1685	12	A9 TEST FOR 800I
107	0	1685	12	SL	16+	1613	01	1686	12	A9A10 INST
108	0	1686	12	BRMIN	46+	1688	12	1687	12	A10A12A11
109	0	1687	12	SL	16+	1614	01	8002	00	A11
110	8	8002	00	RAL	65+	0000	00	1689	12	A11A13 STORE 800I INST
111	1	1688	12	RAL	65+	0000	00	1689	12	A12A13 STORE INSTRUCTION
112	0	1689	12	STL	20+	1628	02	1690	12	A13A14 IN P2
113	0	1690	12	SRT	30+	0008	00	1691	12	A14
114	0	1691	12	AL	15+	1621	01	1692	12	A14A15 TEST FOR SUB
115	0	1692	12	BRNZ	45+	1693	12	1711	12	A15A16A24 ROUTINE ENTRY
116	0	1693	12	RAL	65+	1617	01	1694	12	A16
117	0	1694	12	LD	69+	1628	02	1695	12	A16 STORE MODIFIED
118	0	1695	12	STIA	23+	1603	01	1696	12	A16A17 INST IN SM

CD NO	M	LOC		ABBR	OP	DA		IA		REMARKS
119	0	1696	12	RAABL	67+	1628	02	1697	12	A17
120	0	1697	12	LD	69+	8003	00	1698	12	A17 STORE NEXT INST
121	0	1698	12	STIA	23+	1605	01	1699	12	A17A18 ADDRESS IN SA
122	0	1699	12	SLT	35+	0004	00	1700	12	A18
123	0	1700	12	LD	69+	1688	12	1701	12	A18 STORE NEXT INST
124	0	1701	12	STDA	22+	1688	12	1702	12	A18A19 ADDRESS IN A12
125	0	1702	12	SRT	30+	0003	00	1703	12	A19
126	0	1703	12	SL	16+	8002	00	1704	12	A19 CHECK FOR BRANCH
127	0	1704	12	SU	11+	1611	01	1705	12	A19A20 INST
128	0	1705	12	BRNZU	44+	1706	12	1708	12	A20A21A23
129	0	1706	12	SU	11+	1612	01	1707	12	A21A22 CHECK FOR BRANCH
130	0	1707	12	BRNZU	44+	1719	13	1708	12	A22B1A23 INST
131	0	1708	12	RAL	65+	1617	01	1709	12	A23 MODIFY DATA ADD
132	0	1709	12	LD	69+	1603	01	1710	12	A23 OF CURRENT INST
133	0	1710	12	STDA	22+	1603	01	1878	16	A23T8 IF BRANCH INST
134	0	1711	12	RAL	65+	1628	02	1712	12	A24 STORE SUBROUTINE
135	0	1712	12	LD	69+	1688	12	1713	12	A24 RETURN
136	0	1713	12	STDA	22+	1688	12	1714	12	A24A25
137	0	1714	12	LD	69+	8003	00	1715	12	A25
138	0	1715	12	SRT	30+	0004	00	1716	12	A25 STORE RETURN
139	0	1716	12	STIA	23+	1605	01	1717	12	A25A26 LOCATION IN SA
140	0	1717	12	RAL	65+	1620	01	1718	12	A26
141	0	1718	12	LD	69+	1628	02	1710	12	A26A23/3 CHANGE RETURN
142	0	1719	13	RAABL	67+	1628	02	1720	13	B1
143	0	1720	13	SLT	35+	0002	00	1721	13	B1
144	0	1721	13	SL	16+	8002	00	1722	13	B1
145	0	1722	13	SU	11+	1755	13	1723	13	B1B2 TEST FOR DIV INST
146	7	1723	13	BRNZ	45+	1724	13	1729	13	B2B4ORB6B3
147	0	1724	13	RAU	60+	1757	13	1725	13	B4
148	0	1725	13	SLT	35+	0001	00	1726	13	B4
149	0	1726	13	AU	10+	1629	02	1727	13	B4 RESTORE PREVIOUS
150	0	1727	13	AL	15+	1630	02	1728	13	B4B5 RESULTS
151	0	1728	13	LD	69+	1631	02	1603	01	B5SM
152	0	1729	13	RAABL	67+	1628	02	1730	13	B3
153	0	1730	13	LD	69+	1756	13	1731	13	B3 STORE PREVIOUS
154	0	1731	13	STDA	22+	1603	01	1732	13	B3 ACC RESULTS
155	0	1732	13	RAL	65+	1629	02	1733	13	B3 TEMPORARILY
156	0	1733	13	STL	20+	1760	13	1734	13	B3 AND STORE MOD
157	0	1734	13	RAL	65+	1630	02	1735	13	B3 INST IN SM
158	0	1735	13	STL	20+	1761	13	1724	13	B3B4
159	0	1736	13	RAL	65+	1737	13	1738	13	B6 TEST FOR NO OF
160	0	1737	13	CONST	00+	0000	00	0001	00	B6B7 DIVISIONS
161	0	1738	13	BRMIN	46+	1745	13	1739	13	B7B11B8
162	0	1739	13	SL	16+	1610	01	1740	13	B8B9 REDUCT COUNT
163	0	1740	13	STL	20+	1737	13	1742	13	B9B10 BY ONE
164	0	1741	13	STD	21+	1632	02	1848	15	B16I SET P6 TO ZERO
165	0	1742	13	RAU	60+	1760	13	1743	13	B10 RESTORE ACC
166	0	1743	13	AL	15+	1761	13	1744	13	B10 AND DIVIDE
167	0	1744	13	DIV	14+	1762	13	1728	13	B10B5
168	0	1745	13	RAL	65+	1759	13	1746	13	B11 SET CONNECTOR
169	0	1746	13	STL	20+	1723	13	1724	13	B11B4
170	0	1747	13	STD	24+	1762	13	1748	13	B12 STORE RESULTS
171	0	1748	13	STD	24+	1631	02	1749	13	B12 TEMPORARILY
172	0	1749	13	STU	21+	1629	02	1750	13	B12 AND FOR PUNCH
173	0	1750	13	STL	20+	1630	02	1751	13	B12

CD NO	M	LOC	ABBR	OP	DA	IA	REMARKS
174	0	1751 13	LD	69+	1610 01	1752 13	B12
175	0	1752 13	STD	24+	1737 13	1753 13	B12
176	0	1753 13	LD	69+	1758 13	1754 13	B12 SET CONNECTOR
177	0	1754 13	STD	24+	1723 13	1792 14	B12B13/4
178	0	1755 13	CONST	00+	0000 00	0014 00	K32 00 0000 0014
179	0	1756 13	CONST	14+	0000 00	1747 13	K33 14 0000 B12/1
180	0	1757 13	CONST	10-	0000 00	0000 00	K34 10 0000 0000 NEG
181	0	1758 13	CONST	45+	1736 13	1729 13	K35 45 B6 B3
182	0	1759 13	CONST	45+	1724 13	1729 13	K36 45 B4 B3
183	9	1760 13	TEMP	+			S1 CONTENTS P3
184	9	1761 13	TEMP	+			S2 CONTENTS P4
185	9	1762 13	TEMP	+			S3 DIST AFTER DIVISION
186	0	1763 13	CONST	00+	1607 01	0000 00	K37 00 L/0/ 0000
187	0	1764 13	LD	69+	1673 12	1637 17	T10T11
188	0	1765 14	STD	24+	1631 02	1766 14	B13 STORE RESULTS
189	0	1766 14	STL	20+	1630 02	1767 14	B13
190	0	1767 14	STU	21+	1629 02	1792 14	B13B16
191	0	1768 14	RAL	65+	1628 02	1769 14	B14 STORE NEXT INST
192	0	1769 14	LD	69+	8003 00	1770 14	B14 LOC OF BRANCH
193	0	1770 14	STDA	22+	1605 01	1771 14	B14B15 INST
194	0	1771 14	LD	69+	1688 12	1772 14	B15 SET UP NEXT INST
195	0	1772 14	STDA	22+	1688 12	1792 14	B15B13/4 ADDRESS
196	0	1773 14	RAL	65+	1634 02	1774 14	B16 INCREASE CARD
197	0	1774 14	AL	15+	1610 01	1775 14	B16 COUNT BY 1
198	0	1775 14	STL	20+	1634 02	1741 13	B16B16I 2
199	7	1776 14	PCH	71+	1627 02	1777 14	B17B18/B19/B25/B28
200	0	1777 14	LD	69+	1605 01	1778 14	B18 STORE INST LOC
201	0	1778 14	STD	24+	1627 02	1673 12	B18A1 IN P1
202	0	1779 14	RAL	65+	1688 12	1780 14	B19 STORE FIRST
203	0	1780 14	LD	69+	1622 01	1781 14	B19 UNMONITORED
204	0	1781 14	STDA	22+	1635 02	8001 00	B19 INST IN SM
205	8	8001 00	LD	69+	0000 00	1782 14	B19
206	0	1782 14	STD	24+	1603 01	1783 14	B19B20
207	0	1783 14	RAL	65+	1634 02	1784 14	B20 RESET CARD CT TO
208	0	1784 14	SRT	30+	0003 00	1785 14	B20 ZERO
209	0	1785 14	SLT	35+	0003 00	1786 14	B20
210	0	1786 14	AL	15+	1623 01	1787 14	B20 INCREASE LOOP CT
211	0	1787 14	STL	20+	1634 02	1788 14	B20B21 BY ONE
212	0	1788 14	LD	69+	1615 01	1789 14	B21 SET CONNECTOR
213	0	1789 14	STD	24+	1776 14	1790 14	B21B22 TO B18
214	0	1790 14	RAL	65+	1606 01	1791 14	B22
215	0	1791 14	LD	69-	1821 09	1803 10	B22B23 SET RETURN ADD
216	0	1792 14	LD	69+	1773 14	1637 17	B13T11
217	0	1793 14	RAL	65+	1645 08	1794 14	B25 INCREASE TABLE
218	0	1794 14	AL	15+	1609 01	1795 14	B25 INDEX BY ONE
219	0	1795 14	STL	20+	1645 08	1796 14	B25B26
220	0	1796 14	LD	69+	1615 01	1797 14	B26 SET CONNECTOR
221	0	1797 14	STD	24+	1776 14	1798 14	B26B27 TO B18
222	0	1798 14	RAL	65+	1688 12	1799 14	B27 STORE FIRST
223	0	1799 14	LD	69+	1620 01	1800 14	B27 UNMONITORED
224	0	1800 14	STDA	22+	1635 02	8001 00	B27 INST IN SM
226	0	1801 14	LD	69+	1624 01	1802 14	B28 SET CONNECTOR
227	0	1802 14	STD	24+	1682 12	1796 14	B28B26 TO B19
228	0	1803 10	STD	24+	1635 02	1804 10	SA1SA2 STORE FOR RETURN
229	0	1804 10	LD	69+	8003 00	1805 10	SA2 STORE LOCATION

CD NO	M	LOC		ABBR	OP	DA		IA		REMARKS	
230	0	1805	10	STDA	22+	1627	02	1806	10	SA2SA3	OF F IN P1
231	0	1806	10	LD	69+	1688	12	1807	10	SA3	SET UP A12/1
232	0	1807	10	STDA	22+	1688	12	1808	10	SA3SA4	
233	0	1808	10	SLT	35+	0004	00	1809	10	SA4	STORE LOCATION
234	0	1809	10	STDA	22+	1602	01	1635	02	SA4P9	OF L IN SL
248	0	1821	09	LD	69−	1878	16	1822	09	T7SB1	
249	0	1822	09	STD	24+	1635	02	1823	09	SB1	STORE FOR RETURN
250	0	1823	09	RAL	65+	1606	01	1824	09	SB1SB2	LOAD SP
251	0	1824	09	LD	69+	1832	09	1825	09	SB2	STORE LOCATION
252	0	1825	09	STDA	22+	1832	09	1826	09	SB2SB3	OF F IN SB5/2
253	0	1826	09	LD	69+	1672	07	1827	09	SB3	STORE LOCATION
254	0	1827	09	STDA	22+	1672	07	1828	09	SB3SB4	OF F IN T10/5
255	0	1828	09	LD	69+	1618	01	1829	09	SB4	STORE FIRST
256	0	1829	09	STDA	22+	1618	01	8001	00	SB4	MONITORED INST
257	8	8001	00	RAL	65+	L/F/		1830	09	SB4	IN SF
258	0	1830	09	STL	20+	1601	01	1831	09	SB4SB5	
259	0	1831	09	LD	69+	1668	07	1832	09	SB5	REPLACE FIRST
260	1	1832	09	STD	24+	0000	00	1833	09	SB5SB6	INST
261	0	1833	09	SL	16+	1603	01	1834	09	SB6SB7	TEST FOR F EQUAL
262	0	1834	09	BRNZ	45+	1635	02	1835	09	SB7P9SB8	TO R
263	0	1835	09	LD	69+	1671	07	1836	09	SB8	REPLACE INST
264	0	1836	09	STD	24+	1603	01	1635	02	SB8P9	IN SM
265	0	1846	15	RAL	65+	0029	05	1847	15	G1	COMPUTE INST LOC
266	0	1847	15	AL	15+	1609	01	1849	15	G1G2	
267	0	1848	15	STU	21+	1633	02	1776	14	B16IB17	SET P7 TO ZERO
268	0	1849	15	STU	21+	1629	02	1850	15	G2G3	SET P3 TO ZERO
269	0	1850	15	STDA	22+	1627	02	1851	15	G3G4	STORE LOC IN P1
270	0	1851	15	RAABL	67+	8001	00	1852	15	G4	STORE LOC IN
271	0	1852	15	LD	69+	1688	12	1853	15	G4	PROGRAM
272	0	1853	15	STDA	22+	1688	12	1854	15	G4G5	
273	0	1854	15	AL	15+	1838	03	8002	00	G5	STORE INST IN P2
274	8	8002	00	RAL	65+	0000	00	1855	15	G5	
275	0	1855	15	STL	20+	1628	02	1856	15	G5G6	
276	0	1856	15	LD	69+	1839	03	1857	15	G6	STORE FACTOR A
277	0	1857	15	STDA	22+	1635	02	8001	00	G6	FOR PUNCHING
278	8	8001	00	LD	69+	0000	00	1858	15	G6	
279	0	1858	15	STD	24+	1630	02	1859	15	G6G7	
280	0	1859	15	SLT	35+	0006	00	1860	15	G7	
281	0	1860	15	SU	11+	8003	00	1861	15	G7G8	TEST FOR B ADD
282	0	1861	15	BRNZ	45+	1862	15	1864	15	G8G9G10	
283	0	1862	15	SRT	30+	0002	00	1863	15	G9	IF B FACTOR STORE
284	0	1863	15	SL	16+	1840	03	8002	00	G9	IN P5 / IF NOT
285	8	8002	00	RAL	65+	0000	00	1864	15	G9G10	STORE ZERO
286	0	1864	15	STL	20+	1631	02	1865	15	G10G11	STORE ZERO
287	0	1865	15	RAL	65+	1627	02	1866	15	G11	LOAD NEXT INST
288	0	1866	15	AL	15+	1841	03	8002	00	G11	AND STORE IN P6
289	8	8002	00	RAL	65+	0000	00	1868	15	G11	
290	0	1867	15	RAL	65+	1763	13	1889	16	G13/2G13/3	
291	0	1868	15	SLT	35+	0002	00	1869	15	G11G12	TEST FOR SECOND
292	0	1869	15	BRNZU	44+	1867	15	1870	15	G12G13/2G13/1	INST
293	0	1870	15	SRT	30+	0002	00	1871	15	G13	STORE NEXT INST
294	0	1871	15	STL	20+	1629	02	1889	16	G13/1	
295	0	1872	15	LD	69+	0057	05	1873	15	G15	STORE K IN P7 FOR
296	0	1873	15	STD	24+	1633	02	1891	16	G15G15/1	
297	0	1874	15	RAL	65+	1634	02	1875	15	G16	INCREASE CD

SAM 2 LISTING

CD NO	M	LOC		ABBR	OP	DA		IA		REMARKS	
298	0	1875	15	AL	15+	1610	01	1876	15	G16	CT BY ONE
299	0	1876	15	STL	20+	1634	02	1877	15	G16G17	
300	0	1877	15	PCH	71+	1627	02	1846	15	G17G1	PUNCH RESULTS
301	0	1878	16	LD	69+	1842	03	1879	16	T8	RESTORE INTERP
302	0	1879	16	STD	24+	0026	05	1880	16	T8	INST
303	0	1880	16	LD	69+	1843	03	1881	16	T8	
304	0	1881	16	STD	24+	0439	05	1895	16	T8	
305	0	1882	16	LD	69+	1844	03	1883	16	T11	SET INT ROUTINE
306	0	1883	16	STD	24+	0026	05	1884	16	T11	FOR MONITORING
307	0	1884	16	LD	69+	1845	03	1885	16	T11	
308	0	1885	16	STD	24+	0439	05	1893	16	T11	
309	0	1886	16	RAL	65+	1688	12	1887	16	A7	CHECK FOR INT
310	0	1887	16	SL	16+	1837	03	1888	16	A7A8	ROUTINE
311	0	1888	16	BRNZ	45+	1684	12	1846	15	A8A9G1	
312	0	1889	16	LD	69+	1891	16	1890	16	G13/3	
313	0	1890	16	STDA	22+	1891	16	1842	03	G13/3K23	
314	1	1891	16	LD	69+	0000	00	1892	16	G15/1	STORE C OR ZERO
315	0	1892	16	STD	24+	1632	02	1874	15	G15/1G16	IN P6
317	0	1893	16	LD	69+	1898	16	1894	16	T11	
318	0	1894	16	STD	24+	0426	05	1635	02	T11A1	
319	0	1895	16	LD	69+	1897	16	1896	16	T8	
320	0	1896	16	STD	24+	0426	05	1719	13	T8B1	
321	0	1897	16	CONST	22+	0029	05	8001	00	K37	NORMAL BR EXIT
322	0	1898	16	CONST	22+	0029	05	1899	16	K38	MOD BR EXIT
323	0	1899	16	RAL	65+	0029	05	1810	11	G19	I MINUS ONE
324	0	1810	11	SL	16+	1609	01	1811	11	G19	
325	0	1811	11	STL	20+	0029	05	1872	15	G19G15	

M CODE - INDICATES THAT THE OPERATION CODE - DATA ADDRESS OR INSTRUCTION ADDRESS OR ANY COMBINATION OF THESE IS TO BE MODIFIED

M - 1 DA IS TO BE MODIFIED
2 IA IS TO BE MODIFIED
3 OP IS TO BE MODIFIED
4 DA - IA ARE TO BE MODIFIED
5 DA - OP ARE TO BE MODIFIED
6 IA - OP ARE TO BE MODIFIED
7 DA-IA-OP ARE TO BE MODIFIED
8 8000 INSTRUCTION NOT ENTERED INTO 650
9 CODING IDENTIFICATION NOT ENTERED INTO 650

NOTE - CARDS APPARENTLY MISSING IN ABOVE ROUTINE WERE NOT NEEDED IN SAM 2 BUT ARE USED IN SAM 1

THE MIT INSTRUMENTATION LABORATORY AUTOMATIC CODING 650 PROGRAM

R. H. Battin, R. J. O'Keefe, and M. E. Petrick
Massachusetts Institute of Technology

I. Introduction

At the MIT Instrumentation Laboratory we are faced with a situation which is undoubtedly not unique in the computing business. From our many fields of activity come a wide variety of scientific problems requiring machine computation; at the same time, our staff of programmers is quite small indeed. Therefore, it is necessary for the various engineers, with whom these problems originate, to prepare their own programs for the machine. We have always felt that many advantages might accrue from such an arrangement if the problem of coding and mistake diagnosis could be considerably simplified. Having then, as our prime objective, that of making programming as natural and inartificial as possible, we have developed a mnemonic general purpose floating decimal interpretive routine for the IBM Type 650.

This routine consists of many arithmetical, functional and logical operations and occupies approximately 1500 storage locations. Approximately 350 of these storage registers are required by the "read-in"routine which is used to translate the programmer's program. and enter it on the drum. This portion of the interpretive routine may be used by the programmer to store data. Therefore, the programmer has roughly 850 storage locations in which to store both his program and data. These were chosen within the first one thousand drum storage locations so that a three digit address is sufficient to specify the location of the programmer's data.

Among the outstanding features of this routine is the mnemonic coding system used in preparing programs. The ease with which programming is accomplished may be illustrated by means of a few examples.

1. To add the contents of storage locations 201 and 202 we write:

P201 A P202 .

We use P for plus and M for minus to affect sign control on the factors used in the various arithmetical and functional operations. Thus, if we wished to subtract the contents of storage location 202 from the contents of 201, we would write:

P201 A M202 .

2. The result of each computation is automatically stored in location 000. To multiply the previously computed quantity by the negative of the contents of storage location 201 and store the result in register 203 we write:

M201 X P000 S 203 .

3. To compute the sine of the quantity in storage location 202 and store the result in register 204 we write:

SIN P202 S 204 .

In all there are provided twenty-five possible arithmetical operations and seven functional operations which are discussed in more detail below. In addition to these there are sixteen so-called "logical" operations which are used to control data input and output, to facilitate conditional branching and cycling, to enable the programmer to modify his own interpretive instructions and to facilitate checking procedures. In general, the programmer will write his entire program in this symbolic form. If he wishes to incorporate any normal non-interpreted basic 650 instructions with his program, he writes, for example:

BR650 250 .

The next instruction will be taken from register 0250 and it and subsequent instructions are not interpreted. This mode of operation continues until control is transferred back to the interpretive routine by means of an appropriate instruction address.

II. The Read-In Program

The programmer prepares a program for the 650, using this symbolic notation, on the form shown on page 12 . At the bottom of this form is a summary of the various arithmetical operations and the "correct" spelling of functional and logical mnemonic codes. Each instruction is punched on a separate card. The last instruction of every program must be FINIS. This must not only be the last instruction executed but must also be physically located at the end of the program deck. After the card containing the FINIS instruction are placed cards containing the floating decimal data and basic 650 non-interpreted instructions, if any. These are punched six to a card and are read into the machine under the control of the programmer's program.

During the read-in phase each program card is read and converted into a single ten-digit instruction. Since no alphabetical device is employed, appropriate selector wiring is used to convert alphabetic characters into numeric codes. A single instruction is dissected into ten parts and enters the machine as ten separate words. The division is as follows:

Word 1:	Card number	(cols. 1 - 5)
Word 2:	Entry point number	(cols. 6 - 7)
Word 3:	Sign of first operand	(col. 8)
Word 4:	Location of first operand	(cols. 9 - 11)
Word 5:	First operation	(col. 12)
Word 6:	Sign of second operand	(col. 13)
Word 7:	Location of second operand	(cols. 14 - 16)
Word 8:	Second operation	(col. 17)
Word 9:	Sign of third operand	(col. 18)
Word 10:	Location of third operand	(cols. 19 - 21)

The card number is used only for sequence checking so that the machine will stop on read-in if the cards are out of normal sequence. Entry point numbers are assigned by the programmer only to those instructions which are referred to by the program itself as when branching or modifying instructions. When not used, these columns are left blank and zeros are filled in by means of selector wiring. The entry point artifice makes it unnecessary for the programmer to know precisely where each instruction is being stored on the drum. Thus, instructions may be added or removed from a program deck without, in general, necessitating any alteration in the remainder of the program.

The signs of the three operands enter the machine coded as an 8 for plus and a 9 for minus. The distinction between arithmetical operations and functional or logical operations is made in the Type 533 by means of selector wiring. If Word 4 is negative, the numerical portion of the word is a unique code representing one of the functional or logical instructions. The particular instruction is then determined by a table look-up operation.

The first operation code stored in Word 5 has significance only for arithmetical instructions. There are four possibilities:

No Operation:	88	Multiply:	8 9
Add:	99	Divide:	9 8

The second operation code stored in Word 8 has significance for arithmetical and functional operations. Here, there are five possibilities:

No Operation: 888 Multiply: 898

Add: 998 Divide: 9 88

Store: 889

If the instruction calls for a functional operation, the second operation code can be only No Operation or Store.

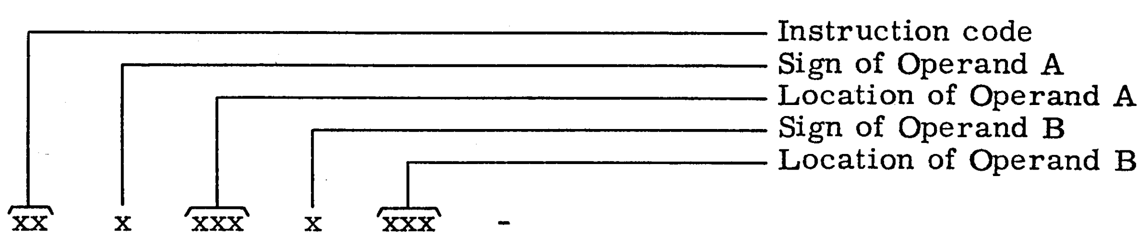
After the read-in program has translated each program card and stored the coded instructions in consecutive locations, the entire program is then searched to replace entry point numbers, counter numbers, etc., by actual drum addresses. When this has been accomplished, the computer stops and a signal is provided to inform the operator that computation is ready to begin.

The read-in routine is programmed so that the card reader will operate at 200 cards per minute. This has been accomplished in part by doing as much of the translating as possible on the control panel. The entire selector capacity of a standard Type 650 is utilized for this purpose.

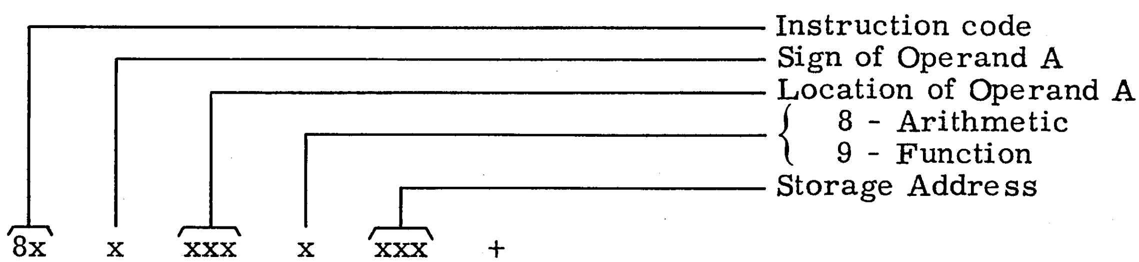
III. Instruction Form and the General Interpretation Routine

The various interpreted instructions are divided into three catagories and are assembled in the machine by the read-in program. They appear in storage in the following form:

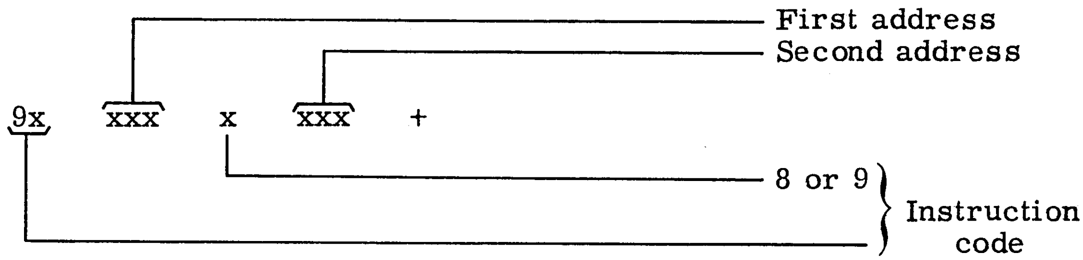
Class I. Arithmetic (no store)



Class II. Arithmetic (store) and functional operation



Class III. Logical operations



The general interpretation routine selects instructions in sequence from consecutive storage locations. The sign of the instruction separates Class I from Classes II and III instructions. An 8 or 9 in the tenth digit position separates Class II from Class III instructions. An 8 or 9 in the fourth digit position of a Class II instruction distinguishes between arithmetical and functional operations.

This particular catagorical break-down and instruction form was selected so that the general interpretation routine would be as fast as possible. Each class of instruction is handled as follows:

Class I: Operands A and B are stored in fixed locations with appropriate signs and control is transferred to the relevant subroutine.

Class II: Operand A is stored with appropriate sign in a fixed location and the address location for storing the computed result is inserted in the data address portion of a certain instruction. Control is then transferred to the relevant subroutine.

Class III: Control is transferred immediately to the appropriate subroutine.

IV. Description of the Computer Operations

At the bottom of the programming form on page 12 is shown a complete list of the possible arithmetic operations. The factors A and B may be selected from any storage locations with complete freedom on the control of the algebraic sign. The factor K is the result of the immediately preceding computation and has the address 000. The result of each arithmetical and functional computation is always automatically stored in this location. Sign control for the factor K is possible only where indicated in the table of arithmetic operations. For arithmetic computations involving three operands the first operation indicated is always the first to be performed. Thus

$$A \times B + K \equiv (A \times B) + K .$$

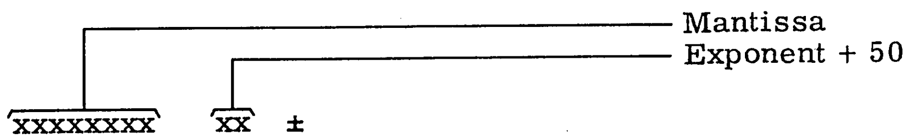
The various functional operations provided require but little comment. Complete sign control on the argument is provided. The only possible second operation is Store. Since the result of the computation is also automatically stored in register 000, we need not store it again elsewhere if it is to be used only on the immediately following instruction. In such a case the second operation and corresponding location may be left blank.

A variety of logical operations are included so that flexibility and versatility of programming may be achieved in the interpreted mode of operation. Sample instructions describing these logical operations follow.

1. Input and Output Control Operations

READ 200

The location specified must be the initial address of any band of storage. The six floating decimal numbers on the data card are read into the first six words of storage of the specified band. The number form as it appears on the data card is as follows:



However, in the machine, the exponent and the mantissa are in the reverse order. The word DATA is punched in columns 77-80 for all cards containing floating decimal numbers. Cards which contain basic 650 instructions have the numbers 650 punched in columns 78-80.

A six digit identification is read into the seventh word together with a 50 exponent emitted from the control panel to form a floating decimal identification which may be used at the programmer's discretion. Words 8, 9, and 10 of the read band are left undisturbed by this operation.

PUNCH 200

The location specified must be the initial address of any band of storage. The form of the output card is the same as that of an input data card. The first six words of the punch band are punched as data while the seventh word is punched as a six digit identification. The eighth word is punched as a five digit card number so that output cards may be ordered if desired.

2. Counters and Cycling Operations

RSCT 001 050

Ten counters are provided in the computer to facilitate cycling operations. The instruction, Reset Counter (RSCT) in the example above, resets counter number one to the value fifty. The counter values are stored in locations 901-910 and may be used, for example, to identify output cards.

BRCNT 003 005

The Branch and Count (BRCNT) operation is used to affect the cycling. First, the value of counter number three is reduced by one. If the counter value is now zero, no branching occurs and the next instruction is taken in normal sequence. If the counter value is not zero, the normal sequence of operations is broken and the next instruction executed is the one having the entry point number 5. Instructions are then taken in normal sequence from this point.

3. Branching Operations

BRNCH 006

The Branch (BRNCH) instruction causes the normal sequence of instructions to be interrupted unconditionally. The next sequence of orders begins with the instruction having the entry point 6.

BRMIN 225 004 BRNOZ 225 004

The two conditional branching instructions, Branch Minus (BRMIN) and Branch Non-Zero (BRNOZ), test the contents of storage location 225. If the number tested is negative or different from zero, respectively, the normal sequence of operations is interrupted and the next instruction executed is the one having the entry point number 4.

BR650 250

The usefulness of this instruction is discussed at the end of the introduction.

TA 008

Quite often the programmer would like to incorporate, within his main program, a subroutine consisting of interpreted instructions to which he would branch from several different points in his main routine.

After executing the subroutine, he would then like to return to the point in his main routine from which he branched. This may be accomplished quite easily by means of the instruction Transfer Address (TA). The method by which this is done may be seen from the following example:

Main Program	Sub-Program
. . .	01 TA 008
BRNCH 001	. . .
. . .	08 BRNCH 000
BRMIN 252 001	
. . .	

In this example we interrupt the main program at two different points, each time branching to entry point 1. The TA instruction then causes the return address for the branch instruction located at entry point 8 to be set so that control will be transferred back to the main program to the point from which it left after execution of the subroutine. The branch instruction at the end of the sub-program may be of either the conditional or unconditional variety.

4. Instruction Modification Operations

AOL 005 AOR 005

The instructions Add One Left (AOL) and Add One Right (AOR) are used to enable the programmer to modify his own interpreted instructions. The effect of these operations is to increase by one the left or right hand addresses of the instruction having the entry point 5. For the case of an arithmetic instruction having three operands, there is no ambiguity since one of these addresses must be 000. Clearly, we never wish to modify this address.

RSTL 005 RSTR 005

The instructions Restore Left (RSTL) and Restore Right (RSTR) are used to restore the left or right hand addresses of the instruction having the entry point 5 to the value originally assigned by the programmer. Since the machine must remember these original values, all instructions referred to by Restore Operations are stored in their original form at the end of the program. This is handled automatically by the read-in program and need be of no concern to the programmer.

5. The Check Stop Operation

STOP 010

The Check Stop instruction (STOP) is used by the programmer to aid in checking out his program. Any number of stop orders may be placed throughout a program at convenient spots. They must be followed immediately by an arithmetical or functional operation and be numbered consecutively beginning with 010. Thus, if three stop orders are used, the addresses must be 010, 011 and 012 but do not necessarily have to appear in the program in that order.

Under normal operation the stop order will have no effect on the computation. However, if the Address Selection Switches are set at 0010 and the Control Switch set at Address Stop, the machine will execute the instruction immediately following the relevent stop order and then stop computation. The two operands A and B and the results of the computation will be stored in the upper and lower accumulator and the distributor. Then, by means of the Display Switch, the programmer may check these quantities. Because the sign of the two halves of the accumulator must be identical, the order of the factors A, B, and K cannot always be the same. There are three possible arrangements and the order section of the program register indicates the appropriate arrangement to the programmer according to the following code:

Code	Upper	Lower	Distributor
00	A	B	K
10	A	K	B
11	K	B	A

If the programmer is satisfied with the result, he may depress the Program Start button and the computation will continue from that point.

Because the Address Stop Switch is used with the STOP instruction, the address set into the Address Switches must be one which is encountered only during the STOP order. The read-in routine counts the number of STOP instructions as the program cards are read and shifts the program in storage by this amount. Storage locations 001 through 009 are used to store some frequently used constants as indicated at the bottom of the program form. Then beginning with register 010 a block of storage is cleared to be used by the STOP instructions. After this the program is stored consecutively and immediately following the FINIS instruction is placed a duplicate set of those instructions referred to by RSTL and RSTR operations. This is all handled automatically by the read-in program. However, the programmer must be cognizant of these facts when planning his data storage.

V. Programming Examples

We shall now illustrate the ease with which programs may be prepared by means of a few examples.

Example 1: Consider the problem of finding the root of the transcendental equation

$$\cos x = x$$

We shall program this as an iteration process using the well-known Newton's method. In general, to solve the equation

$$f(x) = 0$$

it can be shown that if x_i is in approximation to the root, then

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

will be a better approximation. In our case

$$f(x) = \cos x - x$$

so that

$$x_{i+1} = x_i + \frac{\cos x_i - x_i}{\sin x_i + 1}$$

As a criterion for stopping the iteration let us require that

$$|x_{i+1} - x_i| \leqq 10^{-6}$$

An annotated program to solve this problem is shown on page 12. From this form program cards are prepared and listed on an IBM Type 402 tabulator. The programmer now has the opportunity to proofread and check his program for error in analysis and punching. A copy of the proof for this problem is reproduced below.

```
10        READ        100
20        P009             S   127
30   01   COS         P127 S   100
40        SIN         P127
50        P001   A    P000
60        P100   A    M127 D  P000
70        P127   A    P000 S   102
```

```
 80        P127  A  M000
 90        ABVAL    P000   S    103
100        P102            S    127
110        P101  A  M103
120        BRMIN     000        001
130        PUNCH     100
140        FINIS
```

These cards are then read into the machine under the control of the read-in program and computation begins. Six iterations were required to obtain the desired accuracy with a total computation time of six seconds not including the time required to read in the program.

Example 2: To illustrate the use of a few of the logical operations, consider the problem of reading in 60 numbers into the machine and storing them away in consecutive storage locations beginning with register 301. Ten data cards are required and the following program will do the job:

```
      RSCT     001        010
02    RSCT     002        006
      READ     200
01    P201            S   301
      AOL      001
      AOR      001
      BRCNT    002        001
      RSTL     001
      BRCNT    001        002
```

Example 3: Two ten dimensional vectors are to be multiplied together. The elements of the first vector are stored in locations 201-210 while those of the second are stored in 301-310. The following program performs the multiplication and stores the result in register 350.

```
      P201  X  P301
      RSCT      001         009
01    P202  X  P302   A   P000
      AOL       001
      AOR       001
      BRCNT     001         001
      P000            S     350
```

INSTRUMENTATION LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Page ____ of ____

MITILAC 650 PROGRAM

Problem No. ________ Written by ________________________ Project No. ________

Description: Root of cos x = x

Card Number 1	2	3	4	5	Entry Point Number 6	7	Operation / Sign 8	Location 9	10	11	Oper 12	Sign 13	Location 14	15	16	Oper 17	Sign 18	Location 19	20	21	Functions and Logical Operations / Arithmetic Operations / Remarks
0	0	0	1	0			R	E	A	D			1	0	0						$10^{-6} \rightarrow 101$
0	0	0	2	0			P	0	0	9						S		1	2	7	$x_0 = 0 \rightarrow 127$
0	0	0	3	0	0	1	C	O	S			P	1	2	7	S		1	0	0	$\cos x_0 \rightarrow 100$
0	0	0	4	0			S	I	N			P	1	2	7						$\sin x_0$
0	0	0	5	0			P	0	0	1	A	P	0	0	0						$1 + \sin x_0$
0	0	0	6	0			P	1	0	0	A	M	1	2	7	D	P	0	0	0	$(\cos x_0 - x_0) / (1 + \sin x_0)$
0	0	0	7	0			P	1	2	7	A	P	0	0	0	S		1	0	2	$x_1 \rightarrow 102$
0	0	0	8	0			P	1	2	7	A	M	0	0	0						$x_0 - x_1$
0	0	0	9	0			A	B	V	A	L	P	0	0	0	S		1	0	3	$\|x_0 - x_1\| \rightarrow 103$
0	0	1	0	0			P	1	0	2						S		1	2	7	$x_1 \rightarrow 127$
0	0	1	1	0			P	1	0	1	A	M	1	0	3						$10^{-6} - \|x_0 - x_1\|$
0	0	1	2	0			B	R	M	I	N		0	0	0			0	0	1	
0	0	1	3	0			P	U	N	C	H		1	0	0						
0	0	1	4	0			F	I	N	I	S										

Arithmetic Codes
P: Plus
M: Minus
A: Add
X: Multiply
D: Divide
S: Store

Functional Codes
SIN
COS
ARCTN
SQRT
ABVAL
EXP
LOG

Logical Codes
READ AOL
PUNCH AOR
BRNCH TA
BRMIN RSTL
BRNOZ RSTR
BRCNT RSCT
BR650 STOP
DIFEQ FINIS

Location of Constants
001: 1
002: 2
003: 3
004: $180/\pi$
005: 5
006: $\sqrt{2}$
007: e
008: π
009: 0

Arithmetic Operations

$A + B$	$A \times K + B$	$A \div K + B$
$A + B + K$	$A \times B - K$	$K \div A + B$
$A + B - K$	$A \times B \times K$	$A \rightarrow B$
$A + B \times K$	$A \times B \div K$	$A + K \rightarrow B$
$A + K \times B$	$A \times K \div B$	$A - K \rightarrow B$
$A + B \div K$	$A \div B$	$A \times K \rightarrow B$
$A + K \div B$	$A \div B + K$	$A \div K \rightarrow B$
$A \times B$	$A \div B - K$	$K \div A \rightarrow B$
$A \times B + K$		

IL-124

VI. Checking Procedures

In Section IV we described the operation of the STOP instruction and how it is used in checking out programs. A number of other procedures have been devised for this purpose to facilitate checking which will now be discussed.

One of the most common errors in programming occurs in arithmetic operations which involve three operands. If none of the three addresses is 000, the read-in program will translate this instruction into a different order and not detect the error. To combat this, we have wired our tabulator to check this kind of error when preparing a listing of the program. Thus, if any instruction of this kind appears in a program, the tabulator will print an asterisk beside it.

The read-in routine is designed to check for several kinds of program errors during the read-in phase. The card number sequence of program cards is checked and the machine will stop if the cards are out of order. Also the spelling of functional and logical mnemonic codes is checked. In most cases if information is punched in the wrong columns of the card, this will be detected by the validity checking circuits of the 650 itself.

Several kinds of errors are checked while the actual computation is being performed. The interpreted READ instruction checks the numbering sequence of the data cards as they are read into the computer. The machine will stop if the cards are out of order. Also the machine will stop if instructed to take the square root of a negative number. In this case the lower accumulator will contain the number whose square root is called for and in the distributor will be the instruction itself. The operation code in the program register will be 50 to indicate the reason the machine stopped.

If a functional operation using a series computation has an operand which exceeds 10^{10} in magnitude, the machine will stop. The argument will be in the lower accumulator, the instruction in the distributor, and a 40 in the operation section of the program register.

The most useful device available to the programmer for checking out his program is the list mode of operation. When the read-in program has translated all program cards, the machine stops. By a proper setting of the storage entry switches, the programmer then has the option to have each arithmetical and functional operation punched out on a separate card. The programmer will then have a complete listing of just how his program is functioning. By using this procedure in conjunction with the STOP instructions, the programmer may alternate between the list mode and the normal mode of operation at will. Each card that is punched while listing contains the operation code, the addresses of the operands, the operands themselves, and the result of the computation. On this and the following pages is shown a partial listing of the program for Example 1 discussed in Section V. Because of paper size limitations the listing from the tabulator is shown in this paper in two parts.

Operation	L(A)	L(B)	L(C)
A	P009		S127
F2	P127		S100
F1	P127		S000
A + B	P001	P000	
A + B / C	P100	M127	P000
A + B	P127	P000	S102
A + B	P127	M000	
F5	P000		S103
A	P102		S127
A + B	P101	M103	
F2	P127		S100
F1	P127		S000
A + B	P001	P000	
A + B / C	P100	M127	P000
A + B	P127	P000	S102
A + B	P127	M000	
F5	P000		S103

Operand A	Operand B	Operand C	Result
00000000 00			
0			10000000 50
			*00
10000000 50	*00		10000000 50
10000000 50		10000000 50	10000000 50
00000000 00	10000000 50		10000000 50
0	10000000 50		10000000*50
10000000*50			10000000 50
10000000 50			10000000 50
10000000 44	10000000 50		99999900*49
10000000 50			54030230 49
10000000 50			84147098 49
10000000 50	84147098 49		18414710 50
54030230 49	10000000 50	18414710 50	24963613*49
10000000 50	24963613*49		75036390 49
10000000 50	75036390 49		24963610 49
24963610 49			24963610 49

VII. Differential Equations

A subroutine has been incorporated in our interpretive program in an effort to simplify the problem of solving simultaneous differential equations. The procedure used is due to S. Gill* and is essentially a Runge-Kutta fourth order process. To use this routine the equations to be solved must be of the form

$$\frac{dy_i}{dt} = f_i (y_1 , y_2, \dots y_n ; t)$$

where i=1, 2, . . . , n. The programmer stores the initial values of the dependent variables in consecutive locations beginning with register 827. The initial value of t is placed in location 849 while the increment in t, designated by h, is stored in register 949. (h/2 must also be placed in register 899.) At the start of the program a block of storage beginning with register 927 is cleared and is reserved to be used by the subroutine to store the so-called "bridging q's". The programmer then computes the right hand sides of the differential equations $f_1, f_2, \dots, f_n$ and stores them consecutively beginning with register 877. The instruction

DIFEQ 002 004

is all that is required to initiate the subroutine. The second location of the instruction is used to specify the number of differential equations to be solved. The first location specifies the entry point corresponding to that instruction which initiated the computation of the f's. This is needed because, in the Gill routine, four sub-steps are required to advance the integration by one time step h and the f's must be recomputed for each of these sub-steps. This is all handled automatically by the DIFEQ routine. After execution of the Differential Equations instruction, the previous values of the dependent variables will be replaced by the new values corresponding to h units of time later and t will be replaced by t + h.

This technique will be clarified by an example. Suppose that the following two differential equations together with the indicated initial conditions are to be solved.

$$\frac{dy_1}{dt} = y_2 \qquad \frac{dy_2}{dt} = -y_1$$

$$y_1(0) = 0 \qquad y_2(0) = 1$$

---

* A Process for the Step-by-Step Integration of Differential Equations in an Automatic Digital Computing Machine , S. Gill, Proceedings of Cambridge Philosophical Society, Vol. 47, Part 1.

Taking h = 0.2, the following program will compute five values of y_1 and y_2 and punch five output cards:

```
 10          P009                S   927    Set q1 = 0
 20          P009                S   928    Set q2 = 0
 30          P001   D   P005
 40          P000                S   949    Set h = 0.2
 50          P000   D   P002     S   899    Set h/2 = 0.1
 60          P009                S   849    Set t = 0
 70          P009                S   827    Set y1 = 0
 80          P001                S   828    Set y2 = 1
 90          RSCT        001         005
100   01     P828                S   877    Compute f1
110          M827                S   878    Compute f2
120          DIFEQ       001         002
130          P849                S   829    Transfer t
140          PUNCH       800                  to a punch
150          BRCNT       001         001        location
160          FINIS
```

AN INTEGRATED COMPUTATION SYSTEM FOR THE IBM 650

C. K. Titus
Westinghouse Electric Corporation

In order to utilize fully the IBM 650 in the pursuit of general engineering problems, an integrated computation system has been set up at the Air Arm Division of the Westinghouse Electric Corporation. Such a set of procedures increases reliability and efficiency by standardizing and integrating many of the details of computation. There will always be a few special problems, of course, that still may best be treated on an individual basis.

This system to be described is a further step in a continuing expansion of the use of numerical techniques to aid in the Division's primary task of developing, designing, and manufacturing airborne electronic armament. Typical problems encountered are the computation of fire-control pursuit courses by numerical integration and the study of aircraft-autopilot stability by evaluating the system characteristic equation and solving for its roots. The calculation of radar antenna and lens configurations, radar detection probabilities, and power density spectral analyses of experimental data are also representative problems.

This standardization of techniques, which has been developed and is currently in use, has been extended to the following computation phases:

1. Card Formats.
2. Loading Routine.
3. Program Checking Routines.
4. Floating-point Subroutines.
5. Interpretive Program.
6. Presentation of Results.

One of the basic problems in setting up such a system for the IBM 650, since it is a card input and output machine, is the standardization of the card formats. The type of loading routine used governs the layout of the loading cards. The available word space is limited if columns are set aside for card identification, which is a desirable feature. Similarity in form between input and output cards will greatly facilitate punching input data directly through the computer for output identification. The output format must be compatible with the printing requirements. The cards punched by storage unloading routines should preferably be in the correct form for reloading.

Consideration of all of these aspects led to a choice of an "all purpose" five-word-per-card standard at the Air Arm Division. The details of the format are presented in Table I. This basic card is used for five major purposes: loading, data entry, result punching, storage punch-out, and identification printing. The various features will be discussed with the use to which they apply.

Primary in the decision to use this format was the development of an individually assignable five-word-per-card loading routine. This technique was developed as the result of an effort to utilize the wasted pre-punched columns of the standard four-word routine. The program is essentially the same, but the fundamental difference is that the information which formerly had to be pre-punched is emitted from the 533 control panel. Co-selectors, picked up by an X-punch in column 11, channel the correct digits into the proper word-entry hubs. This wiring, which requires no extra features on the panel, can be devised by anyone familiar with IBM wiring procedures, and so is not illustrated here.

This method allows the full use of the ten-word input of the 650. The five ten-digit words are read into Words 1 through 5, and the five corresponding four-digit location addresses, packed with appropriate constant information, enter Words 6 through 10. Since each of the words thus requires only 14 columns on the card, the loading data occupies a total of 70 columns, conveniently leaving ten columns for identification, as indicated in Table I.

The loading routine, which requires only six words permanently located in storages 1994 through 1999, is presented in Table II. The numbers in the brackets are those which are emitted. It is to be noted that once the routine is on the drum, entry is through 1999, a number easy to remember and to set into the console.

In order to load this routine, the program listed in Table III, which is placed on two standard loading cards, can be used. The numbers emitted from the control panel simultaneously with the first card are superfluous; nevertheless zeros must be filled in the location addresses. The second card, however, makes full use of this information so that, in a sense, the loading routine loads itself.

In standardizing on the five-word loading routine, it was felt that the sequential limitations of the seven- or eight-word methods nullify their speed advantages. In addition, an element of confusion is eliminated by using only one type of routine. It is believed, therefore, that the five-word technique represents an optimum compromise.

The purpose of the data-entry card, as distinguished from the loading card, is to allow read-in of data, under control of the program, during the course of a computation. For this application, the same format of Table I is used with the exception that the five location addresses are

TABLE I. IBM 650 "ALL PURPOSE" CARD FORMAT

Columns	Function
1 - 6	Job number
7 - 10	Card number
7X	Unconditional skip to next page
8X	Conditional skip to next page
9X	Double space
10X	Single space
11X	Loading card
12X	Title card
13 - 52	Alphameric field, Title card
11 - 14	Word 1, Location
15 - 24	" Contents
24X	" Sign
25 - 28	Word 2, Location
29 - 38	" Contents
38X	" Sign
39 - 42	Word 3, Location
43 - 52	" Contents
52X	" Sign
53 - 56	Word 4, Location
57 - 66	" Contents
66X	" Sign
67 - 70	Word 5, Location
71 - 80	" Contents
80X	" Sign

TABLE II. FIVE-WORD-PER-CARD LOADING ROUTINE

Location	Instruction			Data Locations
1999	70	1951	1994	
1994	69	1951	1956	C(1951) = W_1
1956	[24]	L(W_1)	[1995]	
1995	69	1952	1957	C(1952) = W_2
1957	[24]	L(W_2)	[1996]	
1996	69	1953	1958	C(1953) = W_3
1958	[24]	L(W_3)	[1997]	
1997	69	1954	1959	C(1954) = W_4
1959	[24]	L(W_4)	[1998]	
1998	69	1955	1960	C(1955) = W_5
1960	[24]	L(W_5)	[1999]	

TABLE III. PROGRAM FOR PLACING LOADING ROUTINE ON THE DRUM

	Location	Instruction			Data Locations
Console	8000	70	1951	1951	
Card #1	1951	69	1954	1952	C(1954) = 69 1951 1956
	1952	24	1994	1955	C(1953) = 00 0000 0000
	1955	70	1951	1994	
Card #2	1994	69	1951	1956	C(1951) = 69 1952 1957
	1956	[24]	1995	[1995]	
	1995	69	1952	1957	C(1952) = 69 1953 1958
	1957	[24]	1996	[1996]	
	1996	69	1953	1958	C(1953) = 69 1954 1959
	1958	[24]	1997	[1997]	
	1997	69	1954	1959	C(1954) = 69 1995 1960
	1959	[24]	1998	[1998]	
	1998	69	1955	1960	C(1955) = 70 1951 1994
	1960	[24]	1999	[1999]	

left blank. The five words enter Words 1 through 5. To be available for use in identification of results, the job number and card number columns are also wired to enter Word 6. Although it would be possible to read in as many as eight ten-digit words from a data card, the use of five was chosen as standard in order to be compatible with the result cards, which are restricted by printing requirements. Primarily, this choice eliminates a very difficult bookkeeping situation in punching data straight through the computer in order to have input values available with the corresponding output values. The data card, then, and the loading card, are the two methods available for entering information into the computer under this integrated system.

An integrated computation system must also include routines to aid in program checking. One of the most important of these for the 650 is the storage punch-out program because it allows the programmer to examine his problem in detail at his desk rather than to use an excessive amount of time at the console. Within a group just "graduated" from the CPC, finding errors from a storage print-out is a new concept. The programmers are used to checking their problem with a step-by-step detail, the only method available on the CPC. Consequently there is a tendency toward spending too much time at the console because of inefficient use of step-by-step procedures such as the half-cycle feature, or a tracing or auto-monitor routine. Such techniques, when judiciously used, can be very effective, but finding errors from a storage print-out is a method which should be assimilated for all-around efficient utilization of the 650.

In conjunction with the storage punch-out routine, a storage-erase program is useful so that discrimination can be employed to punch out only those registers which contain significant data. For this system, the erase routine places 8's in all locations except those containing the subroutines. The choice of all 8's was based on the consideration that this configuration is more unique than 9's or 0's. It enables one readily to discern, for instance, those quantities which have become zero as a result of the calculations. This particular routine was written so that it enters directly into the load routine, thus encouraging its use as a preliminary step to loading.

Control of the punch-out routine is effected by setting the data-address portion of the storage entry switches to the address of the first instruction to be punched out. Following logically, the last location to be punched is set on the instruction address switches. The program then punches out the location and contents of all storage units within these limits set on the console, bypassing those which contain all 8's. In a manner analogous to the loading routine, the five words are punched from Words 1 through 5, and their location addresses from Words 6 through 10, making full use of the ten-word output of the 650.

As was indicated previously, it is desirable to have the punch-out card format in exactly the same form as the loading card. This feature is very useful for "roll-back" or restart purposes. It is also useful for obtaining a consolidated loading deck after many changes and additions have been made to a given program. Use of the Word 10 control information hubs and selector wiring on the control panel enables one, under control of the program, to punch out in the correct loading form.

Another program checking aid, which has been briefly mentioned, is the tracing, or auto-monitor, routine. The purpose of such routines is to provide an instruction-by-instruction punch-out of the arithmetic details of each step. If properly used, these routines can be very helpful. However, for the present, use of a tracing routine at the Air Arm installation is being discouraged for two reasons: first, it is too often badly misused, resulting in inefficient computer operation; and second, the more general storage punch-out technique has certain advantages which should be encouraged.

For the majority of engineering calculations at this installation, floating-point subroutines have been found to be almost indispensable. The ability to eliminate scaling considerations and thus permit extensive generality in programming has proven to be of considerable importance. Consequently the integrated system includes floating-point subroutines for all of the following functions which experience with the CPC has shown to be frequently used in aircraft armament engineering calculations.

1. Add
2. Subtract
3. Multiply
4. Divide
5. Square Root
6. Exponential
7. Logarithm
8. Sine
9. Cosine
10. Inverse Tangent

It is not within the purpose of this paper to consider these subroutines in detail, but rather to show how they have been coordinated with the system. In particular, because the 650 is a relatively medium-size machine on the scale of present-day computers, it is important to have calling sequences which are efficient in computing time and storage space. Yet, flexibility is desirable also, in the form of being able to call factors from any location, and to return to any location. As a result of considering these requirements, a system of calling sequences listed in Table IV was evolved. There are two fundamental types, corresponding to one- and two-operand routines, respectively. Although these calling sequences were devised specifically for floating-point subroutines, their use is obviously not confined to this application. The same forms would be used for any type of one- or two-operand subroutines.

Return from the subroutines is made by storing the next instruction in a location (1875) which each subroutine refers to for its last instruction. For two operands, the storage of the next instruction must be done during the calling sequence; but for the one-operand type, it is done within the individual routine. The results of all subroutines are placed in the lower accumulator. Thus, for continuing calculations, the RAL or RSL instruction can often be omitted, shortening the calling

TABLE IV. FLOATING-POINT CALLING SEQUENCES

Two Operand Type

General Form:

Location	OP	DA	IA
α_1	LD	α_5	α_2
α_2	STD	1875	α_3
α_3	β	L(B)	α_4
α_4	LD	L(A)	γ
α_5	Next Instruction		

Specific Data:

Operation	β	γ
A + B	RAL	1850
A - B	RSL	1850
A x B	RAL	1900
A x (-B)	RSL	1900
A / B	RAL	1950
A / (-B)	RSL	1950
Arc Tan A/B	RAL	1600
Arc Tan A/(-B)	RSL	1600

One-Operand Type

General Form:

Location	OP	DA	IA
α_1	β	L(A)	α_2
α_2	LD	α_3	γ
α_3	Next Instruction		

Specific Data:

Operation		β	γ
$\sqrt{A}$	$(A \geq 0)$	RAL	1800
$\sqrt{-A}$	$(A \leq 0)$	RSL	1800
e^A		RAL	1650
e^{-A}		RSL	1650
Ln A	$(A > 0)$	RAL	1550
Ln (-A)	$(A < 0)$	RSL	1550
Sine A		RAL	1750
Sine (-A)		RSL	1750
Cosine A		RAL	1700
Cosine (-A)		RSL	1700

sequence accordingly. Other arrangements of the two-operand calling sequence are also possible. The inverse tangent function was written as a two-operand routine in order to allow placement of the angle in the correct quadrant, within the range 0 to 2π. Even if the factor "B" is zero the routine gives the correct answer.

Another important floating-point routine concept that should be standardized is that of providing automatic stops for operations that are invalid. For the routines listed, stops have been provided for all the following improper calculations:

1. Exceeding radix bounds on any operation.
2. Division by zero.
3. Square root of a negative number.
4. Logarithm of a negative number, or zero.
5. Exceeding angular argument bounds on sine or cosine.

The stops are purposely coded to be of the improper storage-selection type so that they cannot be bypassed. Each is numerically coded so that the source of trouble can readily be determined. A further standardization that has been incorporated is the location of each subroutine within a certain band on the drum. Thus, if storage space becomes critical, omitting those routines which are not needed will free complete blocks of instructions for other use. All of these described features have made this floating-point system one which has proven very satisfactory.

In order to facilitate programming, it is often desirable to formulate an interpretive program. Such a program was prepared for the 650 to work in conjunction with the floating-point routines. After considerable study of the possibilities, it was found that the best program technique seems to be one wherein the calling sequences, as presented previously, are built up from the information furnished by the interpretive instructions.

The system was kept simple by standardizing on a two-address code for every operation. Thus for two operands the addresses specify the location of "A" and "B"; and storage in "C", when desired, is accomplished by a separate transfer instruction. For one operand the addresses specify "A" and "C", and thus the extra store instruction is not necessary. In addition, the results of each operation are always placed in 0000, where they can be called out for the next arithmetic operation, or for permanent storage in the case of two operands.

Under this particular system only three extra interpretive instructions were added to the ten mathematical functions. These are the transfer, the branch-on-minus (BRMIN), and the branch-on-non-zero (BRNZ) instructions. Several useful features are provided in this system, such as the ability to punch out a step-by-step detail through setting the console to minus. In addition, by requiring that interpretive instructions be negative, transfer to regular 650 operation can be effected by changing to a positive instruction.

However, a detailed description of this program is not warranted, because experience with it has not been satisfactory for two major reasons. First, interpretation increases calculation time about 50 percent, as compared with calling-sequence programming. Second, since one still has to fall back to straight 650 programming for many bookkeeping operations, the system offers little advantage over calling-sequence coding. Adding more interpretive bookkeeping instructions might improve this situation, but would use too much storage and further slow the calculations. As a result, the interpretive program has been more or less discarded.

Since the methods for entering data into the 650, checking programs, and proceeding with engineering calculations have been discussed, the final phase of presentation of the results remains to be considered. The philosophy adopted for this phase has been that an answer is often no better than its final printed form. Results from a simple calculation presented neatly and well identified can be more impressive than weighty computations poorly marked and printed in haphazard fashion. Of course, this philosophy can be carried to such an extreme that it interferes with system operation, but much can be done before approaching this limit.

Experience with the CPC has shown that printing results on standard 8-1/2 by 11 sheets has marked advantages. It allows results to be conveniently assembled in a standard three-ring binder for easy consultation, and to be included directly, or in a one-to-one reproduced form, in standard size reports. The sheets do not become dog-eared or torn, or even lost, because they do not fit with other material. Consequently, this goal was set for the 650, and was attained by using standard tabulating forms which allow printing sidewise on 8-1/2 by 11 sheets. The eleven-inch width conveniently allows printing of the four-digit card number and five ten-digit words across one line. While it is true that a wider form would permit up to seven words per line, the advantage of the standard size outweighs the occasional advantage of seven words. Printing of the card number with each line provides a ready check on the order of the cards.

Since the output of the 650 under this system is standardized at five words per card, the format of the result card can be made compatible with all card forms, resulting in the "all purpose" form presented in Table I. The five output words are punched from Words 1 through 5 into the five word-content fields on the card. A six-digit job number and a four-digit card number are punched from Word 6. Punching of the card number, or some other sequential information, is particularly important with card output, so that if the cards get out of order before printing they can easily be restored. From Word 10, control information in the form of 8's placed in the proper locations by the program, causes X-punches to be placed above the card number, for printing controls. This feature permits direct control of the 402 printing by the 650 program.

In order to extend the idea of well-identified results, the full alphameric features of the tabulator are employed. Provision for printing titles and headings along with numerical results, without changing control panels, is made by the use of extensive selector wiring. For instance, under this system, an X-punch in column 12 causes the next card to be interpreted as a title card. The one-cycle delay is required in order to be able to select zone impulses.

Spacing and skipping controls on the tabulator provide further refinements in the printed output. By means of the X-punches above the card number, the following four features can be controlled:

1. Unconditional skip to next page.
2. Conditional skip to next page.
3. Double spacing.
4. Single spacing.

The conditional skip occurs only when printing is within a certain distance from the bottom of the page, as determined by the carriage tape. This feature is very useful to cause skipping only after a group of results has been completely printed. These four controls, then, permit a considerable flexibility in the form of the printed results.

For the purpose of printing storage punch-out cards, a separate 402 control panel is required, in order to tabulate the storage location as well as its contents. The programmer is thus provided with an easy-to-read listing of the location and contents of each of the storage units punched out. Wide paper must be used to allow room for the extra information printed, but since such tabulations are usually destroyed once the program is working, the former objections concerning paper size do not apply.

With this description of the printed output standards, the discussion of an integrated computation system for the IBM 650 is complete. Techniques applicable to every phase of the use of this computer in engineering calculations have been presented. Ideas for card formats, loading routines, program checking aids, floating-point subroutines, interpretive programs, and final presentation of results have all been considered. The system is currently in use at the Westinghouse Air Arm Division. It is hoped that this discussion will be of benefit to personnel of similar IBM 650 installations.

DATAMATIC CORPORATION LIBRARY ROUTINES FOR THE 650

R. F. Clippinger and E. E. Comerford
Datamatic Corporation

Datamatic Corporation operates a rather unusual computing service which programs, codes and solves a variety of problems coming from any source. Consequently it has been forced to create a library of routines to make it easy to locate routines in the 650, weed out coding errors, watch the growth of errors and minimize the time to solve a problem.

<u>Subroutines, directory and assembly program.</u> As a method of organizing the use of subroutines we have chosen that outline used by the group with Turing of Manchester. A directory defines where each routine will be placed in the machine. The assembly routine uses the directory to modify the orders as they are read in so that they will work wherever placed. A routine changing sequence is used to pass from one routine to another and keep track, by means of a link list, of the current status of the problem. A word $D_i = O\alpha_i\beta_i$ is associated with each subroutine S_i. α_i is the data address and β_i is the instruction address. When the coder writes his code, he marks some addresses by adding 4000 to the relative address (the relative address is the address that would be used if the subroutine began in register 0). He may mark other addresses by adding 2000 to the relative address. The input routine then removes the marker and adds β_i if the marker was 4000 and α_i if the marker was 2000. Consequently, by changing the single word D_i one can move the subroutine to an arbitrary position in the machine and one can move its working space to an entirely different arbitrary spot in the machine. If two or three people who are trouble shooting problems are sharing the use of the machine, they can adjust their directories so as to avoid conflict and not have to read-in their problems each time. Trouble-shooting routines, to be described later, will make it possible for each of them to get several periods of use of the machine within an hour.

The passage from one routine to a subroutine is affected by an order in register S which loads the distributor with the contents of S + 1 and transfers control to the routine changing sequence. The word in S + 1 has been called by the Manchester group a False Line. As written by the coder it is in the form 88 D_iR, where R is the relative entry point. D_i defines which subroutine is to be entered, and since the subroutine may do different things depending on where it is entered, the relative entry determines what it accomplishes.

At the time of reading the problem into the machine the assembly routine deletes the 88, looks up the directory, adds β_i to the relative entry and creates the true entry. At the same time it uses the information that it has concerning the location of the false line to create a return address so that the false line in the machine is in the form 00, return point, true entry to subroutine.

When the problem is being executed, the Routine Changing Sequence uses the false line to create a link in a link list for later transfer of control to the return point. If the subroutine requires parameters, they are placed following the false line in positions S + 3, S + 4, etc. 8 parameters may be used. The Routine Changing Sequence moves these parameters to registers 1 through 8, if the false line is negative. It then enters the subroutine.

On leaving the subroutine one simply goes to a different point in the Routine Changing Sequence which uses the link list to return to the proper place. At any time in the course of a problem one can tell which subroutine he is in by examining the link list. Since the 650 keeps track only of the order it is trying to execute and not the order that was done before that, any derail leaves one in the embarrassing spot of not knowing where he was. Under these circumstances, the link list is quite a help aided by the state of the bound variables associated with the subroutines.

The assembly routine in addition to modifying addresses and false lines creates a check sum for the routine being read in and compares it with the check sum on an input card. Information being entered into the machine by the assembly routine is preceded by a descriptive card which gives the check sum, the directory number and tells how many orders and numbers are to be inserted. Each card gives the relative address of the second word on the card and how many words are to be put away. The 7 remaining words are useful words to be inserted into the machine. The assembly program has been force programmed (that is optimum programmed) so that the entire machine can be loaded in 2.8 minutes. If the same routine is to be used many times, it can be assembled once, read out by our post mortem routine and after that read in, using absolute addresses. In this case, the machine can be loaded in 1.5 minutes.

Trouble shooting routines. Stop and Punch. Stop and punch is similar to codes available on other computers. The function of stop and punch is to enable one to find out what is going on in the execution of the problem. In using stop and punch the coder prepares a set of words of the form $A_i\ B_i\ N_i$, which are typed in successive word positions on successive cards. A stop and punch marking routine will use these descriptive cards to plant derails at A_i and store the information $B_i\ N_i$. When the problem is executed the result will be that every time the control arrives at A_i, N_i words will be printed starting at B_i. The derails that were planted by the marking routine can be removed by entering stop and punch at a different point. This routine can be used in many ways. The first time a problem is run in the trouble-shooting process a set of descriptive cards will make it print out key quantities which enable one to deduce if certain segments of the code are correct. Having corrected the coding errors that have been discovered, a second set of stop and punch descriptive cards will enable one to punch out further information to check whether the corrections have cleared up all the difficulties. Printing associated with correct portions of code can thus be by-passed. The program is not executed interpretively; consequently, it is not slowed down very much by this trouble shooting mechanism.

A second use for stop and punch is for problem analysis. It may be that a code is correct in the sense that it does what one asked it to do but one may not have been sure how the numbers would behave. For example, in solving a non-linear differential equation it is difficult to predict the growth of the numbers. Without the use of stop and punch it is awkward to provide for all printing which would be useful in this regard. And, in fact, one cannot always predict what quantities one will want to see. With stop and punch one can control the output of the computers with a twist of the wrist, and this brings us to the third use of stop and punch. After one has complete understanding of a problem, the code is correct, and one knows how the numbers behave, he may still decide on a different output from what he had originally planned. Stop and punch makes this possible without elaborate code changes.

A Generalized Monitoring Routine. An interpretive program used throughout the computing industry to locate trouble in desperate situations is known as a monitoring or tracing routine. Such a routine prints out what each order does every time the order is executed. The result is a wealth of information which enables one to find any errors. However, it is extremely wasteful since it is very slow. It is particularly wasteful on a machine like the 701 but it is bad enough on the 650. A simple generalization used in many places consists of marking some of the orders so that one only prints out what these orders do. However, it is usually not very important to see the results of a specific order over and over as one goes around a loop. Such tracing routines are therefore not very useful.

About eight years ago at the Computing Laboratory at Aberdeen Proving Ground one of the authors introduced a different generalization, which we call the code checker. The code checker prints out what every order does the first time it comes to the order and never again. It is therefore much faster than the tracing routine and much more economical of the programmer's time in looking through the output. We have such a code checker for the 650. The specific code checker that we have written for the 650 is used as follows: The coder prepares descriptive cards containing words in the form O C_i N_i ϵ_i or 80 D_i O ϵ_i. A code checker marking routine uses these descriptive cards to mark N_i orders starting at register C_i or the false line at D_i. If ϵ_i is minus marks are inserted; if it is plus, marks are removed. One then inserts the address of the first order in its program into a certain register and enters the code checker, which then prints out for every marked order the address of the order, the order, and the contents of the distributor and lower and upper accumulators. If a false line is marked, the corresponding subroutine is also code checked. If it is not marked, the subroutine is executed at full speed and the routine changing sequence derails the problem back into the code checker upon return from the subroutine. It is thus a simple matter to code check any part of a problem which is causing trouble and not code check the rest of it. These two trouble shooting routines give great flexibility and enable one to trouble shoot problems using a minimum of 650 times. For 650 users having many different uses for the machine such routines are quite important.

<u>Automatic optimum programming</u>. For problems involving considerably more computer time than reading or punch time the 650 can be made to go roughly <u>twice as fast</u> by placing the orders in the right position. This can be done manually. George Trimble has prepared cards as aids to the

optimum programmer which many of you are using for this purpose. An expert like George Trimble can perform this manual optimum programming nearly as fast as random programming. For the rest of us it is handy to make the 650 do it. Consequently, E. E. Comerford has written an optimum programming routine which will force code a routine using 350 registers in about 10 minutes. Using this routine, we are able to give our customers more answers per dollar.

Our optimum programming routine is built around our assembly program and structure of subroutines. One recognizes five different kinds of words: numbers, ordinary orders, orders whose instruction address is made up, orders whose data address is made up and false lines. To use the optimum programming routine, one simply prepares descriptive cards which determine how many of each class of words there are. A second kind of descriptive card determines how many registers are to be used by the optimum programming routine and which pieces of the code are to be improved first. The pieces which are programmed first are programmed best because they have the widest choice of registers. Also, the constants which are common to more than one order are allocated for optimum timing in the first order in which they appear. Consequently, one can speed up the inner loop of a multiple loop induction by improving it before other sections of the program.

AN AUTOMATIC METHOD OF OPTIMUM PROGRAMMING FOR THE 650 USING THE 650

Elmer F. Shepherd
John Hancock Mutual Life Insurance Company

Optimum or minimum latency coding is a technique whereby data or instructions are assigned drum locations in such a manner as to minimize access time.

Optimum programming is accomplished manually by reference to an optimum program table. This provides increments, based on the word time required to carry out operations, to add to an address to determine the module of the next address. For instance, in order to determine where to locate a data address it is first necessary to determine whether the location of instruction address is an odd or even number. Reference is then made to a table by operation code and the increment found is added to the location of instruction. With the module thus formed, an array of all available locations of this module is consulted, one selected, checked off and used as the data address. The instruction address is determined in like manner. The manual technique accomplishes coding one line at a time. The determination of the necessary repetitive use of the same address (constants interim storage or branching) is accomplished visually. Certain addresses, for read in words, punch words table locations, distributions, etc., are, of course, specifically assigned before coding is begun. The manual method is a slow process exposed to large error.

Once coding has been accomplished, a deck of load cards is key punched showing in each card the location of instruction or data and the instruction or data.

The manual method, then makes use of the following elements.

1. Specific assignment of address in advance.
2. Table look up to determine module.
3. Consultation of an array for the determination of arbitrary assignments.
4. Scanning or cross reference for necessary repetitive use of previously assigned addresses.

In the machine method described here the same elements are present.

Necessarily, specific addresses are coded as usual.

An optimum increment table is provided and stored in locations of the 0950 band.

By storing two drum locations in each word, an array of all 2000 locations can be stored in 1000 words. All addresses of one module may be stored in 20 consecutive drum locations. Drum locations 1000 to 1999 are reserved for the entire array. Any locations not available for a particular program are stored as zeros. This is accomplished by first storing zeros in locations 1000 to 1999 (two load cards required) and then selecting from a 2000 card load deck representing all locations, the ones available for the particular program being made optimum.

The operation portion of an instruction is coded normally. In order to accomplish cross reference pseudo addresses consisting of the 9000 series are used. A 9000 series address causes the machine to select the best normal address available and store it in the drum location specified by the three low order positions of the pseudo address if such drum location contains zeros, or to recall the address stored there if such drum location does not contain zeros. Drum locations 0100 to 0699 and 0000 are reserved for this purpose and are filled with zeros, by two load cards, at the start of the process. Locations 0100 to 0699 provide 600 numbers for cross reference. A program may be of considerable length, but since the instruction address of one instruction is most frequently the drum location of the next following instruction (as written) and this is the <u>only</u> cross reference, drum location 0000 is assigned for the store and recall sequence, and is filled with zeros after each recall. This diminishes the need to use 0100 to 0699 locations. As addresses are selected from the array in 1000 to 1999, they are replaced with zeros.

In the case of 800X addresses, equivalent drum locations must be calculated to proceed to build the next following address module. However, it is seldom necessary to carry such equivalent to the next instruction since an 800X referred to instruction would most frequently contain a normal data address. Drum locations 0001 to 0099 are provided for storage and recall of equivalent 800X addresses if needed and the two low orders of such addresses replace the two middle digits of the 800X address. There is no objection to using 01 to 99 to expand the 600 locations available, if not used for their reserved purpose.

Since the arbitrary assignment of a data address by a 9100 to 9699 code is the location address of data, the location addresses of data can be recalled. Data can be any constant, interim storage or dummy instruction. It is necessary to provide two recognition punches to differentiate between programmed instructions and data. One such punch indicates that the data address positions are to remain unchanged and the other indicates that the instruction address portion is to remain unchanged. For example - a constant or the segment of a dummy instruction which is not subject to change during the program. By use of 9000 series codes and the absence of these punches previously determined address portions of dummy instructions can be recalled.

Having accomplished the coding, a deck of cards is prepared from the code sheets. The machine is loaded to carry out the required instructions to optimum program. The code deck is run with constants at the end and a load deck for the problem is produced. It is possible to feed the code deck in any desired block sequence as long as cross reference addresses are suitably adjusted.

It is believed that this method is adaptable to the majority of situations that will arise. It has the element of simplicity and is as easy to use as sequential coding and in addition to saving time, eliminates many of the hazards inherent in manual optimum coding.

RULES for USING PSEUDO CODE

1. A normal address is required as the drum location of the first instruction.

2. Normal addresses may still be used where desired.

3. All arbitrary assigned data is referred to as 9100 to 9699.

4. 9000 may be used as the instruction address, or data address if branch operation, whenever the next instruction referred to follows immediately in which case the drum location of the following instruction is 9000. The foregoing is, provided there is no other cross reference to the latter instruction, 9100 to 9699 being required in such case.

5. 800X are used normally, the only exception being when the equivalent is required on a subsequent instruction, in which case 801X to 899X are used.

AUTOMATIC OPTIMUM PROGRAM

OPERATING PROCEDURE

I. A program is written in pseudo coding. Punching is done in the following manner (no "R" punches necessary): -

Program Code	= Col.	1 (also "X" if location is 8000 type)
69 0004 0003	=	11-20
24	=	21-22
8000	=	27-30
Segment codes	=	41-42
Punch page number	=	43-45
Pseudo drum location	=	47-50
Pseudo inst. or Data	=	51-60

II. A pre-punched card with program type code in col. 1, block number in cols. 2-4 and an "R" punch in col. 44 is manually placed in front of each program block.

Constants and interim storage constitute the last two blocks.

III. Four instruction cards can be fed into the 650, and the words 1000 to 1999 which are to contain the addresses of available locations (two to a word) and 0100 to 699 for reference location storage - cleared to zeros.

IV. There exists a deck of 2000 cards each one of which represents a location from 0000 to 1999 to be made available for locations of Optimum programming.

From this deck will be manually extracted cards for locations that are to be used for:

1. Read words
2. Punch words
3. Tables
4. Distributions

5. Constants with normal codes assigned.

6. Interim storage with normal codes assigned

7. Regular instructions with normal codes assigned

8. Where possible the 1950 band (or a band) for the storage of a trace routine

The remainder represents available location words and are loaded to fill in the words (or parts of words) 1000 to 1999 to be available for Optimum Programming.

V. A single load card contains the instruction necessary to cause the 650 to punch out cards for locations still available after new optimum coded load deck has been produced.

VI. Cards are fed into the 650 in this order:

1. Optimum program load cards.

2. Zero restore cards from step III.

3. Available location cards from step IV.

4. Program detail cards from steps I and II.

5. Remaining available location card from step V.

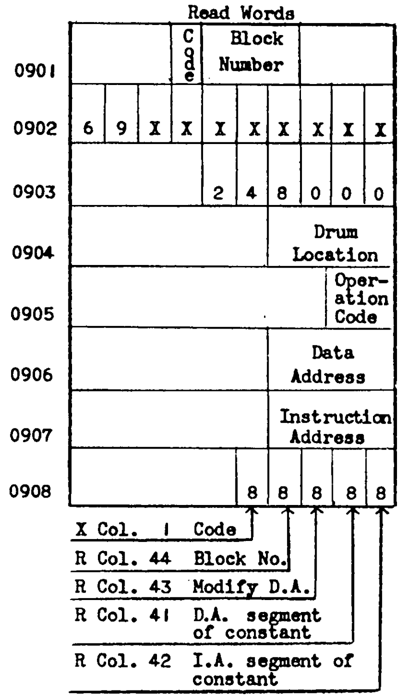
VII. Optimum coded load deck produced:

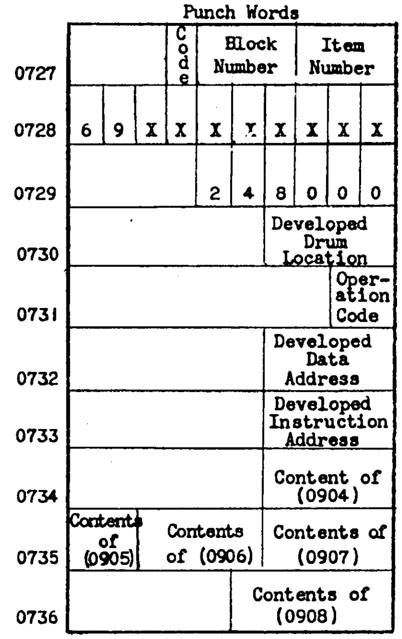
Program Code	Cols. 1	(X also for 8000 type locations)
Block Number	2- 4	
Item	5- 7	
69 0004 0003	11-20	(R in 20)
24	21-22	
Optimum drum location	23-26	
8000	27-30	(R in 30)
Optimum instructions or data	31-40	(R in 40)
Segment codes (if used)	41-42	
Pseudo drum location	47-50	

Pseudo instruction or data   Cols. 51-60

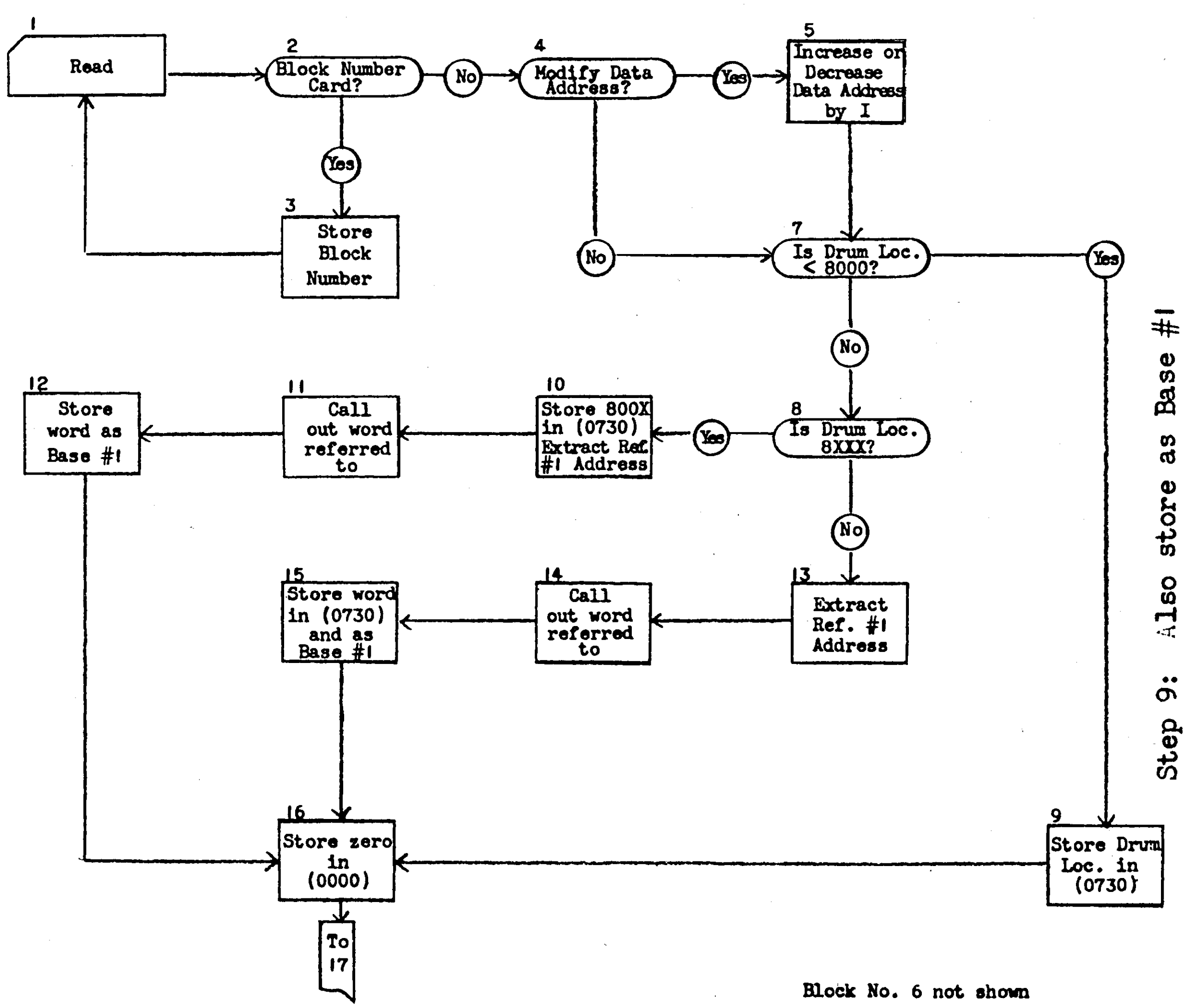
(Note that the pseudo codes are preserved to make corrections or changes, it is only necessary to change the pseudo coding and run the deck again for a new deck of optimum coded load cards.)

VIII. Cards with available locations after optimum coding are produced for reference and use.



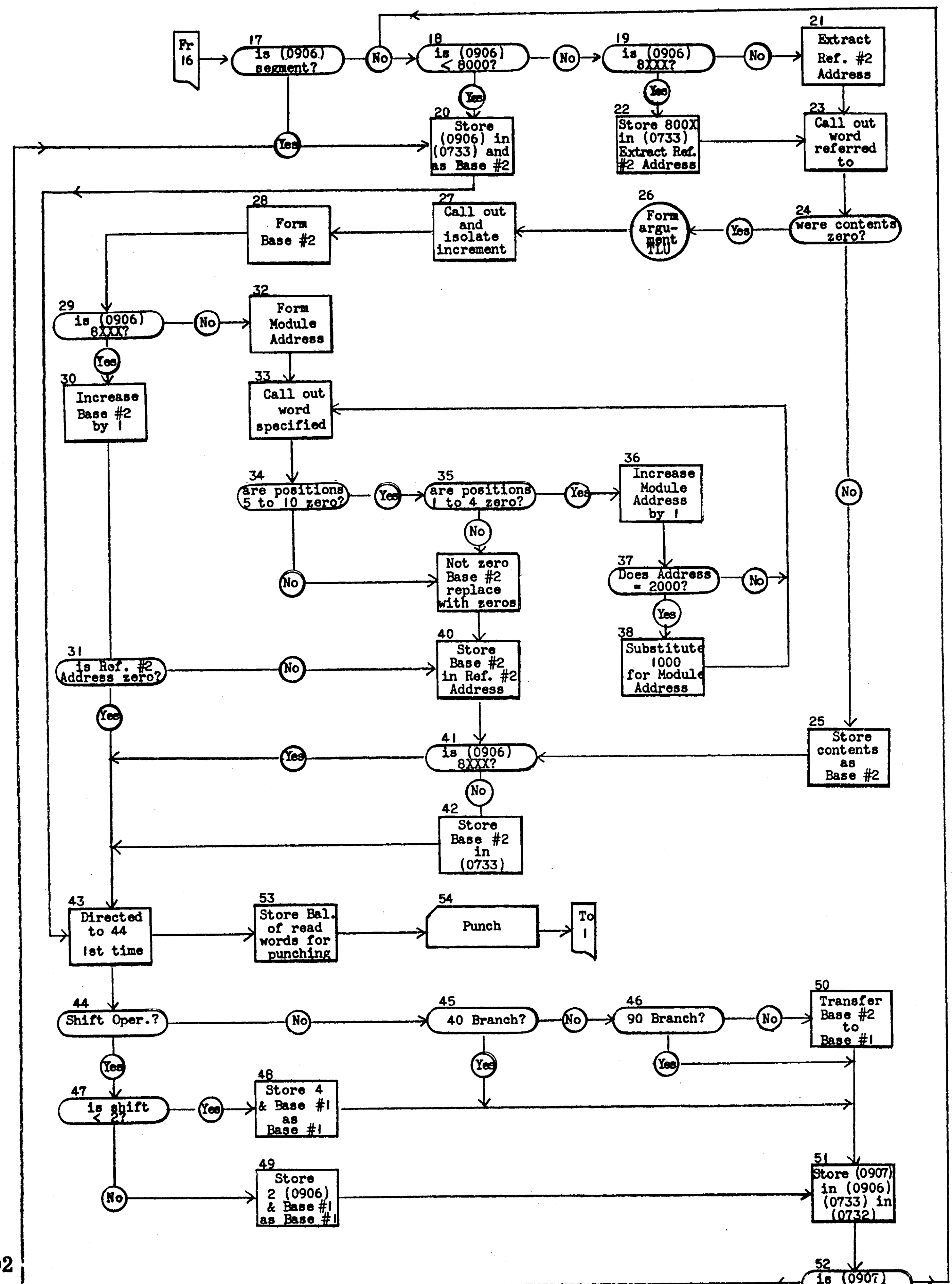


DEVELOPMENT of DRUM LOCATION

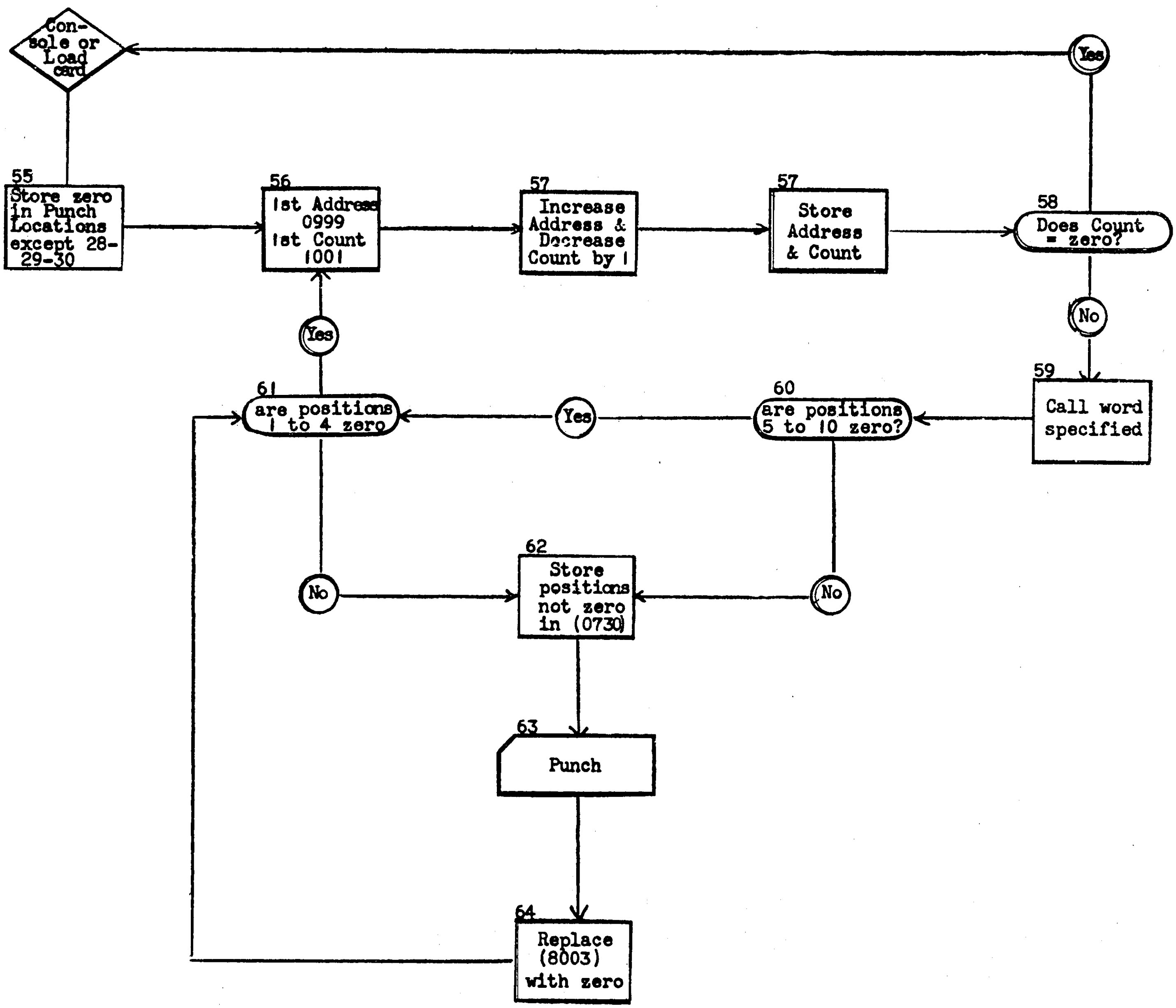


Block No. 6 not shown

DEVELOPMENT of DATA ADDRESS and INSTRUCTION ADDRESS



OPTIMUM PROGRAMING for I.B.M. 650 ROUTINE TO PUNCH UNUSED LOCATIONS

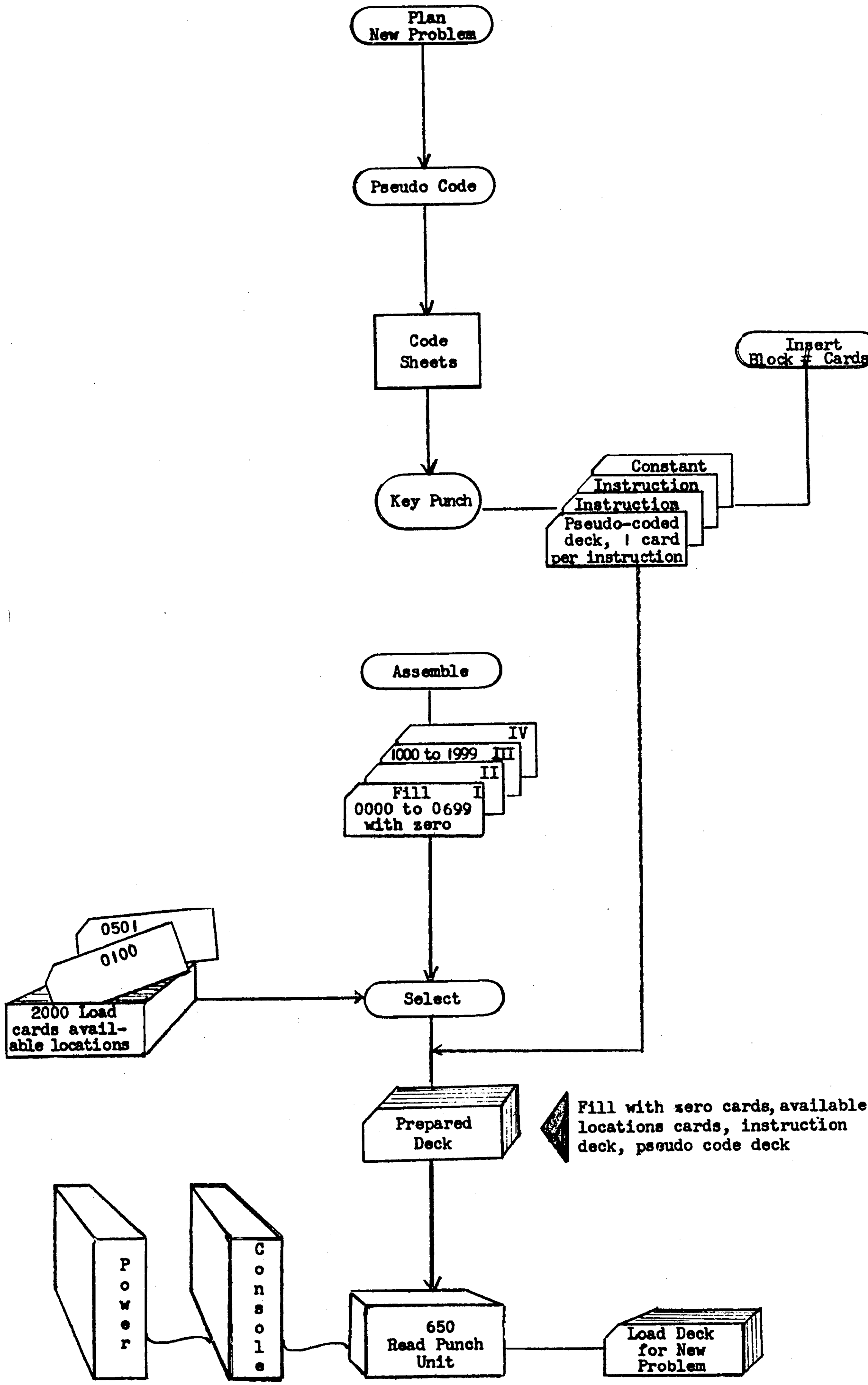


Sample Table

	Argument			DA Function			IA Function			
0955	0	2	0	0	0	5	0	0	0	3
	Odd Even	Oper. Code								
0973	1	1	1	0	0	3	0	0	0	4

Sample Array (Module 47)

			Drum Loc.				Drum Loc.			
1941	0	0	0	1	4	7	0	1	9	7
1942	0	0	0	2	4	7	0	2	9	7


Plan
New Problem
Pseudo Code
Code
Sheets
Insert
Block # Cards
Key Punch
Constant
Instruction
Instruction
Pseudo-coded
deck, 1 card
per instruction
Assemble
IV
1000 to 1999 III
II
Fill I
0000 to 0699
with zero
0501
0100
2000 Load
cards avail-
able locations
Select
Prepared
Deck
Fill with zero cards, available
locations cards, instruction
deck, pseudo code deck
Power
Console
650
Read Punch
Unit
Load Deck
for New
Problem

A NOTE ON OPTIMUM PROGRAMMING AND THE IBM TYPE 650 OPERATION CODE USAGE

Dura W. Sweeney
International Business Machines Corporation

As a part of the continuing analysis of the IBM Type 650 questions arose as to the relative usage of the various operation codes and the effect of optimum programming in achieving significant increases in computing speed.

An answer to the first question was gained by tracing several programs and counting the various operation codes executed by the computor in processing these programs. Since a limited amount of machine time was available only ten programs were traced. These were chosen so that each was written by a different coder to eliminate any bias in the habits of the coder as to his preference for using certain operation codes. The programs were also chosen to cover a wide range of applications in the commercial, engineering, and scientific fields to eliminate any bias in the needs of a program for certain operation codes in a particular application. Approximately three million executed operations were traced. The following table shows the results of this investigation:

Operation	% Used
Multiply	4.7
Divide	1.9
Shift	11.5
Store	22.8
Branch	8.0
Add	49.7
Table Look-Up	0.8
Read	0.3
Punch	0.3

These percentages do not include the instruction executions necessary to load the program. The only input-output traced was the reading of data required for the calculation and the punched results.

Now using these figures and the times required to execute these instructions for both serial coding and optimum coding, an estimate for the average operation time can be made. Assume an average six digit multiplier and quotient and an average shift of four positions.

For serial coding there will be an average of 24.5 word times for access to either the data or the next instruction. Both of these access times are taken only in the case of store and add operations, and since add operations have some parallel compute time even this is reduced three word times. The following table gives the average number of word times to execute the operations.

Operation	Percent	Optimum	Serial
Multiply	4.7	77.5	77.5 + 24.5
Divide	1.9	117.5	117.5 + 24.5
Shift	11.5	10.5	10.5 + 24.5
Store	22.8	7.5	7.5 + 49.0
Branch	8.0	4.0	4.0 + 24.5
Add	49.7	7.5	7.5 + 46.0
	98.6		

By crossmultiplying columns one and two and one and three and dividing each by 98.6 the following times result.

Optimum Coding: 13.0 word times (1.25 ms) per operation.

Serial Coding: 53.8 word times (5.17 ms) per operation.

This gives a ratio of $\frac{5.17}{1.25}$ or 4.1 times as fast for optimum coding over serial coding.

Considering that data and instructions cannot always be located in the best optimum locations, the next question that arose was what had been or could be achieved. The matrix inversion program written by R.W. DeSio was traced during the inversion of a 10x10 matrix. This program uses the interpretive floating point routines described in Technical Newsletter #8 for floating point operations and serial coding for the manipulation of the addresses of the matrix elements. The program required the execution of 154,000 instructions in 278 seconds. This gives an average operation time of 1.81 ms.

Next a program was written to do matrix inversion by the same method but the address arithmetic was coded optimumly and the floating point operations were included directly in the routine so that no interpretation time was required. This program was traced during the inversion of the same 10x10 matrix. Here, 49,000 instructions were executed in 71 seconds giving an average operation time of 1.45 ms. This program required 3300 accesses to get or store the matrix elements in a 110 position serial array in drum storage.

It appears obvious then that optimum coding can give significant increases in speed. The first matrix inversion program gives a ratio of $\frac{5.17}{1.81}$ or 2.9 times faster than serial coding. The second program gives a ratio of $\frac{5.17}{1.45}$ or 3.6 times faster than serial coding. From the figures for optimum coding versus serial coding, the ratio of 4.1 compared to the ratio of 3.6, for the matrix inversion program, indicates that in spite of the large number of accesses to serially stored data a speed increase very close to the maximum may be achieved.

AUTOMATIC FLOATING DECIMAL ARITHMETIC IN THE IBM TYPE 650

George R. Trimble, Jr. and Dura W. Sweeney
International Business Machines Corporation

The present operations of the IBM Type 650 have proven to be satisfactory for the great majority of problems which it has solved. There are, however, many problems involving lengthy, complex calculations which require extensive analysis to determine the size and range of intermediate and final quantities. This analysis and the subsequent scaling of these quantities frequently requires a larger percentage of the total time required to solve the problem than the actual calculations.

Floating decimal arithmetic circumvents this difficulty by tagging each number with a 2 digit characteristic. This characteristic specifies where the decimal point should be. Use of this technique virtually eliminates the need for scaling numbers mentioned above.

Floating decimal numbers in the 650 look exactly like fixed point numbers. The only difference between them is the way in which the arithmetic unit interprets them when a floating decimal operation is called for. Seven new instructions have been added to operate upon floating decimal numbers. They are add, subtract, add absolute, subtract absolute, multiply, divide and non-normalize add. Whenever one of these operations is called for the numbers operated upon are interpreted as follows:

$$(M, C) = \underbrace{.xxxxxxxx}_{M}\ \underbrace{xx}_{C}\ \underbrace{+}_{\text{Sign of } M}$$

The mantissa, M, is eight decimal digits in length. The decimal point of the mantissa lies to the left of the eighth digit. The sign of the number is always associated with the mantissa. Thus the range of the mantissa is

$$0.1 \leq M < 1.0$$

The exponent, e, is a two digit integer in the range

$$-50 \leq e \leq 49$$

Since the sign is associated with the mantissa it cannot be used to indicate the exponent sign. By adding 50 to the exponent, a positive number, C, in the range

$$0 \leq C = e + 50 \leq 99$$

is obtained. It is the two digit characteristic C that is carried as a tag to specify where the decimal point of the number really is.

To summarize then, the fixed point number, N, being represented by the floating point number (M,C) is determined by

$$N = M \times 10^{C-50}.$$

For example: 1.0 would be represented as 1000000051.

Since there is no difference between fixed and floating decimal numbers, fixed point operations can be performed upon floating decimal numbers if desired. For example, it is possible to test the floating point number to determine whether it is zero or non-zero; positive or negative. It is simply up to the programmer to determine what he wishes to do and to write the proper sequence of instructions to perform that operation. Similarly the characteristic can be separated from the mantissa by shifting and examined. It can be modified by programming, or whatever else is desired can be done. This facility of operating upon numbers with either type of arithmetic provides great flexibility.

The following descriptions tell how each of the seven new instructions function. Any operation which results in a zero mantissa will force a zero exponent.

32, FA Floating Add.

The floating decimal number specified by the data address is added to the floating decimal number in the upper accumulator. The rounded result will be retained in the upper accumulator. The lower accumulator is ignored for this operation and will contain zeros after its completion.

33, FS Floating Subtract.

The floating decimal number specified by the data address is subtracted from the floating decimal number in the upper accumulator. The rounded result will be retained in the upper accumulator. The lower accumulator is ignored for this operation and will contain zero after its completion.

34, FD, Floating Divide.

The floating decimal number in the upper accumulator is divided by the floating decimal number specified by the data address. The rounded quotient will be retained in the upper accumulator. The lower accumulator is ignored for this operation and will contain zero after its completion.

37, FAAB, Floating Add Absolute.

The absolute value of the floating decimal number specified by the data address is added to the floating decimal number in the upper accumulator. The rounded result will be retained in the upper accumulator. The lower accumulator is ignored for this operation and will contain zero after its completion.

38, FSAB, Floating Subtract Absolute.

The absolute value of the floating decimal number specified by the data address is subtracted from the floating decimal number in the upper accumulator. The rounded result will be retained in the upper accumulator. The lower accumulator is ignored for this operation and will contain zero after its completion.

39, FM Floating Multiply.

The floating decimal number in the upper accumulator is multiplied by the floating decimal number specified by the data address. The rounded result is retained in the upper accumulator. The lower accumulator is ignored for this operation and will contain zero after its completion.

02, FASN, Floating Add, Suppress Normalization

This code operates exactly the same as 32 (FA) except that the normalization, which occurs after adding the shifted numbers, is suppressed. This makes it possible to attach the same exponent to a group of numbers for fixed point output.

The times required to execute the above operations are essentially the same as for corresponding fixed point operations. Since multiply uses only an 8 digit multiplier it will be faster than fixed point multiply. The add type operations will vary in length depending upon the number of shifts required to line up the decimal points or to normalize the sum. The minimum time is approximately 1.0 ms. and the maximum 2.4 ms, a good average is probably 1.7 ms. Thus, floating point add is about half as fast as fixed point add when optimum programmed. Random programming, of course, requires the same time for both types of addition, namely, 5.2 ms.

COMPLEX ARITHMETIC ROUTINES FOR THE IBM 650 MAGNETIC DRUM DATA PROCESSING MACHINE

Tsai Hwa Lee
The Detroit Edison Company

Introduction

Many machine computation problems require operations in complex arithmetic rather than in real arithmetic. For example, complex arithmetic is used in the steady-state and transient analysis of electrical networks, and especially in the studies of the A-C power system. This paper presents some programming aids to use the 650 as if it were a complex arithmetic machine instead of one processing ten digit factors. Part I presents The Complex Arithmetic Interpretive Routine which provides twelve instructions. These include add, subtract, multiply, divide, shift left, shift round, store complex accumulator, transfer of complex number from memory to memory, sum of a block of complex numbers, square of absolute value, vector-vector multiplication, and unconditional transfer of control. This routine is optimumly coded.

Part II presents a complex arithmetic matrix inversion program which makes use of the interpretive system. It is possible to obtain the inverse of a matrix up to the order of 27 x 27.

Part I - The Complex Arithmetic Interpretive Routine

The complex arithmetic interpretive routine was designed so that the programmer can use the 650 as if it were a machine that could recognize and execute a list of twelve complex arithmetic instructions besides the forty-four normal 650 instructions. This routine not only makes the 650 a more versatile machine, but also facilitates coding whenever it is necessary to perform complex arithmetic operations.

Complex instructions are tagged with a minus sign. A complex instruction may be a one, two, or three-address operation. The operation code occupies 2 digits, while the location of the address part occupies 4 digits. The address specified refers to the real part of the factor to be operated upon. The interpretive routine automatically selects the imaginary part from the next memory location. Thus the real and imaginary parts of a complex number will always be stored in successive memory locations.

The interpretive routine is entered by transferring control to (0054). The instruction to be interpreted will be obtained according to the instruction address counter (0157). If the instruction obtained is plus, it will be executed as a normal 650 instruction, and subsequent instructions will not be interpreted until the interpretive routine is entered again. This is the normal manner of leaving the interpretive routine. If the instruction obtained is minus, the interpretive routine will analyze it, obtain the real and imaginary factors specified, and transfer control to the proper sub-routine. After the sub-routine is executed, a new instruction will be obtained from the location next to where the last instruction was obtained. Once the interpretive routine is entered, the instruction address counter is incremented automatically. Thus complex instructions are stored sequentially in the memory according to the order of their execution.

For example, consider the following sequence:

Location	Contents
n	complex instruction (-)
n+1	complex instruction (-)
n+2	complex instruction (-)
n+3	normal 650 instruction (+)

Instructions n, n+1, n+2 will be interpreted in sequence. n+3 is also interpreted but it is executed as a normal 650 instruction. However, by this time, the instruction address counter has been incremented, so that if the interpretive routine is entered again, the first instruction to be interpreted will be obtained from location n+4.

The number form is as follows:

$$.xxxxxxxxxx + j.xxxxxxxxxx = A_1 + jA_2$$

All the operations are designed to handle numbers with the decimal point set at the extreme left. It is necessary to scale the numbers so that the results will be less than one. If any answer should exceed one, the machine will stop. Also the condition $|A| < |B|$ must be satisfied for the divide operation A/B. The two shift instructions are for scaling where the problem decimal point is different than the decimal point of the interpretive routine (extreme left). Generally, multiplication is followed by a shift left instruction, and division is preceded by a shift round instruction.

LIST OF INSTRUCTIONS FOR THE COMPLEX ARITHMETIC INTERPRETIVE ROUTINE

Code		Operations		Estimated Time (msec.)
00 A 0000		(C) → (A)	Store complex accumulator	28.3
00 A 0001		BR (A)	Unconditional transfer to (A)	26.4
01 A B		(A) + (B) → (C)	Add	45.6
02 A B		(A) - (B) → (C)	Subtract	45.6
03 A B		(A) x (B) → (C)	Multiply	96.5
04 A B		(A) / (B) → (C)	Divide	175.4
05 A B		(A) → (B)	Memory to memory transfer	60.2
06 A M	$\sum_{i=1}^{M}$	$(A_i) \rightarrow (C)$	Block summation	36.2 + 43.2 M
07 A B 00 M 0	$\sum_{i=1}^{M}$	$(A_i) x (B_i) \rightarrow (C)$	Vector-vector multiplication	50.6 + 115 M
08 A B		$\lvert(A)\rvert^2 \rightarrow (B)$	Square of absolute value	93.8
09 A B		Shift round (A) B positions, result in (C)		79.4
10 A B		Shift left (A) B positions, result in (C)		84.2

Notes:
1. (C) is complex arithmetic accumulator.
2. Instruction address counter (0157) = 11 n 0122
3. Transfer to (0054) to enter interpretive routine.
4. Complex instruction is tagged with minus sign.
5. Storage required for routine: (0000) to (0283)

Part II - Complex Arithmetic Matrix Inversion Program

This program uses the standard elimination method to obtain the inverse of complex matrix. Basically, the original matrix is first reduced to a triangular form and then to the unit matrix by successive operations on rows. The same operations are applied to the unit matrix simultaneously. When the original matrix is reduced to the unit matrix, the unit matrix will be reduced to the inverse of the original matrix. By augmenting the original matrix with a -I matrix below it, no back substitution is necessary and the inverse is obtained after n reductions. The effect of the -I matrix is to transfer the pivot row after each reduction to the n+1 row.

Given the original matrix:

$$A = \begin{Vmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ \vdots & & & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{Vmatrix}$$

Augment the matrix as follows:

$$\begin{Vmatrix} A & I \\ -I & 0 \end{Vmatrix}$$

For example, the original matrix:

$$\begin{Vmatrix} A_{11} & A_{12} & \cdots & A_{1n} & 1 \\ A_{21} & A_{22} & \cdots & A_{2n} & 0 \\ \vdots & & & & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} & 0 \\ -1 & 0 & \cdots & 0 & 0 \end{Vmatrix}$$

becomes the following matrix after the first reduction:

$$\begin{Vmatrix} 1 & A'_{12} & \cdots & A'_{1n} & ; & A'_{1,n+1} \\ 0 & A'_{22} & \cdots & A'_{2n} & ; & A'_{2,n+1} \\ \vdots & \vdots & & & & \vdots \\ 0 & A'_{n+1,2} & \cdots & A'_{n+1,n} & ; & A'_{n+1,n+1} \end{Vmatrix}$$

Two types of operations are performed in the reduction:

For the pivot row: $A'_{1j} = A_{1j}/A_{11}$

For all other rows: $A'_{ij} = A_{ij} - A_{i1} A'_{1j}$

Note that the pivot row (except the pivot element) is translated to the n+1 row. The next reduction will work on the matrix enclosed by the rectangle with A'_{22} as the pivot element. After n reductions, the inverse of the original matrix will be obtained. This is in effect a process of sliding, where the matrix being worked on slides down along the main diagonal one step each reduction.

In the 650 program, the elements of the matrix are stored by row, e.g., A_{11}, A_{12} ... A_{1n}, A_{21}, A_{22} etc. Each element of the matrix occupies two memory locations, the imaginary immediately following the real. Thus $2n^2$ locations are required for storing the matrix. In addition, 2n locations are needed for working storage in handling the n+1 row. The I and -I matrices are not stored, their effect is programmed.

Each reduction performs the following operations in sequence:

1. Operate on first row and store in n+1 row.
2. Operate on second row and store in first row.

$$A'_{22} \longrightarrow A_{11}$$
$$A'_{23} \longrightarrow A_{12} \quad \text{etc.}$$

3. Operate on third row and store in second row in a similar manner. Operate on the fourth and remaining rows.
4. Transfer n+1 to n row.

This procedure stores the reduced matrix in the same locations as the original matrix. For example the new pivot row for the next reduction is stored in the locations used by the previous pivot row. The same program can be used in the second and subsequent reductions.

Furthermore, the original matrix can be augmented with b column vectors which are the constant terms of the linear equations.

$$\left\| A \quad b_1 \quad b_2 \quad \cdots \quad b_m \right\|$$

The solution of the linear equations after n reductions will result as well as the inverse.

$$\left\| X_1 \quad X_2 \quad \cdots \quad X_m \quad A^{-1} \right\|$$

The storage assignment is as follows:

(0455): A_{11}, real component of pivot element.
(0441): (n) 0000 (b+n) n= order of matrix
b= number of column vectors.
b= 0 if there are no vectors.
(0300): Beginning of program; first instruction.
(0410): End of program; last instruction. xx xxxx exit address.

The decimal point of the matrix elements is set as follows:
xx.xxxxxxxx

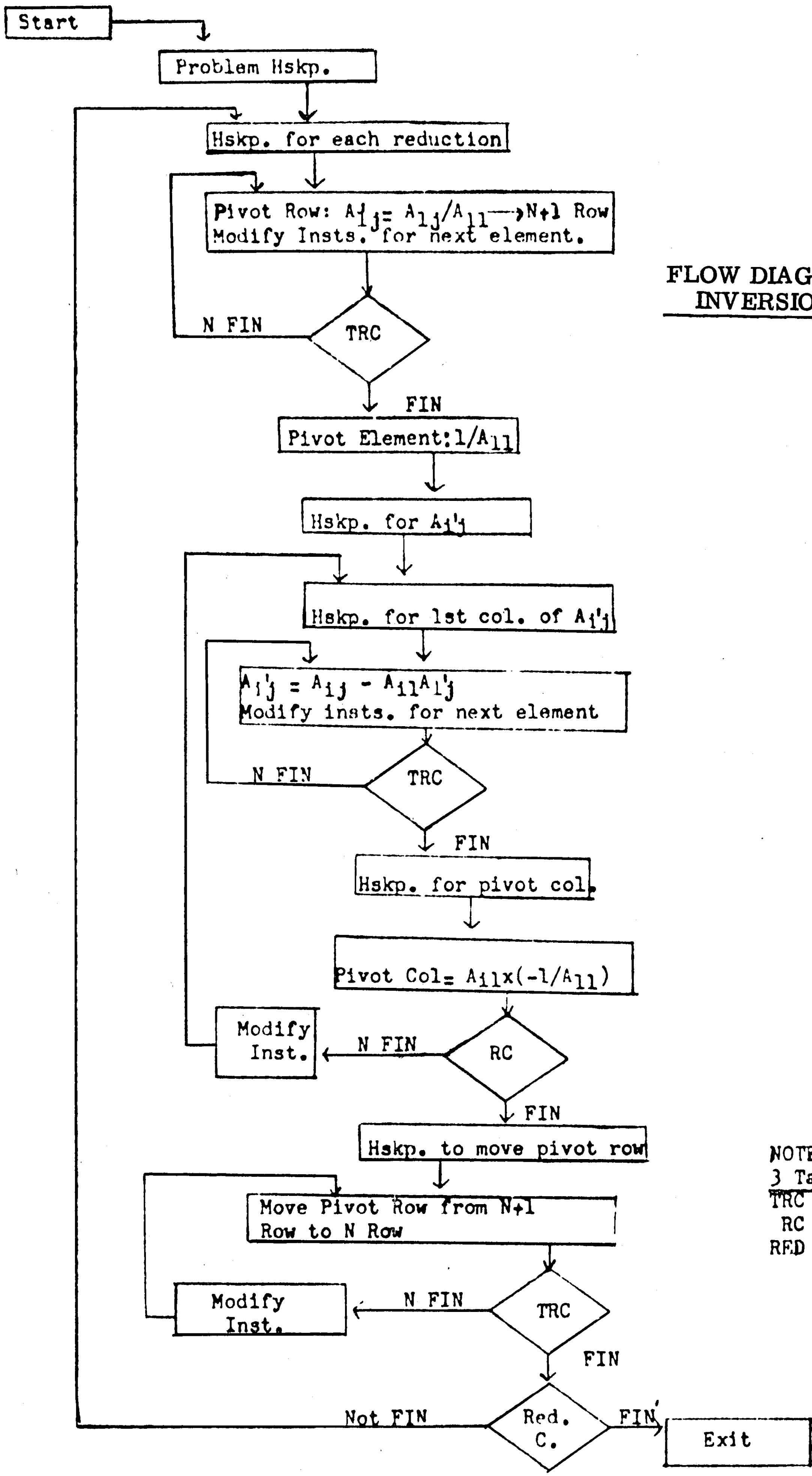
The time required for inverting a 10 x 10 matrix is approximately 5 minutes, and 92 minutes is required for the maximum size of 27 x 27.

Acknowledgments

1. To G. R. Trimble, Jr. of IBM, whose "A Method for Performing Complex Arithmetic on the IBM Type 650", published in IBM Technical Newsletter No. 8, forms the basis of The Complex Arithmetic Interpretive Routine.

2. To R. W. DeSio of IBM. The logic of the Complex Matrix Inversion Program is similar to the "Floating Decimal Point Matrix Inversion Program" written by Mr. DeSio.

FLOW DIAGRAM FOR MATRIX INVERSION BY SLIDING



NOTE:
3 Tallys Used
TRC - Traverse Row Count
RC - Row Count
RED C - Reduction Count

MATRIX MULTIPLICATION WITH THE IBM 650

R. H. Morris and C. H. Remilen
Eastman Kodak Company

Multivariate statistical problems such as multiple regression analysis, least squares analysis, multivariate tests of hypotheses, etc, require as a basic computation the multiplication of a matrix by its own transpose.

In the classical least squares problem, for example, one is given a n x s matrix A ($n > s$) and a n x 1 vector y and is required to find a s x 1 vector x of unknowns which best satisfy the matrix equation:

$$Ax = y.$$

The best solution, in the least squares sense, is that which makes the sum of the squares of the residuals a minimum,

$$x \rightarrow (y-Ax)'(y-Ax) = \min.$$

As is well known, the solution is found by solving the normal equations

$$A'Ax = A'y$$

and the residual sum of squares R is found by direct substitution, or more easily,

$$R = y'y - x'A'y.$$

The quantities required as intermediates for the solution of the problem are, then,

$$A'A,\ A'y,\ \text{and}\ y'y.$$

These are conveniently determined by considering the partitioned matrix:

$$B = \left[\begin{array}{c|c} A & y \end{array}\right]$$

and the product C, of B by its transpose:

$$C = B'B = \left[\begin{array}{c|c} A'A & A'y \\ \hline y'A & y'y \end{array}\right]$$

The 650 routine to be described presupposes the internal storage of B (the B matrix is usually augmented by an extra column consisting of the row sums so as to provide a check on the computations), it forms B'B, omitting those elements which may be obtained by symmetry, repeats the calculation of any row which fails to check, and leaves the matrix C stored in the machine for further computations.

The routine can also be used for multiplications such as C= A B where A'A is not required, by loading B completely and loading A one row at a time.

The restrictions on the matrix B for the multiplication:

C= B'B are:

1. Any element c_{ij} of C must not exceed ten digits. Since all calculations are done in fixed decimal, any element b_{ij} of B must not exceed five digits. The number of digits allowable in b_{ij} will vary with number of products (n) to be summed over.

2. The order, n x s, of B must satisfy $s(n+1) \leq 1770$. The constants, and auxiliary storage for the load, unload and B'B program decks consume the first 230 cells. Element c_{ij} may be stored in location 0231, element b_{11} must be stored in 0231+ s or further. Only storage enough for one row of C is reserved. After the first row of C is calculated we no longer need the first column of B and because $n \geq s$ the second row of C may replace the first column of B, etc. Thus, the elements of C "chase" the data of B.

Matrix B will be entered with a crossfoot column of row totals which will be used to check each row as it is calculated. In the event that the check fails, the program will stop the machine. However, programs are included in the appendix which will adjust the routine so that it will recalculate the row. These programs were not tested when this routine was checked out.

If the column totals are needed to "mean correct" matrix C then a column of 1's should be included in B. Note that this "1" should also be included in the check column.

The general operating procedure for making this calculation is:

1. load the load-unload program
2. read a card that describes the next deck to be loaded. This card contains the following data:
   a. how many cards are to be loaded
   b. how many instructions per card
   c. into which location to start loading
   d. to which location to go after completion of loading
3. The above card will describe matrix B which will be loaded in successive cells in order by column.
4. read a card that describes the program B'B deck as in step 2.

5. load program deck
6. calculate B'B = C
7. read a card that describes next deck to load if C must be further operated upon or read a card which describes how to punch C. The punch routine is just the reverse of the load, so this card must contain similar information.

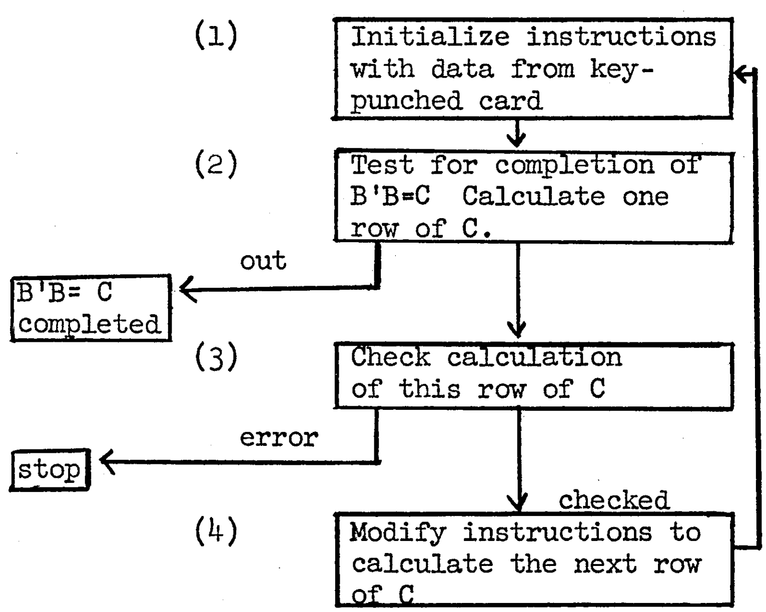
The program for B'B = C is stored in locations 0099 to 0226. The contents of each location is found in Appendix A. The 128 instructions are punched eight to a load card, the first fifteen cards being permanent and the sixteenth card keypunched to identify the order of the B matrix.

The keypunched card contains the following data:

1. n, the number of rows of B
2. s, the number of columns of B
3. s, the number of columns of B
4. location of b_{11}
5. location of c_{11}
6. stop code if check fails.

The program deck and original data, punched eight instructions or eight elements per card, load at the rate of 200 cards /min. assuming the whole drum is to be loaded with data, the maximum load time is $1\frac{1}{2}$ min. for data and instructions.

The B'B = C program is represented by the following block diagram:



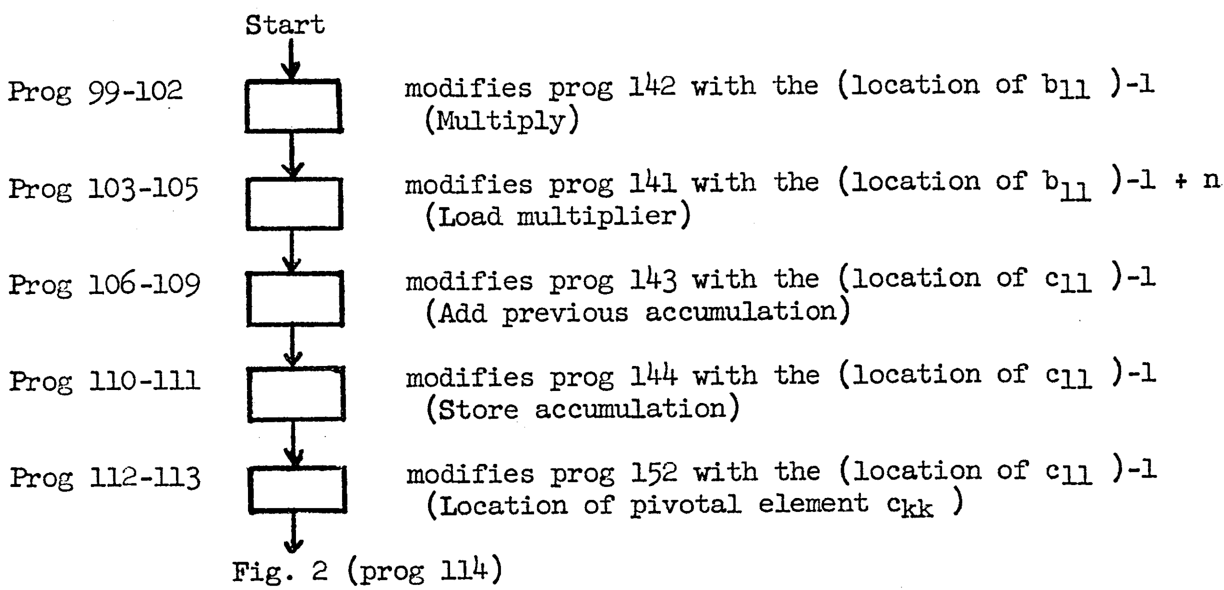
Appendix A shows a more detailed set of flow charts, Figures 1, 2, 3, 4. Each detailed block in Appendix A refers to locations of instructions which are required.

The program was checked out with a 42 x 33 matrix. The calculate time for B'B=C was about 25 minutes. This matrix required 23,561 multiplications.

APPENDIX A

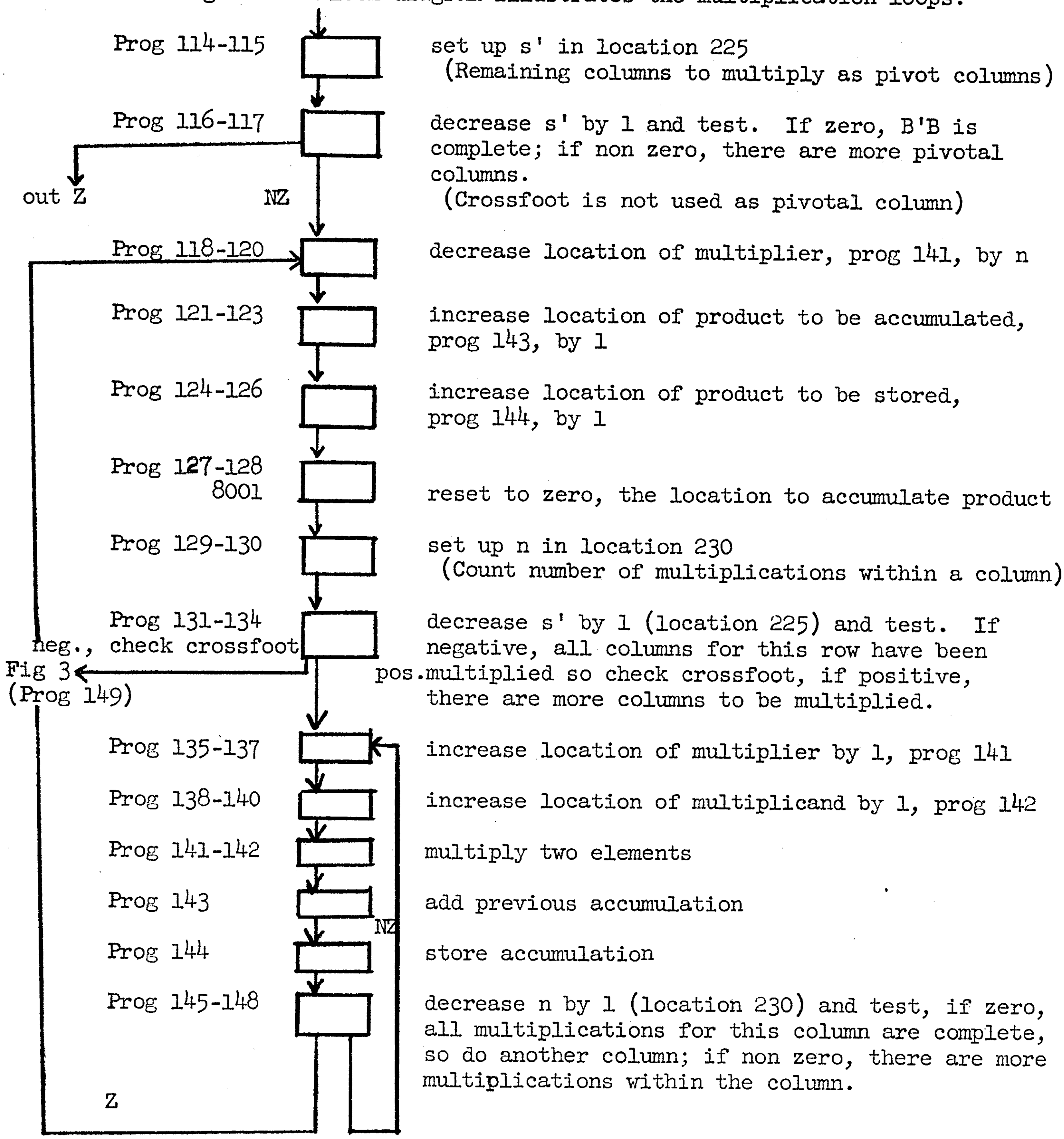
Figure 1 - Block diagram illustrates modifications to programs with the location of element b_{11} and the location of c_{11}.

These locations are backed off the proper distance so that all the elements' locations may be generated, locating the first column the same as the rest.



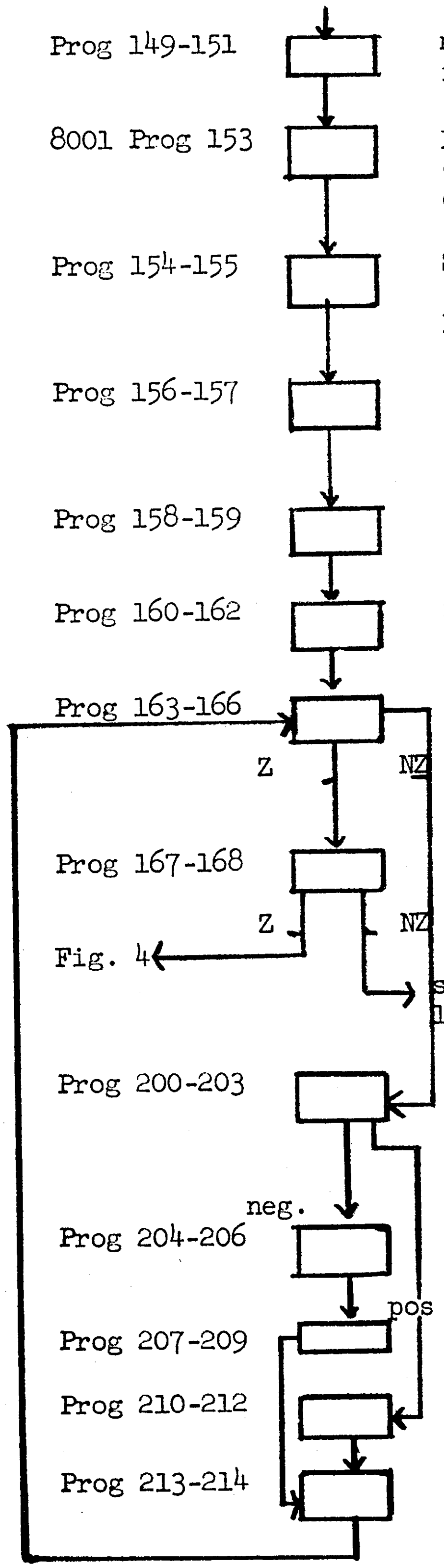
APPENDIX A

Figure 2 - Block diagram illustrates the multiplication loops.



APPENDIX A

Figure 3 - Block diagram illustrates the subtraction loop to check crossfoot of each row of C as it is completed.



Prog 149-151 — modify prog 152 with the location of the crossfoot for the row to be checked

8001 Prog 153 — place value of crossfoot in an auxiliary storage cell, location 229, to hold differences as each element of row is subtracted out.

Prog 154-155 — Set up s in location 226 to be called r_1 (Count total number of subtractions to be made) NOTE: this is now one too high

Prog 156-157 — set up s' in location 228 to be called r_3 (Used to locate location of column elements to be subtracted out)

Prog 158-159 — set up s'-1 in location 227 to be called r_2 (Count number of row elements to be subtracted out)

Prog 160-162 — modify prog 214 with location of the first element to be subtracted out <u>plus</u> <u>one</u>

Prog 163-166 — decrease r_1 by 1 (location 226) and test. If zero, all subtractions have been made so check crossfoot for zero balance; if non zero, there are more elements to subtract out for this row

Prog 167-168 — test location 229 for zero. If zero, start another row, if non zero, stop machine or recalculate row

stop or location 169-184 (recalculate row)

Prog 200-203 — decrease r_2 by 1 (location 227) and test. If positive, subtract a row element, if negative, calculate location of column element to be subtracted out.

Prog 204-206 — decrease location of column element to be subtracted, prog 214, by r_3 (location 228)

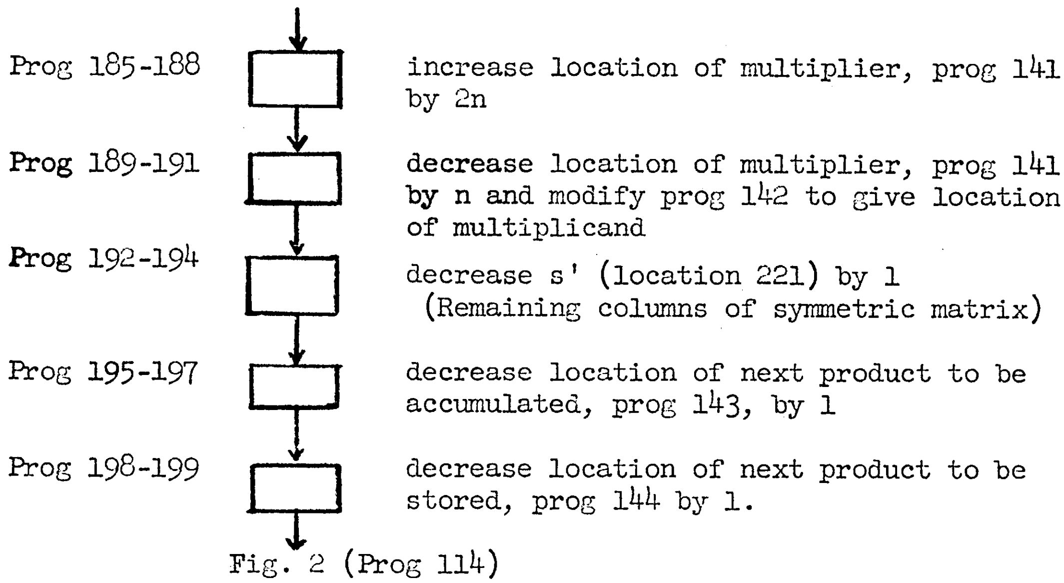
Prog 207-209 — increase r_3 by 1 (location 228)

Prog 210-212 — decrease location of row element to be subtracted out, prog 214, by 1

Prog 213-214 — subtract element out of previous difference, location 229

APPENDIX A

Figure 4 - Block diagram illustrates modifications of programs in order to calculate another row of C.



DETERMINING THE EIGENVALUES OF MATRICES

Mark Robinson
Bell Aircraft Corporation

I. Eigenvalues and Eigenvectors

In applied mathematics, a problem that frequently occurs is that of solving, for a given square matrix M, an equation of the type

(1) $MV = \lambda V,$

where V is a vector and λ is a scalar. It is obvious that if for some λ, V_0 satisfies equation (1), then k V_0 where k is any constant also satisfies the equation. In such a case λ would be called an eigenvalue, (or characteristic value, or root of the characteristic equation) of M and V_0 or k V_0 would be called an eigenvector corresponding to λ. For purposes of counting, V_0 and k V_0 are considered to be the same eigenvector.

The first and most important theorem on the subject is that every square matrix over an algebraically closed field has at least one eigenvalue and a corresponding eigenvector. The proof follows from the theory of linear systems. Equation (1) may be rewritten:

(2) $(M-I\lambda)\ V = 0$

For any given λ, this has a non-trivial solution if and only if the determinant: $|M-I\lambda| = 0$. Expanding the determinant by minors we get a polynomial in λ of degree n called the characteristic equation of M. As the matrix was assumed to be over an algebraically closed field this polynomial has n roots one or more of which may be multiple roots. To each distinct root, λ_i, of the characteristic equation corresponds at least one eigenvector, V_i.

If $\lambda_1, \lambda_2, \ldots, \lambda_m$ are distinct eigenvalues and $V_1, V_2, \ldots, V_m$ are corresponding eigenvectors then the V_i are linearly independent. If all n eigenvalues are distinct, then all of the eigenvectors are linearly independent and as the characteristic equation of an n x n matrix has n roots, the eigenvectors of a matrix whose roots are distinct span the space over which it operates. This is the case most commonly encountered in practice. Only two other cases may occur: matrices with one or more multiple roots whose eigenvectors span the space, and matrices whose eigenvectors do not span the space upon which they operate. Some methods of obtaining eigenvalues may fail or have excessive rounding error for one or both of these cases.

An operation that has considerable application is the change of basis in the vector field over which a given matrix operates. Let a given vector V be represented by the column X in one coordinate system and by Y in another, where the transformation is $Y = CX$. Then MX expressed in the Y coordinate system becomes $CMX = CMC^{-1}CX = CMC^{-1}Y$; that is M is transformed into CMC^{-1}. If V is an eigenvector of M, then CV is an eigenvector of CMC^{-1}. Further, if P(M) is a polynomial in M and if $P(M) \circ X = 0$ then $P(CMC^{-1}) \circ X = 0$; for if $P(M) = \Sigma a_k M^k$, then $P(CMC^{-1}) = \Sigma a_k CM^k C^{-1}$ and $P(CMC^{-1}) \circ CX = (\Sigma a_k CM^k C^{-1})\ CX = (\Sigma a_k CM^k) \circ X = C(\Sigma a_k M^k) \cdot X = C(0) = 0$. For matrices of a general type, this transformation is most useful in reducing them to a simpler form, as it does not change the eigenvalues. Note that as the transformation is reversible a vector which is not an eigenvector cannot be transformed into an eigenvector by change of basis, which, of course, is intuitively obvious.

The change of basis may be used to transform a matrix to a form with zeros below the main diagonal and also with every element $m_{ij} = 0$ if $m_{ii} \neq m_{jj}$. The process goes as follows: denote M by M_1, M_1 having an eigenvector V_1. Let V_1 be normalized so that the leading element is one. If the leading element is zero, rows and columns of M_1 may be interchanged. Let C_1^{-1} be the matrix obtained by replacing column 1 of the identity matrix by V_1. Then $C_1MC_1^{-1}$ has λ_1 in the position (1,1) and zeros elsewhere in the first column. Let $M_{i+1} = C_iM_iC_i^{-1}$ where for $i > 1$ the C_i^{-1} are chosen as below. Consider the $(n+1-i) \times (n+1-i)$ matrix in the lower right hand corner of M_i. It has an eigenvector in $(n+1-i)$ dimensional space. We may normalize it as before. Now it may or may not be possible to add elements $\alpha_1, \alpha_2 \ldots \alpha_{i-1}$ to V_i in such a way as to make it an eigenvector for all of M_i. We want $\lambda_{i-1}\alpha_{i-1} + \Sigma m_{i-1,j} \cdot v_j = \lambda_i\alpha_{i-1}$, and similarly for the other α_i. These equations have solutions except where $\lambda_i = \lambda_j$; $(j < i)$. Where no solutions exist, we may put $\alpha_j = 0$. Then V_i with the α_i adjoined is used to replace the i'th column of the identity matrix to give C_i^{-1}. Thus, the transformation $M_{i+1} = C_iM_iC_i^{-1}$ gives M_{i+1} with zeros in the i'th column except on the diagonal which has λ_i and above the diagonal in cases where $\lambda_i = \lambda_j$; $(j \neq i)$. M_{n+1} is a triangular matrix whose characteristic equation is seen by inspection to be: $\prod_{i=1}^{n} (\lambda - \lambda_i) = 0$. Let $P(M_{n+1}) = \prod_{i=1}^{n} (M_{n+1} - I\lambda_i)$. Then $P(M_{n+1}) \circ X = 0$ for any vector X. It is sufficient to show this for the "basis" vectors, $(1,0,\ldots,0)$, $(0,1,0\ldots0)$ etc. Let u_i be the basis vector which has 1 for its i'th component and zeros elsewhere. Let p be the number of λ's above λ_i on the main diagonal and equal to λ_i. Then $P(M_{n+1})$ contains the factor $(M_{n+1} - I\lambda_i)^{p+1}$. Now $(M_{n+1} - I\lambda_i)\ u_j$ is zero except possibly in those positions, where $\lambda_j = \lambda_i$ and $j < i$. This means for $p = 0$ the vector is annihilated by $(M_{n+1}-I\lambda_i)$, and hence by $P(M_{n+1})$. If we assume the annihilation proven for $p = k$, then if $p = k+1$ multiplying by $(M_{n+1}-I\lambda_i)$ reduces u_i to a sum of vectors which are annihilated by $(M_{n+1}-I\lambda_i)^k$. Thus M_{n+1} satisfies its characteristic equation. But M_{n+1} is of the form CMC^{-1} so $P(M)Cu_i = 0$ for all i, and M also satisfies the characteristic equation. (Cayley-Hamilton theorem).

II. Various Methods of Solving the Eigenvalue Problem

We have programmed and tested at Bell Aircraft Corporation a great many methods for finding the eigenvalues of matrices. The reason that we have used so many methods is, of course, that so few methods have proven completely satisfactory. Except for special methods for symmetric and Hermitian matrices, mentioned in Section J below, all the methods have been tested and used on our CPC or 650 or both.

A) Direct expansion by minors.

For small matrices, it is possible to expand $|M - I\lambda|$ by minors and get the characteristic equation of M directly. This is probably the best approach for 2x2 matrices and on the CPC for 3x3 matrices. It has been programmed on the 650 for 3x3 complex matrices and it works quite satisfactorily, but it uses so much storage for instructions, that we plan to run many of our 3x3's by Danielewsky's method discussed below. For matrices of size 4x4 and higher, the method is too laborious for wide use as the number of operations goes up about as $n \cdot n!$. However, we once attempted this method for 5x5's on the CPC. While this method is quite accurate when applied to the small size matrices for which it is suitable, it was not remarkably accurate when applied to 5x5's. In all cases we solved the characteristic equation by Newton's method.

B) Another Method Using the Determinant

It is well known that it is easier to evaluate the determinant of a large matrix of numbers (as opposed to a matrix of Polynomials) by elimination than by minors, as elimination takes about $n^3/3$ multiplications instead of about $n!$. This technique has been applied to determinants of the form $|M - I\lambda|$ at Bell Aircraft Corporation, where we used elimination down the opposite diagonal, to avoid dividing by polynomials in λ. When the elimination was one half complete, we shifted to direct evaluation of the coefficients of the characteristic equation from the reduced matrix. We used this method for complex 5x5's with moderate success on the CPC, but the rounding error was treacherous, and a lengthy checking process was found necessary to guarantee accuracy. We do not consider it suitable for the 650 as it appears that the program would take too many instructions.

C) Numerical Evaluation of Determinant

If one is given Λ_0 as an unverified eigenvalue of M_1 a simple way to check the value is to compute $|M - I\Lambda_0|$, usually by elimination. This has a number of variations. For example, we frequently improve the approximation of Λ_0 as a root of $|M - I\lambda| = P(\lambda) = 0$ by evaluating $P(\Lambda_0)$ and setting $\Lambda_1 = \Lambda_0 - (P(\lambda)/\frac{dP}{d\lambda})_{\lambda = \Lambda_0}$, that is by Newton's method. However, if we have only approximate values for the coefficients of $P(\lambda)$, then we may be able to evaluate $|M - I\Lambda_0|$ more accurately than $P(\Lambda_0)$. Thus, direct evaluation of the determinant can be used to refine approximate eigenvalues. Also we may replace $dP/d\lambda$ by:

$$(|M - I\Lambda_0| - |M - I\Lambda_1|) / (\Lambda_0 - \Lambda_1) \text{ if } |\Lambda_0 - \Lambda_1| \text{ is small.}$$

Both of these methods have been used to check the results of the calculations of the type described in section B. If $(\Lambda_1 - \Lambda_0) / \lambda_{max} < 10^{-6}$ (where λ_{max} is the largest computed eigenvalue) the root is accepted; otherwise the process is repeated until this criterion is satisfied. Most roots are accepted at once, but occasionally roots close together give considerable difficulty.

A variation of the above method is being programmed for the 650 to check the results of Danielewsky's method, and simultaneously obtain the eigenvectors. We have $(M - I\Lambda_0) V = 0$ where V may be written in the form $(x_1, x_2, \ldots, x_{n-1}, -1)$. If we perform our elimination working down the main diagonal we arrive at a matrix of the form:

$$\begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & x_2 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 1 & x_{n-1} \\ 0 & 0 & 0 & \epsilon \end{bmatrix}$$

where the x_i give the eigenvector and ϵ times the divisors used in the elimination is the value of the determinant, $|M - I\Lambda_0|$. The quotient of this by $\lambda_0 P'(\lambda_0)$ will be used as an indication of the accuracy of the root.

D) A Method Using the Cayley-Hamilton Theorem

The theorem: $P(M) = \sum_{k=0}^{n} a_k M^k = 0$ (where $P(\lambda)$ is the characteristic polynomial $|\lambda I - M|$) can be used to determine the a_k. Since the $\Sigma a_k M^k \cdot X = 0$ where X is any vector, repeated premultiplication of a vector by M yields the set of vectors $M^k X$. Denoting the i'th component of the k'th vector of the set by b_{ik} we have the set of equations: $\sum_{k=0}^{n} b_{ik} a_k = 0$; and a_o is known to be 1. If the matrix B is not too badly conditioned, the a_k may be obtained from this set.

This method is simple, easy to program, and occasionally gives good results. It is treacherous with respect to rounding error as the matrix B is seldom well conditioned. For example, if we have two equal eigenvalues then B is singular, if some of the eigenvalues are much larger than others, then the columns of B representing the higher powers of M will tend to lie in the same general direction; B will be nearly singular if by accident the vector X is chosen so that it has a nearly zero component in the direction of some eigenvector. As a result of the uncertainty about the accuracy of the results, we have been forced to abandon this method.

An extension of this method has been suggested, but not tried out, so far as we know. Instead of taking one vector X, we may start with two or more initial vectors and get a 2n x n or 3 n x n redundant set of equations which could be solved by a least squares process. This method may merit further investigation, as it is one of the few that appear suitable for fixed decimal type of operation.

E) Leverrier's Method

If we study the diagonalized or triangularized form of M, that is CMC^{-1}, we observe that $(CMC^{-1})^n = \begin{bmatrix} \lambda_1^n & & & \\ & \lambda_2^n & & \\ & & \ddots & \\ 0 & & & \lambda_n^n \end{bmatrix}$ where λ_i are the roots of the characteristic equation. But $(CMC^{-1})^n = CM^nC^{-1}$, that is the transformation of M^n. Now the trace of M^n is (-1) times the coefficient of λ^{n-1} in the characteristic equation of M^n and is invariant under the transformation: $M^n \rightarrow CM^nC^{-1}$. Thus the trace of $M^n = \Sigma \lambda_i^n$. Now the coefficients of a polynomial, a_n, are symmetric functions of its roots: If we denote the trace of M^n by S_n we get

$$a_1 = - S_1$$

$$a_2 = - \frac{a_1 S_1 + S_2}{2}$$

$$a_3 = - 1/3 \, (a_2 S_1 + a_1 S_2 + S_3) \text{ etc.}$$

This method suffers from two major defects. In the first place, the computation is lengthy, as the matrix must be raised to the nth. power, requiring about n^4 multiplications. Also, in many cases, the rounding error is very bad. In cases where all the eigenvalues are positive, the S_i are all positive but the a_i alternate in sign. This means that after the first, each a_i is the difference between positive quantities, and quite frequently, the small difference between large positive quantities. In aircraft flutter work matrices with real positive eigenvalues are infrequently encountered but one often meets complex matrices with the real parts of the eigenvalues positive and large compared with the imaginary part. This is another method we were forced to abandon.

F) Danielewsky's Method

The transformation $M_{i+1} = C_i M_i C_i^{-1}$ may be used to reduce a matrix in such a way that the coefficients of the characteristic equation are obtained. Take $M_1 = M$ and let C_i^{-1} be the matrix obtained by replacing the (i+1)st. column of the identity matrix by the vector obtained by dividing the ith. column of M_i by its (i+1)st. element. After this transformation M_2 contains only zeros in the first column except for row 2; the second column of M_3 is zero except in its third element; and finally M_n has zeros everywhere except for the last column and the diagonal below the main diagonal. This process can fail only if one or more of the "pivotal" elements is zero. We perform another set of simple transformations to reduce M_n to Frobenius standard form:

$$\bar{M}_n = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & 0 & \cdots & 0 & -a_2 \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ 0 & 0 & \cdots & 0 1 & 0 & -a_{n-2} \\ 0 & 0 & \cdots & 0 0 & 1 & -a_{n-1} \end{bmatrix}$$

where a_i is the coefficient of λ^i in the characteristic equation $P(\lambda) = \lambda^n + \sum_{i=0}^{n-1} a_i \lambda^i = 0$. To see this, consider $P(\lambda) / (\lambda - \lambda_o) = \lambda^{n-1} + \sum_{i=0}^{n-2} d_i \lambda^i$ (λ_o any root of $P(\lambda)$. As may be verified by multiplying $(\lambda - \lambda_o)(\lambda^{n-1} \Sigma d_i \lambda^i)$, $a_i = -\lambda_o d_i + d_{i-1}$; $a_o = -\lambda_o d_o$. Now $\bar{M}_n \cdot \{d_o \; . \; . \; . \; . \; . \; d_{n-2}, 1\}$ = $-a_o, d_o - a_1, \ldots d_{i-1} - a_i, \ldots, d_{n-2} - a_{n-1}$. But, from the previous statement, this is just equal to $\{\lambda_o d_i\}$. Hence, λ_o is an eigenvalue of $\bar{M}_n$, and, since the transformation of M into $\bar{M}_n$ did not affect the eigenvalues, it is an eigenvalue of M. Consequently, $P(\lambda)$ is the characteristic equation of M.

As was pointed out earlier, this method fails only when the pivotal element is zero. However, the method can be extended to cover even these cases. Suppose the pivotal of M_i is zero, but there is a non-zero element below it in the i'th. column, and k'th. row. Then we may interchange the (i+1) st. row with the k'th. row and also the (i+1)st. column with the k'th. column. This does not disturb any of the zeros in the columns numbered less than i, and the interchange of both rows and columns is another example of change of basis which does not affect the characteristic equation.

In order to handle the case where the whole column is zero below the main diagonal, we split the matrix, using an auxiliary theorem:

Let any square matrix M be partitioned as below so that A and D are square principal minor matrices containing between them all the diagonal elements, and let C = 0, then every eigenvalue of M is an eigenvalue of A or D.

$$M = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

Proof: Let m be the number of rows in A. Then every eigenvector of M falls into one of two classes; (1) all elements after number m equal to zero, and (2) all other cases. It is obvious that the eigenvectors of class (1) are eigenvectors of A with the trailing zero's omitted. If we take vectors of class (2) and consider only the last (n-m) elements, then they are eigenvectors of D. Also with all of these vectors, there is associated the same root in both M and A or D. The converse is also true. Obviously, an eigenvector of A is an eigenvector of M (by adding zeros). Also if an eigenvalue of D is at the same time an eigenvalue of A, it is therefore an eigenvalue of M. In the case of an eigenvalue of D which is not one for A, we may transform M so that A is in diagonal or triangular form as in part I without affecting matrices B, C or D. Now, as in Part I, any eigenvector of the lower, right-hand corner may be extended to an eigenvector of M (transformed) provided, as we assumed, that the eigenvalue associated does not equal any of the upper left-hand diagonal elements.

Using the above theorem, we see that when we carry out the reductions of Danielewsky's method until we encounter a column which has all zeros below, we may split the matrix and find the eigenvalues of part A and D separately. A is already reduced to standard form for the characteristic equation and D is of lower degree than M, even if the splitting occurs on the first column. Therefore, if we reduce D as we had been reducing M, the process will terminate, although we may have to split D several times. After splitting we have a factorization of the complete characteristic polynomial.

This extended process appears difficult to apply directly on the 650. Even if we neglect the problem of storing the extra instructions, we have the old problem of digital computers: "what is a zero ?". Usually, when the true answer is a zero, the computed answer will contain rounding error. In problems that are carefully scaled, it is frequently possible to set a reasonable upper bound for the rounding error, but in routine matrix work, scaling each matrix would be laborious, and the results might not mean much, as the operation of dividing a column by its pivotal element causes irregular and sometimes large increases in the size of the numbers used.

We have adopted Danielewsky's method as our principal means of attacking medium size matrices on the 650. We have run several hundred 5x5 complex matrices with good results using our provisional program, and we are now adding checks on the determinant, as outlined in Section C. We know that there are some matrices, (for example, matrices with two or more equal roots, whose eigenvectors span the space) that cannot be handled by this method, but we have been lucky enough not to be presented with any of these.

G) The Power Method

If the characteristic vectors of M span the space, then any vector V may be written: $V = \Sigma a_i V_i$ where the V_i are eigenvectors of M. Hence, $M^p V = \Sigma a_i \lambda^p_i V_i$. If one of the λ_i, say λ_1 has much larger absolute value than the others, the term $\lambda_1^p a_1 V_1$ will dominate the others for large p, assuming of course, that $a_1 \neq 0$. Thus, by repeatedly multiplying a matrix by an arbitrary vector, we get an approximate eigenvector, which we can make as accurate as available machine time will allow. It is customary to normalize each time by dividing through by some element, frequently the last. Then the approximate eigenvalue appears as the last component of the next product vector.

This method may be extended to find other eigenvectors in several ways. For example, if M is symmetric, its eigenvectors are or may be chosen orthogonal. Thus, after V_1 is determined, we may choose a V such that $V \cdot V_1 = 0$. Then, the component $a_1 = 0$, so that $\Sigma a_i \lambda^p V_i$ approaches the vector corresponding to the second largest λ. Because of rounding error, it is usually necessary to re-orthogonalize occasionally. This can be extended to find all the roots and corresponding vectors of a symmetric matrix. If M is not symmetric and we have found one of its eigenvectors, we may reduce it as in Part I and consider only the (n-1) x (n-1) lower right-hand corner. This process, when applied repeatedly, reduces M to triangular form.

The convergence of this process depends upon $\left|\lambda_1/\lambda_2\right|$ where λ_1 and λ_2 are the two largest roots of $P(\lambda)$. When this ratio approaches one, convergence becomes very poor. At one there is no convergence. To accelerate convergence quadratic or higher formulas may be used. Take X and Y any two non-parallel vectors. We assume that eigenvectors corresponding to eigenvalues other than λ_1 and λ_2 have been substantially eliminated from V. Then we solve the equation:

$$(M^2V)X + (MV)\cdot X\cdot p + V\cdot X\cdot q = 0$$
$$(M^2V)Y + (MV)\cdot Y\cdot p + V\cdot Y\cdot q = 0$$

for p and q. The roots of the equation $\lambda^2 + p\lambda + q = 0$ are approximate values for the two largest eigenvalues λ_1 and λ_2. We choose the smaller to eliminate. We take as our new V_0 the quantity $M^2V - \lambda_2 MV$. The cubic reduction is similar.

Except perhaps for ultra high speed machines, the power method is too slow to be practical except in special cases. Probably the most important of these special cases is when only one or two of the eigenvectors are needed. For example, some vibration problems are set up so that the reciprocals of the squares of the frequencies are the eigenvalues of a matrix. An engineer may be interested in only the lowest two frequencies. These frequencies are often well separated; for example, the vibration frequencies of a cantilever beam have about the ratio 1:6.2:17.4. We have programmed this method for both the

650 and the CPC and use it for these special applications.

H) An extension of the Power Method

Except as applied to symmetric matrices, the complete power method has the reputation of being inaccurate, at least for the smaller values obtained later in the process. The reason is that most matrices have at least one pair of eigenvalues nearly the same size. If $|\lambda_1/\lambda_2|$ is 1.1 may take sixty or seventy matrix vector multiplications to get three digit accuracy. Under these circumstances there is a strong temptation to stop the iteration as soon as a reasonably good value is obtained. This may be permissible for the first two or three roots, but when the matrix has been reduced several times using approximate eigenvectors, the roots of the lower right-hand corner are no longer close to the roots of the original matrix. At Bell Aircraft Corporation, we have developed and tested a method for getting around this difficulty. Essentially what we do is recognize that when we reduce with an approximate eigenvector, we obtain only approximate zeros. We save the whole matrix after each reduction, including the terms near zero.

The process thus has three parts: find a reasonable approximation to the eigenvector for the lower right-hand corner matrix, extend this to an eigenvector for M_i, and then reduce the whole matrix. When the process is complete we have an "almost diagonal" matrix, that is a matrix whose elements on the main diagonal are large compared with all others.

The approximate roots may now be improved by a simple iteration process. If we want to improve λp we take:

$$v_{i,o} = 0; \quad i \neq p; \quad v_{p,o} = 1; \quad \lambda_{p,o} = m_{pp}$$

(the second subscript is the number of the iteration).

$$v_{i,j+1} = \sum_{k \neq i} m_{ik} \cdot v_{k,j} / \lambda_{p,i} - \bar{m}_{ii}); \quad i \neq p$$

$$v_{p,j+1} = 1; \quad \lambda_{p,j+1} = \sum_{k=1}^{n} \bar{m}_{pk} \, v_{k,j+1}$$

The advantage of this last type of iteration is its very rapid convergence. In the case of two equal diagonal elements we simply set the corresponding v_i equal to zero. However, it is clear that this method fails for matrices whose eigenvectors do not span the space as these cannot be reduced to diagonal form. The failure occurs when we try to extend the eigenvector of the lower right-hand corner to one for M.

This method has been tested on the CPC. It is too laborious for general use, but for medium-sized matrices, it is the most accurate we have found, using a fixed number of digits.

J) Conjugate Gradient and Special Methods for Symmetric Matrices

In the past few years, there has been a great development of special methods for symmetric matrices; most of it by men who worked at one time or another for the National Bureau of Standards.

The process for finding the characteristic equation is an extension of the process of solving a linear system. Given AX = B, one takes an arbitrary x_o and sets

$$p_o = r_o = B - AX_o$$
$$a_i = (r_i \cdot p_i)/(p_i \cdot Ap_i)$$
$$X_{i+1} = X_i + a_i p_i$$
$$r_{i+1} = r_i - a_i \; Ap_i$$
$$b_i = (r_{i+1} \cdot Ap_i)/(p_i \cdot Ap_i)$$
$$p_{i+1} = r_{i+1} + b_i p_i$$

If the process can be carried out n steps, neglecting rounding error, X_n is a solution, that is $AX_n = B$, and $r_n = 0$. The a_i and b_i can be used to obtain the characteristic function as follows:

$$R_o = P_o = 1$$
$$R_{i+1} = R_i - \lambda a_i P_i$$
$$P_{i+1} = R_{i+1} + b_i P_i \qquad P_n = P(\lambda)$$

Although we have programmed this conjugate gradient method for the solution of simultaneous equations, we have not used it for eigenvalues because most of our matrices are not of the right kind. However, it appears very promising.

III. The Program Using Danielewsky's Method

We have in operation a program applying Danielewsky's method to the solution of eigenvalue problems for 3x3 to 5x5 complex matrices. The work is done in floating decimal using an interpretive routine prepared by Mr. Bruce Blasdell of Bell Aircraft Corporation. This system uses an eight digit mantissa and a two digit exponent. To simplify coding and reduce storage requirements, complex arithmetic subroutines were worked out for the operation, $a \cdot b \rightarrow k$, $1/a \rightarrow k$ and $a - k \rightarrow a$, which were the operations most used. The use of these subroutines simplifies coding the main problem, but at some expense in speed.

There are many formulations of Danielewsky's method. For example, instead of working with matrices that differ from the identity only by a column, we may use for C and C^{-1} matrices that differ from the identity only by a row. This is equivalent to working on the transpose of M, and the choice appears arbitrary. Again, equally arbitrarily, we might start with the last column, to set all but one of its elements equal to zero, instead of the first column. Because our research on this problem was done on the CPC, the arbitrary choices were made in such a way as to simplify CPC coding and card handling.

There is one respect in which our method differs from methods that I have seen published. Usually the matrix C_i^{-1} is obtained by replacing the (i+1)th. column in the identity matrix by the ith. column of M_i. We first divide this column by its (i+1)th. element. The only change that this makes in C_i is that for the (i+1)th. diagonal element, instead of one we get $(1/m_{i+1,i})$. An advantage of the usual method is that the final reduced matrix M_n has 1's on the sub-diagonal, so the final multiplication by these elements is not needed. We feel that our method is superior, however, as it is shorter, the coding appears to be simpler, and rounding error should be less. The last factor was decisive. The reason why our revision of Danielewsky's method should produce less rounding error is that in the reduction: $M_{i+1} = C_i M_i C_i^{-1}$ there are two possible sources of error, the first is that C_i may not be the exact inverse of C and the second is the rounding error in the indicated calculation. Our version eliminates the first source of error because C_i is the exact inverse of C_i^{-1} instead of a computed inverse. This should cut the per-step rounding error in half.

The actual operations contained in the program are:

1) Compute V_i = the i'th. column divided by its (i+1)st. element
2) $M_i \cdot V_i$ and place answer in (i+1)st. column of M_i
3) $m_{j,k}^{i+1} = m_{j,k}^{i} - v_j \cdot m^i_{(i+1),k} \quad j \neq i+1$

$m^{i+1}_{(i+1),k} = m_{(i+1)}{}^{i}{}_{,k}$

The first step is a loop, and steps 2) and 3) are both loops within loops. The three steps are iterated (n-1) times; the final step in the reduction to Frobenius form is the multiplication by the subdiagonal elements.

Next, the roots of the characteristic equation are obtained. Newton's method is used and each root of the reduced equation is substituted back into the original equation for correction and checking. In addition, the program contains special parts to handle the particular problems of aircraft flutter. We get M by dividing a given matrix by a diagonal matrix. We also compute w, V, M and g, functions of each root. The whole existing program runs about eight minutes per matrix.

There are at least two plausible methods of checking the results of the program. The first is to add an (n+1)th. row to M, a check row whose elements are the sums of the elements in their columns. For the product $M_i C_i^{-1}$ the check row remains a check row. For the multiplication $C_i(M_i C_i^{-1})$ it is necessary to augment C_i by adding an (n+1)th. row and column. The latter is zero except on the diagonal, which is one, and the former is zero except on the diagonal and the (i+1)st. column which is the sum of the off-diagonal elements in C_i. The product now contains a valid check row. Examination of the check row after (n-1) reductions will locate gross errors, and give a rough idea of the total rounding error in reduction. This check was used on our CPC program. The check we plan for our 650 program is the direct one using the determinant, which also gives the eigenvectors. We make this choice because there is enough demand for the eigenvectors to justify the extra work involved. Meanwhile we have been running the program without check, except that any value questioned by the customer is checked on the CPC. So far we have run several hundred cases without finding a serious error.

V. The Modified Power Method

The extension of the power method described in II-H has been programmed and tested on the CPC. For a calculation using a given number of digits, we regard the method as the most accurate we have tried. One difficulty in analysis of rounding error is getting examples whose exact answers are known. We constructed by hand an 8 x 8 matrix whose eigenvalues were complex integers. The example contained five and six digit numbers to ensure normal rounding error for multiplication. It was not weighted on the main diagonal and the largest element was about ten times the largest eigenvalue. While no pair of roots were close in the complex plane, one set of three roots and another of two were of similar absolute value. On the whole, we considered this matrix as a moderately bad case. Using the modified power method, we obtained the eigenvalues to about seven digit accuracy.

We have not programmed this process for the 650 because we regard Danielewsky's method as suitable for matrices of order 5 or smaller; and with double precision routines, we expect to use it for matrices of any order within the capacity of the 650.

As both the modified power method and Danielewsky's method involve same type of reduction, we might ask, "How can one be more accurate than the other?" The answer is that the modified power method is essentially an elimination process working down the main diagonal while Danielewsky's method takes its pivotal elements down the sub-diagonal. Most matrices arising from physical problems tend to be weighted on the main diagonal which means the pivotal elements of the power method tend to be large, while for Danielewsky's method we usually seem to get at least one small one. Also when a matrix is reduced by the power method, the elements normally become smaller and smaller as the larger eigenvalues are eliminated first, while with Danielewsky's method they gorw, as the coefficients of the characteristic equation tend to be large compared to the individual roots.

The rounding error of the basic reduction process is easy to study for the case of distinct roots. Let $M_{i+1} = C_i M_i C_i^{-1} = KDK^{-1}$ where D is the diagonal matrix of eigenvalues. Let $\bar{M}_{i+1}$ be the computed matrix obtained from the reduction using machine calculations and define $\epsilon_{i+1} = \bar{M}_{i+1} - M_{i+1}$. Now $K^{-1} \bar{M}_{i+1} K = D + K^{-1} \epsilon_{i+1} K$. If the ϵ_{i+1} terms are small compared with the differences between the λ_i, then the eigenvalues of $D + K^{-1} \epsilon_{i+1} K$ depend almost entirely on the diagonal terms, and hence the errors depend primarily upon the diagonal terms of $K^{-1} \epsilon_{i+1} K$. It is possible to scale K, so that all of its columns have length one. Then we get a bound for the errors in the roots of: The square roots of the sum of the squares of the elements of ϵ_{i+1} times the length of the rows in K^{-1} corresponding to these roots. From this it can be seen that the closer a matrix is to symmetric or diagonal, the less is the rounding error in this reduction. The power method reduces a matrix to a form which more and more closely approaches diagonal, while Danielewsky's method produces an unsymmetric form.

In principle, both methods may be improved by proper choice of pivotal element. In Danielewsky's method the largest element in the I'th. column which is below the main diagonal may be used. If the only large element lies on the main diagonal this is not very helpful. In the power method we may interchange rows and columns so that the largest element of the eigenvector of the reduced matrix is the pivotal element, so the system is always well conditioned in this respect. Frequently, vibration matrices come so arranged naturally. We have done a limited amount of experimenting with the results of interchanging rows and columns in Danielewsky's method with discouraging results. It seems that frequently what is gained in one step, is lost on succeeding steps. However, when we have added everything we want to our program, if we have storage for more instructions, we may add the interchange of rows and columns, to give protection in the rare case of extra bad luck.

VI. Conclusions

If trivial multiplications and divisions are avoided a matrix may be reduced by Danielewsky's method using about n^3 multiplications. Another number of multiplications, usually smaller are needed to get the roots of the polynomial. With the power method the number of operations is indeterminant. However, if we assume twelve iterations per eigenvector, it comes out about $2n^3$ multiplications for reductions and $4n^3$ for iterations. In addition, there are the extra iterations at the end to refine the roots.

Now a double precision routine uses three single precision multiplications to get a double precision product and from this point of view could be considered as three times as slow, and similarly for division. Therefore, for matrices from 6x6 complex to 12x12, where we have a choice of single precision power method or double precision Danielewsky, we plan to use double precision and are programming routines using 18 digit numbers with a 2 digit exponent.

DATA REDUCTION OF TELEMETERED INFORMATION ON THE IBM TYPE 650

Essor Maso and Raymond C. Clerkin
Hughes Aircraft Company

One of the byproducts of all experimental work is the seemingly endless amounts of data. The information may be recorded on paper tape, magnetic tape, oscillographs or film. But somehow, if there happens to be a computing facility in the area, the data eventually is placed on punched cards and it becomes the task of the computer to reduce these numbers. The first impulse one has is to run and hide and maybe the engineer will go away, but on closer inspection it becomes clear that here is a job that although not glamorous, is highly important and should be studied very carefully from all points of view. Many engineers are waiting for these calculations, since they will be used in further studies. One particular case in point is the job of reducing data for a Hughes missile. Since the early part of 1952, Hughes Aircraft has been using a system whereby telemetered information has eventually found its way to punched cards. Just how this is done is not of immediate importance, but the data is eventually punched from a 521 summary punch. The cards are brought to the machine room for calculation and then returned to another facility for automatic plotting. The data has usually been sampled at a rate of thirty points a second on as many as twenty-eight channels. The early attempts at data reductions generally involved calculations of the order Ax + B, A and B being parameters dependent on the particular channel being sampled and x the value at a certain time t. Since the portion of the flight that was interesting rarely exceeded five or six seconds and the number of channels involved usually was around twenty, and in view of the simple linear calculations that were to be performed, this problem was rather easily performed on a 604. It is of interest at this point to note that prior to the use of the 604, a 602A was used and before that a 602. However, as time passed the calculations became more and more involved and the number of channels were increased to 28. At present the equations used may take any one of the following six forms:

1. $A\,(x - x_1) + B$

2. $$\frac{A\,(x - x_1) + C}{(x_5 - x_1) + D\,(x - x_1) + E} + B$$

3. $$\frac{A\,(x - x_1)^2 + B\,(x_4 - x_1)\,(x - x_1) + C\,(x_4 - x_1)^2}{(x_6 - x_1) + D\,(x - x_1)}$$

4. $$\frac{A\,(x - x_1)^2 + B\,(x_4 - x_1)\,(x - x_1) + C\,(x_4 - x_1)^2}{(x_6 - x_1) + D\,(x - x_1)\,x_2}$$

5. $$\frac{A\,(x_6 - x_1) + C}{(x - x_1) + D\,(x_6 - x_1) + E} + B$$

6. $$\frac{b}{2}\left(-1 + \sqrt{1 - \frac{4c}{b^2}}\right)$$

where $b = A\,x + D\,x_{11} + B$

and $C = E\,x^2 + F\,x + G\,xx_{11} + H\,x_{11} + I\,x_{11}^2 + J$

where the x without a subscript refers to the data point, the x with a subscript refers to certain channels which are for reference purposes, and the remaining letters are empirical constants. Needless to say that work on a 604 now became almost prohibitive. However, since each card contained only one data point, it wasn't economically feasible to use the 650 for these calculations since the input time would slow the machine to a point where the total elapsed time of processing would not be appreciably faster than the 604.

To overcome this problem, an old device was used. The data, as was stated before, is punched on a 521 summary punch. By offset gangpunching, as many as fourteen points may be placed on one card. Therefore, an entire frame or twenty-eight channels may be punched on two cards. Every fourteenth card may now be selected on a sorter and the card volume is decreased by one-fourteenth. Directly following the input of the program instructions are a group of special instruction cards that may change for each job run. These instructions indicate which of the six types of calculations are to be performed on the various channels. All of the identifying information for listing purposes is placed on the detail cards. The output is also punched fourteen per card. This is easily accomplished by placing two numbers in one storage location (since the numbers never exceed four digits) and saving the sign in one of the numbers in word ten. This involves the use of punch code selectors.

Comparative speeds indicate an over-all improvement in the neighborhood of ten to one over the old system. Not enough work has been done as yet to justify a more accurate estimate. The main increase in speed stems from the elimination of the majority of the card handling as might be expected. Yet the actual computing speedup is of the order of four to one.

From this example it becomes fairly evident that there are many other problems in data reduction that will in time be transferred to our 650 for faster and more varied analysis.

THE DETERMINATION OF THE AUTOCORRELATION AND POWER SPECTRUM BY USE OF THE IBM TYPE 650

Essor Maso and William J. Drenick
Hughes Aircraft Company

The problem of estimating the power spectrum from the autocorrelation, or more precisely the autocovariance, function starting from a set of raw data has been of frequent occurrence at Hughes Aircraft Company. In the past four and one-half years we have proceeded from hand claculations to the use of the 402 tabulator and 602A calculating punch, the CPC, and finally to our present use of the 650.

This method of determining the power spectrum of a discrete stationary time series is due to John W. Tukey. It may be found in the "Symposium on Applications of Autocorrelation Analysis to Physical Problems", pp 47-67, Office of Naval Research.

Data is received in the form

$$y_1, y_2, y_3, \ldots, y_N$$

then

$$\bar{y} = \frac{1}{N} \sum_{i=1}^{N} y_i$$

$$x_i = y_i - \bar{y}$$

$$R_p = \frac{1}{N-p} \sum_{j=1}^{N-p} x_j x_{j+p}$$

We compute $R_o, R_1, R_2, \ldots, R_m$ where m is some predetermined number.

From these we calculate the apparent line powers $L_o, L_1, L_2, \ldots,$ L_m by the equations:

$$L_o = \frac{1}{2m}(R_o + R_m) + \frac{1}{m}\sum_{p=1}^{m-1} R_p$$

$$L_h = \frac{1}{m}R_o + \frac{2}{m}\sum_{p=1}^{m-1} R_p \cos\left(\frac{ph\pi}{m}\right) + \frac{1}{m}R_m \cos h\pi$$

for $0 < h < m$

$$L_m = \frac{1}{2m}\left[R_o + (-1)^m R_m\right] + \frac{1}{m}\sum_{p=1}^{m-1} (-1)^p R_p$$

Now we have $L_o, L_1, L_2, \ldots, L_m$.

Next we calculate

$$U_o = .54\, L_o + .46\, L_1$$

$$U_k = .54\, L_k + .23\,(L_{k-1} + L_{k+1}) \quad \text{for } 0 < k < m$$

$$U_m = .54\, L_m + .46\, L_{m-1}$$

Originally we found the autocorrelation function by a method known as progressive digiting on the 402 tabulator. This was discovered to be faster and more economical than 604 calculations. However, the power spectrum analysis was done on the 604 at first and later was done on the CPC. The problems encountered on the CPC were the tedious card preparation and the equally tedious card handling during the running of the problem. We were later able to speed up the autocorrelation computations with the advent of the 407 tabulator, but this was not a large scale improvement. The length of time from the arrival of the data until the engineer had the final answers would at best be two days. The actual computation would take two men one complete day working at top speed.

Prior to the arrival of our 650 the code was written and then checked out on a machine made available at Endicott, New York. The example or test case used at Endicott was one that took approximately one day to compute by our old method. This time was reduced to about 15 minutes on the 650.

The optimum programming features were used throughout the coding. We felt that this was necessary at least in the autocorrelation portion of the program because of the repetitive nature of the problem. The storage locations were allocated to allow for any number of input items up to 1,000. In addition, we allowed for any number up to 100 lags in the autocorrelation. Since the maximum number of cards to be punched would never exceed 100, we withheld punching until the end of the entire calculation period. It is only fair to note that the methods using the tabulator did not allow such large values of m and N.

An interesting observation made by Mr. Donald Criley of the Los Angeles Applied Science Office of IBM indicated that by eliminating the Fourier transformation and making a few minor changes in the autocorrelation coding, the cross correlation may be found in a similar fashion.

It should be mentioned that Tukey has given other methods of estimating the empirical power spectrum and at present prefers a different method of estimation rather than the U's given. These are given in his unpublished manuscript "Measuring Noise Color". Alternative techniques are discussed in the excellent paper of Ulf Grenander and Murray Rosenblatt, "Comments on Statistical Spectral Analysis", Skandinavisk Aktuarietidskrift, 1953, parts 3-4, pp 182-202. All of these publications discuss statistical tests of signification.

The use of the 650 in the autocorrelation and power spectrum solutions has been very useful to many members of the Research and Development Laboratories in radar and missile problems.

NUMERICAL SOLUTION OF AN INTEGRAL EQUATION CONCERNING VELOCITY DISTRIBUTION OF NEUTRONS IN A MODERATOR

D. B. MacMillan and R. H. Stark
Knolls Atomic Power Laboratory[1]

Wigner and Wilkins[2] have obtained an integral equation governing the energy distribution of neutrons that are being slowed down uniformly throughout the entire space by a uniformly distributed moderator whose atoms are in motion with a Maxwellian distribution of velocities. The effects of chemical binding and of crystal reflection were ignored (to put it another way, the moderator was assumed to be a monatomic gas.)

The atomic number of the moderator appears as a parameter in the expression for the kernel of the integral equation. In case the moderator is hydrogen, cancellations occur in that expression, and Wigner and Wilkins were able to derive from the integral equation a differential equation having the same solution as the integral equation. They solved that differential equation, and discussed the solution.

For moderators other than hydrogen, we know of no differential equation equivalent to the integral equation.

We were asked (by members of the Theoretical Physics Staff at Knolls Atomic Power Laboratory) to obtain numerical solutions to the integral equation for other moderators. It is our purpose here to discuss numerical techniques we are using in this problem. Our difficulties with this problem were all connected with the evaluation of the error function. This function is usually evaluated by means of a power series around the origin, and an asymptotic series for large values of the argument. In the middle range of values of the argument the first of these series converges very slowly and the second gives inaccurate values. We used a continued fraction expansion in place of the asymptotic series. The continued fraction bridged the gap nicely. Furthermore, it enabled us to circumvent a cancellation of terms which would have made it difficult to obtain sufficient accuracy. In the discussion which follows, we emphasize the considerations involved in the use of the continued fraction.

The integral equation given by Wigner and Wilkins is

$$(1) \qquad \left[V(v) + \gamma(v)\right] N(v) = S(v) + \int_0^\infty P(v_1 \longrightarrow v)\, N(v_1)\, dv_1$$

Here N(v) is defined by the statement that N(v)dv equals the number of neutrons per cubic centimeter having a velocity between v and v + dv (N(v) = neutrons per $cm.^3$ per unit velocity). V(v) is the probable rate of scattering collisions for neutrons at velocity v, γ is the probable rate of neutron absorptions, and $P(v_1 \longrightarrow v)$ is the probable rate at which neutrons with velocity v_1 will be scattered to velocity v. S(v) is a uniformly distributed source of neutrons.

1. The Knolls Atomic Power Laboratory is operated by the General Electric Company for the U. S. Atomic Energy Commission.

2. Wigner, E. P. and Wilkins, J. E. Jr., "Effect of the Temperature of the Moderator on the Velocity Distribution of Neutrons with Numerical Calculations for H as Moderator. AECD 2275." Oak Ridge, 1949.

Thus, the equation is just a statement of equilibrium; the rate at which neutrons of a given velocity disappear equals the rate at which they appear.

The kernel, $P(v_1 \longrightarrow v)$, is given by the formulas

$$(2) \quad P(v_1 \to v) = \theta^2 \frac{v}{v_1} \left\{ e^{(\beta^2 v_1^2 - \beta^2 v^2)} \left[\operatorname{erf}(\theta\beta v_1 - \zeta\beta v) + \operatorname{erf}(\theta\beta v_1 + \zeta\beta v) \right] + \operatorname{erf}(\theta\beta v - \zeta\beta v_1) - \operatorname{erf}(\theta\beta v + \zeta\beta v_1) \right\}$$

when $v_1 \leq v$, and:

$$(3) \quad P(v_1 \longrightarrow v) = \theta^2 \frac{v}{v_1} \left\{ \operatorname{erf}(\theta\beta v - \zeta\beta v_1) + \operatorname{erf}(\theta\beta v + \zeta\beta v_1) + e^{(\beta^2 v_1^2 - \beta^2 v^2)} \left[\operatorname{erf}(\theta\beta v_1 - \zeta\beta v) - \operatorname{erf}(\theta\beta v_1 + \zeta\beta v) \right] \right\}$$

when $v_1 \geq v$.

In these equations, $\theta = \frac{M+1}{2\sqrt{M}}$ and $\zeta = \frac{M-1}{2\sqrt{M}}$, where M is the ratio of the mass of a moderator nucleus to the mass of a neutron, $\beta = \frac{1}{\sqrt{2KT}}$, where K is Boltzmann's constant and T is the temperature of the moderator. By erf, we mean the error function,

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

The overall accuracy requirements for this problem are not severe, since the infinite moderator postulated here is a poor approximation to the finite moderator regions that appear in practice. However, there are spectacular cancellations. The trouble occurs in (3), in the term

$$(4) \quad e^{(\beta^2 v_1^2 - \beta^2 v^2)} \left[\operatorname{erf}(\theta\beta v_1 - \zeta\beta v) - \operatorname{erf}(\theta\beta v_1 + \zeta\beta v) \right]$$

For some values of v_1 and v, the evaluation of this results in numbers like:

$$1.6 \times 10^{15} [.99999\ 99999\ 99999\ 999996 - .99999\ 99999\ 99999\ 99999\ 99999\ 99999\ 95]$$

From our point of view, the problem was this: We were interested in obtaining values of the kernel accurate to, say, four decimals. If there was a factor of the order of 10^{15}, then we would have to obtain the error function accurate to nineteen or twenty decimals, which would be a nuisance on a machine with ten-digit words.

We used a continued fraction expansion for the error function:

$$(5) \quad \operatorname{erf}(x) = 1 - \frac{e^{-x^2}}{\sqrt{\pi}} \left[\cfrac{1}{x + \cfrac{1}{2x + \cfrac{2}{x + \cfrac{3}{2x + \cfrac{4}{x + \cdots}}}}} \right]$$

valid for $x>0$.[3] If we substitute this expression in the term in question we get

$$(6)\quad e^{\beta^2 v_1^2-\beta^2 v^2-(\theta\beta v_1-\zeta\beta v)^2}\, g(\theta\beta v_1-\zeta\beta v) - e^{\beta^2 v_1^2-\beta^2 v^2-(\theta\beta v_1+\zeta\beta v)^2}\, g(\theta\beta v_1+\zeta\beta v)$$

where we have used the notation

$$g(x) = \frac{1}{\sqrt{\pi}}\left[\cfrac{1}{x+\cfrac{1}{2x+\cfrac{2}{x+\cfrac{3}{2x+\cdots}}}}\right]$$

A short calculation, using the fact that $\beta v_1>0$, $\beta v>0$, and $M\geq 1$, shows that both exponents in (5) are negative. This means that the two values of $g(x)$ are multiplied by numbers less than 1; to insure, say, four decimals accuracy in the value of (3), we have only to get $g(x)$ accurate to five decimals.

Now $g(x)$ converges rapidly for large x, and slowly for small x. The series expansion (around $x=0$) of the error function converges rapidly for small x, and slowly for large x. It turned out to be convenient, in terms of the magnitude of intermediate quantities, to set $|x|\leq 1.97$ as the range of the program which evaluates the series. We chose $x=1.9$ as the boundary, and wrote a subroutine to evaluate the error function by means of the series when the argument is smaller than 1.9, and by means of the continued fraction when the argument is greater than 1.9. Also, we substituted expression (6) for expression (4) in equation (3) whenever $(\beta^2 v_1^2-\beta^2 v^2)>(1.9)^2$ -which assured that the arguments of the function $g(x)$ in (6) would always be >1.9.

We studied the convergence of approximations to the continued fraction by means of numerical experiments. Finally we chose the expression

$$g(x) \sim \frac{1}{\sqrt{\pi}}\left[\cfrac{1}{x+\cfrac{1}{2x+\cfrac{2}{x+\cfrac{3}{2x+\cdots+\cfrac{6}{x+\cfrac{7}{2x+A}}}}}}\right]$$

where the constant A was to be chosen to yield adequate accuracy over the range $x>1.9$. By approximations involving additional terms and by numerical experiments we chose $A=2.5$, which gives a maximum error of less than 2×10^{-7} in that range. (The elementary mathematical theory of continued fractions would use the approximations obtained by setting $A=0$ or $A=8$. These bracket the value of the infinite continued fraction, but differ from it by about 1×10^{-5}.)

3. H. S. Wall, Continued Fractions, New York, 1948, p. 357.

After the computation of the matrix which represents the kernel, we use the conventional iterative process,

$$N^{n+1}(v) = \frac{1}{V(v) + \gamma(v)} \left[S(v) + \int P(v_1 \longrightarrow v)\, N^n(v_1)\, dv_1 \right]$$

to get from a first guess at the solution, $N^0(v)$, to successively better approximations to the solution, $N^1(v)$, $N^2(v)$, . . .

The computation of values of the kernel is entirely (including all subroutines) sequentially programmed. It takes about two seconds per point, or about half an hour for a 25 x 25 mesh. The iteration procedure is optimum programmed. It takes about 1/30 second per point of the kernel, or about thirty seconds per iteration for a 25 x 25 mesh.

At the present time, only exploratory calculations have been made using the program discussed here. Further calculations will be made, and results will be published later.

APPLICATIONS OF THE 650 MAGNETIC DRUM DATA PROCESSING MACHINE AT MARQUARDT AIRCRAFT COMPANY

Richard A. De Santis
Marquardt Aircraft Company

The development of ramjet engines and the facilities for testing these engines have reached a point where it is imperative that engine development work be supplied with equipment and facilities for quickly and efficiently recording and calculating large amounts of test data, and for providing development engineers with an efficient tool for determination of engine design and performance criteria.

The Marquardt Jet Laboratory has facilities for testing engines ranging in size from 1 inch to several feet in diameter. Under various conditions, altitudes may be simulated from sea level up to 100,000 feet, temperatures up to 970°F, and fuel flows up to several hundred gpm. During a test run 80 channels of data may be recorded every 0.2 second. Typically, these might represent 74 pressures at various stations throughout the engine being tested, the remaining 6 being temperature and fuel flow readings. Thus, for a test lasting 20 seconds, 8,000 data readings could be recorded. Within approximately 2½ hours after completion of an engine test run, printed lists of pressure ratios, combustion efficiencies, fuel/air ratios, Mach numbers, etc. are available for engineering analysis by the Test Engineer.

Figure No. 1 is a functional diagram of the data recording and computing system incorporated at the Marquardt Aircraft Co.

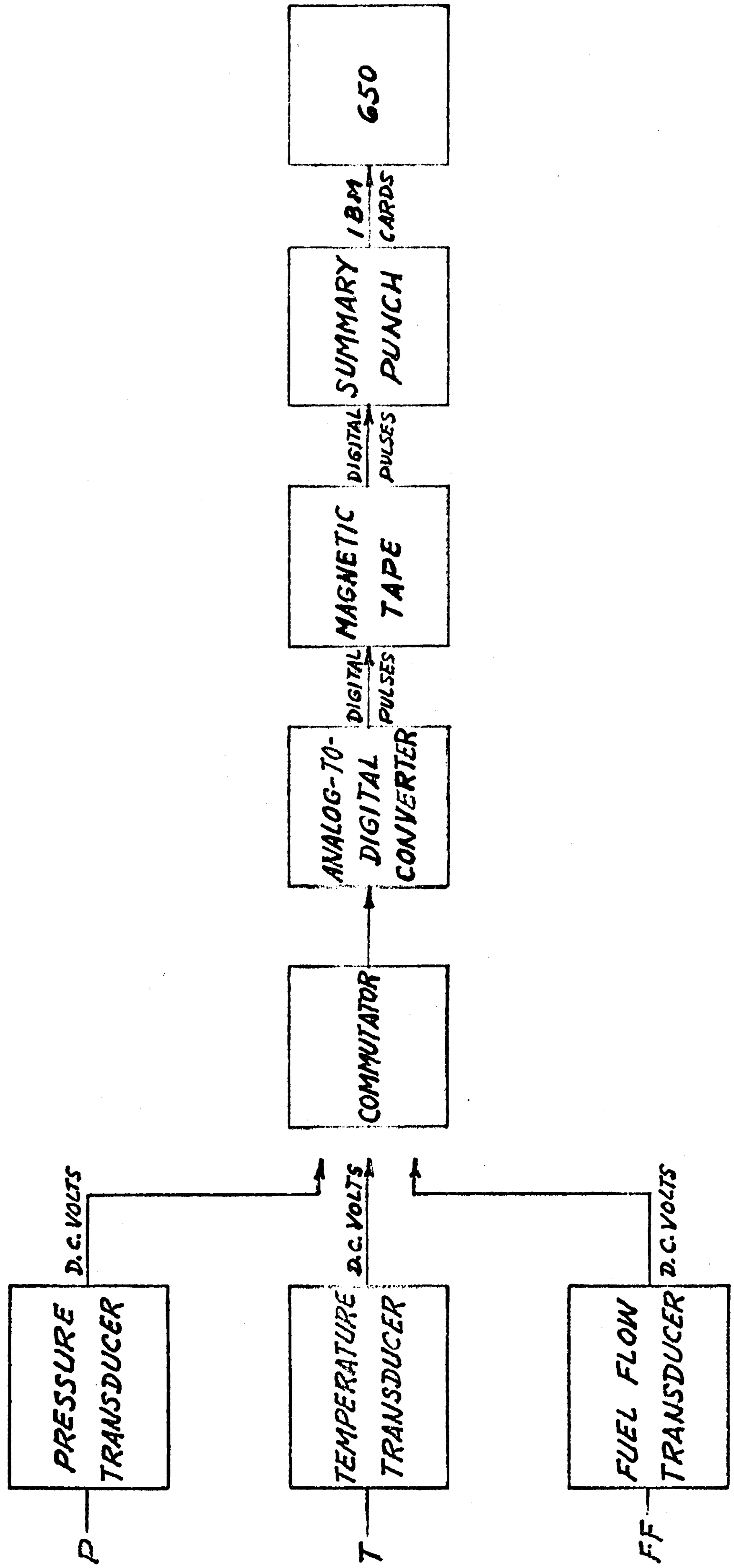

P
PRESSURE TRANSDUCER
D.C. VOLTS
T
TEMPERATURE TRANSDUCER
D.C. VOLTS
FF
FUEL FLOW TRANSDUCER
D.C. VOLTS
COMMUTATOR
ANALOG-TO-DIGITAL CONVERTER
DIGITAL PULSES
MAGNETIC TAPE
DIGITAL PULSES
SUMMARY PUNCH
IBM CARDS
650

FIGURE 1

Referring to the diagram, the quantities of interest to be recorded during a test run are fed into transducers where the signals are amplified. A commutator selects the d.c. voltages and transmits them to an analog-to-digital converter, which emits digital pulses which are recorded on magnetic tape. After the test run, the magnetic tape is played back into a summary punch which produces IBM cards on which the recorded quantities are scaled from 0 - 999. For maximum resolution, the amplification can be doubled or quadrupled. The cards are then loaded into the 650 where the numbers are entered into a cubic equation which simultaneously applies an error correction and shifts the scale and range to correspond to the measured quantities. On the next pass through the 650 the desired calculations are performed. The fixed decimal system is read in the data processing problem. A typical engine test run would produce 400 IBM cards, each containing 20 data readings which would be completely processed on the 650 in a period of approximately 2½ hours.

In addition to its use as a data processing computer, the IBM 650 is used at Marquardt Aircraft Co. for the solution of a wide range of engineering design and performance problems. A typical problem is the design of an axially symmetric, supersonic nozzle with continuous wall curvature. Whereas the previous problem used a fixed decimal mode to compute a large number of sums and quotients, this method employs a floating decimal abstract interpretive routine (FLAIR) and involves circular and inverse trigonometric functions, logarithms and exponentials, and an iterative solution, by the difference method, of four simultaneous nonlinear differential equations.

Specifically, it is desired to calculate the shape of a nozzle under the following conditions:

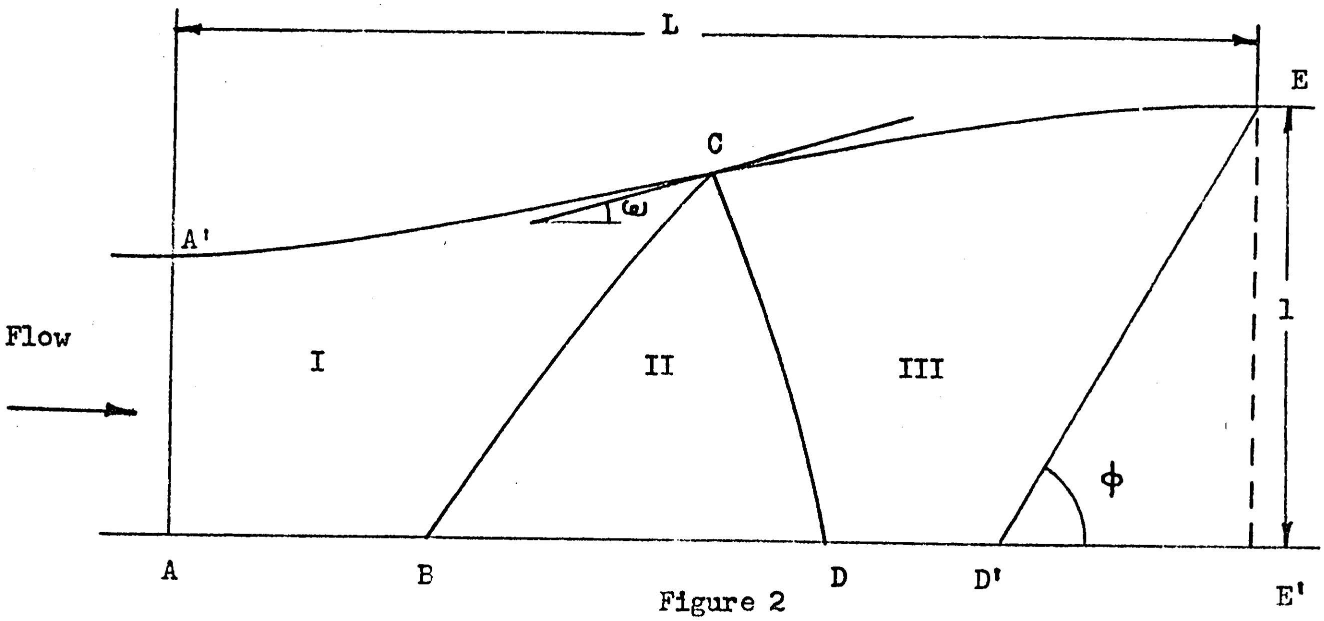


Figure 2

Referring to Figure No. 2, the flow field is first specified as being radial in Region II, bounded by the left and right characteristic lines BC and DC. Flow through A-A', the throat, is assumed at sonic uniform velocity, a condition frequently assumed in nozzle calculations and closely attainable in practice. Finally, flow is specified as being plane and parallel to the axis as it passes through D'E at the design Mach number $M_{D'}$. The problem is to find the wall shape A'CE which will transform the plane parallel flow at AA' to radial flow at BC, and the radial flow at CD to plane parallel flow at D'E at the design Mach number. D'E is also a left characteristic line and the acute angle at D' is defined by $\phi = \text{arc cot} \sqrt{M^2_{D'} - 1}$. The exit radius is taken as unity, the length of the nozzle is L, and the maximum wall angle ω is obtained at C, which is thus the point of inflection. With $M_{D'}$, L and ω given, enter a family of curves especially prepared for this purpose and determine a value M_D. Enter the 650 with M_D', M_D, ω, and γ, followed by the program deck. The solution begins with the following equations: (Ref. [1])

(1) $$\nu = \frac{1}{2}\sqrt{\frac{\gamma+1}{\gamma-1}}\tan^{-1}\left[\frac{\gamma-1}{\gamma+1}(M^2-1)\right]^{\frac{1}{2}} - \frac{1}{2}\tan^{-1}\sqrt{M^2-1}$$

(2) $$\begin{cases} \nu_C = \nu_D - \omega \\ \nu_B = \nu_C - \omega \end{cases}$$

(3) $$\tau = \left\{ \frac{\left[\frac{2}{\gamma+1}\left(1+\frac{\gamma-1}{2}M^2\right)\right]^{\frac{\gamma+1}{2(\gamma-1)}}}{M} \right\}^{\frac{1}{2}}$$

(4) $$\left.\frac{dM}{d\alpha}\right|_{(y=0)} = \frac{2MK_1}{\tau}\cdot\frac{1+\frac{\gamma-1}{2}M^2}{M^2-1}, \quad K_1 = 2\tau_{D'}\sin\frac{\omega}{2}$$

(5) $$\begin{cases} \alpha_A = 0 & \alpha_D = \alpha_B + \dfrac{\tau_D - \tau_B}{K_1} \\ \alpha_B = \dfrac{2(M_B - 1)}{M_B' = \left.\frac{dM}{d\alpha}\right|_{(\alpha=\alpha_B)}} & \alpha_{D'} = \alpha_D + \dfrac{2(M_{D'} - M_D)}{M_D' = \left.\frac{dM}{d\alpha}\right|_{(\alpha=\alpha_D)}} \end{cases}$$

Equations (1) and (2) are readily seen to be one half the Prandtl-Meyer expansion angle; equation (4) is valid for radial flow.

Now perform calculations tabulated as follows:

Use equation (1) to compute ν_D

(2) . . . . ν_C, ν_B

(1) . . . . M_C, M_B *

(3) . . . . $\tau_{D'}$, τ_D, τ_B

(4) . . . . M'_B, M'_D

(5) . . . . α_B, α_D, $\alpha_{D'}$

(*use Newton's method, with $M^{\circ} = 5\nu + 1$ as the first approximation)

Note that L = $\alpha_{D'} + \sqrt{M_{D'}^2 - 1}$

Let

$$f(\alpha) = \frac{(M'_B)^2}{4(M_B - 1)}\alpha^2 + 1$$

$$g(\alpha) = -\frac{(M'_D)^2}{4(M_{D'} - M_D)}\alpha^2 + \left(M'_D + \frac{(M'_D)^2 \alpha_D}{2(M_{D'} - M_D)}\right)\alpha + \left(M_D - \frac{(M'_D \alpha_D)^2}{4(M_{D'} - M_D)} - M'_D \alpha_D\right)$$

It can be seen that

$$f(\alpha_A) = 1 \qquad f(\alpha_B) = M_B \qquad g(\alpha_D) = M_D \qquad g(\alpha_{D'}) = M_{D'}$$

$$f'(\alpha_A) = 0 \qquad f'(\alpha_B) = M'_B \qquad g'(\alpha_D) = M'_D \qquad g'(\alpha_{D'}) = 0$$

Hence, f(x) and g(x) are used to define M on AB and DD', respectively.

To obtain starting values on BC, select about 20 equally spaced values of M between M_B and M_C, and compute the corresponding values of ν and τ using (1) and (3). Then,

$$\tan\theta = \tan(\nu - \nu_D) \qquad \theta = \text{flow angle}$$

$$x = \frac{\tau\cos\theta}{K_1} + K_2 \qquad K_2 = x_D - \frac{\tau_D}{K_1}$$

$$y = \frac{\tau\sin\theta}{K_1}$$

Compute CD in like manner, substituting

$$\tan\theta = \tan(\nu_D - \nu)$$

The flow field is computed by the familiar method of characteristics, using

$$dy = \lambda^L dx \qquad dy = \lambda^R dx$$

$$d\theta - A\,dM + B^L dx = 0 \qquad d\theta + A\,dM - B^R dx = 0$$

(left characteristic) (right characteristic)

where

$$\lambda^{L,R} = \frac{\sqrt{M^2-1}\cdot\tan\theta \pm 1}{\sqrt{M^2-1} \mp \tan\theta}$$

$$A = \frac{\sqrt{M^2-1}}{M\left(1+\frac{\gamma-1}{2}M^2\right)}$$

$$B^{L,R} = \frac{\tan\theta}{y\left(\sqrt{M^2-1}\mp\tan\theta\right)} \text{ if } y \neq 0; \quad \frac{1}{2}A\frac{dM}{dx} \text{ if } y = 0$$

Having $x_L, y_L, \tan\theta_L, M_L$ and $x_R, y_R, \tan\theta_R, M_R$ solve by iteration for $x_N, y_N, \tan\theta_N, M_N$ using

$$\alpha_N = \frac{y_L - y_R + \alpha_R \bar{\lambda}_R - \alpha_L \bar{\lambda}_L}{\bar{\lambda}_R - \bar{\lambda}_L}$$

$$(6) \qquad y_N = \frac{y_L \bar{\lambda}_R - y_R \bar{\lambda}_L - \bar{\lambda}_L \bar{\lambda}_R (\alpha_L - \alpha_R)}{\bar{\lambda}_R - \bar{\lambda}_L}$$

$$\theta_N = \frac{\bar{A}_R \theta_L + \bar{A}_L \theta_R + \bar{A}_L \bar{A}_R (M_R - M_L) + \bar{A}_L \bar{B}_R (\alpha_N - \alpha_R) - \bar{A}_R \bar{B}_L (\alpha_N - \alpha_L)}{\bar{A}_R + \bar{A}_L}$$

$$M_N = \frac{\theta_R - \theta_L + \bar{A}_L M_L + \bar{A}_R M_R + \bar{B}_L (\alpha_N - \alpha_L) + \bar{B}_R (\alpha_N - \alpha_R)}{\bar{A}_R + \bar{A}_L}$$

where

$$\bar{\lambda}_L = \tfrac{1}{2}(\lambda_L^L + \lambda_N^L) \qquad \bar{\lambda}_R = \tfrac{1}{2}(\lambda_R^R + \lambda_N^R)$$

$$\bar{A}_L = \tfrac{1}{2}(A_L + A_N) \qquad \bar{A}_R = \tfrac{1}{2}(A_R + A_N)$$

$$\bar{B}_L = \tfrac{1}{2}(B_L^L + B_N^L) \qquad \bar{B}_R = \tfrac{1}{2}(B_R^R + B_N^R)$$

Solve equations (6) in the order given, using $\bar{\lambda}_L = \lambda_L^L$ etc. on the first iteration. On all other iterations, compute λ_N, A_N, B_N using the values of $\tan\theta_N$, M_N from the previous iteration.

To compute the flow net in Region I, start with the known points on BC and $b_1 = \alpha_B - 0.1$ (see Figure 3). Use b_1 and a_1 to get b_2, then b_2 and a_2 to get b_3, etc. Suppose b_n is the first point which lies above the tangent to the streamline at C. (This can be determined by testing the slope of b_iC after each point b_i is computed.)

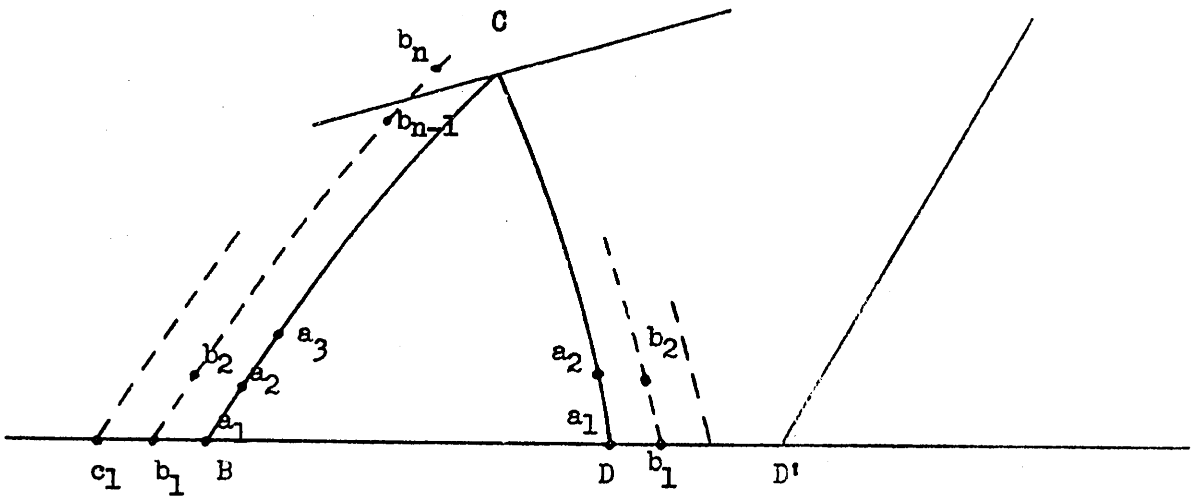


Figure 3

Then determine a new streamline (wall) point S(x, y, tan θ , M) by solving

$$\frac{y - y_{m-1}}{\alpha - \alpha_{m-1}} = \frac{1}{2}\left(\lambda^L_m + \lambda^L_{m-1}\right) \qquad \lambda^R \text{ in Region III}$$

$$\frac{y - y_c}{\alpha - \alpha_c} = \frac{1}{2}\left(\tan\theta + \tan\theta_c\right) \qquad \theta_c = \omega$$

$$\frac{\tan\theta - \tan\theta_{m-1}}{\tan\theta_m - \tan\theta_{m-1}} = \frac{\alpha - \alpha_{m-1}}{\alpha_m - \alpha_{m-1}}$$

for x, y, tan θ . Having thus determined s so that, simultaneously,

$$\alpha = s\,\alpha_m + (1-s)\,\alpha_{m-1}$$

$$y = s\,y_m + (1-s)\,y_{m-1}$$

$$\tan\theta = s\tan\theta_m + (1-s)\tan\theta_{m-1}$$

this value of s can be used to determine

$$M = s\,M_m + (1-s)\,M_{m-1}$$

This method assumes that: (1) the characteristic line is straight between b_{n-1} and b_n and (2) that x, y, tan θ , and M vary linearly on this segment.

No appreciable error is introduced by these assumptions. Having x, y, tan θ and M, compute K_3x, K_3y and θ , where K_3 is simply a constant used to scale the dimensions up to the desired exit diameter. Then punch a card containing K_3x, K_3y, θ , M and M_D'.

This procedure is repeated using the left characteristic just constructed, $c_1 = b_1 - 0.1$, and the tangent to the streamline at S.

With appropriate substitutions, the same instructions can be used to compute Region III, taking points $a_1 = \alpha_D$, b_1, c_1, ...along DD'E and constructing right characteristics. The results can be compared with the theoretical values at A' ($0, \frac{1}{\tau_{D'}}, 0, 1$) and at E($L, 1, 0, M_{D'}$).

It can be seen from Figure 4, which describes the computation procedure using the 650, that this entire program can be run with no decision required by the operator, after one card is loaded, containing M_D', M_D, ω , γ , K_3.

Depending, of course, upon the fineness of the flow net constructed, the time required to design a nozzle by this method is approximately 2 hours.

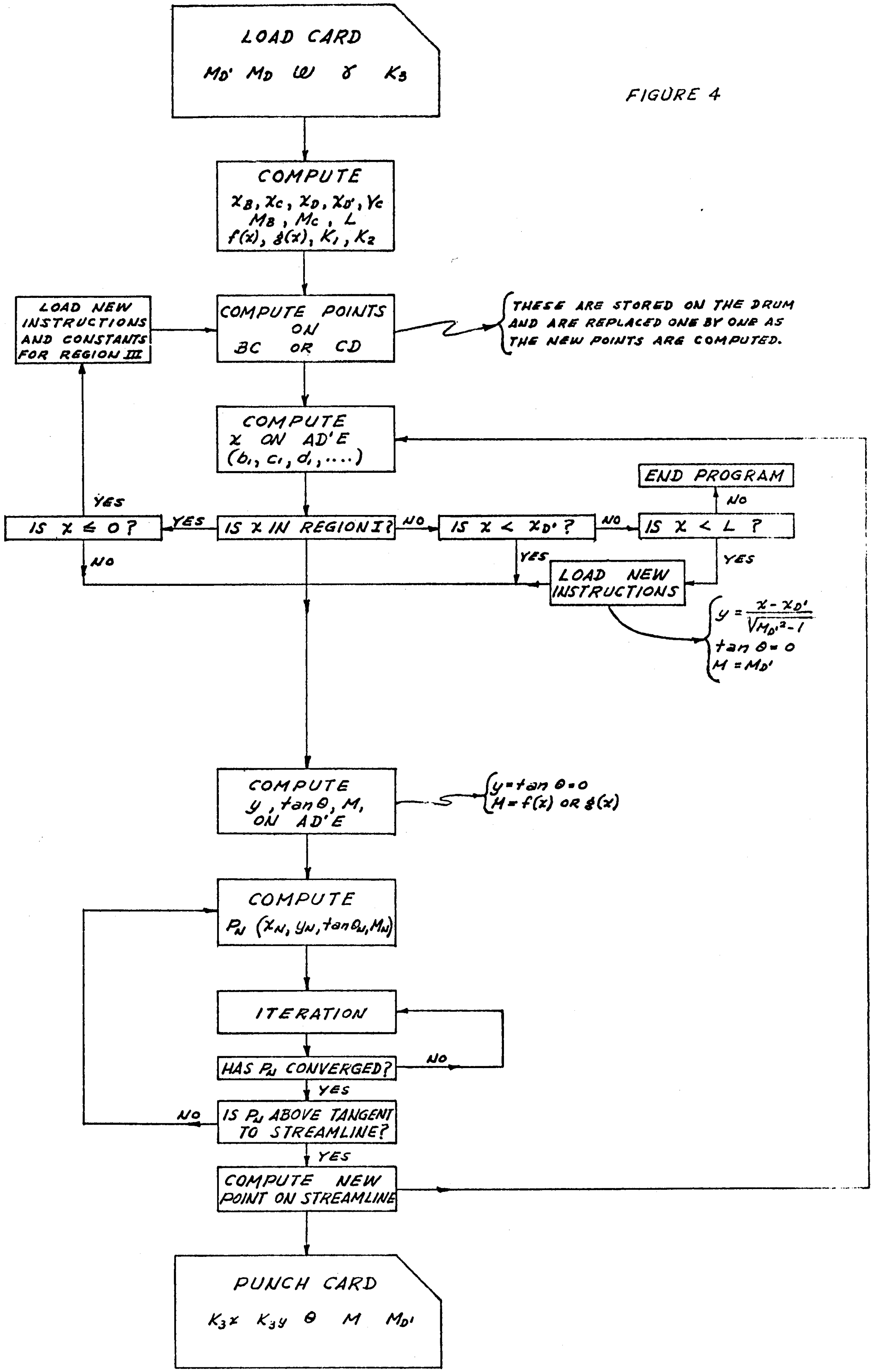

LOAD CARD
MD' MD ω γ K3
COMPUTE
xB, xC, xD, xD', YC
MB, MC, L
f(x), g(x), K1, K2
LOAD NEW INSTRUCTIONS AND CONSTANTS FOR REGION III
COMPUTE POINTS ON BC OR CD
THESE ARE STORED ON THE DRUM AND ARE REPLACED ONE BY ONE AS THE NEW POINTS ARE COMPUTED.
COMPUTE x ON AD'E (b1, c1, d1, ...)
END PROGRAM
YES
NO
IS x ≤ 0?
YES
IS x IN REGION I?
NO
IS x < xD'?
NO
IS x < L ?
NO
YES
LOAD NEW INSTRUCTIONS
YES
y = (x - xD')/√(MD'^2 - 1)
tan θ = 0
M = MD'
COMPUTE y, tan θ, M, ON AD'E
y = tan θ = 0
M = f(x) OR g(x)
COMPUTE PN (xN, yN, tan θN, MN)
ITERATION
HAS PN CONVERGED?
NO
YES
NO
IS PN ABOVE TANGENT TO STREAMLINE?
YES
COMPUTE NEW POINT ON STREAMLINE
PUNCH CARD
K3x K3y θ M MD'

FIGURE 4

In conclusion, two points should be noted. First, the shortest possible nozzle can be designed by this method with a given M_D' and ω, by choosing $M_D = M_D'$. This will introduce a discontinuity in $\frac{dM}{dx}$ at x_D', but experience has not indicated any serious difficulty. Second, ω and M_D must be chosen so that $\omega < \frac{1}{2} \nu_D$. This is to permit expansion in Region I.

It has been shown how the IBM 650 computer is being used to handle current problems at the Marquardt Aircraft Co. Its use will be invaluable in reducing the time of ramjet engine development from the preliminary design phase to its use as a power plant in support of our national defense effort.

Reference [1] NACA TN 2711 "The Aerodynamic Design of High Mach Number Nozzles utilizing Axisymmetric Flow with Application to a Nozzle of Square Test Section" by Ivan E. Beckwith, Herbert W. Ridyard, and Nancy Cromer

DETERMINATION OF CRITICAL SPEEDS IN ROTATING SYSTEMS BY MEANS OF AN IBM TYPE 650

Marshall Middleton, Jr.
Westinghouse Electric Corporation

Introduction

One of the most important problems in the design of large electrical machines is the determination of the critical speeds or natural frequency of vibrations of the rotating system. Coincidence of the normal operating speed and critical speeds or any harmonic thereof, will produce vibrations detrimental to the operation of the machine. These vibrations endanger the strength of the various structural elements and thus make machine operation extremely hazardous. Excessive oil and gas sealing gland wear, bearing surface fatigue, improper commutation, fretting corrosion at joints and fits are but a few of the injurious effects perpetrated by excessive vibrations. To eliminate or minimize these vibrations, it is necessary for the critical speeds to be sufficiently displaced from the operating speed. Hence, it is imperative to know the location of the critical speeds and their position relative to the normal operating speed.

The rotating systems for which the critical speeds must be determined vary in combination of simply supported to multiple span systems with or without overhanging extension on one or both ends. The physical composition of the system greatly effects the critical speed. For instance, the addition of a short overhanging shaft to a system which is simply supported at either end may change the location of the critical speed by as much as 20 to 25 percent. The number of bearings and the flexibility in the bearing mounts both effect the critical speed and must be considered in the calculations.

An important factor contributing to the demagnification of the amplitude of vibration which occurs at a critical speed is the damping produced by the oil film between the rotating system and the bearing surface. This damping is present at all speeds but is most beneficial at or near the critical speeds. Many of the large electrical machines built today have normal operating speeds which lie between the first and second critical speeds. The ease with which these machines cross over the first critical may be attributed almost exclusively to the oil film damping.

In order to obtain the location of the critical speed and the amplitude of its associated vibrations with any degree of accuracy, it is apparent that the physical composition of the rotating system, the bearing characteristic and the oil film damping must be simultaneously considered.

Computational Procedure

The critical speed vibrations of a rotating system may be considered as a special case of forced beam vibrations. The exciting force in this case is produced by any eccentricity or unbalance in the rotating system. This force, however, is not constant but varies as the square of the speed of rotation.

The method used to obtain the critical speeds is an extension of a Holzer type iteration. The rotating system is first broken up into a large number of sections with constant diametrical moments of inertia. For large machines, such as modern tandem compound turbine-generators, it is not uncommon to divide the rotating systems into 80 or more sections. The numerical representation of the system consists of the length, weight and areal inertia of every section. The location of the bearing with respect to these sections, the flexibility constants of the bearings and the weights at the bearings which contribute to the shearing force are also required. When considering the oil film between the rotating system and the bearing, an external force must be applied in order to supply the energy absorbed by the damping.

Solutions of the problem is based on the standard deflection coefficient equations, which express the deflection angle, θ, the displacement, X, the moment, M, and the shear, S, at the n + 1st position in terms of those at the nth position.

$$\theta_{n+1} = \theta_n + \beta_n S_n + \alpha_n M_n$$

$$X_{n+1} = X_n + \ell_n \theta_{n+1} - \gamma_n S_n - \beta_n M_n$$

$$M_{n+1} = M_n + \ell_n S_n$$

$$S_{n+1} = S_n \left(\frac{2\pi f}{g}\right)^2 W_{n+1} X_{n+1}$$

where

$$\alpha = \frac{\ell}{EI}, \quad \beta = \frac{\ell^2}{2\,EI}, \quad \gamma = \frac{\ell^3}{3\,EI}.$$

The frequency of the forced vibration appears in the shear term, and the shear term is contained in the other three equations.

To obtain the critical speed a trial frequency of vibration is selected. Then a set of beginning conditions are established. If the starting end consists of an overhanging extension, the initial moment and shear terms are zero while the slope and deflection are carried as unknowns. When starting at a bearing, the initial moment and deflection are zero with the slope and shear as unknowns. The trial frequency, and the two beginning conditions are substituted into the above equations. The slope, displacement, shear and moment are then calculated at the end of the first section. These conditions are in turn used as initial conditions for the next section and the process repeated. When an intermediate bearing is encountered, a constraint is introduced in the form of a known force depending on the spring constant and the deflection at the previous section. This provides one additional equation at that point and sets the deflection to a value depending upon the spring constant of the bearing.

At the end of the last section, four equations are obtained expressing the slope, deflection, moment and shear in terms of the initial unknowns. An additional equation and unknown shear force were added for each bearing encountered. A set of known conditions exist at this point. If the system terminates in an overhanging extension, the shear and moment must be zero. When the system terminates in a bearing the moment is zero and the deflection is proportional to the spring constant of the bearing. With these conditions and the above results, a system of n simultaneous equations are resolved. This system of equations will have a non-trivial solution only if the determinants of its coefficients are zero. Since the coefficients contain the frequency, this determinant defines a polynomial in the frequency whose roots are the natural frequencies of vibrations. Rather than finding the roots of the frequency polynomial, it is easier to select trial frequencies until the value of the close out determinant changes sign. The natural frequency of vibration then lies between those two frequencies which produce sign reversal of the value of the close out determinant.

Because of the wide ranges in the magnitudes of the numbers used in this problem, the floating decimal number system must be used. The IBM floating decimal routine written by G. R. Trimble, Jr., was adopted. The critical speed program deck consists of approximately five hundred instructions which are loaded into the 650, four instructions per card. The selected value of frequency is set on the storage entry switches. After the 650 has completed the calculations, the value of the close out determinant is located on the display lights. From this value, a new frequency is selected and the process continued.

When the rotating system is extensive and must be divided into a large number of sections, round off errors begin to produce erratic results. In these cases it is desirable to close out at each bearing thereby eliminating one unknown by the use of the constraint at that point.

Conclusions

To obtain an indication of the precision of the results, the problem is run in the forward and backwards direction. The IBM 650 machine time required to find a critical speed to within 0.1% is about 30 minutes. This method of solution is not restricted to rotating systems. With slight modifications, the same procedure may be applied to vibrations in jet engines, bridges and other structures.

Bibliography

1. "Holzer Method for Forced-Damped Torsional Vibrations" by T. W. Spaetgens, Journal of Applied Mechanics, Trans. ASME, Vol. 72, 1950, page 59.

2. "Some Vibration Aspects of Lubrication", by A. C. Hagg, Lubrication Engineering, Vol. 4, No. 4, 1948, pages 166-169.

3. "Vibration Problems in Large Electrical Machines" Midwest Power Conference, Proceedings, Vol. 13, 1951, pages 145-152.

650 PROCESSING OF MASS SPECTROMETER DATA

B. R. Faden
International Business Machines Corporation

At the IBM Data Processing Center in Los Angeles we are processing mass spectrometer data for the Union Oil Company of California. The program was written to the specifications furnished to us by Mr. W. C. Ferguson of the Union Oil Company's Research Center at Brea, California.

The application of the mass spectrometer with which we are concerned is the quantitative chemical analysis of mixtures of gases. The samples to be analyzed are known to be made up of certain compounds; the problem is to determine the proportion of each compound in the mixture.

In the mass spectrometer the compounds are broken up into ions, and the ions are subjected to an accelerating voltage and to a magnetic field. The path followed by an ion is determined by its mass, by the accelerating voltage, and by the applied magnetic field. In this application the instrument is focused for ions of a given mass by changing the accelerating voltage, for a given setting of accelerating voltage, ions of a given mass are collected at the plate of the instrument and give a current reading proportional to the number of ions of the given mass present in the sample. In spectroscopic practice this current is called peak height.

A procedure designed to accommodate n constituent compounds must record (at least) n peak heights corresponding to n ionic masses. Let n such ionic masses be chosen and identified by n numbers m_1, $m_2 \ldots m_n$, where the m_i are arranged sequentially according to the mass values they denote but need not be the mass values themselves, since as we shall see, the actual masses of the ions play no part in the computations.

The instrument is then calibrated for each compound as follows. The compound is introduced into the instrument at some particular pressure and readings are taken of the peak height for each m_i. The peak height is linearly proportional to the pressure, so that if the readings observed are divided by the pressure, we obtain normalized peak heights in milliamperes per micron. Let the normalized readings for the j'th compound be denoted by a_{ij}, that is, a_{ij} is the normalized peak height which the j'th substance gives at mass number m_i.

We have said that, for a given substance, the number of ions of a given mass is linearly proportional to the pressure. This is also assumed true of the partial pressure when the substance is mixed with others. Suppose now that we introduce a mixture into the instrument and denote the partial pressure of the j'th substance by y_j and denote the peak height observed for mass i by v_i. Then the law of partial pressures gives, as the equations for the amount of each substance present.

$$\sum a_{ij} y_j = v_i$$

and the percentage of the j'th substance present is, of course,

$$x_j = 100 \frac{y_j}{\sum y_k}$$

The quantity $\sum y_j$ should equal, within the limits of experimental error, the total pressure of the sample. The percentages x_j are the chief desired results of the processing.

Since a great many mixtures of the same set of compounds are to be analyzed, it is worthwhile to compute the inverse of the calibration matrix a_{ij}, and to determine the x_j by multiplying the peak height vector v_i by the inverse matrix.

The calibration matrix, a_{ij}, may be inverted with fully adequate precision by Gaussian Elimination, since it is very well conditioned. Many of the masses observed are chosen to be just the molecular weights of the constituent compounds, and many of these substances undergo comparatively little disassociation. Hence, by ordering the rows of the matrix by increasing masses and the columns by increasing molecular weight, we get a very strongly diagonal matrix. Furthermore, the matrix so arranged is very markedly upper triangular. The molecules undergo some disassociation but little or no, so to speak, agglutination, so there is little contribution to mass i from substances of molecular weight less than i, and hence there are almost all zeros to the left of the diagonal.

Since the need to invert a new calibration matrix occurs infrequently and since the matrix is well conditioned the matrix inversion part of the problem is very suitable for a library routine, and our practice is to perform the inversion using Mr. Dura Sweeney's routine, and convert the results to fixed point for incorporation into the program.

The rest of the work is not so well suited to a utilization of a vector by matrix multiplication library routine. The peak heights which form the multiplying vector are only four digit numbers. A library routine could not be expected to take full advantage of being able to do the multiplication with single precision fixed point arithmetic, and of the possibility of optimization afforded by the small maximum number of digits in the multiplier.

Since there are several special requirements to the problem, and since increased speed is quite an important factor when several hundred samples are to be processed, it seems well worth while to write a special program for the entire processing of the samples. We have presently in use two such programs, one for a twentieth order calibration matrix and one for a twenty-seventh.

The instrument in use at the Union Oil Company's Brea Research Center incorporates a Spectro-SADIC connected to an IBM punch, so that the data is directly recorded on punched cards and comes to us in that form, with one peak height per card, n cards per sample. The cards contain the mass identification number, m_i, the apparent net peak height or deflection, a multiplying or scaling factor, the observed total pressure of the sample, and the chemist number and sample number. The 650 program multiplies the apparent peak height by the scaling factor to obtain the observed peak height, v_i. The observed total pressure does not enter the computations but is punched on the 650 output cards and listed, for comparison with the calculated pressure.

The program counts and sequence checks the cards. If the cards are not in sequence by mass number, or if the number of cards per case is other than n, the program will skip over the calculation of percentages and punch a special error card, so that the listing will indicate that we were furnished an erroneous deck of cards for that case.

One of the special requirements of the program is the processing of what are called "check peaks". The nature of the check peak computations is as follows. Suppose that we have decided on n ionic masses at which to read the n peak heights, and that we solve the equations for these n peak heights to get the n partial pressures. However, suppose that we have also calibrated the instrument for the peak heights for one or two extra ionic masses, and suppose that we read also these extra peak heights or check peak heights when observing the samples.

Then the following check on the accuracy of the spectrometer run can be made. Multiply the solutions found for the partial pressures by the calibration coefficients for the check mass. Now if the sample is pure, that is, contains only the compounds calibrated for, and if these compounds are yielding ion currents in the same pattern and degree as they did under calibration conditions, then this multiplication will give exactly the peak height observed at the check mass. If there is an appreciable difference from the calibration conditions, then there will be a residual difference between the check peak height so calculated and the check peak height observed, and this residual is a measure of the general accuracy of the spectrometer run.

The calculation of this residual may be incorporated directly into the solution. Into the matrix to be inverted is incorporated the calibration equation for the check mass, and also incorporated is a column containing unity in the check equation row and zeros in all other rows. Then when this augmented system is solved, the solution corresponding to the incorporated check column is the desired residual.

Since the columns of the calibration matrix are labeled by the constituent compounds, it may be convenient to think of the check column as corresponding to a fictitious substance. The fictitious substance has the following mass spectrum: at the check mass, a normalized peak height of unity; at all other masses, a normalized peak height of zero. Then the solution found for this column may be thought of as the partial pressure of the fictitious substance. Since the fictitious substance gives unit peak height per unit pressure, its calculated partial pressure is equal to its contribution to the check peak height. Its contribution to the check peak height is a measure of how much of the peak height is not accounted for by the concentrations of the real substances, and hence is the desired residual peak height.

Two such check peaks are included in the n equations in each of the programs so far written. The branch on m equal to 02 or 17, which may be noted in the flow chart, is for the purpose of transferring to punching location the observed check peak heights, so that they may be listed and compared with the calculated check peak residuals. The observed peak heights at the other mass numbers are not punched in the results.

The instrumentation set up as presently used produces an extra card, identified by mass number 01, in addition to the n data cards. This card plays no part in the computations. The branch on m equals 01 shown on the flow chart causes the program to ignore the data on this card.

There are several special requirements in connection with the computation of total pressure. The partial pressures for the check peak substances do not enter this summation. It is desired to report the proportion of substances other than air and water as percentages of total non-air/water, non-check peak substances; and to report the proportion of air and water as percentages of all non-check peak substances. Because of inaccuracies in the experimental set up it is possible for the calculations to yield some negative partial pressures. These have no direct physical significance. A summation including the negative values is made, however, for comparison purposes. Accordingly, three summations are performed as follows: We first form

$Y_a = \Sigma y_j$, with j not equal to the check peak substances.

All negative partial pressures, are then replaced by zero, and we form,

$Y_b = \Sigma y_j$, with j not equal to the check peak substances, and

$Y_c = \Sigma y_j$, with j not equal to the check peak substances or water or air.

The percentages of water and air are then calculated as percentages of Y_b, and the percentages of the remaining non-check peak substances are calculated as percentages of Y_c.

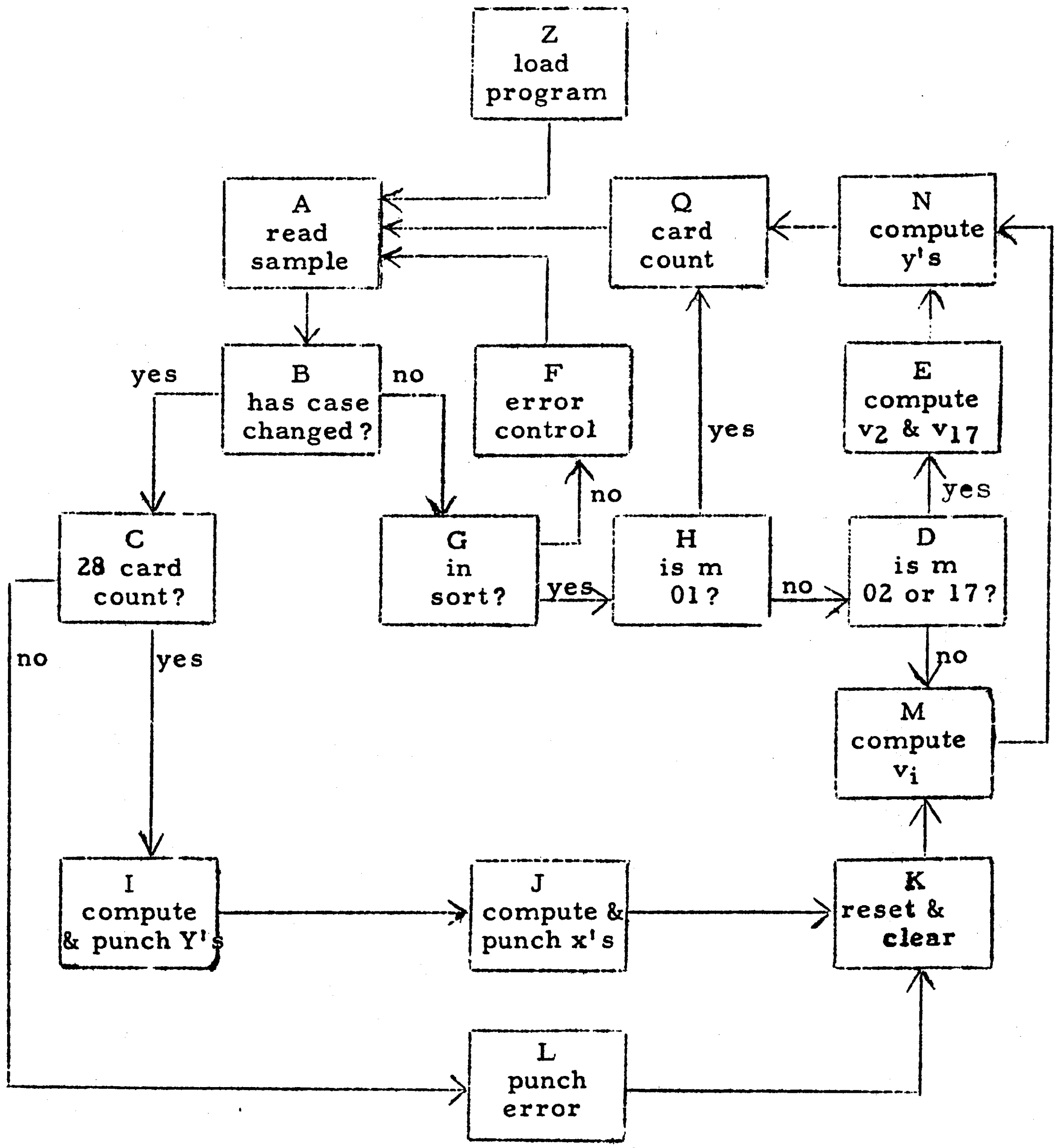
The output is punched as follows: On all cards: sample number and chemist number. On a header card: the observed pressure and the computed pressures Y_a and Y_b; and the observed check peak heights and the calculated check peak partial pressures (the check peak residuals). On detail cards: the calculated percentages. Our present punching and listing practice calls for spreading this output over one header card and five detail cards for both the 20'th and 27'th order set ups. An example of a typical 20'th order listing is attached to this write-up.

Since the elements of the observed peak height vector are presented one per card, the program computes the contribution to the twenty-seven partial pressures after each card is read and adds each contribution to the sum of previous contributions. This is block N of the flow chart, and is the main arithmetic work of the program. An address modification scheme is used to sweep the multiplication down the rows of one column for each card, and then to step from one column to the next on each feed cycle. Block N comprises about 170 instructions.

Block K, which occurs on the first card of a new case, resets the card count to zero, clears the locations in which the y_i are accumulated, and resets the address modification words to the values required to start multiplication at the first row and first column. Block F, which occurs only for a card out of sequence, adds a spurious large number to the card count, so that Block C will cause punching of an error card at the end of the case. It is hoped that the labeling of the remaining blocks on the flow chart is reasonably self-explanatory.

The program for the 27'th order set up comprises all told about 430 instructions and miscellaneous constants, plus the n^2 matrix elements, which are in a sense constants of the program. We take about five minutes to load the program and to run pre-computed test samples to check the loading. The calculations then proceed at approximately 200 samples an hour for the 20'th order set up, and 150 samples an hour for the 27'th order set up.

We like to think that these calculations provide a good example of the high degree of speed and economy which can be achieved when bulky repetitive calculations are routinized to take advantage of automatic digital computing. In conclusion we would like to express our thanks to Mr. W. C. Ferguson and the Union Oil Company of California for their cooperation in working out this procedure with us.


Z
load
program
A
read
sample
Q
card
count
N
compute
y's
B
has case
changed?
yes
no
F
error
control
yes
E
compute
v2 & v17
yes
no
C
28 card
count?
G
in
sort?
yes
H
is m
01?
no
D
is m
02 or 17?
no
no
yes
M
compute
vi
I
compute
& punch Y's
J
compute &
punch x's
K
reset &
clear
L
punch
error

911 45025 26/26/26 000/0235 013/0478

UNION OIL COMPANY OF CALIFORNIA

RESEARCH DEPARTMENT

MASS SPECTROMETER ANALYSIS

COMPONENT	MOL %	COMPONENT	MOL %	COMPONENT	MOL %	COMPONENT	MOL %
H_2	4•2%	CH_4	48•8%	N_2	1•1%	C_2H_6	30•6%
H_2S	0•0%	A	0•0%	CO_2	0•2%	C_3H_8	12•3%
iC_4H_{10}	1•3%	nC_4H_{10}	1•0%	iC_5H_{12}	0•4%	nC_5H_{12}	0•0%
SO_2	0•0%	Benzene	0•0%	Av. C_6	0•0%	Toluene	0•0%
H_2O	0•4%	Air	6•9%		99•9%		0•0%

THE NUMBERS ON THE TOP LINE OF THIS SAMPLE LISTING HAVE THE FOLLOWING MEANING.
CCC NNNNN P/YA/YB Y2/V2 Y17/V17
CCC IS THE CHEMIST NUMBER, NNNNN IS THE CASE NUMBER, P IS THE OBSERVED PRESSURE, YA AND YB ARE COMPUTED PRESSURES. THE REMAINING NUMBERS ARE THE CHECK PEAK INFORMATION.
THE MEANING OF THE COMPUTED PRESSURES AND OF THE CHECK PEAK NUMBERS IS EXPLAINED IN THE WRITE UP.
THE REMAINING NUMBERS ON THE PAGE ARE THE COMPUTED PERCENTAGES.
THE THIRD RESULT ON THE LAST PRINTED LINE IS THE SUM OF THE PERCENTAGES OF THE SUBSTANCES OTHER THAN WATER AND AIR.

CALCULATION OF LOAD STABILITY OF AN ELECTRICAL SYSTEM

J. E. Rowe
Union Carbide and Carbon Corporation

Introduction

In the operation of electrical power systems emergency situations, termed faults, may arise which require rapid automatic switching equipment to reroute power and/or drop part of the load to maintain system stability, that is, return to steady state operating conditions. The problem of transient load stability assumes a particularly important role when the system is composed of large blocks of induction motors operating close to the power limit such as those associated with AEC production facilities.

The problems encountered in stability studies of this type are varied and no attempt is made to give the engineering or developmental aspects other than those of a historical nature. The Atomic Energy Commission, Carbide and Carbon Chemicals Company, Westinghouse Electric Corporation, General Electric Company, and others have devoted considerable effort to the derivation of the equation representation of the electrical system. The Westinghouse Corporation in particular pioneered in the representation of the induction motor and the development of a computing technique (I).

The classical method of solving transient stability problems has been through the use of an AC Network Analyzer; however, due to the size of the electrical systems and nature of the associated problems there are definite advantages to the digital method. These advantages, computing experience, particularly that on the 650, and time and cost

(I) Shankle, D. F., et al, "Transient Stability Studies - I Synchronous Machines", Power Apparatus and Systems, 16, 1563-80, (February 1955).

comparisons between digital and network analyzer methods are given. In addition a particular digital scheme based on the representation of the power system by admittance constants and equation representation of machines and fault busses in the system is discussed. A fault will be considered as an abnormal voltage condition imposed simultaneously on all three phases at the fault bus.

Mathematical Statement of Problem

The voltage of each machine (II) is represented by a differential equation (III) which is dependent upon its characteristics and the system admittance constants.

$$(1)\quad \frac{dE_k(t)}{dt} = \left[\frac{-1}{\alpha_k} + j\ 120\pi s_k(t)\right] E_k(t) - j\frac{\beta_k}{\alpha_k}\ i_k(t),\ k = 1, ---, N,$$

where

t = time as measured from the occurance of the fault,

N = total number of machines in the power system,

k = machine index ($k = 1, ---, m$ represents induction motors, $k = m + 1, ---, N$ represents synchronous machines,

α_k = time constant for motors,

α_k = ∞ for synchronous machines,

β_k = reactance constant,

j^2 = -1.

The slip $s_k(t)$ and current $i_k(t)$ are defined by the following relations:

(II) A machine is used throughout to designate an induction motor, generator, or synchronous condenser.

(III) Concordia, C., "Transient Stability Studies - I Synchronous and Induction Machines - Discussion", Power Apparatus and Systems, 16, 1578-79, (February, 1955).

(2) $$ds_k(t)/dt = \gamma_k Ta_k(t),$$

(3.1) $$Ta_k(t) = \text{accelerating torque} = TL_k(t) - P_k(t),$$

(3.2) $$TL_k(t) = \text{load torque} = c_k \left[1 + s_k(t) \right] ,$$

(3.3) $$P_k(t) = \text{power} = \text{real component of } E_k(t)\ i_k^*(t),$$

and

(3.4) $$i_k(t) = \sum_{n=1}^{q} E_n(t)\ Y_{kn},$$

where

$k = 1, 2, ---, N,$

γ_k = constant dependent on machine inertia,

c_k = constant dependent on load characteristics

$i_k^* = i_k$ conjugate,

$Y_{kn} = Y_{nk}$ = equivalent system admittance constant measured between machine k and point n in the system,

$1 \leq n \leq N$ = machine index,

$N+1 \leq n \leq q$ = fault bus index.

Equation (3.4) introduces fault bus voltages $E_n(t)$ for $N+1 \leq n \leq q$. The equations for these voltages result from an application of Kirchhoff's law around the fault busses and take the form of linear simultaneous equations.

(4) $$\left\| \begin{matrix} Y_{N+1,\ N+1} & --- & Y_{N+1,q} \\ \vdots & & \vdots \\ Y_{q,\ N+1} & --- & Y_{q,q} \end{matrix} \right\| \left\| \begin{matrix} E_{N+1}(t) \\ \vdots \\ E_M(t) \end{matrix} \right\| = \left\| \begin{matrix} \sum_{n=1}^{N} E_n(t)\ Y_{N+1,n} \\ \vdots \\ \sum_{n=1}^{N} E_n(t)\ Y_{q,n} \end{matrix} \right\|$$

Equations (1), (2), and (3) are reducible by differentiation and substitution to a single system of second order non linear differential equations of order N. Although such a reduction results in a more concise mathematical statement, it obscures the approach to the solution. Accordingly, equations (1) through (4) provide an adequate mathematical description of the electrical system.

Approximating the Differential System

In order to obtain a numerical solution to equations (1) through (4), it is necessary to approximate the derivatives in equations (1) and (2). In the case of the induction motor, i.e., equation (1) for k = 1, ---,m, a first order forward difference approximation, $dE_k/dt \approx E_k(t+\Delta t)-E_k(t)\ /\Delta t$, is used, resulting in

$$(5.1)\qquad E_k(t+\Delta t) = \left[\frac{-1}{\alpha_k} + j120\pi s_k(t) + \frac{1}{\Delta t}\right]\Delta t\ E_k(t) - j\,\frac{\beta_k}{\alpha_k}\,\Delta t\ i_k(t).$$

In the case of synchronous machines, i.e., equation (1) for k=m+1,---,N, the representation

$$(5.2)\qquad E_k(t+\Delta t) = \left[\cos(120\pi\Delta\theta_k(t)) + j\,\sin(120\pi\Delta\theta_k(t))\right]E_k(t),$$

is used, where $\Delta\theta_k(t) \approx d\theta_k = s_k(t)dt$.

Approximating the differential $d\theta_k$ by the difference $\Delta\theta_k$ has an advantage over that used in (5.1) because it allows the use of a time increment Δt two to four times as large for the same accuracy. It has the additional advantage of maintaining a constant voltage magnitude, independent of the size of Δt.

The first order difference is used to approximate equation (2), resulting in

$$(6)\qquad s_k(t+\Delta t) = \gamma_k T a_k(t)\Delta t + s_k(t).$$

Equations (3) through (6) describe the transient behavior of a power system, with the accuracy of the approximation in equations (5) and (6) controlled with the choice of Δt.

System Data and Computing Procedure

Data for a study include the steady state voltage and slips for all machines, all constants, and system admittances. The admittance values and constants are not time dependent but are subject to change when the fault is applied or removed, when load is dropped in an effort to maintain stability, and when lines are reclosed to return this load to the system.

Equation (4) is solved repeatedly as t takes on its incremental values making it expedient to use matrix inverse methods. Predetermined faults and load droppings permit all admittance matrix inverse calculations to be made prior to a study; consequently these inverse elements become data rather than the Y's of equation (4).

The first step in the calculation is to solve for $E_k(t)$ from equation (4), then by use of equations (3), $i_k(t)$, $P_k(t)$, $TL_k(t)$, and $Ta_k(t)$ are evaluated in that order. These values are used to compute $s_k(t+\Delta t)$ and $E_k(t+\Delta t)$, $k = 1,---,N$, from equations (5) and (6). The newly calculated quantities $s_k(t+\Delta t)$ and $E_k(t+\Delta t)$ replace the old values, and the entire process is repeated. The calculations are continued until t = one second or less, since the system's behavior can be predicted on the basis of its behavior during this period. This procedure is summarized in the flow chart shown in figure 1.

Types of Problems and Programming Considerations

The analysis and determination of the design of a power system require a computing program which is adaptable to a wide variety of networks. Our experience has ranged from a theoretical system consisting of an infinite bus with one motor to a power network of ten equivalent induction motors, seventeen synchronous machines, and four fault busses representing a system load of 2200 MW. The latter involves approximately 2100 words of data consisting of 961 admittances and approximately 100 associated constants; all are vector quantities having both magnitude and direction.

These and the voltages and slips which must be retained from one iteration to the next, place unusual demands on the memory of the computer.

In order to reduce these memory requirements to the range of the 650 only the distinct nonzero admittances and their indices (kn) are stored. The small number of these and the symmetry of the system reduce the 2100 word requirement of the above problem to approximately 350. Since the nonzero admittances occupy different locations in the network, the table lookup feature of the 650 is used to locate the data. For example nonzero admittances Y_{kn} are stored in increasing order of kn. When the calculation calls for a given admittance, k is compared with n. If $k \leq n$ the machine searches for Y_{kn}. If $k > n$, the machine searches for Y_{nk}, thus taking advantage of symmetry. If the table search does not reveal this admittance, it is treated as zero and the calculation proceeds to the next instruction.

The forward facing methods result in equations (5.1) and (5.2). These were used rather than higher order approximations to minimize the quantities needed for each iteration. This representation also permitted a time increment of .02 seconds for most studies. Motor torque is the calculation most sensitive to the choice of Δt. This effect is illustrated by figure 6. Test calculations indicate that time increments of .01 seconds during the fault and .02 seconds after the fault determine the critical switching time with an accuracy of .01 seconds.

Having established the equations to be solved, calculating time is minimized by the following:

1. Pseudo optimum program.

2. The matrix of Y's in equations (4) are not functions of time; therefore, the matrix inverse is entered as basis data.

3. Addition of complex numbers can be done directly in rectangular form whereas in polar form they must be converted to rectangular, requiring the use of a square root and sine-cosine routine.

All decisions, such as changing line conditions, etc., are interpreted by the computer and no operating or card handling is necessary. Data which change in the course of a study are loaded initially in the read hopper in the order in which they are needed, and reading is controlled by the program.

Analysis of Results

The results of a single study for a large network consist of approximately 20,000 words. In order to analyze these data, the voltage, slip, power, torque, current, and identification associated with a machine at time t are punched into one card. The cards are sorted and listed in order by time under machine.

Stability is best judged from the motor accelerating torques or slips. Figures 2 through 5 are plots of these quantities; figures 2 and 3 indicating motor acceleration and return to equilibrium, and figures 4 and 5 indicating instability.

Comparisons With Other Methods of Solution

In solving stability problems on a network analyzer, it is necessary to spend considerable time setting up a network analogy on the calculating board, and should studies for the same power system be required at a later date, this setup procedure must be repeated. With the digital method, once data are assembled they can be stored on cards or tapes, and studies can be resumed at any future date with minimum effort. The network analyzer method requires a group of engineers supervising the study, taking data, and making analyzer settings. A procedure of this type necessarily introduces error, and is costly from a standpoint of personnel and machine time. The digital method can be carried on with one supervising engineer, one computer supervisor and at most one operator. The time per study is two-five days on the analogue computer, ten hours on the CPC and one hour and twenty minutes on the 650 (based on $\Delta t = .02$ seconds). Digital methods in general appear to have a cost advantage ranging from five to seven over the analogue method.

The 650 has distinct advantages over the CPC in that calculating time has been reduced by a factor of eight, card handling is virtually eliminated, the automatic features eliminate operator errors, machine errors do not go undetected, and costs per study have been reduced by a factor of four. In four months experience with the 650, thirty different stability problems requiring forty hours of machine time were solved. During this period there was evidence of only one error in calculation and it was detected by the computer.

Summary

A digital method for studying power network stability has maximum utility when the computer has a large high speed memory, rapid input-output, and logical orders which minimize data handling and operator decisions. Experience has indicated that such a method is faster, less expensive, more accurate, and more flexible than the analogue method. The program as described for the 650 can be used to study any network represented by admittance constants and equations (3) through (6).

Acknowledgement

The author is indebted to Mr. J. L. Gabbard, Jr., and Mr. P. E. Scott for their cooperation in supplying the engineering information and assembling data, and to Miss Barbara Boatman for her assistance in writing the 650 code.

FLOW CHART

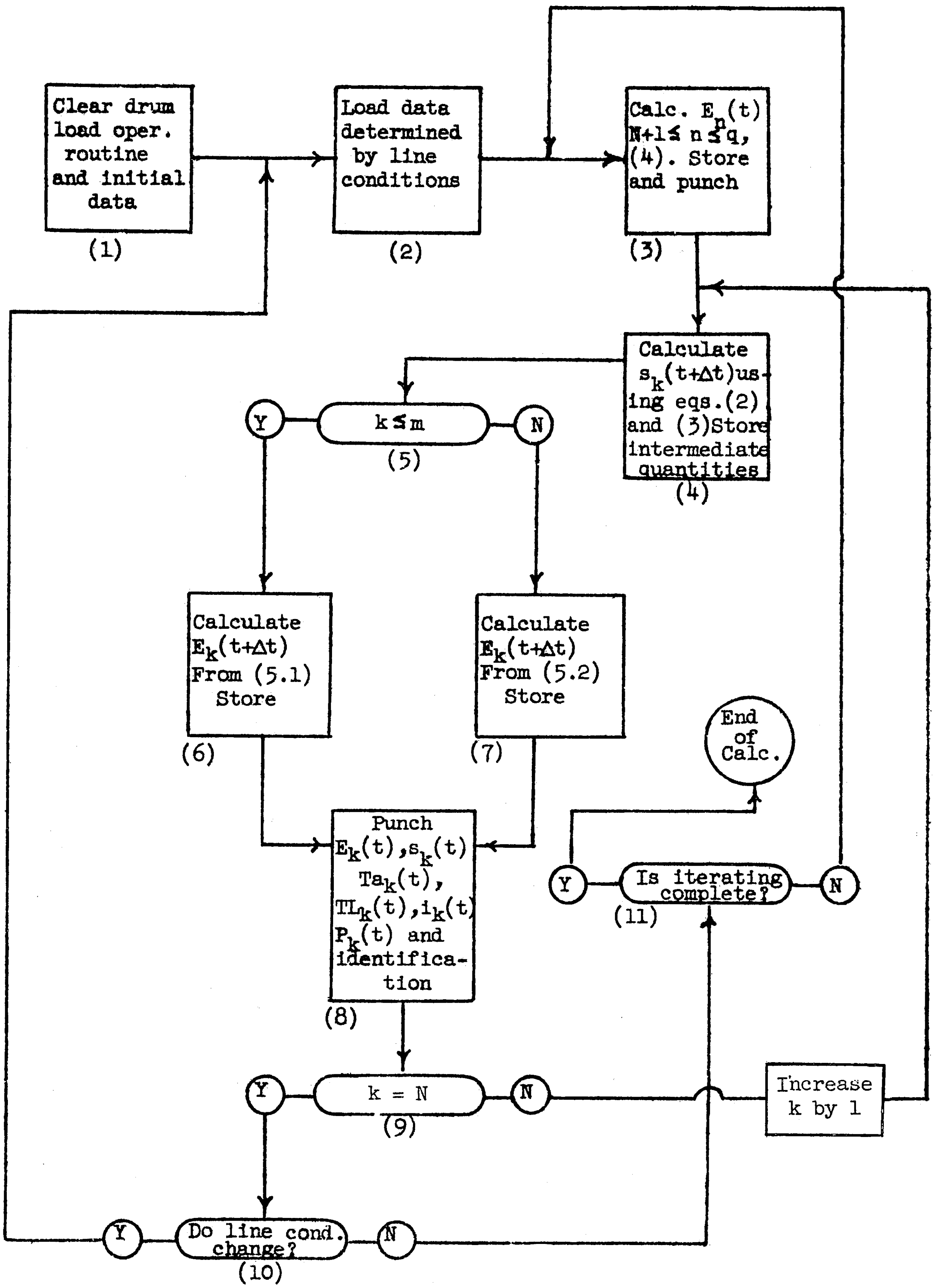
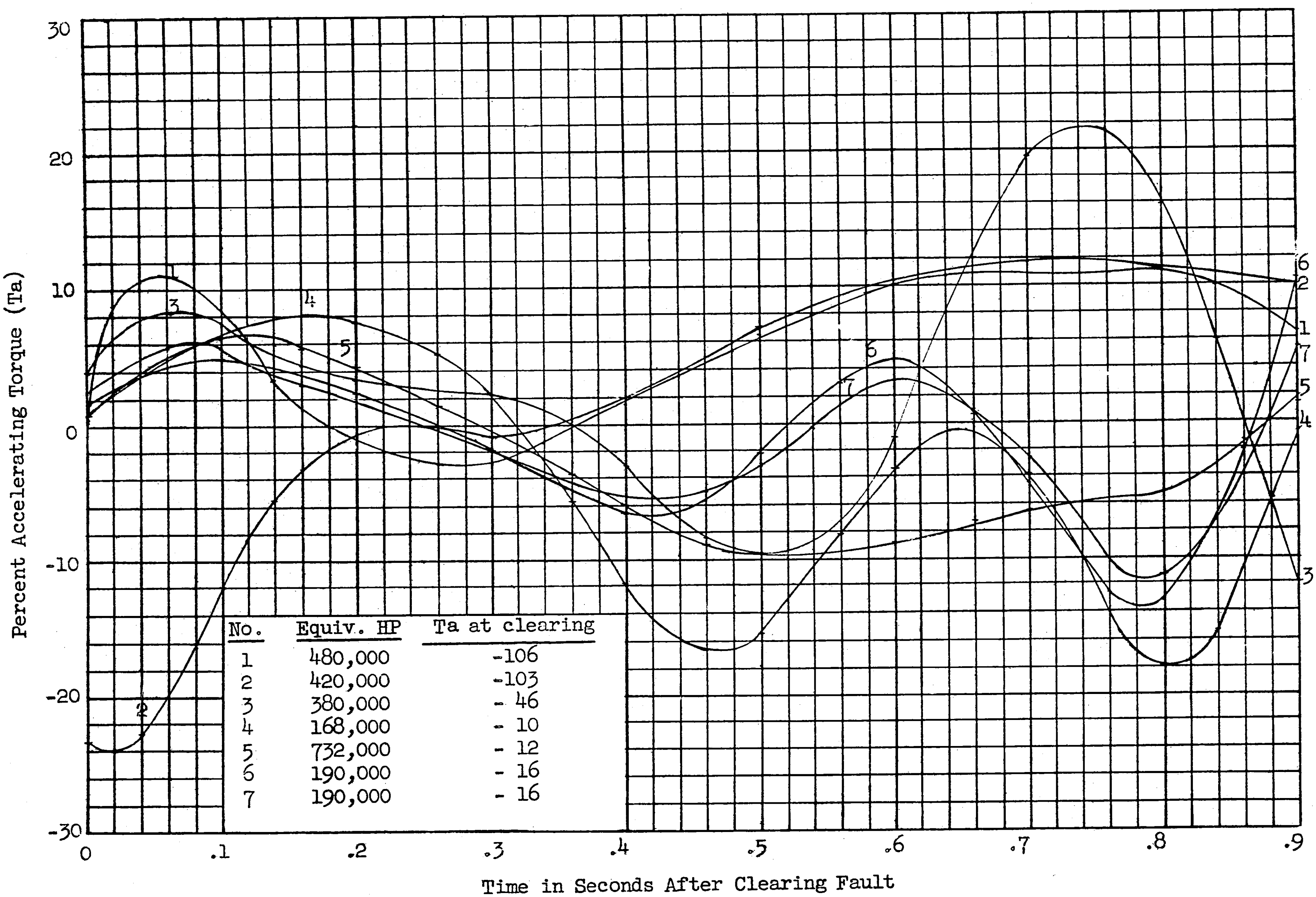


Figure 1

MOTOR ACCELERATING TORQUE VS. TIME - STABILITY INDICATED

(3 phase fault cleared after 0.08 seconds)

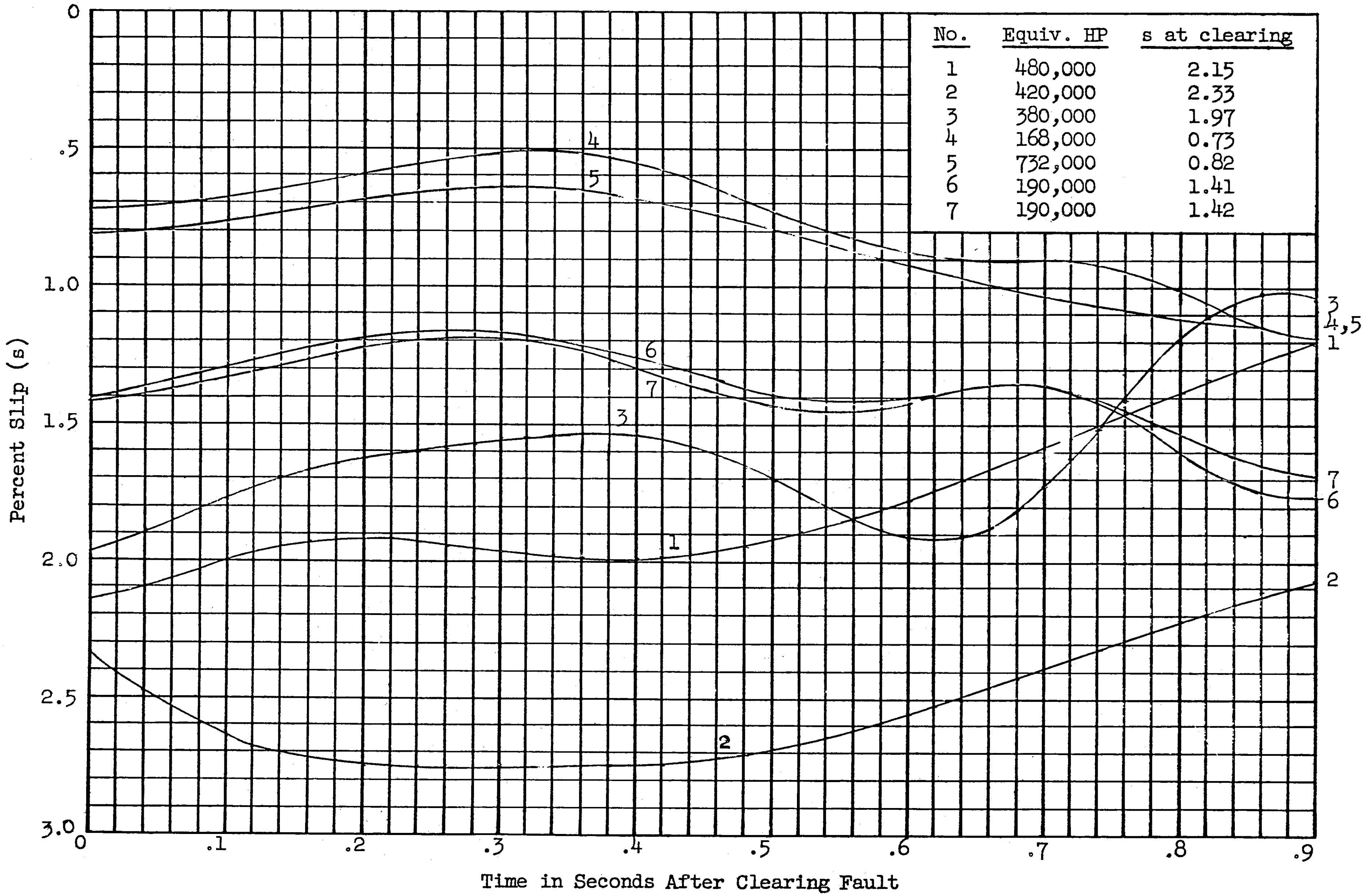


No.	Equiv. HP	Ta at clearing
1	480,000	-106
2	420,000	-103
3	380,000	- 46
4	168,000	- 10
5	732,000	- 12
6	190,000	- 16
7	190,000	- 16

Figure 2

MOTOR SLIP VS. TIME - STABILITY INDICATED

(3 phase fault cleared after 0.08 seconds)



No.	Equiv. HP	s at clearing
1	480,000	2.15
2	420,000	2.33
3	380,000	1.97
4	168,000	0.73
5	732,000	0.82
6	190,000	1.41
7	190,000	1.42

Figure 3

MOTOR ACCELERATING TORQUE VS. TIME - INSTABILITY INDICATED

(3 phase fault cleared after 0.11 seconds)

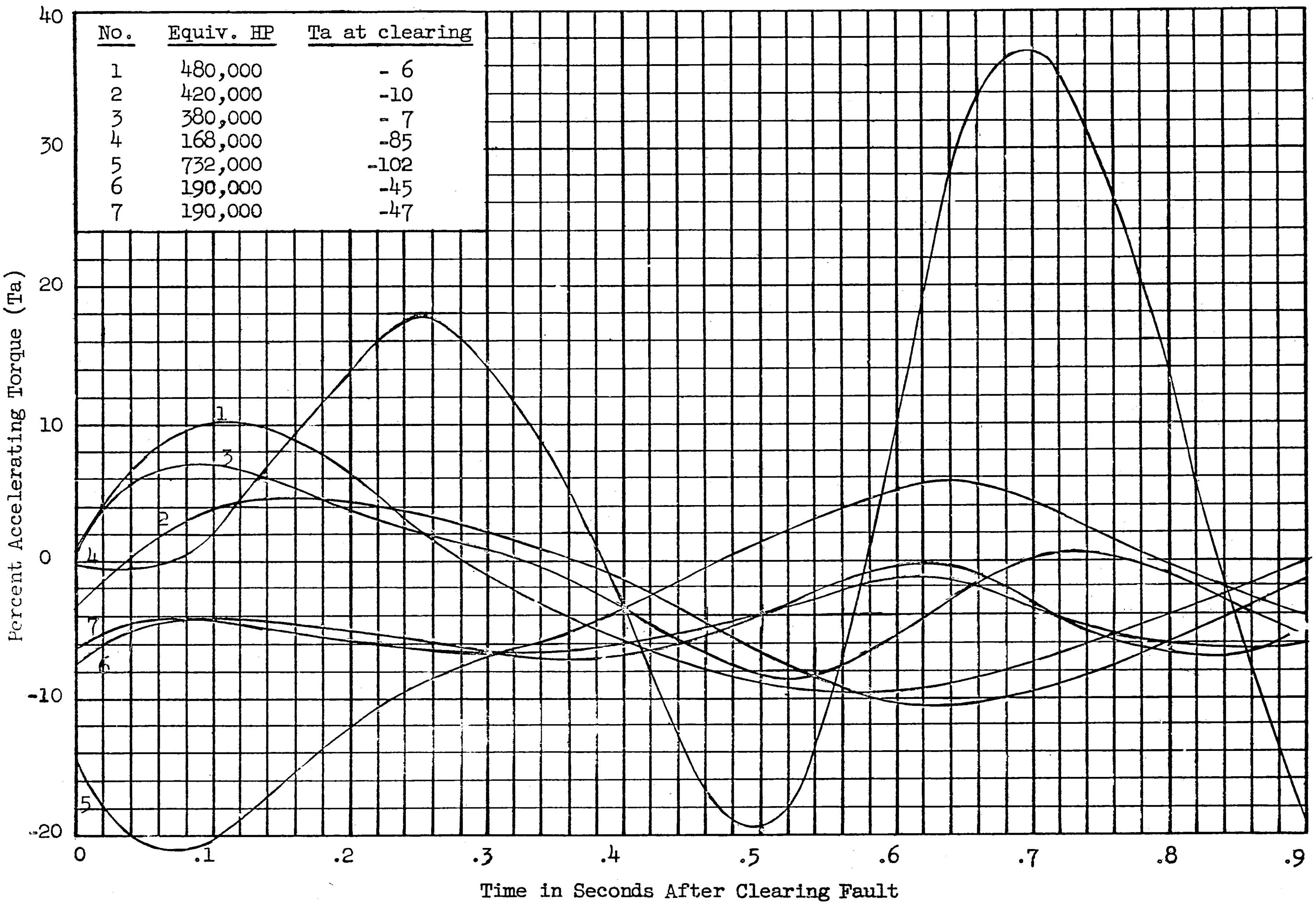
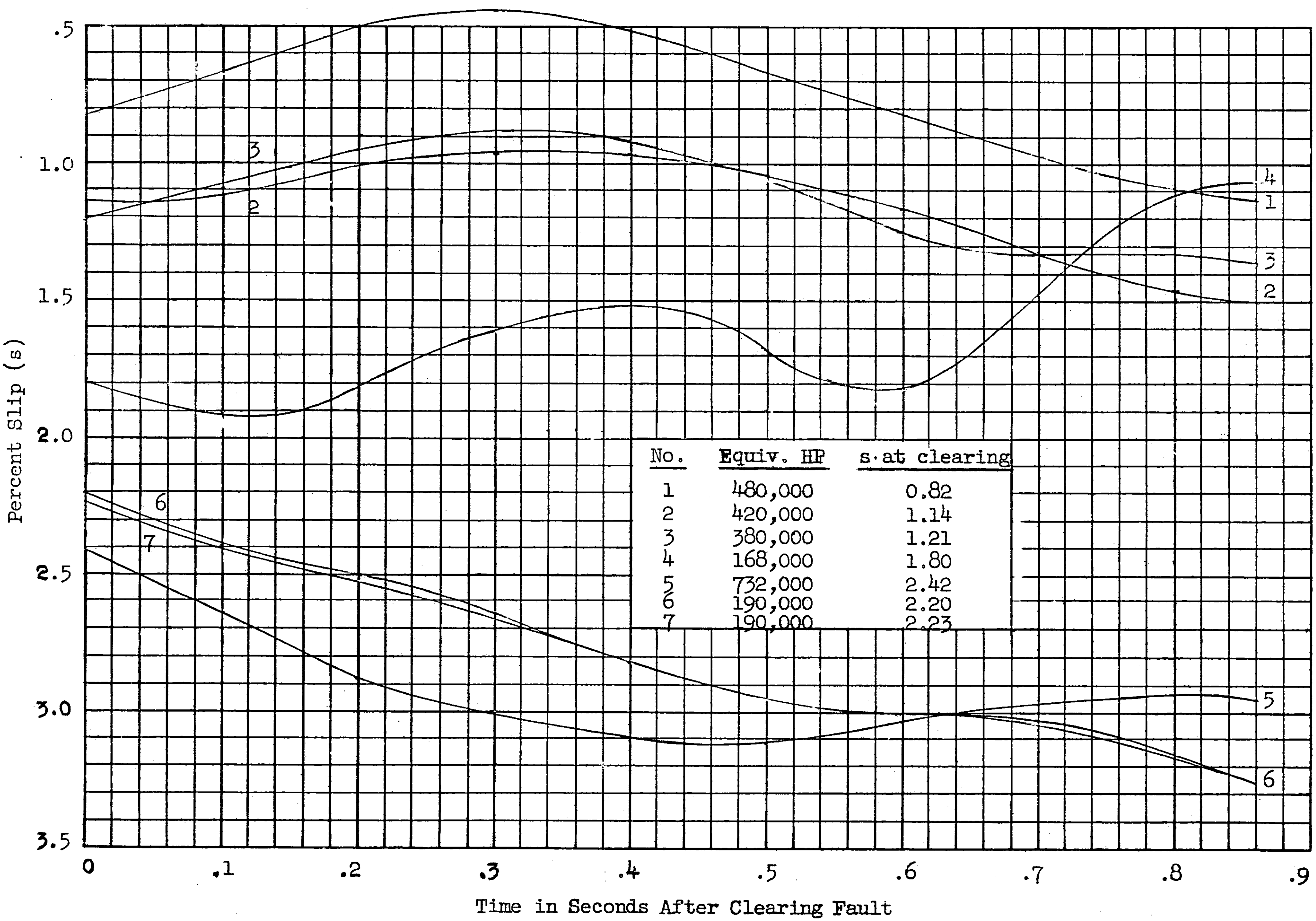


Figure 4

MOTOR SLIP VS. TIME - INSTABILITY INDICATED
(3 phase fault cleared after 0.11 seconds)



No.	Equiv. HP	s·at clearing
1	480,000	0.82
2	420,000	1.14
3	380,000	1.21
4	168,000	1.80
5	732,000	2.42
6	190,000	2.20
7	190,000	2.23

Figure 5

PERCENT ACCELERATING TORQUE FOR CRITICALLY SENSITIVE MOTOR VERSUS TIME

(3 phase fault cleared after 0.12 seconds)

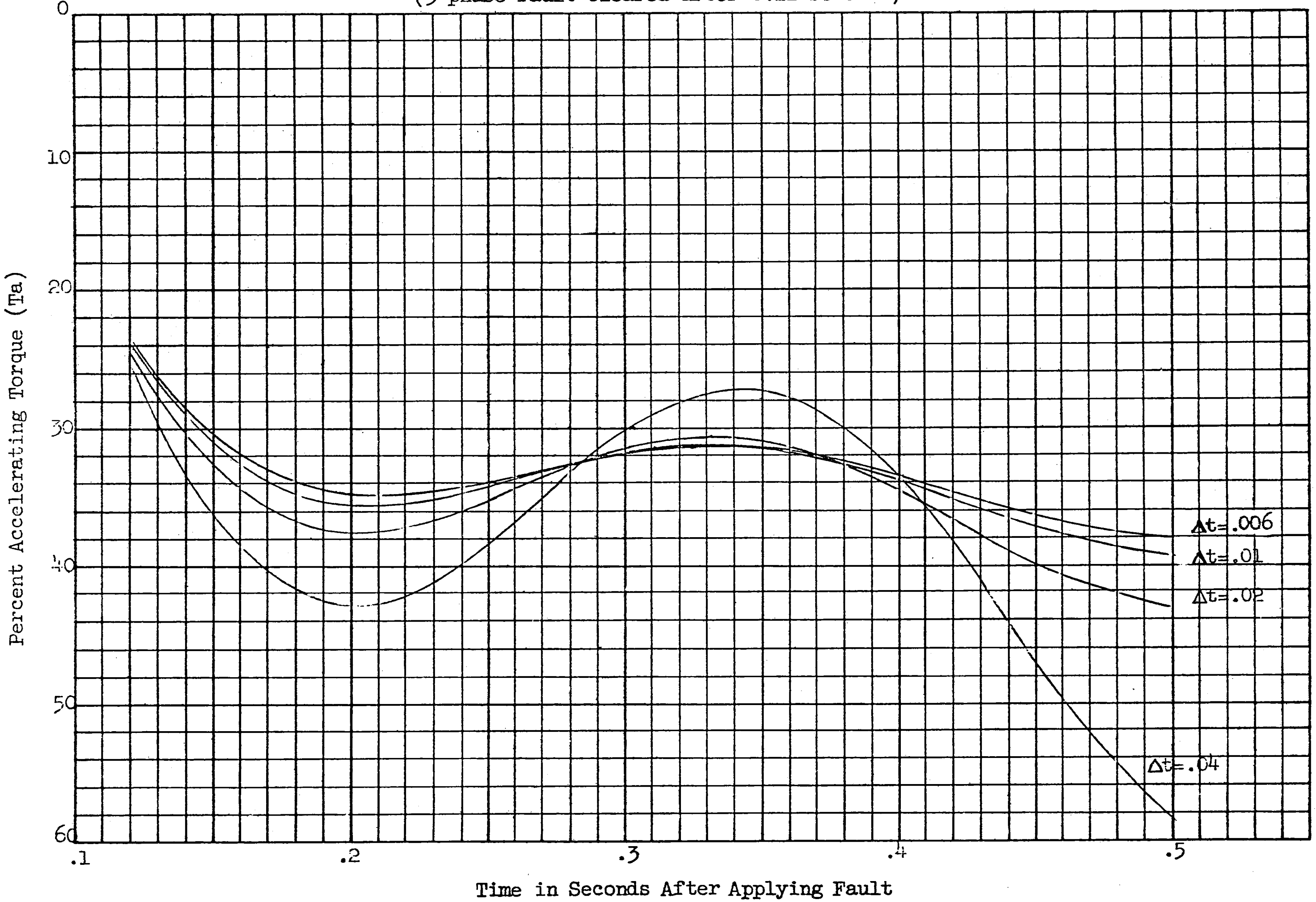
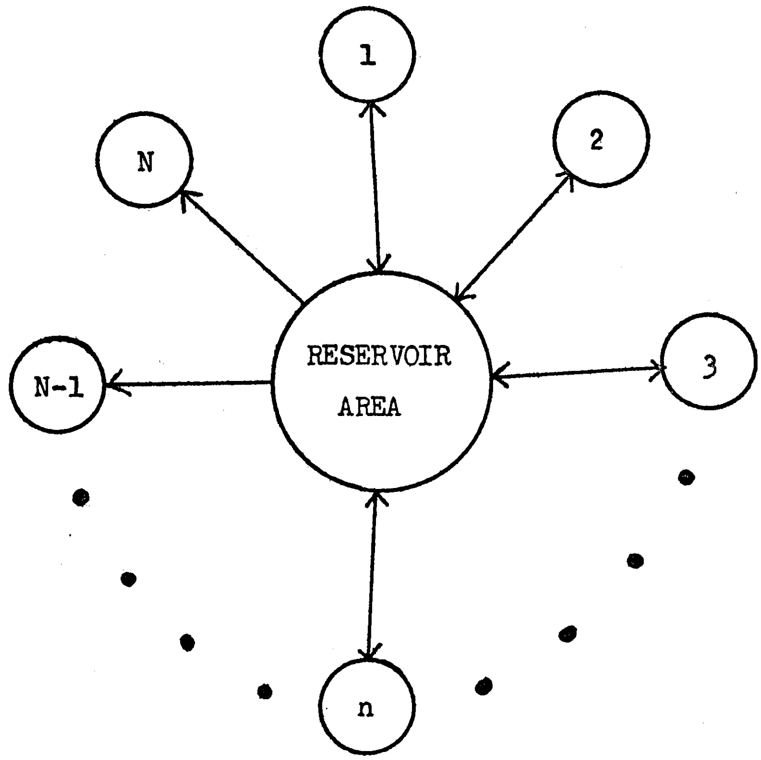


Figure 6

COMPUTATIONS OF UNIT COSTS IN POWER DISTRIBUTION

J. C. English
E. I. du Pont de Nemours and Company

At the Savannah River Plant there are N areas, all of which use electricity and n of which produce electricity. Those areas which produce no electricity consume electricity from the generating areas.



In the above diagram, Generating Areas 1, 2, ..., n may export excess electricity to the reservoir area or they may import electricity from it. This is indicated by the two-directional arrows. The non-generating areas must import their electricity from the reservoir area. This is represented by the unidirectional arrows. In any given month, all arrows are unidirectional.

The cost of generating electricity in any generating area will vary from month to month. The charge for excess electricity to the reservoir area by any excess generating area is the product of the quantity exchanged and the unit cost of generation for that area. The charge to the consuming areas by the reservoir area is the product of the quantity consumed by the area and the average cost per unit of electricity to the reservoir area.

The Accounting Department of SRP is concerned with the accounting for this electrical power. The proper cost codes[1] must be charged and the amount charged must be correct.

---

(1) Number designation to denote a group of people working on a particular problem.

The quantities of raw material and of electricity used are readily determined from meters and other measuring devices. Most overhead costs are also known. But, because of the interdependence of the areas through the reservoir area, the unit costs are not easily determined.

For each of the N areas there is a system of linear equations

$$\sum_j Q_{ij} U_j = X_i$$

expressing relationships among the various unit costs U_j for that area. The unit costs for an area may be the cost per unit of electricity generated, the cost per unit of electrical distribution, the cost per unit of river water, etc. The Q_{ij} are quantities of electricity generated, electricity distributed, river water used, etc. All these Q_{ij} are known since the quantities have been measured. The unit costs U_j are the unknowns which are desired. If all the X_i were known exactly, then the solution U_j could be obtained directly. The X_i's, which can be termed overhead, are not constant but are functions of the unit costs of electrical generation in the excess generating areas. The greater portion of the X_i's are salaries and other overhead items.

While the values of the X_i's are not known exactly, they may be estimated reasonably well when the estimate is based upon the preceding months' values for the unit cost of electrical generation.

If iterations are performed on the X_i's, improved U_j result. If the first guess for the X_i's is moderately good, then the U_i converge rapidly.

The logical procedure used in solving this problem is:

1. A guess is calculated for those X_i's which are functions of the unit costs of electrical generation. This calculation uses the values of the unit costs of electrical generation from past experience.

2. The systems of linear equations for those areas producing more electricity than they use are solved using the assumed X_i's from (1). This yields for each excess generating area an approximate unit cost of electrical generation for that area.

3. The new X_i's are calculated from these new approximations to the unit costs of electrical generation.

4. Again the system of linear equations for each excess generating area is solved. This gives a better approximation to the unit costs U_j for those areas.

5. From the new approximation to the unit costs of electrical generation, new X_i's are calculated. It has been found that these last approximations to the X_i's are sufficiently accurate, so that no further iterations are required.

6. At this point, every system of linear equations is solved with the last approximation to the X_i's. Not only the unit costs of

electrical generation, but all unit costs for each of the N areas are now considered. With these unit costs, the Accounting Department has all the information required to account accurately for the plant power.

7. As a check on the consistency of the data that has been processed, the X_i's for each area are summed. Then $\sum_j C_j U_j$ is computed , where $C_j = \sum_i Q_{ij}$ as determined by the Accounting Department from their original data. If $\sum_i X_i \neq \sum_j C_j U_j$, there must have been an error in the transcription of data.

8. The total cost for each area is computed by adding known quantities to $\sum_j C_j U_j$.

9. The total cost for all areas is obtained by summing costs for each each area. This total should equal the total Power Department expenditures plus inter-area transfers. (Sum of power exchanged with the reservoir area).

10. In the near future, the Computations Group will go on with the problem to show the distribution of power to the proper cost codes for each area.

Before this power accounting problem was calculated on the IBM 650, the Accounting Department maintained a suspense balance. The amount of this balance was the difference between the book value of electricity and the amount the Accounting Department had charged to the areas. Each month the previous month's suspense balance was charged or credited to the various cost codes and each month a new suspense balance was evolved to balance the books. The suspense balance is no longer a necessity.

This problem was done in the floating decimal mode of operation. The N systems of linear equations are solved using the floating decimal routine developed by G. R. Trimble, Jr. and E. C. Kubie of the IBM Corp., and the matrix inversion routine developed by R. W. DeSio also of the IBM Corp.

The computational procedure is outlined below.

The Accounting Department supplies the numerical values of the quantities of excess electricity produced by the areas that generate an excess. At the outset the machine tests to determine whether or not Generating Area 1 produced an excess of electricity. If it did, the machine solves the system of equations for Generating Area 1, and only the unit cost of electrical generation is maintained in storage. If the area does not produce excess, the routine goes on to the next area. The tests continue through the n generating areas after which the machine punches the unit costs of electrical generation for each area.

Next, the routine calculates the unit cost of electricity m distributed by the reservoir area. This unit cost is the weighted average of the cost of electricity from all the excess generating areas. The total cost of transmission in the reservoir area is also calculated. This cost is made up of salaries, material, and transmission losses. Finally, the unit cost of river water is calculated for the reservoir area.

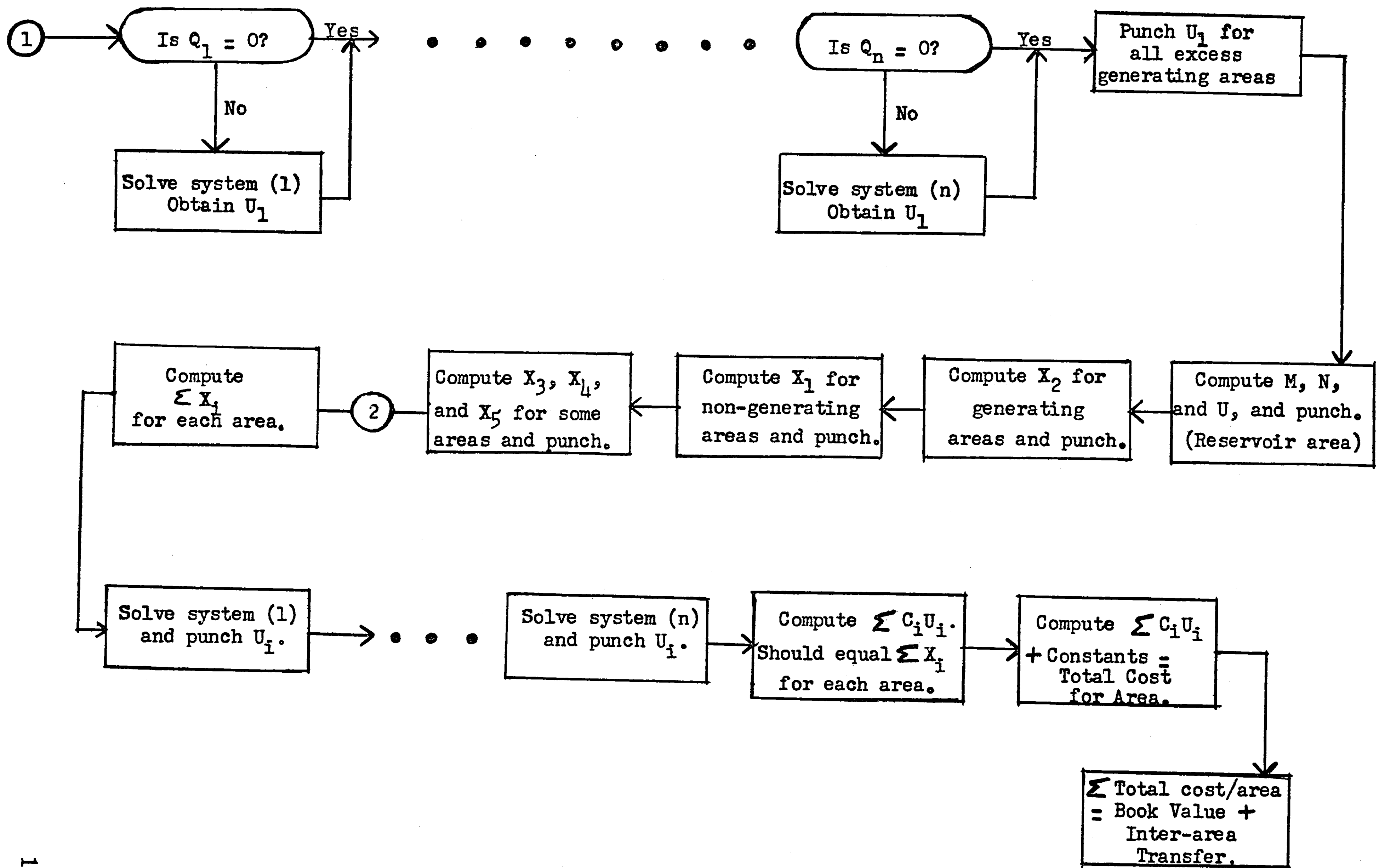
Now those X_i's which are functions of the unit costs of the reservoir area are calculated. This brings us to (2) in the Flow Sheet. With these new values for the X_i's, the routine is re-entered at (1). When the 650 has computed to point (2) again, it is allowed to proceed normally.

At this time, the X_i are calculated for each of the N regions, after which each of the N linear systems is solved for the unit costs. Each set of unit costs U_j is punched as it is calculated.

Originally, the Accounting Department prepared the input numbers in floating decimal form. Now they prepare the data with the decimal fixed, and a short routine transforms the numbers into floating decimal form.

This work was done under contract AT(07-2)-1 with the United States Atomic Energy Commission whose permission to publish is gratefully acknowledged.

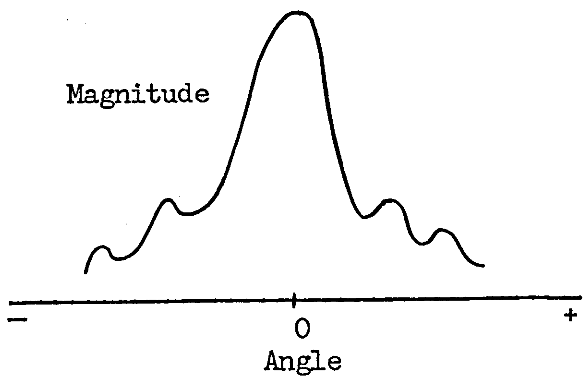
IBM 650 FLOW SHEET FOR POWER COST ACCOUNTING



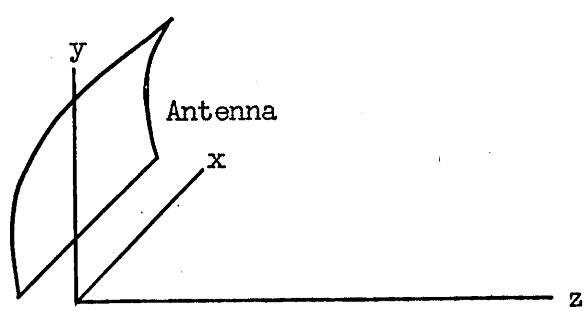
ANTENNA PATTERN CALCULATIONS

S. G. Fleming and R. Habermann, Jr.
General Electric Company

The calculation procedure discussed is a direct method of obtaining the far field antenna pattern from its aperture distribution. The shape of the far field pattern, particularly the gain and nature of the side lobes is of interest in many radar applications. Such patterns may be of the nature of the curve sketched below where the high central lobe and one or more of the small side lobes may be important. The magnitude is usually expressed in logrithmic units (decibels).



This pattern is obtained by integration of the Fourier integral of the aperture signal distribution. However, practical antenna systems do not generally yield expressions which may be directly integrated. Thus, many methods of solution using various measures for approximation have been proposed. A higher measure of precision may be obtained in the procedure described here as it involves only a direct numerical integration which may be done with as much care as is economically prudent.



An arbitrary aperture plane xy is established at the face of the antenna, and the linearly polarized signal phase and amplitude are expressed by:

$$F(x,y) = f(x,y)\ e^{j\psi(x,y)}$$

Generally the most important portion of the antenna far field pattern is that in a small angular region about the z axis. The far field pattern in this case is expressible as:

$$E(\theta,\phi) = \iint_S F(x,y)\ e^{j\ 2\pi/\lambda\ (x \cos \phi + y \sin \phi) \sin \theta}\ dxdy$$

where λ = wave length

It is sometimes convenient to normalize the situation to allow for comparison of related sizes of antenna of the same shape at different wave lengths. This also permits scale change in a single axis with a known effect on the pattern. The normalizing factors may be taken as the maximum dimension on each axis a and b. Thus, we establish new scales

$$X = \frac{x}{a} \qquad Y = \frac{y}{b}$$

Another substitution is made, in some cases, to reduce the trigonometric operations

$$u = \frac{2\pi a}{\lambda} \sin \theta \cos \phi$$

$$v = \frac{2\pi b}{\lambda} \sin \theta \sin \phi$$

The working form of the field expression is then

$$E(u,v) = a,b \iint_S f(X,Y)(\cos \Phi + j \sin \Phi)\ dXdY$$

where $\Phi = uX + vY + \psi(X,Y)$

The far field pattern is obtained by direct numerical integration of the preceding expression. The integration method used was the most elemental; dxdy being replaced by $\Delta x \Delta y$ and the integral becoming a double summation.

The interval size is determined by how fast the integrand changes its slope. The magnitude and phase functions, f and ψ, at the aperture are generally smooth. The rapid variation arises from the alternation of the sine and cosine terms due to the change in value of the uX and the vY terms. Since the integrand has a sine wave-type variation, the interval size is selected on the basis that 14 points per cycle give one percent error and 20 points per cycle give one-half percent error. A maximum error of one percent yields side lobes to 0.1 decibel. The result of the integration is the far field in its complex form.

$$E = g + jh$$

To make the results useful, additional calculations are performed to get the magnitude and phase of the field. It is desired to have it expressed as

$$E = Ae^{j\beta}$$

So, we must calculate

$$\text{db. equivalent of } A = 10 \log (g^2 + h^2)$$

$$\beta = \tan^{-1} \frac{h}{g}$$

We find this problem of particular interest as we have used several different computers over the years to handle various proposed antenna designs.

Experience on this problem started with the 602A. In this operation, the $f(X,Y)$ and $\psi(X,Y)$ were first punched into cards and many duplicate decks of cards were generated. From these the uX and vY products were calculated and Φ was formed for the various u and v values of interest. The sines and cosines were formed from sorting in a table deck and second-order interpolating.

The 604 came along and yielded vast speed-up of the operations. When machine modifications were completed, automatic sine and cosine calculations were used to reduce handling and sorting.

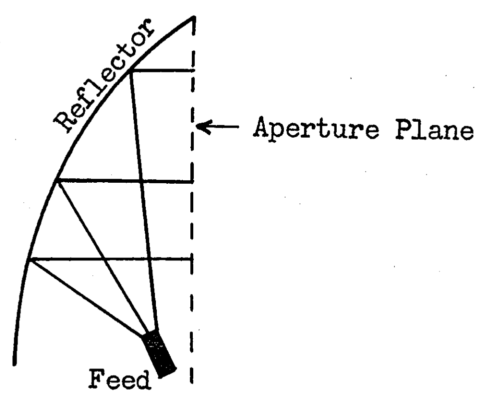
The advent of the CPC cut out the mountain of intermediate calculation result cards. Although the theoretical costs were higher than on the 604, the simplification of the project with the reduced incidence of operator intervention made this machine useful on the project. Again with this machine there was still a considerable amount of card handling as the input hopper did not hold all the instruction cards necessary.

The 650 has made this project fully automatic — the data and instructions are read in, and in due time the results are punched out.

Other 650 applications in the antenna design area have been the calculation of the field due to multiple antennas and the calculation of aperture field distribution.

With multiple antennas, superposition of the fields is done by adjusting the individual complex field values for the proper phase relationships and adding.

The aperture distribution may be calculated by the magnitude versus angle characteristic of the antenna feed. Various rays are established from the feed to the reflector, and thence to the aperture plane.



The magnitude at the aperture comes basically from the angle leaving the feed and the phase is determined by the path length. As small differences in the path length are large compared with the wave length, the calculation of reflectance angle and path length require care in programming to preserve accuracy. In eight-digit floating point, arithmetic results of sufficiently high precision were obtained.

Many of our projects in this field were initiated by Charles C. Allen of the General Engineering Laboratory of the General Electric Company. He has published some results of these calculations, and they appear in the 1953 Convention Record of the Institute of Radio Engineers.

CALCULATION OF PIPING SYSTEM EXPANSION STRESSES ON THE TYPE 650

Marilyn Alfieri, Burton Whipple, and Pierce O'Neill
General Dynamics Corporation

The basic theory needed to develop detailed calculation programs for the evaluation of piping flexibility analysis on the Type 650 Computer is contained in many sources of published literature. The methods of solution presented in the M. W. Kellogg Company manual (Reference 1) were used at Electric Boat for hand calculations and, logically, have been used in the programming for automatic computation.

METHOD OF SOLUTION

The Kellogg method is applicable to piping systems of any shape or configuration, in a single plane or in space. The approach is not restricted to the elementary problem of lines fixed at two ends, but can be used for any number or type of end fixations and intermediate constraints, such as guides, rollers, pivots, links, etc. Constant stiffness throughout the system is not a requirement; the line can be composed of pipes of various sizes, thicknesses and elastic properties. The development is not confined to a consideration of the deformations due to bending and torsion alone, but permits the inclusion of the effects of tension, compression and shear, where these are considered significant as in the case of very stiff lines. The computation program described herein does not include these direct force and shear effects but can be easily modified to do so,

The pipe ends or anchors and the intermediate constraints are termed load points. Complete fixation at a load point introduces three restraining forces acting in the directions of the axis of any orthogonal system of coordinates and an equal number of moments rotating about these axis. Since this equals the number of restraints required to maintain static equilibrium under any system of loading, six unknowns are implied for a line with two fixed ends and each additional load point will introduce six more unknowns. A lesser degree of fixation at a load point will reduce the number of unknowns at that point. For example, a universal joint type of hinge will prevent all translatory movement while not restraining any of the possible angular movements and, accordingly, provides three reactions.

While the scope of the Kellogg calculating method is unlimited theoretically, certain restrictions become apparent in programming a practical general solution for the 650. Branch selection becomes quite involved if more than three load points are considered since several varieties of line configuration are then possible. A routine capable of calculating piping systems with three anchors and no intermediate constraints or the equivalent case of two anchors and one constraint was general enough to satisfy our most pressing computing needs. Our initial effort has been so directed. Elaborations on this basic routine plus modification for special cases will subsequently be initiated.

The operating equations are established by considering one end of the piping system as fixed and located at the origin of the coordinate system. For a three anchor or branched line the origin is usually placed at the end of the common branch. If the other ends of the line are considered free of restraint against translatory and angular displacement the system becomes a cantilever beam fixed at the origin. When subjected to expansion the freed ends will be translated to new positions. In order to bring the freed end back into its original position (modified to include extraneous movements of the origin anchor and end anchor) certain forces and moments must be applied. The problem becomes that of finding the forces and moments necessary to apply at the load point in order to produce known deflections and rotations at the load point.

It is expedient to solve for the internal forces and moments produced at the origin by the external loading instead of solving for the latter directly. The contribution of each individual load point to the total moments and forces at the origin are shown separately to permit proper redistribution.

The equations for deflection and rotation at each load point as finally developed, are in terms of the unknown forces and moments at the origin with coefficients obtained by the summation of derived shaped coefficients reflecting the flexibility characteristics of each piece in the line and its location.

The solution of these simultaneous equations provides the internal moments and forces caused at the origin by each load point. From these the internal moments and forces at any point in the line may be determined and finally the stresses at the point.

CALCULATING PROCEDURE

The program for calculation on the Type 650 is broken down into the three major phases or "runs" of required calculation; computation of the equation coefficients, matrix solution of the simultaneous equations, and computation of stresses and reactions at required points in the line. For each problem to be solved a complete deck of properly arranged program and data cards for all three runs is processed in one continuous operation.

A detailed description of the required input data for each run is provided in Appendix I. All input data cards and output cards are similar. The first fifteen columns are assigned to the various identifications required. Numerical data is punched in the remainder of the card in either ten digit or five digit fields as required.

The program flow chart, provided as Appendix II, shows in detail the operations to be performed and the program selection required for each of the three runs. A brief description of the procedure for a three anchor or branch line follows:

RUN I - Calculation of Shape Coefficients

In setting up the piping system for calculation, a tri-axial coordinate system is established with the origin at the common or "C" Branch anchor. All members, or parts of the line, are assigned to planes parallel to the coordinate planes or in planes which may be rotated about a coordinate axis to achieve this condition. The necessary delineating data for each member together with factors reflecting relative stiffness and flexibility is punched in a card.

The twenty-one shape coefficients of the member, either straight piece or elbow, are calculated and the results stored in memory locations determined by the plane of the member. If no plane rotation is required the results are transferred to the memory locations where the summation of coefficients is accomplished. When rotation is required the results go through the selected rotating routine before being transferred.

When the card for the last member in the line has been calculated we have stored in memory the elements above the diagonal of a symmetrical square matrix of order twelve representing, when augmented by a column vector of constant terms, our set of twelve simultaneous linear equations. In Run II the constant terms are provided and the matrix solution is accomplished.

Approximately 1650 instructions are required in the programming of this Run.

RUN II - Matrix Solution

The constants for our simultaneous equations are derived from the "restoring" rotations and deflections of the "A" branch and "B" branch ends of the line. These "restoring" movements are determined from the thermal expansion of the line and any extraneous movements occurring at the branch end or at the origin. Each constant is the product of the pipe stiffness, EI, and a rotation or deflection.

Each different combination of line temperature and anchor movement establishes an operating condition of the line. An "A" branch and a "B" branch data card provide the set of equation constants for each such condition. The internal pressure for the operating condition, required in the calculations of Run III, is also read in and stored. A maximum of seven different conditions can be treated for solution at the same time. The limitation here is that of drum memory capacity.

A method of solution for simultaneous linear equations with the same matrix of coefficients but different constant terms is described by Eric V. Hankam, (Reference 2). The matrix (A) resulting from Run I is augmented by the several column vectors (b) of constant terms. A composite matrix is then formed by adding an identity matrix (I) under matrix A and adding O-vectors under each b-vector. Reduction of this composite matrix results in the values of the unknowns appearing in the O-vector below the respective b-vector.

Due to the wide range of values expressing the equation coefficients we convert to a floating decimal point number system for the matrix reduction routine. The largest composite matrix contains twenty-four rows and nineteen columns. We take advantage of the memory address system on the 650 by storing the matrix elements in drum bands 01 to 19. The mantissa of the first element is stored in 0101 with the exponent in 0151 and the last element has the mantissa in 1924 with exponent in 1974. The row and column location is thus easily defined. Note, however, that the address indicates the column first and then the row. This is the opposite of usual matrix element notation.

After the matrix solution has been performed the results are converted back to fixed decimal numbers.

For each condition we now have the moments and forces at the origin due to the "A" branch loads and the same for the "B" branch loads. These are stored for use in Run III and also punched out on cards identified as to condition and branch. Corresponding moments and forces for the two branches are summed for use in Run III as common or "C" branch reactions.

The program for this run requires about 500 instructions.

RUN III - Stress Calculation

A data card, for each point in the line at which stress is to be evaluated, provides the necessary coordinate and dimensional information. The moments at the point are obtained from the laws of statics considering that part of the line which is located between the origin and the point investigated. The forces remain the same throughout the system since intermediate constraints are not being considered.

The bending moment in plane of bend, bending moment normal to plane, and torsional moment are then calculated according to the plane of the member containing the point. The stresses due to these moments are computed and punched out.

From these stresses and the calculated longitudinal pressure stress the combined stress following the Principle-Stress Theory (Rankine) is obtained and punched out.

This routine is repeated for each operating condition of the line before feeding the data card for the next point.

Approximately 350 instructions have been used in the program for this Run.

REFERENCES

1. "Expansion Stresses and Reactions in Piping Systems", Anon., M. W. Kellogg Company, (pub.), Jersey City, N.J., 1941 (Now out of print, revised edition in preparation)

2. "Linear Equations and Matrix Inversion", Eric V. Hankam, IBM Technical Newsletter No. 3, December 1951, pp. 26-34.

APPENDIX I

INPUT INFORMATION FOR EACH RUN IN PROGRAM

In addition to specific data noted below all data cards are code punched for their particular run and for job identification.

Run I - Calculation of Shape Coefficients.

Each member in the line has one input data card providing the following information. For straight members K = 1, φ = 1, and R = L (length).

DIGITS		DESCRIPTION
XX	1 to 99	- Piece number of member
X	X, Y, or Z	- Plane of member
X	X, Y, or Z	- Axis of rotation of plane
X	S or E	- Type of member, straight or elbow
X	A, B, or C	- Branch containing member
XX.XXX	K	- Flexibility factor for curved member
XX.XXX	Q	- Relative stiffness factor
XX.XXX	R	- Bend radius of elbow or length of straight
XX.XXX (±)	a, b	- Coordinates of member in plane
XX.XXX (±)	c	- Normal coordinate of plane
X.XXXX	α	- Angle of Tangent to member
X.XXXX	φ	- Arc of curved members
X.XXXX	γ	- Angle of plane rotation

Run II - Matrix Solution

For each different combination of temperature, anchor movements and pressure which establishes an operating condition of the line one input data card for the "A" branch and one for the "B" branch provides the following information.

DIGITS		DESCRIPTION
XX,XXX.	P_N	- Operating pressure
X,XXX,XXX,XXX. (±)	$\emptyset X_A$	- Rotation equation constant, "A" branch
X,XXX,XXX,XXX. (±)	$\emptyset Y_A$	- Rotation equation constant, "A" branch
X,XXX,XXX,XXX. (±)	$\emptyset Z_A$	- Rotation equation constant, "A" branch
X,XXX,XXX,XXX. (±)	ΔX_A	- Deflection equation constant, "A" branch
X,XXX,XXX,XXX. (±)	ΔY_A	- Deflection equation constant, "A" branch
X,XXX,XXX,XXX. (±)	ΔZ_A	- Deflection equation constant, "A" branch
X,XXX,XXX,XXX. (±)	$\emptyset X_B$	- Rotation equation constant, "B" branch
X,XXX,XXX,XXX. (±)	$\emptyset Y_B$	- Rotation equation constant, "B" branch
X,XXX,XXX,XXX. (±)	$\emptyset Z_B$	- Rotation equation constant, "B" branch
X,XXX,XXX,XXX. (±)	ΔX_B	- Deflection equation constant, "B" branch
X,XXX,XXX,XXX. (±)	ΔY_B	- Deflection equation constant, "B" branch
X,XXX,XXX,XXX. (±)	ΔZ_B	- Deflection equation constant, "B" branch

Run III - Stress Calculation

For each point in line at which moments and stresses are to be calculated one input data card provides the following information.

DIGITS		DESCRIPTION
XX	1 to 99	- Point number
X	X, Y, or Z	- Plane of member containing point
X	A, B, or C	- Branch containing point
XX.XXX (±)	x, y, z	- Coordinates of point
X.XXXX	α	- Angle of tangent to point
XXX.XX	SM	- Section modulus of pipe
XX.XXX	OD	- Outside diameter of pipe
XX.XXX	t	- Wall thickness of pipe
XX.XXX	β	- Curved pipe stress intensification factor

APPENDIX II

FLOW CHART FOR PROGRAM

Run I - Calculation of Shape Coefficients

(a) Load program for Run I

(b) Feed data card containing piece number, plane, axis of rotation, type of member, branch containing member and K, Q, R, a, b, c, α, φ, γ.

Feed Data Card

1. $\sin \alpha$
2. $\cos \alpha$
3. 2α
4. $\sin 2\alpha$

Straight		Elbow	
5.	$0 = C_a$	5.	$\cos 2\alpha$
6.	$0 = C_b$	6.	$\alpha + \varphi$
7.	$\frac{\sin 2\alpha}{24} = C_{ab}$	7.	$\sin(\alpha + \varphi)$
8.	$\cos^2\alpha$	8.	$\cos(\alpha + \varphi)$
9.	$\sin^2\alpha$	9.	$2(\alpha + \varphi)$
10.	$\frac{\cos^2\alpha}{12} = C_{aa}$	10.	$\sin 2(\alpha + \varphi)$
11.	$\frac{\sin^2\alpha}{12} = C_{bb}$	11.	$\cos 2(\alpha + \varphi)$
		12.	$\cos\alpha - \cos(\alpha + \varphi) = C_a$
		13.	$\sin\alpha - \sin(\alpha + \varphi) = C_b$

14. $1/4\left[\cos 2\ (\alpha + \varphi) - \cos 2\alpha\right] = C_{ab}$

15. $1/4\left[\sin 2\ (\alpha + \varphi) - \sin 2\alpha\right] = G$

16. $\varphi/2 - G = C_{aa}$

17. $\varphi/2 + G = C_{bb}$

18.	$0 = S'_a$	18.	$KQR^2C_a = S'_a$
19.	$0 = S'_b$	19.	$KQR^2C_b = S'_b$
20.	$QL^3C_{ab} = S'_{ab}$	20.	$KQR^3C_{ab} = S'_{ab}$
21.	$QL^3C_{aa} = S'_{aa}$	21.	$KQR^3C_{aa} = S'_{aa}$
22.	$QL^3C_{bb} = S'_{bb}$	22.	$KQR^3C_{bb} = S'_{bb}$
23.	$0 = u'_o$	23.	$1.3\ QR^2C_a = u'_o$
24.	$0 = v'_o$	24.	$1.3\ QR^2C_b = v'_o$
25.	$QL^3/12 = w'_o$	25.	$1.3\ QR^3\varphi = w'_o$
26.	$QL(1 + 3.6\ C_{bb}) = u$	26.	$QR(KC_{bb} + 1.3\ C_{aa}) = u$
27.	$QL(1 + 3.6\ C_{aa}) = v$	27.	$QR(KC_{aa} + 1.3\ C_{bb}) = v$
28.	$QL\ 3.6\ C_{ab} = w$	28.	$QR\ C_{ab}(K - 1.3) = w$
29.	$QL = S$	29.	$KQR\varphi = S$

30. $S \cdot a + S'_a = S_a$

31. $S \cdot b + S'_b = S_b$

32. $S \cdot ab + S'_a \cdot b + S'_b \cdot a + S'_{ab} = S_{ab}$

33. $S \cdot a^2 + 2\ S'_a \cdot a + S'_{aa} = S_{aa}$

34. $S \cdot b^2 + 2\ S'_b \cdot b + S'_{bb} = S_{bb}$

35. $u \cdot a - w \cdot b + u'_o = u_o$

36. $v \cdot b - w \cdot a + u'_o = v_o$

37. $u \cdot a^2 + v \cdot b^2 - 2w \cdot a \cdot b + 2u'_o \cdot a + 2v'_o \cdot b + w'_o = w_o$

38. $cu = cu$

39. cv = cv

40. cw = cw

41. $cu_o = cu_o$

42. $cv_o = cv_o$

43. $c^2u = c^2u$

44. $c^2v = c^2v$

45. $c^2w = c^2w$

46. $\sin \gamma$

47. $\cos \gamma$

48. $\sin^2\gamma$

49. $\cos^2\gamma$

50. 2γ

51. $\sin 2\gamma$

52. $\cos 2\gamma$

53. $1/2 \sin 2\gamma$

	X Plane	Y Plane	Z Plane
A'_{xx}	= s	= u	= v
A'_{xy}	= o	= o	= w
A'_{xz}	= o	= w	= o
B'_{xx}	= o	= cw	= - cw
B'_{xy}	= s_b	= u_o	= cv
B'_{xz}	= $-s_a$	= -cv	= $-v_o$
A'_{yy}	= v	= s	= u
A'_{yz}	= w	= o	= o
B'_{yx}	= $-v_o$	= $-s_a$	= -cu
B'_{yy}	= -cw	= o	= cw
B'_{yz}	= cv	= s_b	= u_o
A'_{zz}	= u	= v	= s

	Rot. @ X	Rot. @ Y	Rot. @ Z
B'_{zx}	$= u_o$	$= cv$	$= s_b$
B'_{zy}	$= -cu$	$= -v_o$	$= -s_a$
B'_{zz}	$= cw$	$= -cw$	$= o$
C'_{xx}	$= w_o$	$= s_{aa}+c^2v$	$= S_{bb}+c^2u$
C'_{xy}	$= -cu_o$	$= -cv_o$	$= -(s_{ab}+c_2w)$
C'_{xz}	$= -cv_o$	$= -(s_{ab}+c^2w)$	$= -cu_o$
C'_{yy}	$= s_{bb}+c^2u$	$= w_o$	$= s_{aa}+c^2v$
C'_{yz}	$= -(s_{ab}+c^2w)$	$= -cu_o$	$= -cv_o$
C'_{zz}	$= s_{aa}+c^2v$	$= s_{bb}+c^2u$	$= w_o$

For Rotation About X-Axis

1. $A'_{xx} = A_{xx}$
2. $A'_{xy} \cos\gamma - A'_{xz} \sin\gamma = A_{xy}$
3. $A'_{xz} \cos\gamma + A'_{xy} \sin\gamma = A_{xz}$
4. $B'_{xx} = B_{xx}$
5. $B'_{xy} \cos\gamma - B'_{xz} \sin\gamma = B_{xy}$
6. $B'_{xz} \cos\gamma + B'_{xy} \sin\gamma = B_{xz}$
7. $A'_{yy} \cos^2\gamma + A'_{zz} \sin^2\gamma - A'_{yz} \sin 2\gamma = A_{yy}$
8. $A'_{yz} \cos 2\gamma + (A'_{yy} - A'_{zz})\ 1/2 \sin 2\gamma = A_{yz}$
9. $B'_{yx} \cos\gamma - B'_{zx} \sin\gamma = B_{yx}$
10. $B'_{yy} \cos^2\gamma + B'_{zz} \sin^2\gamma - (B'_{yz} + B'_{zy})\ 1/2 \sin 2\gamma = B_{yy}$
11. $B'_{yz} \cos^2\gamma - B'_{zy} \sin^2\gamma + (B'_{yy} - B'_{zz})\ 1/2 \sin 2\gamma = B_{yz}$
12. $A'_{zz} \cos^2\gamma + A'_{yy} \sin^2\gamma + A'_{yz} \sin 2\gamma = A_{zz}$

13. $B'_{zx} \cos \gamma + B_{yx} \sin \gamma = B_{zx}$

14. $B'_{zy} \cos^2\gamma - B'_{yz} \sin^2\gamma + (B'_{yy} - B'_{zz})\ 1/2 \sin 2\gamma = B_{zy}$

15. $B'_{zz} \cos^2\gamma + B'_{yy} \sin^2\gamma + (B'_{zy} + B'_{yz})\ 1/2 \sin 2\gamma = B_{zz}$

16. $C'_{xx} = C_{xx}$

17. $C'_{xy} \cos \gamma - C'_{xz} \sin \gamma = C_{xy}$

18. $C'_{xz} \cos \gamma + C'_{xy} \sin \gamma = C_{xz}$

19. $C'_{yy} \cos^2\gamma + C'_{zz} \sin^2\gamma - C'_{yz} \sin 2\gamma = C_{yy}$

20. $C'_{yz} \cos 2\gamma + (C'_{yy} - C'_{zz})\ 1/2 \sin 2\gamma = C_{yz}$

21. $C'_{zz} \cos^2\gamma + C'_{yy} \sin^2\gamma + C'_{yz} \sin 2\gamma = C_{zz}$

For Rotation about Y-Axis

1. $A'_{xx} \cos^2\gamma + A'_{zz} \sin^2\gamma + A'_{xz} \sin 2\gamma = A_{xx}$

2. $A'_{xy} \cos \gamma + A'_{yz} \sin \gamma = A_{xy}$

3. $A'_{xz} \cos 2\gamma + (A'_{zz} - A'_{xx})\ 1/2 \sin 2\gamma = A_{xz}$

4. $B'_{xx} \cos^2\gamma + B'_{zz} \sin^2\gamma + (B'_{xz} + B'_{zx})\ 1/2 \sin 2\gamma = B_{xx}$

5. $B'_{xy} \cos \gamma + B'_{zy} \sin \gamma = B_{xy}$

6. $B'_{xz} \cos^2 - B'_{zx} \sin^2\gamma + (B'_{zz} - B'_{xx})\ 1/2 \sin 2\gamma = B_{xz}$

7. $A'_{yy} = A_{yy}$

8. $A'_{yz} \cos \gamma - A'_{xy} \sin \gamma = A_{yz}$

9. $B'_{yx} \cos \gamma + B'_{yz} \sin \gamma = B_{yx}$

10. $B'_{yy} = B_{yy}$

11. $B'_{yz} \cos \gamma - B'_{yx} \sin \gamma = B_{yz}$

12. $A'_{zz} \cos^2\gamma + A'_{xx} \sin^2\gamma - A'_{xz} \sin 2\gamma = A_{zz}$

13. $B'_{zx} \cos^2\gamma - B'_{xz} \sin^2\gamma + (B'_{zz} - B'_{xx})\ 1/2 \sin 2\gamma = B_{zx}$

14. $B'_{zy} \cos \gamma - B'_{xy} \sin \gamma = B_{zy}$

15. $B'_{zz} \cos^2\gamma + B'_{xx} \sin^2\gamma - (B'_{zx} + B'_{xz})\ 1/2 \sin 2\gamma = B_{zz}$

16. $C'_{xx} \cos^2\gamma + C'_{zz} \sin^2\gamma + C'_{xz} \sin 2\gamma = C_{xx}$

17. $C'_{xy} \cos \gamma + C'_{yz} \sin \gamma = C_{xy}$

18. $C'_{xz} \cos 2\gamma + (C'_{zz} - C'_{xx})\ 1/2 \sin 2\gamma = C_{xz}$

19. $C'_{yy} = C_{yy}$

20. $C'_{yz} \cos\gamma - C'_{xy} \sin\gamma = C_{yz}$

21. $C'_{zz} \cos^2\gamma + C'_{xx} \sin^2\gamma - C'_{xz} \sin 2\gamma = C_{zz}$

For Rotation About Z-Axis

1. $A'_{xx} \cos^2\gamma + A'_{yy} \sin^2\gamma - A'_{xy} \sin 2\gamma = A_{xx}$
2. $A'_{xy} \cos 2\gamma + (A'_{xx} - A'_{yy})\, 1/2 \sin 2\gamma = A_{xy}$
3. $A'_{xz} \cos\gamma - A'_{yz} \sin\gamma = A_{xz}$
4. $B'_{xx} \cos^2\gamma + B'_{yy} \sin^2\gamma - (B'_{xy} + B'_{yx})\, 1/2 \sin 2\gamma = B_{xx}$
5. $B'_{xy} \cos^2\gamma - B'_{yx} \sin^2\gamma + (B'_{xx} - B'_{yy})\, 1/2 \sin 2\gamma = B_{xy}$
6. $B'_{xz} \cos\gamma - B'_{yz} \sin\gamma = B_{xz}$
7. $A'_{yy} \cos^2\gamma + A'_{xx} \sin^2\gamma + A'_{xy} \sin 2\gamma = A_{yy}$
8. $A'_{yz} \cos\gamma + A'_{xz} \sin\gamma = A_{yz}$
9. $B'_{yx} \cos^2\gamma - B'_{xy} \sin^2\gamma + (B'_{xx} - B'_{yy})\, 1/2 \sin 2\gamma = B_{yx}$
10. $B'_{yy} \cos^2\gamma + B'_{xx} \sin^2\gamma + (B'_{xy} + B'_{yx})\, 1/2 \sin 2\gamma = B_{yy}$
11. $B'_{yz} \cos\gamma + B'_{xz} \sin\gamma = B_{yz}$
12. $A'_{zz} = A_{zz}$
13. $B'_{zx} \cos\gamma - B'_{zy} \sin\gamma = B_{zx}$
14. $B'_{zy} \cos\gamma + B'_{zx} \sin\gamma = B_{zy}$
15. $B'_{zz} = B_{zz}$
16. $C'_{xx} \cos^2\gamma + C'_{yy} \sin^2\gamma - C'_{xy} \sin 2\gamma = C_{xx}$
17. $C'_{xy} \cos 2\gamma + (C'_{xx} - C'_{yy})\, 1/2 \sin 2\gamma = C_{xy}$
18. $C'_{xz} \cos\gamma - C'_{yz} \sin\gamma = C_{xz}$
19. $C'_{yy} \cos^2\gamma + C'_{xx} \sin^2\gamma + C'_{xy} \sin 2\gamma = C_{yy}$
20. $C'_{yz} \cos\gamma + C'_{xz} \sin\gamma = C_{yz}$
21. $C'_{zz} = C_{zz}$

Sum and store the respective shape coefficients, A_{xx}........ to C_{zz}, of all members in each branch. Also add the respective "C" branch coefficients to the "A" branch and "B" branch coefficeints. Subscripts denote branch.

$\Sigma Axx_A = a_{11}$

$\Sigma Axy_A = a_{12}$

$\Sigma Axz_A = a_{13}$

$\Sigma Bxx_A = a_{14}$

$\Sigma Bxy_A = a_{15}$

$\Sigma Bxz_A = a_{16}$

$\Sigma Axx_c = a_{17}$

$\Sigma Axy_c = a_{18}$

$\Sigma Axz_c = a_{19}$

$\Sigma Bxx_c = a_{1,10}$

$\Sigma Bxy_c = a_{1,11}$

$\Sigma Bxz_c = a_{1,12}$

$\Sigma Ayy_A = a_{22}$

$\Sigma Ayz_A = a_{23}$

$\Sigma Byx_A = a_{24}$

$\Sigma Byy_A = a_{25}$

$\Sigma Byz_A = a_{26}$

$\Sigma Axy_c = a_{27}$

$\Sigma Ayy_c = a_{28}$

$\Sigma Ayz_c = a_{29}$

$\Sigma Byx_c = a_{2,10}$

$\Sigma Byy_c = a_{2,11}$

$\Sigma Byz_c = a_{2,12}$

$\Sigma Azz_A = a_{33}$

$\Sigma Bzx_A = a_{34}$

$\Sigma Bzy_A = a_{35}$

$\Sigma Bzz_A = a_{36}$

$\Sigma Axz_c = a_{37}$

$\Sigma Ayz_c = a_{38}$

$\Sigma Azz_c = a_{39}$

$\Sigma Bzx_c = a_{3,10}$

$\Sigma Bzy_c = a_{3,11}$

$\Sigma Bzz_c = a_{3,12}$

$\Sigma Cxx_A = a_{44}$

$\Sigma Cxy_A = a_{45}$

$\Sigma Cxz_A = a_{46}$

$\Sigma Bxx_c = a_{47}$

$\Sigma Byx_c = a_{48}$

$\Sigma Bzx_c = a_{49}$

$\Sigma Cxx_c = a_{4,10}$

$\Sigma Cxy_c = a_{4,11}$

$\Sigma Cxz_c = a_{4,12}$

$\Sigma Cyy_A = a_{55}$

$\Sigma Cyz_A = a_{56}$

$\Sigma Bxy_c = a_{57}$

$\Sigma Byy_c = a_{58}$

$\Sigma Bzy_c = a_{59}$

$\Sigma Cxy_c = a_{5,10}$

$\Sigma Cyy_c = a_{5,11}$

$\Sigma Cyz_c = a_{5,12}$

$\Sigma Czz_A = a_{66}$

$\Sigma Bxz_c = a_{67}$

$\Sigma Byz_c = a_{68}$

$\Sigma Bzz_c = a_{69}$

$\Sigma Cxz_c = a_{6,10}$

$\Sigma Cyz_c = a_{6,11}$

$\Sigma Czz_c = a_{6,12}$

$\Sigma Axx_B = a_{77}$

$\Sigma Axy_B = a_{78}$ $\Sigma Ayy_B = a_{88}$

$\Sigma Axz_B = a_{79}$ $\Sigma Ayz_B = a_{89}$ $\Sigma Azz_B = a_{99}$

$\Sigma Bxx_B = a_{7,10}$ $\Sigma Byx_B = a_{8,10}$ $\Sigma Bzx_B = a_{9,10}$

$\Sigma Bxy_B = a_{7,11}$ $\Sigma Byy_B = a_{8,11}$ $\Sigma Bzy_B = a_{9,11}$

$\Sigma Bxz_B = a_{7,12}$ $\Sigma Byz_B = a_{8,12}$ $\Sigma Bzz_B = a_{9,12}$

$\Sigma Cxx_B = a_{10,10}$

$\Sigma Cxy_B = a_{10,11}$ $\Sigma Cyy_B = a_{11,11}$

$\Sigma Cxz_B = a_{10,12}$ $\Sigma Cyz_B = a_{11,12}$ $\Sigma Czz_B = a_{12,12}$

These are the 78 elements above the diagonal of a symmetrical 12 x 12 A-matrix. Store for matrix solution in Run II after card for last member has been calculated.

Run II - Matrix Solution

(a) Load program for Run II

(b) Feed data cards and store

P_N,

ϕX_{NA}, ϕY_{NA}, ϕZ_{NA}, ΔX_{NA}, ΔY_{NA}, ΔZ_{NA}

ϕX_{NB}, ϕY_{NB}, ϕZ_{NB}, ΔX_{NB}, ΔY_{NB}, ΔZ_{NB}

for each of N operating conditions of line.

Store B-Matrix Elements

$\phi X_{1A} = b_{1,13}$	$\phi X_{2A} = b_{1,14}$	$\phi X_{NA} = b_{1,12+N}$
$\phi Y_{1A} = b_{2,13}$	•	•
$\phi Z_{1A} = b_{3,13}$	•	•
$\Delta X_{1A} = b_{4,13}$	•	•
$\Delta Y_{1A} = b_{5,13}$	•	•
$\Delta Z_{1A} = b_{6,13}$	etc.	etc.

$\emptyset X_{1B} = b_{7,13}$	.	.
$\emptyset Y_{1B} = b_{8,13}$	.	.
$\emptyset Z_{1B} = b_{9,13}$	.	.
$\Delta X_{1B} = b_{10,13}$	.	.
$\Delta Y_{1B} = b_{11,13}$	.	.
$\Delta Z_{1B} = b_{12,13}$	$\Delta Z_{2B} = b_{12,14}$	$\Delta Z_{NB} = b_{12,\ 12+N}$

Solve matrix for the moments and forces at the origin for each condition.

M_{x_a}	M_{x_B}
M_{y_A}	M_{y_B}
M_{z_A}	M_{z_B}
F_{x_A}	F_{x_B}
F_{y_A}	F_{y_B}
F_{z_A}	F_{z_B}

These 12 reactions for each condition are results. Store for use in Run III and punch out.

Add "A" branch reactions to "B" branch reactions to obtain "C" branch reactions for each condition.

$$M_{x_c} = M_{x_A} + M_{x_B}$$

$$M_{y_c} = M_{y_A} + M_{y_A}$$

$$M_{z_c} = M_{z_A} + M_{z_B}$$

$$F_{x_c} = F_{x_A} + F_{x_B}$$

$$F_{y_c} = F_{y_A} + F_{y_B}$$

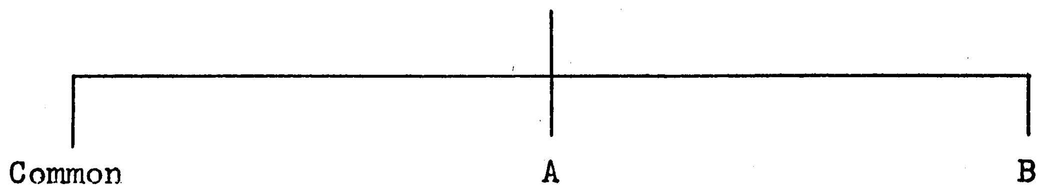
$$F_{z_c} = F_{z_A} + F_{z_B}$$

Store these 6 reactions for each condition for use in Run III.

Run III - Stress Calculation

(a) Load Program for Run III

(b) Feed data card containing stress point number, plane and branch containing point and x, y, z, α, SM, OD, t, and β.



For Point in Common Branch

1. $M_{x_c} + F_{y_c} \cdot z - F_{z_c} \cdot y = M'_x$
2. $M_{y_c} + F_{z_c} \cdot x - F_{x_c} \cdot z = M'_y$
3. $M_{z_c} + F_{x_c} \cdot y - F_{y_c} \cdot x = M'_z$

For Point in "A" Branch

1. $M_{x_A} + F_{y_A} \cdot z - F_{z_A} \cdot y = M'_x$
2. $M_{y_A} + F_{z_A} \cdot x - F_{x_A} \cdot z = M'_y$
3. $M_{z_A} + F_{x_A} \cdot y - F_{y_A} \cdot x = M'_z$

For Point in "B" Branch

1. $M_{x_B} + F_{y_B} \cdot z - F_{z_B} \cdot y = M'_x$
2. $M_{y_B} + F_{z_B} \cdot x - F_{x_B} \cdot z = M'_y$
3. $M_{z_B} + F_{x_B} \cdot y - F_{y_B} \cdot x = M'_z$

These three moments at a point for a condition are results. Store for punch out.

4. Sin α

5. cos α

"X"-Plane "Y"-Plane "Z"-Plane

For Point in "X" Plane

6. $M'_x = M_V$

7. $M'_z \cos \alpha - M'_y \sin \alpha = M_H$

8. $M'_z \sin \alpha + M'_y \cos \alpha = M_Q$

For Point in "Y" Plane

6. $M'_y = M_V$

7. $M'_x \cos \alpha - M'_z \sin \alpha = M_H$

8. $M'_x \sin \alpha + M'_z \cos \alpha = M_Q$

For Point in "Z" Plane

6. $M'_z = M_V$

7. $M'_y \cos \alpha - M'_x \sin \alpha = M_H$

8. $M'_y \sin \alpha + M'_x \cos \alpha = M_Q$

9. $P_N \, (OD-t)/4t = S_p$

10. $\frac{12}{SM} = c$

11. $C\beta \, M_V = S_V$

12. $C\beta \, M_H = S_H$

13. $C \, M_Q/2 = S_Q$

14. $\sqrt{S_V 2 + S_H 2} = S_B$

15. $S_B - S_p$

16. $\sqrt{(S_B - S_p)^2 + (2 \, S_Q)^2} = S_L$

17. $\frac{S_B + 3 \, S_p + S_L}{2} = S_c$

Stresses S_V, S_H. S_Q and S_c are results at a point for a condition. Store for punch out.

If this is not last condition, return to Step 1 after branch selection and calculate same point for next condition.

If it is last condition feed data card for next point.

CATALYTIC REFORMER GAS PLANT EQUILIBRIUM CALCULATIONS

E. V. Merrick and R. B. Perry
The Standard Oil Company of Ohio

The Process Engineering Division wished to calculate the Catalytic Reformer Gas Plant products at design conditions and at a range of operating conditions bracketing the design conditions. The results were to be used for:

1. Current product planning work; and
2. Design specifications for future refinery expansions.

Figure I is a simplified flow diagram of the unit. It was designed to produce both a gasoline product (catalytic reformate) and a heating product (LPG), each of a desired quality and yield at the design operating conditions. If these operating conditions are varied, it becomes necessary to calculate the products obtained under the new conditions in order to see whether they are still within the quality and yield range desired.

The problem programmed here is the calculation of these products. Initially, a feed stream M is taken into the first flash zone. This feed consists of a known quantity and composition of product, F, from the reactor (not shown here), and an assumed quantity and composition for recycle, R_1, as dictated by a fixed hydrogen requirement in R_1. The composition of F will vary according to the degree of hydrocracking used in the reactor section.

The vapor and liquid products (V_1 and L_1) at the Product Separator are then calculated through the use of equations 1, 2, and 3 of Table I. The total recycle quantity R_1^1 is determined by the use of equation (4) and compared with R_1. If not equal, R_1^1 is used as R_1, with a composition proportional to V_1, and the calculation is repeated through the product separator.

Upon obtaining a balance between the equilibrium in the Products Separator and R_1, we then proceed to calculating the rest of the unit.

The equilibrium calculation through the remaining two flash zones is, of course, the same. However, the balancing of products in the unit after flash zone 1 is complicated by the interdependence of R_2 and R_3. R_2 and R_3 are assumed initially and V_3 calculated; these calculations must then proceed until the calculated R_2 and R_3 are constant quantities for each set of operating conditions and so that R_2 equals V_3.

As can be seen, high-powered mathematics are not involved in this problem; it is simply a voluminous arithmetic problem, largely repetitive in nature with a trial-and-error approach used in assuming initial recycle quantities (R_1, R_2, and R_3), followed by equilibrium calculations in three sections of the plant -- the Products Separator, the L. P. Flash Drum, and the Depropanizer Reflux Drum. Material balance calculations are made at the depropanizer and at the deethanizer. Here are used empirical equations based on design conditions (Table I).

The problem was programmed in two parts:

<u>Part I</u> - Calculations through the Product Separator Section;

<u>Part II</u> - Balance of the plant - the calculations being continued upon a basis of selected results from the Part I calculations.

Figure 2 is a block diagram for the flow of calculations. When Figure 1, the simplified flow diagram of the Gas Plant, is compared with Figure 2, it is clearly seen that our 650 program is in essence a simulation of operations in the Gas Plant.

Part I of the program used 422 instructions, 5 constants, and a working area of 110 locations. A total of 60 cases were computed in 39 minutes. The number of cases depended on variations in T_1, P_1 and F (Figure 1).

Part II used 599 instructions, 13 constants, and a working area of 195 locations. It is estimated that the 650 went through more than 75,000,000 operations to compute Part II. A total of 729 cases was computed in approximately 60 hours. The number of cases depended on variations of T_2 and P_2 and T_3 and P_3.

There were no programming difficulties once the problem was defined. Some minor scaling problems, discovered in testing the program, were quickly solved. Five decimals were used throughout the problem wherever possible, so that accuracy was maintained to a finer degree than possible by hand calculations. (Material balances check within 0.01% - See Table II).

The problem analysis and programming totalled approximately 425 man-hours. The job breakdown for this total is as follows:

Problem analysis	-	19 %
Discussion and review	-	18 %
Block Diagram	-	6 %
Coding	-	23 %
Checking of routines	-	20 %
Program testing on 650	-	14 %

We have estimated that it would take an engineering assistant approximately 6000 hours to obtain by hand the results that the 650 has given for this problem. In contrast, the 650 machine time totals less than 61 hours.

TABLE I

Equations Used

Equilibrium Calculations

$$\frac{K_i \; M_i}{L/V * + K_i} = v_i \qquad (1)$$

$$\sum_{1}^{n} v_i = V \qquad (2)$$

$$\frac{M - V}{V} = L/V * \qquad (3)$$

R_1 Determination

$$R_1^1 = \frac{\sum_{1}^{n} v_i}{v_1} (r_1) \qquad (4)$$

Where —

M= total mols/hr. of feed (F [or L] + R).
M_i = mols/hr. of each component in feed.
v_i = mols/hr. of each component leaving flash zone as vapor.
K_i = equilibrium factor for each component in feed at the chosen Temperature and Pressure.
V = total mols/hr. of vapor in equilibrium.
L = total mols/hr. of liquid in equilibrium.
v_1 = mols/hr. hydrogen in V_1.
r_1 = mols/hr. hydrogen in R_1^1 (specified)

Equations Used in De-C_3 and De-C_2 Sections:

$$D + W = L = L_1 + L_2 - - - - - L_n \qquad (5)$$

$$D = a_1 L_1 + a_2 L_2 - - - - - - a_n L_n \qquad (6)$$

$$W = b_1 L_1 + b_2 L_2 - - - - - - b_n L_n \qquad (7)$$

Where —

D = Overhead Product
W = Bottoms Product
L_i = mols/hr. of each component in feed.

a_i, b_i = design constants, $a_i + b_i = 1$

* Assumed value

TABLE II

Example: Material Balance Check

Normal Hydrocracking - Case No. 141

		Mols/Hour
Feed (F)		2520.0
Products:		
G_1	1324.0	
G_2	61.0	
S_1	1062.8	
S_2	44.9	
G_3	27.5	
Total		2520.2

$.2/2520 < 0.01\ \%$

FIG. I - CAT. REFORMER GAS PLANT

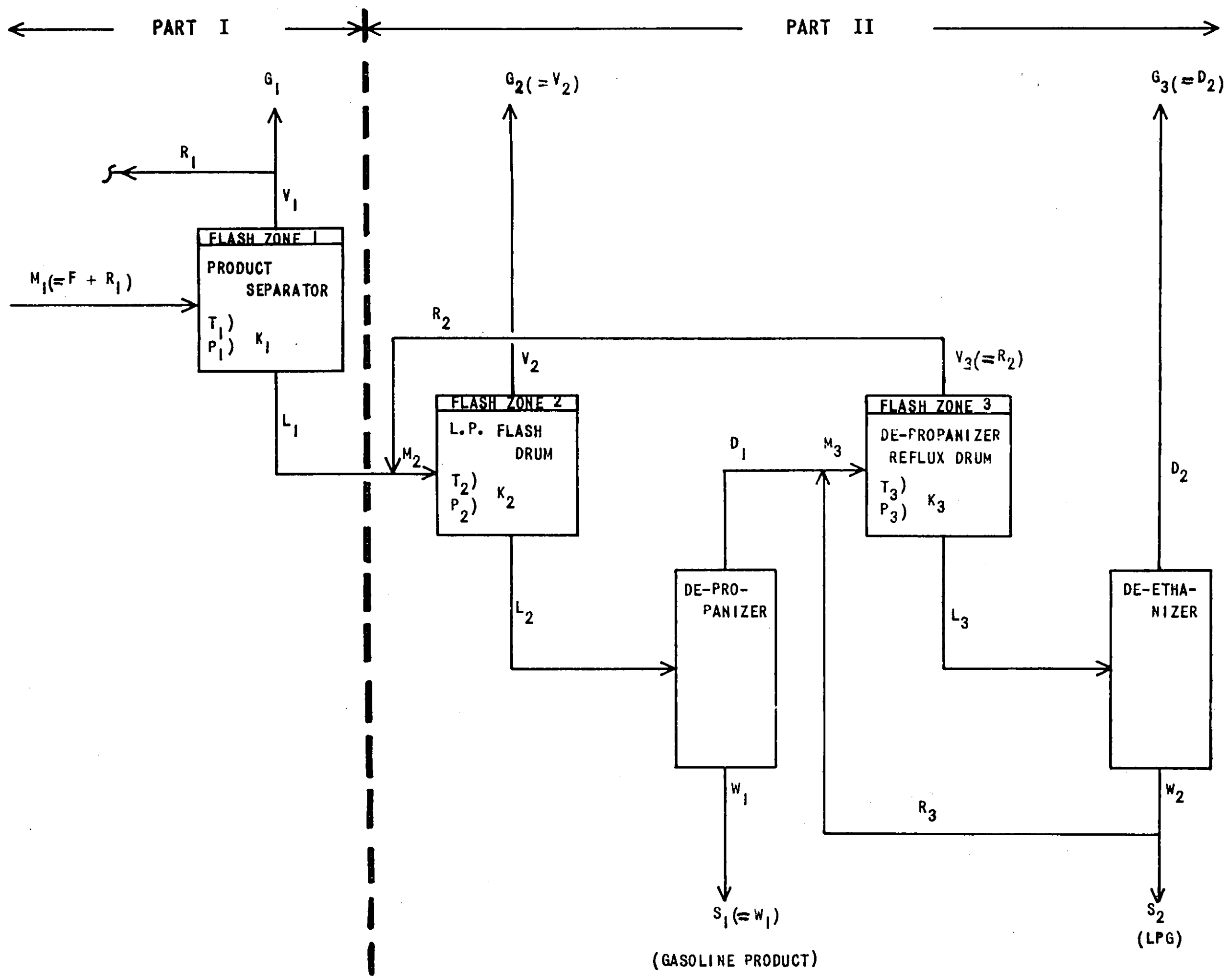


FIGURE 2 BLOCK DIAGRAM CAT REFORMER GAS PLANT

PART I

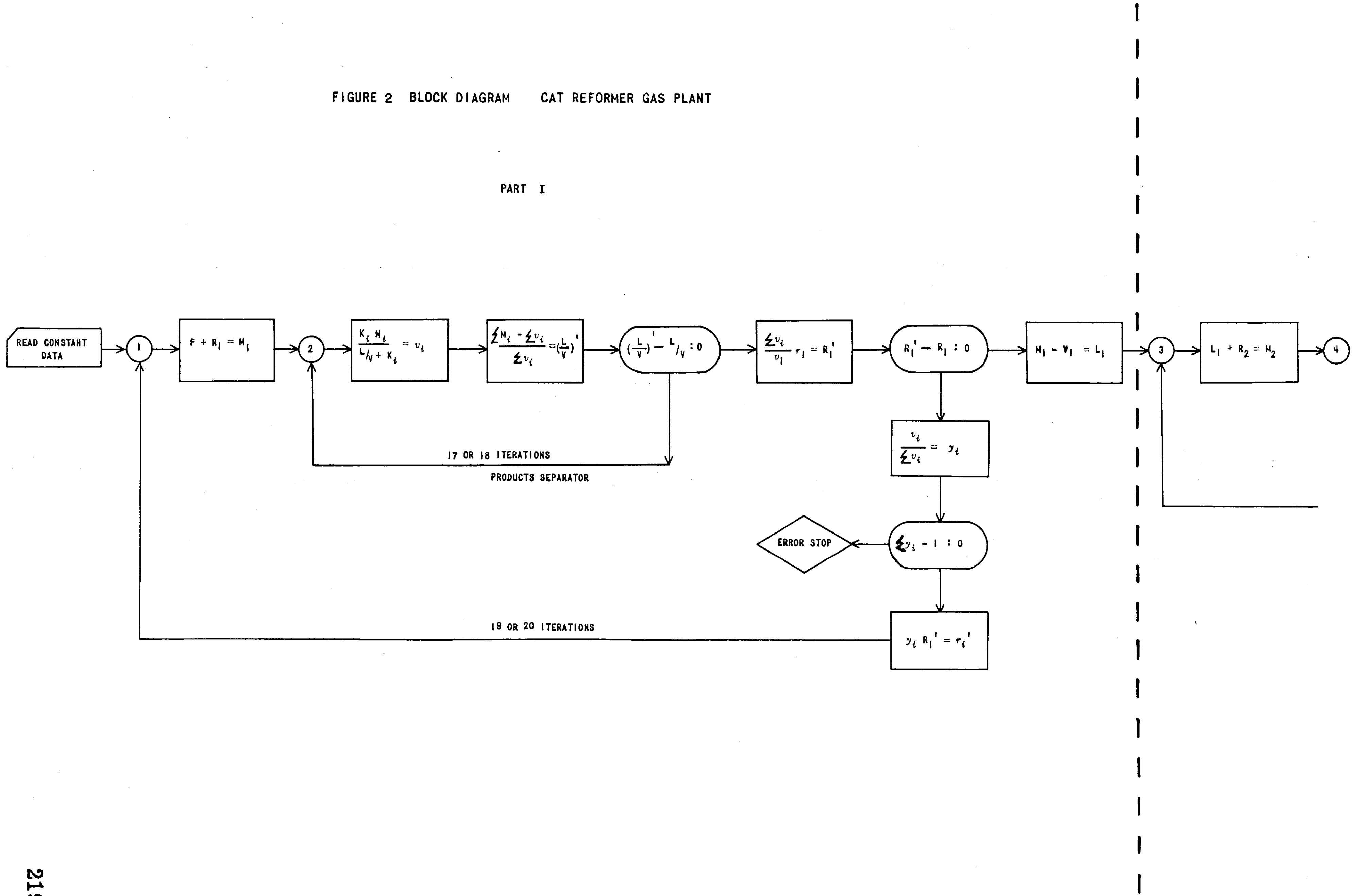


FIGURE 2 - CONTINUED

PART II

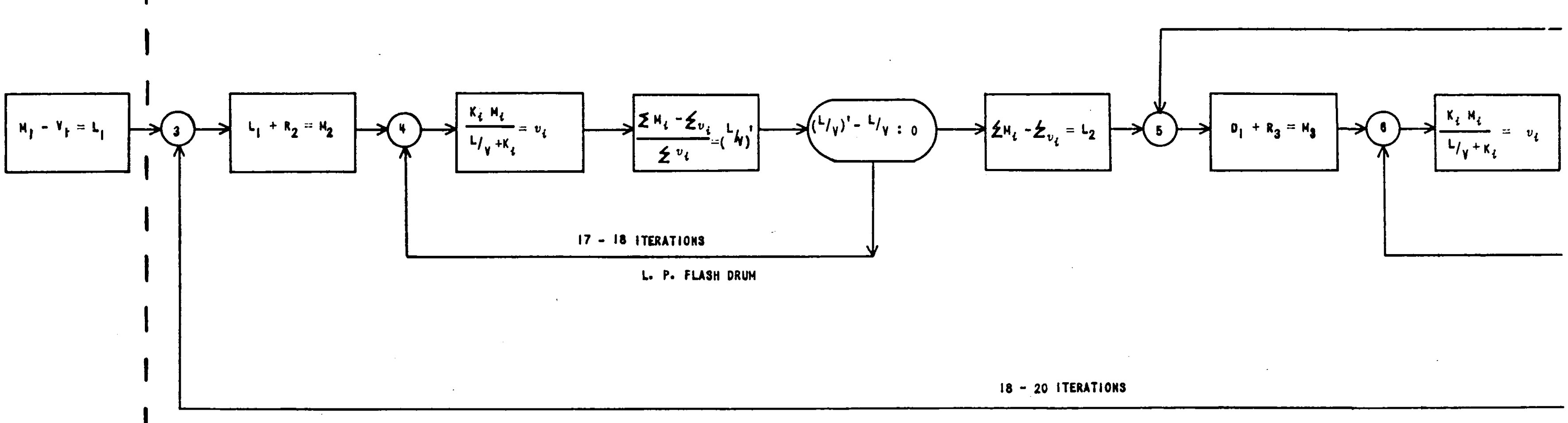
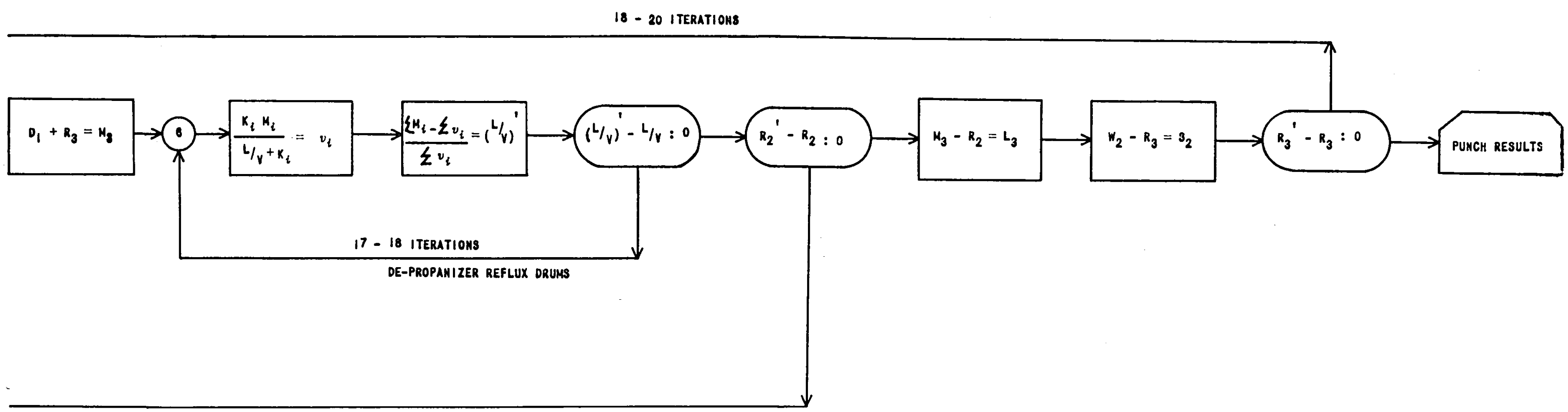


FIGURE 2 - CONTINUED

PART II



A METHOD FOR THE EVALUATION OF NON-LINEAR SERVO-MECHANISMS BY NUMERICAL INTEGRATION

W. Barkley Fritz
Westinghouse Electric Corporation

An early and interesting application of the IBM 650 computer, recently installed at the Air Arm Division of the Westinghouse Electric Corporation, has been the evaluation of non-linear servo-mechanisms by numerically solving the differential equations for the dynamics of these systems. Discontinuities in these servo systems are of particular interest. The mathematical models and computational procedures are presented for the solution of two such problems, each formulated by Dr. K. N. Satyendra of the Air Arm Division. The single-step Runge-Kutta integrating procedure is used to integrate the equations. This procedure is particularly efficient since it requires no special starting values, makes no unreasonable demands on storage, and allows the integration interval to be easily and automatically changed as required.

The first problem concerns the obtaining of a time history, as affected by the non-linearity in the radome, of a servo-mechanism used in the antenna drive of an airborne fire control system.

The equation to be solved may be expressed as follows:

$$\frac{d^2y}{dt^2} = A(y + \frac{dy}{dt}) + B + 2\ (x + \frac{dx}{dt})$$

At $t = 0$, $y = dy/dt = 0$. Moreover, x may be constant or may be a function of time. The problem is to integrate until y exceeds some y_{max}. Typical values of A and B, as they change with the dependent variable y, are as follows:

	A	B
$0 < y \leq 1$	0	-1
$1 < y \leq 2$	-1.5	0.5
$2 < y \leq 3$	4.5	-2.5
$3 < y \leq 5$	0	-1
$5 < y \leq 6$	7.5	-2.5
$6 < y \leq 7$	-10.5	0.5
$7 < y \leq 9$	0	-1

Under the assumptions that y(t) is continuously differentiable and x(t) is known, an analytic solution can be given for each interval. The matching of these solutions at the change-over points requires the solution of a system of transcendental equations for which the error can be controlled. This procedure and another method based on interchanging dependent and independent variables are possible alternatives to fairly straightforward integrating procedure to be described.

In order to get accurate starting values for the new equation to be solved when y leaves one region to enter another, it is necessary to obtain accurate values for t, dy/dt, x, and dx/dt, when y equals the change-over value. These values may be obtained by integrating at some convenient interval until y exceeds the change-over value. Then, returning to the step just before this y value is exceeded, the integration step size is halved and again integrated. If y still exceeds the change-over value, return to the former results, halve the integration interval, and repeat until the newly computed value of y is less than the change-over value. The integration interval is once more halved, and computation proceeds as indicated until the computed value of y is equal to the change-over value to within some ϵ.

The procedure is illustrated in Figure 1.

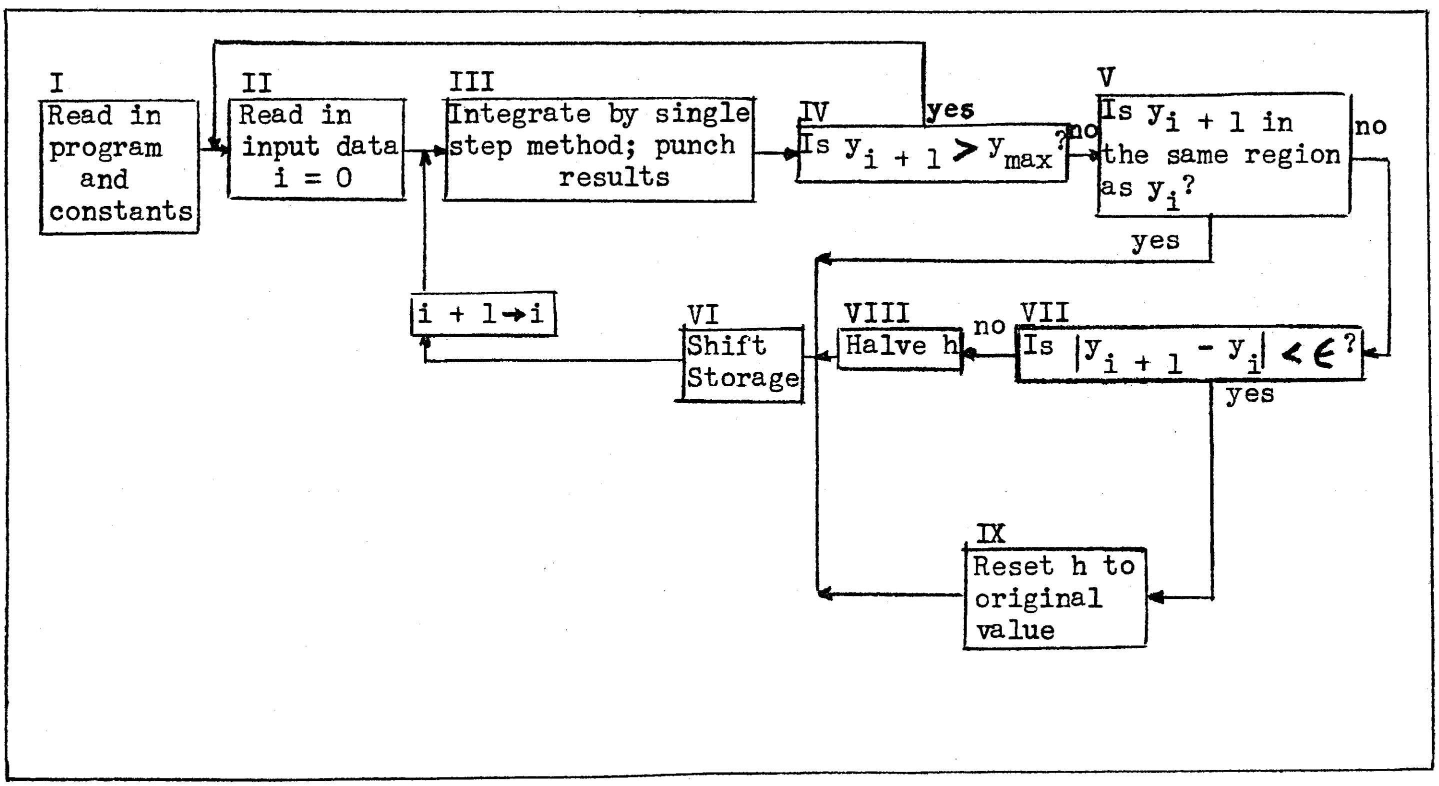


Figure 1. Simplified Flow Diagram of Radome Non-linearity Program.

It has been pointed out that this procedure could give a very poor value for t if dy/dt is small at the change-over point. Fortunately, this is not the case in the physical problem actually solved by this technique; however, in general, it would be best to replace this condition of box VII with some other relation.

An alternate method may be used for the regular step-by-step integration as follows:

1. Integrate, obtaining values $y_{i+1}^{(1)}$ with a single step h.

2. Integrate, obtaining values $y_{i+1/2}$, $y_{i+1}^{(2)}$ with two steps each at the interval h/2.

3. Substitute these values in the Richardson extrapolation to zero grid-size formula

$$y_{i+1} = (2^k y_{i+1}^{(2)} - y_{i+1}^{(1)})/(2^k - 1),$$

in which k is the order of the method used.

The value y_{i+1} is then taken as the starting value for the new step which proceeds as indicated, using the same discrimination procedure to determine when the change-over value of y has been obtained. During the halving process, it is unnecessary to use the Richardson formula as part of the procedure because of the nearness to the change-over value of y and the fact that the new procedure already involves a halving process. Either procedure can be used to integrate the second order equation by using an assigned integration interval. Interpolation might be used to obtain values of t, dy/dt, x, and dx/dt, for which y equals the required change-over value. However, this procedure requires a knowledge of the behavior of the solution to determine the type of interpolation to use as well as the necessary special coding to perform the interpolation. The procedures outlined make use of an integrating process to approach the correct solution and provide control over the accuracy obtained.

A second problem concerns equations that represent the dynamics of a non-linear servo-mechanism with backlash, viscous damping, and coulomb friction. Mathematically the problem can be stated as indicated in Figure 2.

(1) $\frac{dz_m}{dt} = -\frac{(D_m+D_\ell)}{(J_m+J_\ell)} z_m + K_1 (y_m - \frac{b}{2}) \pm F - K_2 x$ $\frac{dy_m}{dt} = z_m$

At $t = 0$, $y_m = y_\ell = \frac{dy_m}{dt} = \frac{dy}{dt} = 0$,

Integrate (1) until $B_i = J_\ell \frac{d^2 y_m}{dt^2} + D_\ell \frac{dy_m}{dt} \pm F$ changes sign. The sign of F is the same as $\frac{dy_m}{dt}$

Then integrate

$$(2) \begin{cases} \frac{dy_\ell}{dt} = z_\ell & \frac{dz_\ell}{dt} = \frac{1}{J_\ell}(D_\ell z_\ell \pm F) \\ \frac{dy_m}{dt} = z_m & \frac{dz_m}{dt} = -\frac{1}{J_m}(D_m z_m + K_1 y - K_2 x) \end{cases}$$

until $G_i = y_m + \frac{b}{2}$ changes sign.

Next, change the sign of b/2 in (1) as well as in the definition of G_i and let y_ℓ replace y_m in B_i and G_i. The systems are again integrated as before completing one cycle.

Figure 2. Non-linear Servo with Backlash Equation.

Obviously y_ℓ, y_m, z_ℓ, and z_m are dependent variables with t the independent variable. Values of D_m, D_ℓ, J_m, J_ℓ, K_1, K_2, b/2, and F are given and remain constant for each case.

In order to keep the emphasis on the solution techniques involved, the engineering aspects of this servo mechanism with a forcing function will not be discussed.

The complete computational procedure is illustrated by Figure 3.

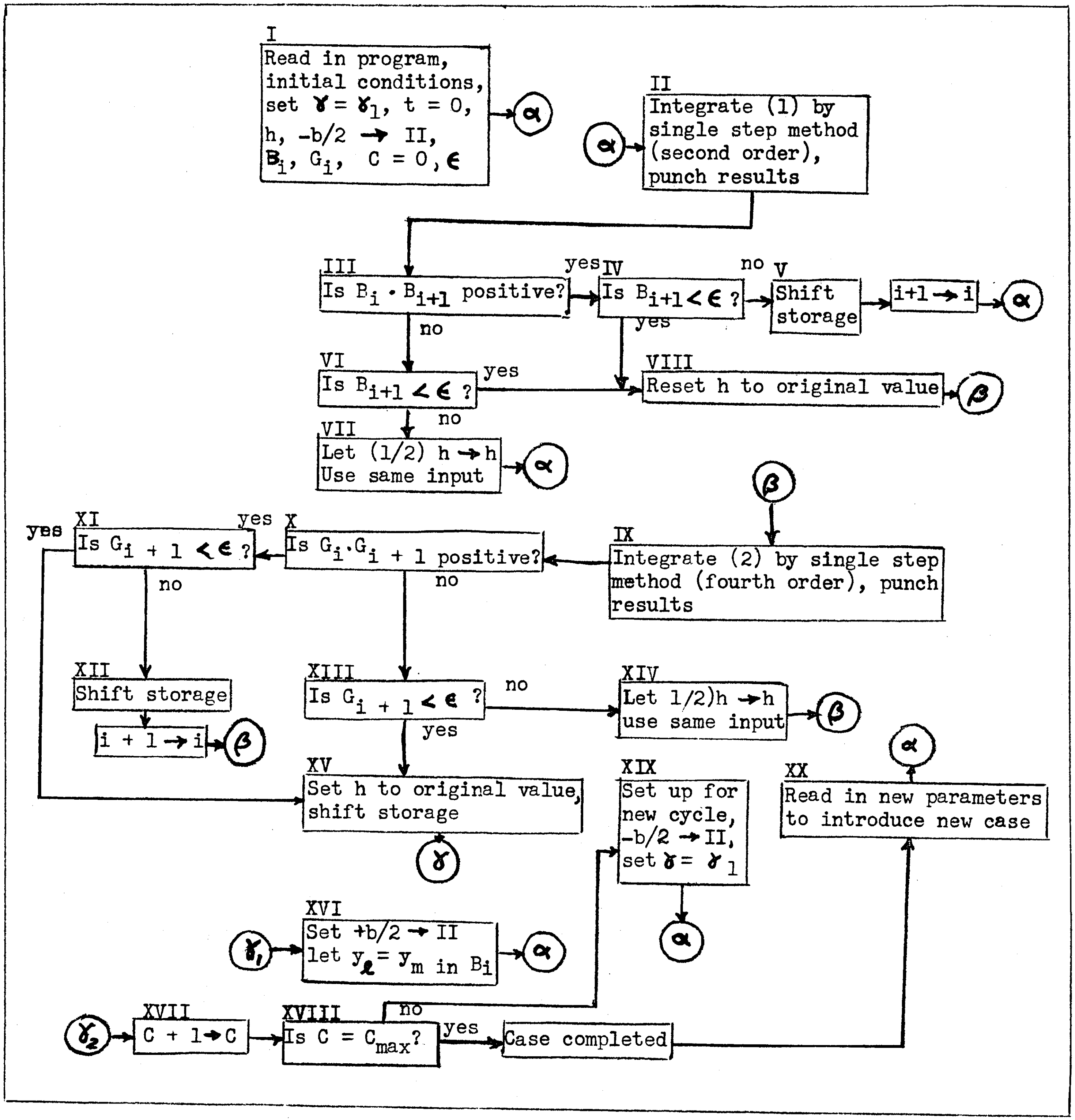


Figure 3. Simplified Flow Diagram for Solution of Non-linear Servo with Backlash.

Each of these problems requires the solution of a series of differential equations, each valid within a clearly defined region of one of the dependent variables. The same type of integrating process is required in every case in order efficiently to take care of the discontinuities present. The basically essential feature is that the integrating procedure be of the single-step type, i.e., that the integration of the system proceed from values of the variables at just one preceding time step.

Among the possible single-step integration procedures is one that is usually designated as the Runge-Kutta method. This method appears to be a particularly attractive one. But after reading the objections presented by Milne in his text, Numerical Solution of Differential Equations [1], it may be considered desirable to try something else. Milne first points out that each step requires four substitutions into the differential equation (s) which, for complicated equations, may demand an excessive amount of labor per step. Milne, in the second place, notes the absence of any simple means for detecting machine failures or for estimating the error.

This first objection is based on the assumption that one is using a desk computer to do the computation. Actually, the very "repetitiousness" of this procedure is an advantage when using a reasonably fast automatic digital computer.

The second objection is a more serious one, however, and must be answered in greater detail. The absence of built-in checks for machine errors is not really very important when one is using a present-day automatic digital computer. As for the estimate of computational error, Bieberbach [2] has found an expression which gives an upper bound for the error at a given step; however this estimate requires, as does an improved bound by Lotkin [3], the computation of additional quantities which do not already appear in the basic computation. Further, this error is only a bound, and may force one to reduce the integration interval even unnecessarily so that the computation time becomes excessive.

The procedure of extrapolation to zero grid size, as used to improve the solution, has been shown by Gorn [4] effectively to transform any integrating procedure into a new integration of one higher order at the cost of multiplying the time approximately threefold. Whether this is practicable or not will depend on such factors as the local situation, the machine time available, the accuracy required, and the independent checks available. What is important is that the error can be controlled in the Runge-Kutta procedure. Moreover, the method is exceptionally efficient. Although care must be exercised in its use, its several advantages require that it be given consideration as a general automatic computing technique for integrating systems of differential equations. Specifically, these advantages are:

1. No special starting procedure is required.

2. The integration formulas do not change when the step size must be varied, as required by the problems just considered.

3. Only the values at the previous time step need to be stored.

Each of these advantages means a saving in coding time and machine space. In fact, the Runge-Kutta procedure requires less machine storage than the methods of Adams, Milne, or other procedures of this type. Putting this another way, the IBM 650, or any automatic digital computer, can store the program for solving a higher order system by using the Runge-Kutta procedure than it is capable of storing by using methods which require the saving of values from a number of previous integration steps.

It is important to point out that the Runge-Kutta method is a fourth order method ($k=4$ in the previous notation); that is, the error excluding round-off goes to zero as the fifth power of the step size. It is therefore equivalent to approximating the solution locally on intervals of length h (the step size) by polynomials of the fourth degree. Adams' and Milne's methods are equivalent in this respect.

It should be mentioned that Gill [5] has modified the Runge-Kutta procedure so that only three registers of storage per dependent variable are required instead of the four required by the unmodified method. Gill's modification serves to emphasize the usefulness of the Runge-Kutta procedure.

To conclude, information on the stability of non-linear servo mechanisms has in the past been obtained by graphical methods like the describing-function technique as exemplified by the work of Johnson [6] and others. Such graphical methods fail to yield data on the transient behavior of the system as a whole. The technique presented in this paper provides a powerful tool in the hands of automatic control systems engineer since the complete time-history of the system, both in its transient and in its steady state, can be accurately described. Finally, the single-step integration, as exemplified by the Runge-Kutta procedure, has been successful in solving several of these discontinuous problems and offers considerable promise of helping the engineer to exploit the digital computer for the more effective design, development, and evaluation of airborne electronic control equipment.

References:

1. Milne, W. E., Numerical Solution of Differential Equations, John Wiley & Sons, Inc. (1953), p 74.

2. Bieberbach, L., Theorie der Differentialgleichungen, Dover, (1946), p 54.

3. Lotkin, M., "On the Accuracy of Runge-Kutta's Method", MTAC, Vol. V, (1951), pp 128-133.

4. Gorn, S., "The Automatic Analysis and Control of Computing Errors", J. SOC. INDUST. APPL. MATH., Vol. II, (1954), pp 69-81.

5. Gill, S., "A Process for the Step-by-Step Integration of Differential Equations in an Automatic Digital Computing Machine", PROC. CAMB. PHILOS. SOC., Vol. 47, (1951), pp 96-108.

6. Johnson, E. C., "Sinusodial Analysis of Feedback Control Systems Containing Non-Linear Elements", AIEE TRANS., Vol. 71, Pt. II, (1952), pp 169-81.

APPLICATION OF THE TYPE 650 TO FOURIER SYNTHESIS IN X-RAY CRYSTAL STRUCTURE ANALYSIS

Howard T. Evans, Jr.
United States Department of the Interior

Abstract

The electron density ρ at a point (x,y) in a crystal lattice as projected on a plane is given by

$$\rho(x,y) = \frac{1}{S} \sum_h \sum_k F(h,k) \cos 2\pi(hx+ky)$$

for a centrosymmetric crystal, where F(h,k) is a function of the diffraction intensity of the x-ray spectrum with order indices h,k as produced by the crystal for a given x-ray wavelength, and S is the area of the periodic unit cell of the crystal lattice. For practical purposes, the Fourier synthesis must be evaluated at all the points of a grid sufficiently fine to allow the electron density to be mapped in detail. A complete program was evolved for the type 650 which automatically computed the sum (which may contain over 200 terms) at each point in turn to cover 5000 points of a 100 x 100 line grid. Such a program was found to be unfeasible for this type of machine because of its great length. Alternatively, a program was developed based on the expanded form of the series:

$$S.\rho(x,y) = \sum_h \sum_k A_1(hk) \cos 2\pi hx \cos 2\pi ky - \sum_h \sum_k A_2(hk) \sin 2\pi hx \sin 2\pi ky$$

Each of the above summations is carried out in four parts for which h,k are even-even, even-odd, odd-even and odd-odd respectively, over one quarter of the cell in each dimension (676 points). The results are punched with 13 sums on one card, the cards merged and tabulated to extend the calculation over the whole unit cell and obtain the printed report. The computer program reads the A(hk) values from single-entry cards (bracketed by lead and trailer cards) into a table on the drum. Cosine values are stored in a table for which only 26 entries are required. By means of a series of loops the values of

$$C = \sum_h A(h,k) \left|\begin{matrix}\cos\\ \sin\end{matrix}\right| 2\pi hx$$

are evolved for each k at 26 values of x, and stored in a new table. The C values form the coefficients for the second

calculation

$$\sum_{k} C(x,k) \begin{vmatrix} \cos \\ \sin \end{vmatrix} 2\pi ky$$

which is initiated as soon as the C table is completed. These final sums are punched in groups of 13 on one card as they are formed. Tests of the program indicated that the type 650 will complete one series with 50 terms at 676 points in approximately 20-25 minutes, without optimum programming.

I. Introduction

The science of the deduction of the arrangement of atoms in crystals from x-ray diffraction data depends on the routine execution of large numbers of calculations based on moderate amounts of experimental data. These calculations fall into a number of well-defined groups, as follows:

1. Data preparation: the correction, scaling, normalization and adjustment of experimental data for subsequent analysis. The body of experimental data, which consists of the diffraction intensities, may vary from a hundred or less to several thousand.

2. Fourier synthesis: the synthesis of the electron density in the crystal, or a function of it, from the adjusted data.

3. Structure factors: the determination of theoretical values of the diffraction intensities from a given model crystal structure.

4. Least squares analysis: the refinement of the accepted structure by least squares analysis of the structure parameters based on the structure factor function.

For a given crystal structure problem, the last three types of computation may be repeated a considerable number of times in an interative fashion. All crystal structure analyses will make use of the same types of calculations, with only slight modification from one problem to another. It is natural therefore, that considerable effort has been made in recent years by various laboratories to adapt various IBM machine types to these calculations. With the advant of the type 650, it is apparent that a far more powerful computing instrument than has heretofore been generally available will soon be accessible to many x-ray crystallographers.

The work to be described herein was initiated in anticipation of the need for crystal structure computing programs for the type 650. The Fourier synthesis, while not as important to the science as least squares analysis, was programmed first as a fair and well-defined example with which to test the capabilities of the magnetic drum calculator.

II. The Fourier Synthesis Function

In a crystal structure the electron density at any point may be expressed as a Fourier series:

$$\rho(x,y,z) = \frac{1}{V}\sum_{h=-\infty}^{\infty}\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} |F(h,k,l)| \cos 2\pi[hx+ky+lz+\alpha(h,k,l)]$$

where $\rho(x,y,z)$ = electron density at the point x,y,z;
V = volume of the unit repeat cell of the crystal;
F(h,k,l) = structure amplitude, function of diffraction intensity;
h,k,l = orders of diffraction image (integers);
$\alpha(h,k,l)$ = phase angle associated with F(h,k,l);
x,y,z = coordinates of the point in question expressed in fractions of the unit cell edges.

This is the most general expression, but usually the crystal has symmetry which allows certain simplifications. For example, if the crystal has a center of symmetry, then for each atom at x_j, y_j, z_j (origin a symmetry center), there will be an identical atom at $-x_j$, $-y$, $-z_j$, and the structure amplitude becomes real, with $\alpha(h,k,l)$ restricted to 0 or 1/2. Further, it is frequently convenient to set one index h,k or l equal to 0, making $\rho(x,y,z)$ independent of the corresponding coordinate x,y or z, thus reducing the problem to a two dimensional one. Nowadays, however, the solution of three-dimensional non-centrosymmetric problems is becoming more and more frequent.

In any case, a considerable amount of calculation is involved since $\rho(x,y,z)$ must be evaluated at a large number of points in order to permit the electron density to be mapped in sufficient detail for the purpose of the investigation. To obtain such a map, it has long been customary to divide the unit cell edges into 60 parts each, thus forming a three dimensional grid of 216,000 points. When symmetry is present, it may be necessary to compute $\rho(x,y,z)$ at only 1/2, 1/4, or 1/8 of these points. But each series will contain several hundred or several thousand terms.

III. Adaptation of the Fourier Synthesis Problem to the Type 650

In this early phase of the study the type 650 was programmed to carry out a two-dimensional synthesis, in order to evolve a procedure which can be readily extended to the third dimension. The first effort resulted in a program designed to compute $\rho(x,y)$ at each point in turn beginning at 0,0. In all these tests, the unit cell edge interval was 1/100th instead of 1/60th. The values of $\rho(x,y)$ were gathered and punched out 13 to a card. All possible modifications required for the 17 different symmetry groups were included in the program. Without giving any more detail, it will suffice to say that the program was rejected because a simple calculation showed that the synthesis of a 200 - term series at 5000 points would require approximately 150 hours continuous machine time. The conclusion is that this approach to the problem is not satisfactory for any magnetic drum calculator, unless results are required for only a few specific points.

Recourse was then made to a procedure which has long been used in hand methods of computation. The electron density is expressed as follows:

$$\rho(x,y) = \frac{1}{S} \sum_{h=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} [A(h,k) \cos 2\pi(hx+ky) + B(h,k) \sin 2\pi(hx+ky)]$$

where S = area of unit cell in projection; and

$$A(h,k) + iB(h,k) = F(h,k).$$

When there is a center of symmetry, $B(h,k) = 0$ and the function may be written:

$$\rho(x,y) = \frac{2}{S} \sum_{h=0}^{\infty} \sum_{k=0}^{\infty} [A_1(h,k) \cos 2\pi hx \cos 2\pi ky - A_2(h,k) \sin 2\pi x \sin 2\pi ky]$$

where $A_1 = A(h,k) + A(\bar{h},k)$;
$A_2 = A(h,k) - A(\bar{h},k)$.

Each summation is now carried out in two parts, with the results of the first part being used as the coefficients for the second, e.g.:

$$C(x,k) = \sum_{h=0}^{\infty} A_1(h,k) \cos 2\pi hx;$$

$$S_1 = \sum_{h=0}^{\infty} C(x,k) \cos 2\pi ky$$

The series may conveniently be further separated into groups according as h and k are odd or even. This procedure results in the summation being required for only the first quadrant in each dimension, or at 676 points for two dimensions ($x = 0$ to 0.25, $y = 0$ to 0.25). The various odd and even subsums may then be combined with appropriate signs to expand the calculations over the whole unit cell on the tabulator.

The procedure for computation is as follows:

1. Determine $A_1(h,k)$ and $A_2(h,k)$ (also $B_1(h,k)$ and $B_2(h,k)$ if crystal is noncentrosymmetric).

2. Divide $A_1(h,k)$(and each of the other groups of coefficients) into four groups where h,k are respectively even, even; even, odd; odd, even; and odd, odd.

3. Feed one set of coefficient cards (say A_1(even, even)) in the type 650 and compute the subseries.

4. Punch out the results of the subseries on 52 cards and read in the next set. Continue (automatically) until all subseries are calculated.

5. Sort the sum cards and tabulate to produce the final table of electron density values.

IV. The Type 650 Program

In computing the subseries at 676 points, the program carries out the following operations:

1. Read in a lead card (indicating type of sum), the coefficient cards carrying h, k and A(h,k), and a trailer card (indicating end of group and initiating calculation). h|k|A(h,k) stored in a table.

2. Compute the first sum ($\Sigma A\cos 2\pi hx$) in parallel for each k, for 26 values of x in turn, and store the results in a table.

3. Compute the second sum ($\Sigma C\cos 2\pi ky$) in parallel for 26 values of y using the coefficients stored in (2) and punch the result on two cards. Repeat this calculation for 26 values of x, and return to (1).

Further details are listed below:

1a. x and y are initialized at 0; h and k are initialized at 0 or 1, according as they are even or odd respectively (digital indication on lead card: 1 for even, even; 2 for even, odd; 3 for odd, even; 4 for odd, odd).

1b. Program set to calculate appropriate cosine-sine combination (digital indication on lead card: 1 for cos-cos; 2 for cos-sin; 3 for sin-cos; 4 for sin-sin).

2a. hx formed and cos [sin] $2\pi hx$ determined.

2b. A(hk) found by table look-up on hk. Table word is: $\pm \underbrace{xx}_{h}\underbrace{xx}_{k}00\underbrace{xxxx}_{A(h,k)}$. Sign is sign of A(h,k).

2c. Find partial sum (address 1900 + k), add A $\cos 2\pi hx$, restore. Add 2 to k, return to (2b). At end of k, add 2 to h, return to (2a).

2d. At end of h, add 1 to x, transfer completed sums to intermediate table, and return to (2a). At end of x, continue to (3).

3a. ky formed and cos[sin]$2\pi ky$ determined.

3b. C(x,k) formed by table look-up on xk. Table word is: $\pm \underbrace{xx}_{x}\underbrace{xx}_{k}00\underbrace{xxxx}_{C(x,k)}$. Sign is sign of C(x,k).

3c. Find partial sum (address 1900 + x), add C $\cos 2\pi ky$, restore. Add 1 to x, return to (3b). At end of x, add 2 to k, return to (3a).

3d. At end of k, transfer completed sums for x = 0 to 0.12 to punch band (2 values per word), with series indications, and punch. Repeat for x = 0.13 to 0.25. Add 1 to y, return to (3a).

3e. At end of y, return to (1).

An overall block diagram is illustrated in Fig. 1.

The cosines and sines are evaluated by table look-up. All angles $2\pi hx$ and $2\pi ky$ are formed as cycles ($360^\circ = 1.00$), whereof the mantissa only is retained, varying from 0 to 0.99. Thus, with first

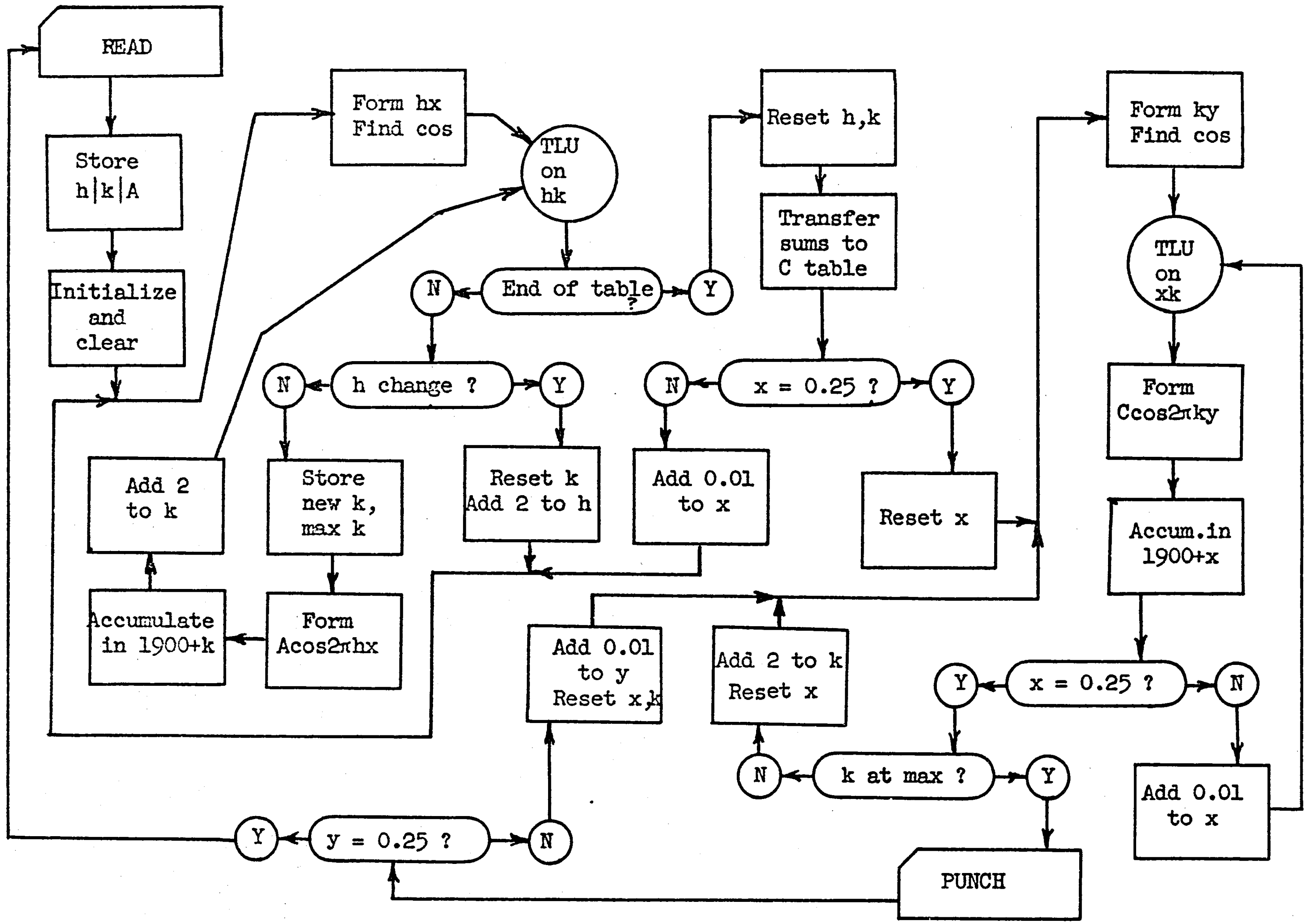


Figure 1. Block diagram for type 650, to compute subseries for Fourier synthesis.

quadrant reduction, only 26 cosine values are required for the table.

The coefficient cards are sorted on h and k, k within h, before being read into the machine. Only significant coefficients are included. In step (2b), if no coefficient is present for the given hk index, the next highest k is obtained, and the new k substituted for the old in temporary storage. Thus, no computing cycles are carried out for zero coefficients.

V. Tabulation of the Results

To obtain the final printed table of electron density values, the punched results of the calculation of the various subseries as obtained from the type 650 must be reassembled and tabulated. The entire output of cards if first separated (by means of an indicating punch) into those carrying the results for y = 0 to 0.12 and those for y = 0.13 to 0.25. Each of these decks is then sorted on x (punched by the type 650 as an indication). With an appropriate lead card, this deck will produce in the type 407, for example, any desired 16th section of the projected unit cell. In the most general centrosymmetric case, there will be eight subseries, or eight cards to be tabulated for each x value. The extension of the summation over the various sections of the unit cell is obtained by the proper control of sign of each card, as controlled by the digit punches associated with odd and even h and k, and cosine-sine combination. For the centrosymmetric case, the eight subseries will be combined in eight different ways, according to the scheme of Table I. This table can easily be extended to include the noncentrosymmetric case.

While such a program has not yet been tested, it is obvious that the combination of the subseries can be most efficiently carried out on the Type 650. The punched sums of the subseries would be sorted together on x, fed back into the Type 650, which would then punch out final totals. These could then be listed directly on the Type 407.

VI. Performance Tests

A synthesis of the electron density of the mineral liebigite, $Ca_2UO_2(CO_3)_3 . 10H_2O$, as projected on the xy plane, was used as a test problem for the procedure described above. The synthesis had previously been carried out (at 1/60th cell edge intervals) by means of our present IBM system, using the types 602A and 407. The data consists of approximately 200 F(h,k,0) terms, which divide into 4 subseries of 50 terms each. It was found that each subseries was completed in about 20-25 minutes without optimum programming. With optimum programming it is estimated that all four series can be calculated in somewhat less than an hour. This rate is about 20 times that required by our present system, or 10 times faster than a similar system based on the type 604.

TABLE I
Subseries Combinations for Complete Fourier Synthesis

Unit Cell Section		Subseries							
		Cos-Cos				Sin-Sin			
x	y	ev,ev	ev,od	od,ev	od,od	ev,ev	ev,od	od,ev	od,od
$0-\frac{1}{4}$	$0-\frac{1}{4}$	+	+	+	+	-	-	-	-
$0-\frac{1}{4}$	$\frac{1}{2}-\frac{1}{4}$	+	-	+	-	-	-	-	-
$0-\frac{1}{4}$	$\frac{1}{2}-\frac{3}{4}$	+	-	+	-	-	+	-	+
$0-\frac{1}{4}$	$1-\frac{3}{4}$	+	+	+	+	-	+	-	+
$\frac{1}{2}-\frac{1}{4}$	$0-\frac{1}{4}$	+	+	-	-	-	-	-	-
$\frac{1}{2}-\frac{1}{4}$	$\frac{1}{2}-\frac{1}{4}$	+	-	-	+	-	-	-	-
$\frac{1}{2}-\frac{1}{4}$	$\frac{1}{2}-\frac{3}{4}$	+	-	-	+	-	+	-	+
$\frac{1}{2}-\frac{1}{4}$	$1-\frac{3}{4}$	+	+	-	-	-	+	-	+

THE TRANSPORTATION PROBLEM

Charles W. Swift and Stanley Poley
International Business Machines Corporation

Introduction:

The transportation problem is essentially a special type linear programming problem. Given certain specified requirements at various destinations and amounts available at specific origins, an allocation of the products over all possible routes is desired such that the total cost of transporting the goods is minimized. In order to solve such a problem, three variables must be known. First, it is essential that the amounts made available at each of the origins be specified. Secondly, the demand at each destination should be given. Finally, it is necessary to specify a unit cost of shipping over each of the possible routes. Once provided with this information, it is possible to determine the optimum allocation of the products over the various routes or modes of transportation. Furthermore, it is possible to obtain alternate optimum solutions, i.e., one or more additional solutions which yield the same minimum total cost.

This description of the transportation problem makes obvious its importance in the field of commercial applications. For this reason, programs have recently been written by the IBM staff for the solution of the transportation problem on IBM 701, 702, 704 and 650 Electronic Data Processing Machines.

This report concerns itself with the application of the IBM 650 Magnetic Drum Calculator in obtaining a solution of the transportation type linear programming problem. Section I concerns itself with the preliminary mathematics involved in computing a minimum solution, while Section II is devoted to a description of the input and output data and other special features of the program.

Of interest to the programmer is the method used to solve the transportation problem by the IBM 650 Magnetic Drum Calculator. The iterative method employed is essentially the same as the "stepping stone" method proposed by A. Charnes and W. W. Cooper.[(1)] All operations are performed on a fixed-point basis, and all input data supplied by the programmer must be restricted to a maximum size of five digits. The latter is imperative in view of the fact that all data is stored in the form of two such five digit numbers per word on the magnetic drum. A further restriction inherent in the present program is that the number of origins must be less than 100, while the number of destinations plus the number of origins must be less than 450. Consequently, the maximum size problem which the 650 is capable of processing is explicitly defined.

Two important features of the program for the solution of the transportation problem bear mentioning. The first is the option of having the 650

[(1)] A. Charnes and W. W. Cooper: "The Stepping Stone Method of Explaining Linear Programming Calculations in Transportation Problems" - "Management Science," October, 1954 compute an initial distribution or using an initial distribution which is supplied by the programmer. The latter may be used provided that the initial distribution is not degenerate. Should this be the case, serious consideration should be given to the former in view of the fact that the program for computing the initial distribution enables the machine to handle a degenerate case.

The second feature of importance to the programmer is that necessary information may be punched out of memory at random intervals specified by the programmer, so as to enable him to restart computing in the event that one of the internal checks causes the machine to stop. This has the effect of reducing the time lost due to machine error to the interval covering operations since the last restart information was punched out of memory.

Finally, in connection with the form of all input data, it should be pointed out that all such data is entered into the machine in the same punched form. The output, in this case the solution, is punched in a form which renders it readily available for further processing on other IBM machines.

Section I: Preliminary Mathematics:

In order to illustrate the mathematics involved in solving a transportation problem, an example will be proposed and a solution arrived at according to the method programmed for the IBM 650 Magnetic Drum Calculator. As previously mentioned, this method is essentially the "stepping stone" method, although some minor modifications have been made in order to facilitate processing the problem on IBM Electronic Data Processing Machines.

Following the requirements specified in the introduction of this report, let us assume that we have $n = 3$ origins from which we wish to ship certain goods, and that the amount available at each of these origins is 5 units, 60 units and 40 units respectively. Furthermore, let us assume that there are $m = 4$ destinations to which we wish to ship these goods and that the amount demanded at each of these destinations is 35 units, 10 units, 35 units and 25 units. One condition imposed on this problem is that the total amount available must equal the total amount required. Table I shows the initialization of the problem.

Finally, let us assume that we are provided with a cost matrix as shown in Table II. Each i (row), j (column) element represents the unit cost required to ship from origin j to destination i. We have now fulfilled the necessary and sufficient requirements for solving a transportation problem. All the computations which follow are essentially made possible by the information provided in Tables I and II below.

n

m

D_i \ S_j	5	60	40
35			
10			
35			
25			

Table I

21	18	40
46	28	28
36	20	24
20	35	22

cost matrix

Table II

The first step in solving a transportation type linear programming problem is to obtain an initial distribution. This distribution may be computed from the information provided in Tables I and II, or it may be supplied by the programmer. Since a given concern will undoubtedly be operating on a relatively inexpensive basis, in many cases the initial distribution will be supplied. If the given initial distribution is not degenerate, it is preferable to use it as a starting point since the closer the initial distribution is to the optimum solution, the less time required to arrive at the minimal solution. However, if we assume for the present that the initial distribution is not given, we may proceed to compute it as follows:

Examining Table II, we commence by searching the first row of the cost matrix and selecting the minimum c_{ij} element in that row. We then proceed to the corresponding (i, j) position in Table I and enter in that position the amount D_i or S_j, whichever is smaller. If the D_i requirement is fulfilled, we proceed to the next row of the cost matrix and repeat this procedure. Otherwise, we return to the cost matrix, select the next minimum c_{ij} in the same row and enter in the corresponding (i, j) position in Table I either an amount equal to S_j or what remains to be filled at D_i, whichever is smaller. We continue in this same manner row by row in the cost matrix until we have completed the last row. Once we have reached this point, we should have the initial distribution, and the example chosen here does yield such an initial distribution. However, there is one case which we shall cover which does not yield a satisfactory initial distribution and must, consequently, be treated in a slightly different manner. Tables III and IV show the results of the above method for computing the initial distribution. The values which are circled in the cost matrix represent those minimum c_{ij}'s which were used to compute Table III. The results in

D_i \ S_j	5	60	40
35		35	
10		10	
35		15	20
25	5		20

Table III

21	18	40
46	28	28
36	20	24
20	35	22

cost matrix

Table IV

Table III show that all the boundary conditions have been satisfied and that, consequently, an initial distribution has been computed.

Close examination of Table III reveals the fact that there are (m + n -1) basis elements. If this condition is not satisfied, the problem is said to be degenerate and steps must be taken to remedy this situation. The example given below in Tables V and VI is degenerate in view of the fact that the above method yields only four basis elements instead of the required six elements.

D_i \ S_j	5	4	2
5	5		
4		4	
1			1
1			1

Table V

2	7	3
5	1	4
3	6	2
5	3	4

cost matrix

Table VI

The degenerate case may be handled as follows: We select an $\epsilon > o$ and add ϵ to each of the S_j's. We then add $n\epsilon$ to D_1 and proceed in exactly the same manner as above. If we choose $\epsilon = .001$, we obtain Table VII, whereupon rounding, we conclude with Table VIII. There are now two elements in the basis distribution which are zero. The problem is still degenerate but, due to the positions these zeros occupy, we are able to handle the problem and arrive at an optimum solution. If the problem is originally degenerate, this implies that there exists an infinite number of optimum solutions. We wish to arrive at one such solution and must, therefore, formulate an initial distribution according to the prescribed method above.

$D_i \backslash S_j$	5.001	4.001	2.001
5.003	5.001		.002
4		4	
1			1
1		.001	.999

Table VII

$D_i \backslash S_j$	5	4	2
5	5		0
4		4	
1			1
1		0	1

Table VIII

Having obtained an initial distribution in Table III, we will now proceed according to the flow chart shown in Figure I and obtain and optimum solution, i.e., a solution which minimizes the total cost of shipping. As shown in Figure I, there are four phases per iteration in obtaining such a solution. Although the method should be considered in its entirety, each of these will be covered separately for purposes of clarification. Table IX is first constructed from Tables III and IV since, with the exception of the cost matrix, it will be convenient to consider only those elements which are directly related to the initial distribution.

i	j	c_{ij}	x_{ij}	$c_{ij}x_{ij}$
1	2	18	35	630
2	2	28	10	280
3	2	20	15	300
3	3	24	20	480
4	1	20	5	100
4	3	22	20	440

Initial Distribution

Table IX

We are now in a position to proceed with the (uv) phase of the program. We wish to compute a u_i and a v_j for each of the elements in the basis table, i.e., there will be m u_i's and n v_j's, such that:

(1) $$u_i + v_j = c_{ij}$$

We start by assigning to u_1 an arbitrary value M. An initial pass is then made through Table IX, the basis table, and u_i or v_j is computed depending on whether or not a u or a v is available for any particular i or j. Provided we have not already completed the (u, v) table, a second pass is made through the basis table. We continue to make passes through the basis table until the (u, v) table is completed. Table X shows the completed (u, v) table for our example. The subscripts on the numbers indicate the pass on which they were obtained.

i or j	u_i	v_j
1	+100	-80_2
2	$+110_1$	-82_1
3	$+102_1$	-78_1
4	$+100_1$	X

Table X

For example, originally we started with i = 1; j = 2. We set M = 100 so that $u_1 = 100$. This enabled us to compute v_j from equation (1), i.e., $v_2 = c_{12} - u_1$, $v_2 = 18 - 100$ or -82. We then proceeded to the case where i = 2, j = 2 in the basis table. Since we first formed $v_2 = -82$, this enabled us to compute u_2. In this manner, two passes through the basis table were sufficient to complete the uv table.

We now pass to the (w_{ij}) phase of the problem. For each c_{ij} in the cost matrix, we wish to compute a w_{ij} which satisfies the following relationship:

(2) $$w_{ij} = u_i + v_j - c_{ij}$$

Table XI shows the results of these calculations, and gives us a maximum $W_{IJ} = 4$; I = 2; J = 3; and $C_{IJ} = 28$,

-1	0	-18
-16	0	+4
-14	0	0
0	-17	0

w_{ij}

Table XI

where the capital letter subscripts are used to denote the maximum w_{ij}.

Following the completion of the (w_{ij}) phase, we proceed next with the $\bar{\mu}$ table. We must first compute the elements shown in Table XII. Each $\bar{u}_i$ stands for the number of i entries in the basis table, while each $\bar{v}_j$ stands for the number of j entries in the basis table. Thus, there is one i = 1 entry, one i = 2 entry, two i = 3 entries, etc. and similarly for the j's there is one j = 1, three j = 2, etc. As in the uv table, there will be m $\bar{u}_i$'s and n $\bar{v}_j$'s.

Calculating μ_i and ν_j is performed in the following manner:[1] We first enter +1 in Table XII for μ_I and ν_J. Proceeding in the order of the basis elements, we pass through the $(\bar{u}_i, \bar{v}_j)$ part of Table XII asking ourselves first, is $\bar{u}_i = 1$? If the answer is in the affirmative we set $\bar{v}_j' = \bar{v}_j - 1$, $\bar{\mu} = \mu_i$ and $\nu_j' = \nu_j - \mu_i$. If the answer to the above question is "no," we ask if $\bar{v}_j = 1$. For a positive answer, we set $\bar{u}_i' = \bar{u}_i - 1$, $\mu_i' = \mu_i - \nu_j$, and $\bar{\mu} = \nu_j$.

For example, starting with i = 1, j = 2 in the basis table, we see that $\bar{u}_1 = 1$. We, therefore, set $\bar{u}_1' = \bar{u}_1$ and $\bar{v}_2' = \bar{v}_2 - 1$ or 2. We then set $\bar{\mu}_1 = \mu_1 = o$ and $\nu_2' = \nu_2 - \mu_1 = o$. As explained above, we follow a similar procedure for the case where $\bar{u}_i \neq 1$ and $\bar{v}_j = 1$. In order to obtain all the $\bar{\mu}$'s, more than one pass through the basis table will usually be required. However, in this example, one pass was sufficient to complete the $\bar{\mu}$ table.

[1] The μ_i and ν_j tables are first reset to zero.

i	j	$\bar{\mu}$	x_{ij}
1	2	0	35
2	2	+1	10
3	2	-1	15
3	3	+1	20
4	1	0	5
4	3	0	20

$\bar{\mu}$

Table XIII

i or j	$\bar{u}_i$	$\bar{v}_j$	μ_i	ν_j
1	1	1	0	0
2	1	~~3~~ ~~2~~ 1	+1	-1
3	~~2~~ 1	~~2~~ ~~1~~ 0	+1	~~+1~~ 0
4	~~2~~ 1	X	0	0

Table XII

The first part of the final phase consists of selecting the minimum x_{ij} corresponding to those $\bar{\mu}$'s which are +1. In our example, this minimum x_{ij} is $x_{22} = 10 \equiv \Theta$. We now replace the element i = 2, j = 2; $c_{ij} = 28$ in the basis table with the element I = 2; J = 3; $c_{ij} = 28$. We then adjust those x_{ij}'s for which $\bar{\mu} = +1$ by $\mp \Theta$. This retains the proper row and column balance between the S_j's and the D_i's as seen in Table XIV below.

	5	60	40
35		35	
10		10	
35		15	20
25	5		20

	5	60	40
35		35	
10			10
35		25	10
25	5		20

Table XIV

The program then transfers back to the (u, v) phase and passes through the (w_{ij}) phase. Now the maximum w_{ij} is zero which implies that the minimal solution has been obtained. The total cost has been reduced from $2,230 to $2,190.

This essentially completes the mathematical description of the iterative process. Generally, several iterations will be necessary before arriving at the minimal solution. The time required to solve such a problem is obviously a function of the size of the problem and the number of iterations. The example above required approximately ten seconds per iteration, while the m = 5 by n = 10 problem shown in the back of this report required approximately 35 seconds per iteration.

Section II: Special Program Features:

The complete input deck both for the case where the initial distribution is computed and for the case where the initial distribution is supplied by the programmer is shown in Figure II. Deck A contains the necessary loading routines and the program for computing the initial distribution. Deck B contains the program for computing the minimum solution as does Deck C. In addition to this, Deck C also contains the necessary loading routines. All three decks are fixed and, therefore, require no alterations. The programmer need only punch a card containing m and n, the cards containing his D_i's and S_j's, and his cost matrix. If the programmer prefers to use his own initial distribution, he must also punch Σcx and ΣS on the same card containing m and n, and he must enter his initial distribution on the drum in the proper locations.

There are essentially three forms of output. The final solution is punched on two separate card forms. The first card contains t, the iteration count and the final cost, and is of the form indicated in Figure IVd. There will also be m + n - 1 cards of the form indicated in Figure IVc, each card containing one i, j, c_{ij}, x_{ij} element of the solution. In the event that alternate optimum solutions are desired, I', J', and $C_{I'J'}$ are punched on cards of the form indicated in Figure IVb for each non basis element for which w_{ij} is equal to zero. These cards will then become the input for computing the alternate optimum solutions on a future run using a separate program, to be completed soon.

The third classification of output is the restart information. If the sign of the storage entry switches is negative, restart cards of the form indicated by figure IVa will be punched at the end of the Θ phase of the program. The i, j, c_{ij} elements will first be punched, seven words per card. Following these cards, the x_{ij} elements will be punched two to a word and seven words per card. In addition, t and the cost will be punched on a card of the form indicated in Figure IVd. In the event that during the modification check (see Figure I), a machine error is detected, the machine stops and the programmer may then transfer to the restart procedure. The restart deck is shown in Figure III and is loaded into the machine; control is then transferred to the beginning of the uv phase. This enables the programmer to effectively reduce the amount of time lost due to machine error.

The program herein described is being modified so that the cost matrix for small or medium sized problems, e.g., m = 60, n = 30, can be entirely stored on the drum.

In way of conclusion, a transportation problem with m = 10 by n = 5 has been solved and the results are shown on the last page of this report. The time required per iteration of this problem was approximately 35 seconds.

Basic Flow Chart

Transportation Problem

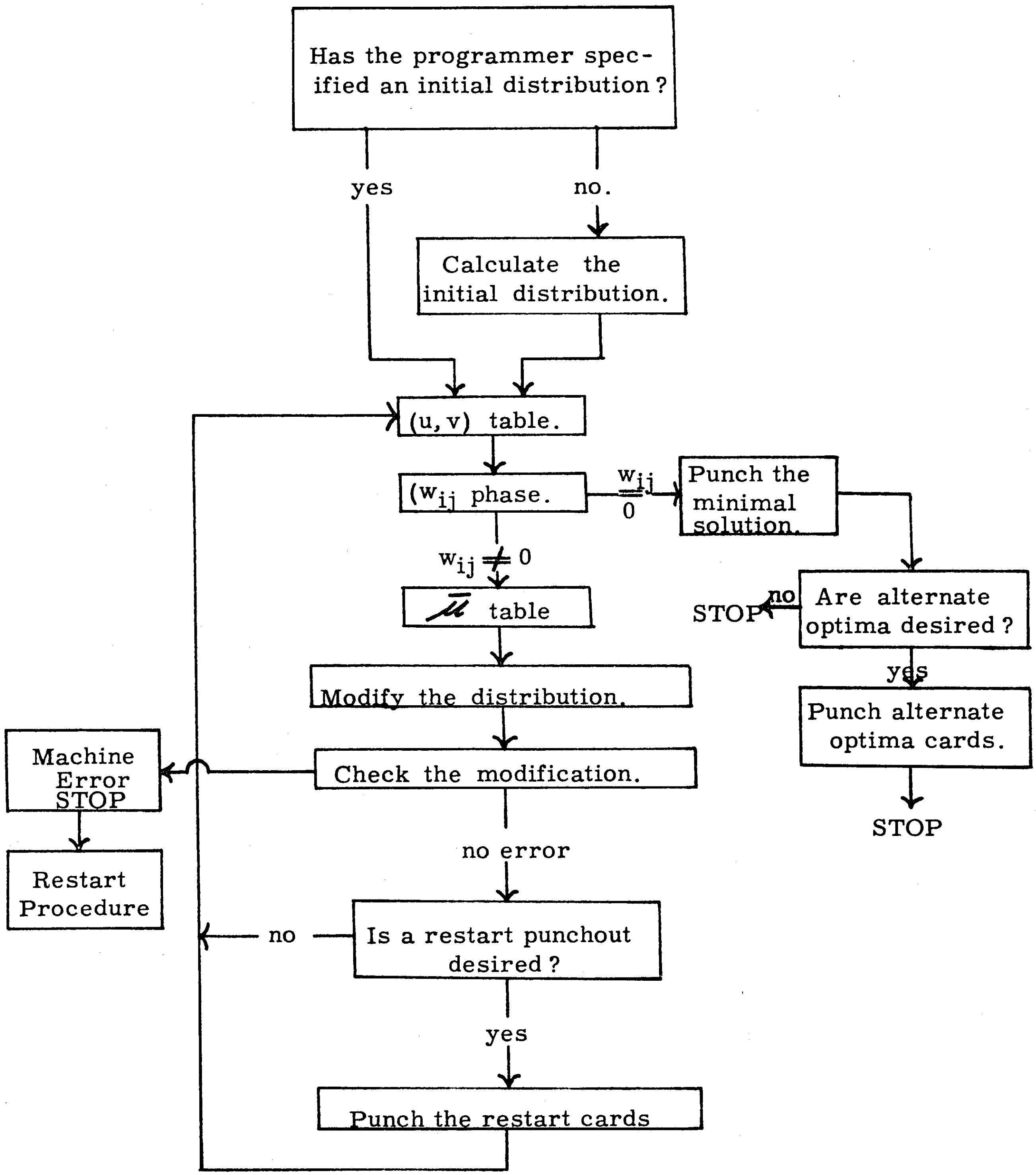


FIGURE I.

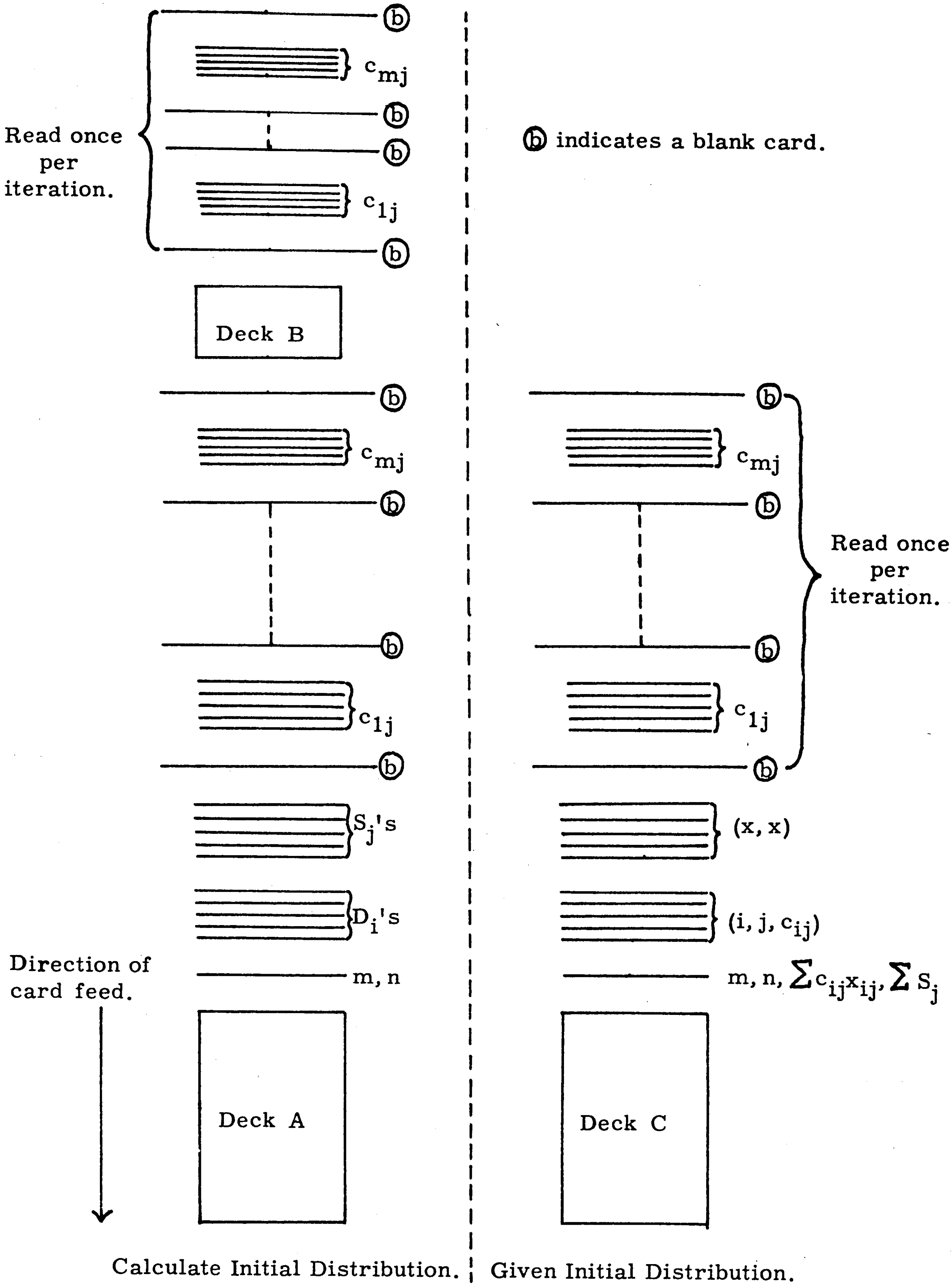


FIGURE II: Description of the Input Deck.

Restart Procedure

Input Deck

ⓑ indicates a blank card.

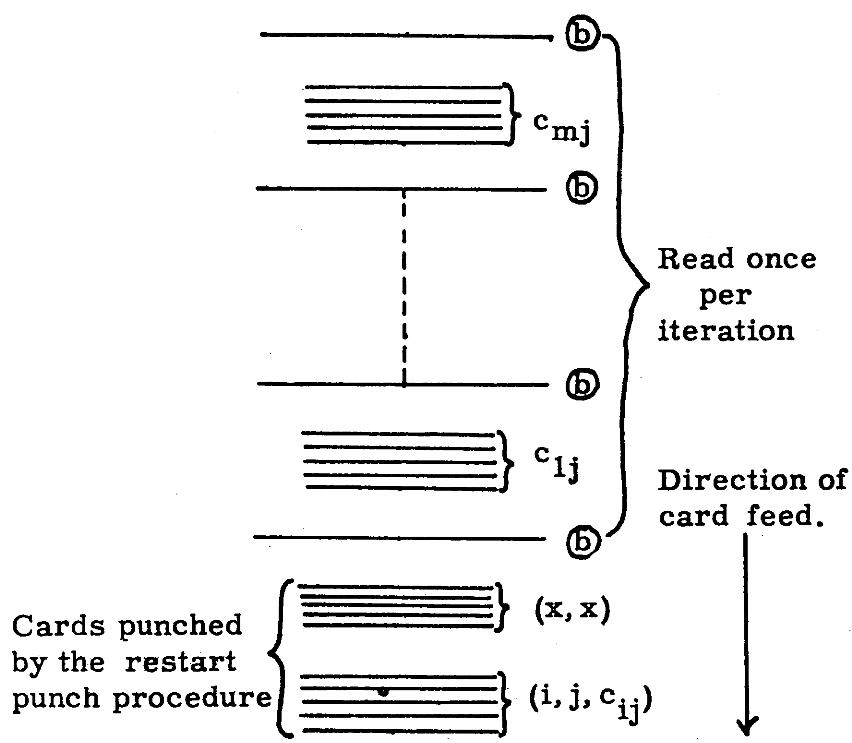
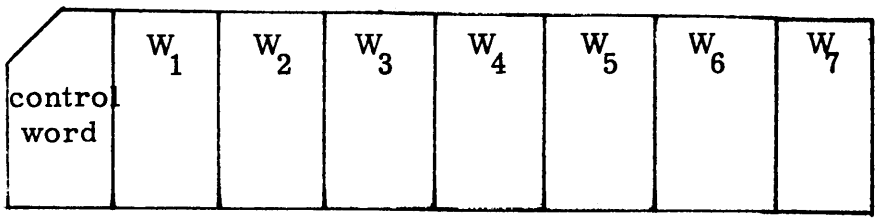


FIGURE III.

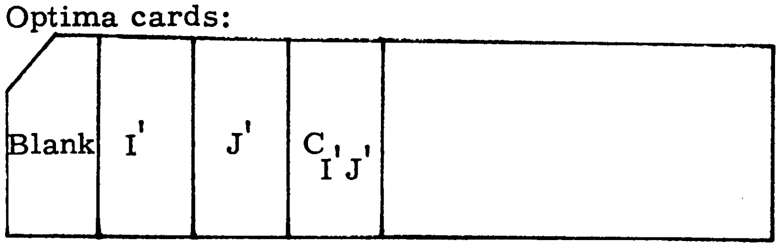
Output Form

(A) Restart punch out:

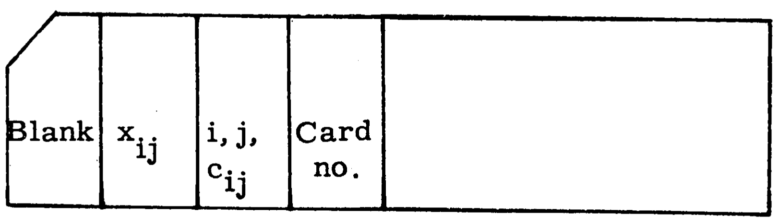


$$W_p = (i, j, c_{ij}) \text{ or } (x, x)$$

(B) Alternate Optima cards:



(C) Final Punch Out:



(D) Cost card:

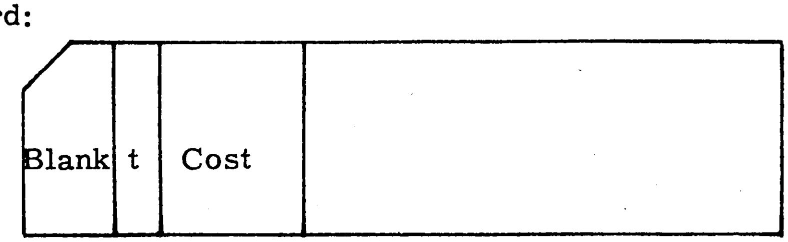


FIGURE IV.

Final Solution for m = 10, n = 5 Transportation Problem

Showing the Decrease in Cost Per Iteration

No. Iterations	Total Cost
1	$4480
2	3950
3	3950
4	3950
5	3390
6	3390
7	3220
8	3220
9	3130
10	3090
11	3090
12	3070

Origin	Destination	Quantity	Unit Cost
1	2	30	14
1	3	0	19
2	1	30	9
2	3	10	19
2	4	0	28
2	8	10	35
3	3	20	9
3	7	20	0
4	4	20	23
4	5	10	19
4	6	20	19
4	9	10	31
5	9	10	16
5	10	10	16

Note: See Table XV on the following page.

D_i \ S_j	30	50	40	60	20
30		30			
30	30				
30	0	10	20		
20		0		20	
10				10	
20				20	
20			20		
10		10			
20				10	10
10					10

Table XV

INDEXING ACCUMULATORS FOR THE IBM TYPE 650 MDDPM

George R. Trimble, Jr. and Dura W. Sweeney
International Business Machines Corporation

General Description

Many problems require the same operations to be performed on ordered arrays of data. When this is the case a large amount of address arithmetic must be done to modify instructions so that they will operate upon the proper data. Index registers are devices which will automatically modify addresses and greatly facilitate the necessary address arithmetic.

The Indexing Accumulators provided for the IBM Type 650 MDDPM incorporate all of the characteristics usually found in index registers plus the ability to be used as separate accumulators. As accumulators they may be used for accumulating small totals, holding group multipliers, or as small high speed storage devices. Programming is simplified, the number of instructions required is reduced, and, therefore, programming errors are reduced. Since fewer instructions are executed the problem solution time will be less. Also, the logic of a program using Indexing Accumulators is simpler than the logic for a corresponding non-indexed program. This, of course, eases the burden on the programmer and tends toward faster, more accurate programming.

Three Indexing Accumulators (I.A.) are provided for the 650. Each I.A. contains four decimal digits and an associated algebraic sign. Factors may be added to or subtracted from the contents of an I.A., or new factors may be inserted in an I.A. by reset add or reset subtract operations. It is possible to test each I.A. for a zero or non-zero, or a positive or negative state by means of branch operations. Each I.A. is addressable so that its contents may be used as a factor in other operations. The primary use of I.A. however, is to automatically modify addresses of instructions.

Addresses of Indexing Accumulators

The addresses assigned to the I.A. are as follows:

I.A.	Address
A	8005
B	8006
C	8007

These addresses may be used as the instruction address of any instruction or as the data address of the following instructions: 00-02, 10-11, 14-19, 30-49, 54, 60-61, 64-69, 90-99. Use of 8005, 6, or 7 as the data address of any other operation will cause a storage selection error. When one of these addresses is used as an instruction address or as the data address of a branch instruction, the next operation executed will be a NO OP whose instruction address is the contents of the I.A. addressed. For example, suppose I.A. A contains +1234 and the operation 65 0100 8005 is given. Following the reset add as specified by code 65, the operation 00 0000 1234 will be executed. Thus, the contents of I.A. A specifies that the next instruction is to be executed from location 1234.

8005, 6, or 7 may be used as the data address of the instructions 00-01, but nothing will happen since these instructions do not use the data address. Since the shift instructions only use the units digit of the data address an 8005, 6, or 7 data address on codes 30, 31, 35, 36 causes normal shifting of 5, 6, or 7 places respectively.

Use of 8005, 6, or 7 as the data address of any of the other instruction will cause the contents of the I.A. to be used as a factor in the operation. When used in this manner the four digits of the I.A. will be in the four low order digits of a word which has six zeros in the high order positions. For example, consider the instruction 65 8006 1234. This instruction will cause the contents of I.A. B to be reset added into the four low order positions of the lower accumulator and zeros to be inserted elsewhere. Since the addition is performed via the distributor, it will also contain six zeros and the contents of I.A. B in the four low order positions. Signs are manipulated just as with any other word.

Automatic Address Modification

The primary use of I.A. is to automatically modify addresses by adding the contents of an I.A. to an address. Since the I.A. can contain either positive or negative values, addresses can be modified by adding to them or subtracting from them depending on the sign of the I.A. Both data addresses and/or instruction addresses can be modified by the contents of any I.A. or by two different I.A.

It is necessary to tag each address by an indicator so that the 650 may know which I.A. should be added to the address. Addresses 2000 through 7999 have been reserved for this purpose. A "Basic" drum address is defined to be one in the range 0000-1999. In order to tag the basic drum address either 2000, 4000, or 6000 is added to indicate that the contents of I.A. A, B, or C respectively is to be added to the basic drum address. Tagging of high speed storage addresses is accomplished by adding 200, 400, or 600 to the basic high speed storage address to indicate the use of I.A. A, B, or C respectively.

The "Effective" address is that address which results after a basic address has been modified by the contents of an I.A. The following table lists all meaningful actual addresses and the resulting effective addresses.

Actual Address	Effective Address
0000-1999	0000-1999
2000-3999	0000-1999 + Contents of I.A.A
4000-5999	0000-1999 + Contents of I.A.B
6000-7999	0000-1999 + Contents of I.A.C
8000-8003	8000-8003
8005-8007	8005-8007
8010-8015	8010-8015
9000-9059	9000-9059
9200-9259	9000-9059 + Contents of I.A.A
9400-9459	9000-9059 + Contents of I.A.B
9600-9659	9000-9059 + Contents of I.A.C

The effective address determined as indicated in the above table must be a meaningful address for the operation called for. The following table lists the possible address that may be used with each meaningful operation code.

Addresses	Code	Description
0000-1999	D	Drum
8000-8003	A	Arithmetic Unit and Console Switches
8005-8007	I	Indexing Accumulator
8010-8015	T	Tapes
9000-9059	B	Buffer

Op Code	Abbvr.	Meaningful Data Address	Name
00	NO OP	D,A,I,T,B	No Operation
01	STOP	D,A,I,T,B	Stop
02	FASN	D,A,I,B	Floating Add Suppress Normalization
03	RCT	T	Read Check Tape Record
04	RT	T	Read Tape Record
05	RAT	T	Read Alphanumeric Tape Record
06	WT	T	Write Tape Record
07	WAT	T	Write Alphanumeric Tape Record
08	LBB	D	Load Buffer Block
09	LB	D	Load Buffer
10	AU	D,A,I,B	Add Upper
11	SU	D,A,I,B	Subtract Upper
12	Not Used		
13	Not Used		
14	DIV	D,A,I,B	Divide
15	AL	D,A,I,B	Add Lower
16	SL	D,A,I,B	Subtract Lower
17	AABL	D,A,I,B	Add Absolute Lower
18	SABL	D,A,I,B	Subtract Absolute Lower
19	MPY	D,A,I,B	Multiply
20	STL	D,B	Store Lower
21	STU	D,B	Store Upper
22	ST DA	D,B,8001	Store Data Address
23	ST IA	D,B,8001	Store Instruction Address
24	ST D	D,B	Store Distributor
25	Not Used		
26	Not Used		
27	SET	B	Set Buffer Address
28	ST BB	D	Store Buffer Block
29	ST B	D	Store Buffer
30	SRT	D,A,I,T,B	Shift Right
31	SRD	D,A,I,T,B	Shift and Round
32	FA	D,A,I,B	Floating Add
33	FS	D,A,I,B	Floating Subtract
34	FD	D,A,I,B	Floating Divide
35	SLT	D,A,I,T,B	Shift Left
36	SCT	D,A,I,T,B	Shift and Count
37	FAAB	D,A,I,B	Floating Add Absolute
38	FSAB	D,A,I,B	Floating Subtract Absolute
39	FM	D,A,I,B	Floating Multiply
40	BNZA	D,A,I,B	Branch Non Zero A
41	BMNA	D,A,I,B	Branch Minus A
42	BNZB	D,A,I,B	Branch Non Zero B
43	BMNB	D,A,I,B	Branch Minus B
44	BRNZU	D,A,I,B	Branch Non Zero Upper
45	BRNZ	D,A,I,B	Branch Non Zero
46	BRMIN	D,A,I,B	Branch Minus
47	BROV	D,A,I,B	Branch Overflow
48	BNZC	D,A,I,B	Branch Non-Zero C

49	BMNC	D,A,I,B	Branch Minus C
50	AA	D,A,B	Add A
51	SA	D,A,B	Subtract A
52	AB	D,A,B	Add B
53	SB	D,A,B	Subtract B
54	BRNEF	D,A,I,B	Branch Not End of File
55	RWD	T	Rewind
56	WTM	T	Write Tape Mark
57	BSP	T	Backspace
58	AC	D,A,B	Add C
59	SC	D,A,B	Subtract C
60	RAU	D,A,I,B	Reset Add Upper
61	RSU	D,A,I,B	Reset Subtract Upper
62	Not Used		
63	Not Used		
64	DIV RU	D,A,I,B	Divide Reset Upper
65	RAL	D,A,I,B	Reset Add Lower
66	RSL	D,A,I,B	Reset Subtract Lower
67	RAABL	D,A,I,B	Reset Add Absolute Lower
68	RSABL	D,A,I,B	Reset Subtract Absolute Lower
69	LD	D,A,I,B	Load Distributor
70	RD	D,B	Read (533)
71	PCH	D,B	Punch (533)
72	Not Used		
73	Not Used		
74	Not Used		
75	RD PRT	D,B	Read (407)
76	RC PRT	D,B	Read Conditional (407)
77	PRT	D,B	Print (407)
78	Not Used		
79	Not Used		
80	RAA	D,A,B	Reset Add A
81	RSA	D,A,B	Reset Subtract A
82	RAB	D,A,B	Reset Add B
83	RSB	D,A,B	Reset Subtract B
84	TLU	D,B	Table Look Up
85	Not Used		
86	Not .sed		
87	Not Used		
88	RAC	D,A,B	Reset Add C
89	RSC	D,A,B	Reset Subtract C
90	BRD 10	D,A,I,B	Branch Distributor Digit 10
91	BRD 1	D,A,I,B	Branch Distributor Digit 1
92	BRD 2	D,A,I,B	Branch Distributor Digit 2
93	BRD 3	D,A,I,B	Branch Distributor Digit 3
94	BRD 4	D,A,I,B	Branch Distributor Digit 4
95	BRD 5	D,A,I,B	Branch Distributor Digit 5
96	BRD 6	D,A,I,B	Branch Distributor Digit 6
97	BRD 7	D,A,I,B	Branch Distributor Digit 7
98	BRD 8	D,A,I,B	Branch Distributor Digit 8
99	BRD 9	D,A,I,B	Branch Distributor Digit 9

Address modification is accomplished by adding the contents of an I.A. to a basic address. If the contents of the I.A. is positive and the resulting effective address exceeds 9999, the carry would be lost and only the four low order digits of the sum would be kept as the effective address. If the contents of the I.A. is negative, the address is modified by subtraction. Since subtraction is accomplished by adding the 10's complement a carry will always occur when the difference is positive. As above, any such carrys are lost. If indexing by subtraction should result in a "negative" address, the complement result will not be recomplemented. This may result in a storage selection error if the complement is not a meaningful address.

<u>Examples</u>

Actual Instruction	Contents of I.A. A	B	C	Indexed Instruction	Remarks
65 0123 0124	0223+	0075-	0062+	65 0123 0124	No indexing
65 2123 0124	0223+			65 0346 0124	Index D by A
65 0123 6124			0062+	65 0123 0186	Index I by C
65 4123 4124		0075-		65 0048 0049	Index D and I by B
65 4123 6124		0075-	0062+	65 0048 0186	Index D by B and I by C
65 9215 8002	0013+			65 9028 8002	Index D by A
65 0123 9627			0015-	65 0123 9012	Index I by C
65 4015 0124		2345+		65 2360 0124	2360 causes storage selection error
65 9210 0124	1983+			65 0993 0124	D exceeds 10,000 Carry is lost
65 0123 4124		7878+		65 0123 8002	I becomes 8002
65 9615 9218	1011-		0015-	65 9000 8007	I becomes 8007
65 2123 0124	1011-			65 9112 0124	D becomes "negative". Complement 9112 causes Storage Selection Error
65 2123 0124	1111-			65 9012 0124	D becomes "negative". Complement is meaningful, however.

Testing of Indexing Accumulators

The following are the operations by means of which the contents of the indexing accumulators may be tested.

40 BNZA Branch Non Zero I.A.A

If I.A.A contains zeros the next instruction will be taken from the location specified by the instruction address. If the contents of I.A.A are not zero, the next instruction will be taken from the location specified by the data address.

41 BMNA Branch Minus I.A.A

If the sign of I.A.A is plus the next instruction will be taken from the location specified by the instruction address. If it is minus the next instruction will be taken from the location specified by the data address.

42 BNZB Branch Non-Zero I.A.B

If I.A.B contains zeros the next instruction will be taken from the location specified by the instruction address. If the contents of I.A.B are not zero, the next instruction will be taken from the location specified by the data address.

43 BMNB Branch Minus I.A.B

If the sign of I.A.B is plus the next instruction will be taken from the locations specified by the instruction address. If it is minus the next instruction will be taken from the location specified by the data address.

48 BNZC Branch Non-Zero I.A.C

If I.A.C contains zeros the next instruction will be taken from the location specified by the instruction address. If the contents of I.A.C are not zero, the next instruction will be taken from the location specified by the data address.

49 BMNC Branch Minus I.A.C

If the sign of I.A.C is plus the next instruction will be taken from the location specified by the instruction address. If it is minus the next instruction will be taken from the location specified by the data address.

Operations Upon Indexing Accumulators

The effective address of those instructions which operate upon I.A. must be 0000-1999, 8000-8003, or 9000-9059. This effective address specifies what the data is that is to be used in the operation. If the effective address is in the range 0000-1999, the data used by the operation is the actual effective address. If it is in the range 8000-8003, or 9000-9059, the data used by the operation will be the four low order digits and sign of the storage location specified by the address. The meaning of these statements will become clear in the examples which follow. The instruction address has its usual meaning.

50 AA Add to I.A.A

The data specified by the effective data address will be added to the contents of I.A.A.

Examples

Actual Instruction	Indexed Instruction	Contents of I.A.A. Before	After	Contents of I.A.B.	Contents of 8000-8003 or 9000-9059
50 0001 0123	50 0001 0123	0500+	0501+		
50 1623 0123	50 1623 0123	0500+	2123+		
50 2000 0123	50 0500 0123	0500+	1000+		
50 2156 0123	50 0656 0123	0500+	1156+		
50 4000 0123	50 0111 0123	0500+	0611+	0111+	
50 4265 0123	50 0154 0123	0500+	0654+	0111-	
50 8002 0123	50 8002 0123	0500+	1611+		7777771111+
50 9007 0123	50 9007 0123	0500+	7277-		1111117777-
50 9407 0123	50 9004 0123	0500+	1734+	0003-	0202021234+
50 2156 2123	50 0656 0623	0500+	1156+		

These examples show that it is possible to simply add a constant (in the range 0000-1999) to A as in the first two examples, to add A to itself as in the third example, to add A to itself and to another constant (in the range 0000-1999) as in the fourth example, to add B to A as in the fifth example, or B to A and to another constant as in the sixth example. The next three examples show how it is possible to add to A from the four low order positions of a high speed storage location. The last example illustrates how addresses are modified before the operation is executed. Thus D and I are increased by 0500 before the contents of I.A.A are modified.

As described previously the effective address must be of type D,A, or B. The operations performed to obtain this effective address are exactly the same here as with any other kind of instruction. The rules regarding carries and complement (negative) addresses still apply. The final addition to I.A.A is algebraic however when the effective address is type A or B and all of the normal rules regarding signs are true. If the effective address is of Type D, this address is always treated as though it were plus.

All of the other instructions which are described in the following paragraphs operate in an analogous manner. Only a few examples for each will be given. It is left as an exercise for the reader to work out exactly what happens under each of the many possible conditions. The general rules given above are completely sufficient for working out any conceivable conditions.

52 AB Add to I.A.B

The data specified by the data address will be added to the contents of I.A.B.

58 AC Add to I.A.C

The data specified by the data address will be added to the contents of I.A.C.

51 SA Subtract from I.A.A

The data specified by the data address will be subtracted from the contents of I.A.A.

Examples

Actual Instruction	Indexed Instruction	Contents of I.A.A Before	After	Contents of I.A.B	Contents of 8000-8003 or 9000-9059
51 0001 0123	51 0001 0123	0500+	0499+		
51 2000 0123	51 0500 0123	0500+	0000+		
51 4250 0123	51 0125 0123	0500+	0375	0125-	
51 8000 0123	51 8000 0123	0500+	7390-		1234567890+
51 9207 0123	51 9057 0123	0050+	7940+		1234567890-

53 SB Subtract from I.A.B

The data specified by the data address will be subtracted from the contents of the I.A.B.

59 SC Subtract from I.A.C

The data specified by the data address will be subtracted from the contents of I.A.C.

The following operations are analogous to the previous group except that the I.A. is reset to zero before the data is added or subtracted into it.

80 RAA Reset Add to I.A.A

I.A.A will be reset to zero and the data specified by the data address will be added to it.

Examples

Actual Instruction	Indexed Instruction	Contents of I.A.A Before	Contents of I.A.A After	Contents of I.A.C	Contents of 8000-8003 or 9000-9059
80 0000 0123	80 0000 0123	1234-	0000+		
80 1520 0123	80 1520 0123	1234-	1520+		
80 6000 0123	80 0175 0123	1234-	0175+	0175+	
80 6525 0123	80 0350 0123	1234-	0350+	0175-	
80 9027 0123	80 9027 0123	1234-	5021-		001212345021-

82 RAB Reset Add to I.A.B

I.A.B will be reset to zero and the data specified by the data address will be added to it.

88 RAC Reset Add to I.A.C

I.A.C will be reset to zero and the data specified by the data address will be added to it.

81 RSA Reset Subtract from I.A.A

I.A.A will be reset to zero and the data specified by the data address will be subtracted from it.

Examples

Actual Instruction	Indexed Instruction	Contents of I.A.A Before	After	Contents of I.A.C	Contents of 8000-8003 or 9100-9059
81 1234 0123	81 1234 0123	0527+	1234-		
81 7015 0123	81 1927 0123	0527+	1927-	0912+	
81 7015 0123	81 0103 0123	0527+	0103-	0912-	
81 9059 0123	81 9059 0123	0527+	2301+		0123012301-

83 RSB Reset Subtract from I.A.B

I.A.B will be reset to zero and the data specified by the data address will be subtracted from it.

89 RSC Reset Subtract from I.A.C

I.A.C will be reset to zeros and the data specified by the data address will be subtracted from it.

IBM TYPE 650 MAGNETIC TAPE ATTACHMENT

Dura W. Sweeney and George R. Trimble, Jr.
International Business Machines Corporation

The Magnetic Tape Attachment for the 650 consists of the following units:

1 Type 653 High Speed Storage Unit (Buffer).

1 Type 652 Tape Control Unit.

1 to 6 Type 727 Magnetic Tape Drives.

The 653 High Speed Storage Unit is used as a buffer between the magnetic tapes and the drum. It is not necessary to transfer the data read from tape onto the drum, however. In general, it will be better to operate on the data while it remains in the buffer. In this manner records will be read into the buffer, operated on while they are still in the buffer, and written on an output tape, without ever going into drum storage.

Each of the tape drives has an address. They are 8010 through 8015. The address assigned to a tape drive is controlled by a switch on the tape drive. Thus two tapes can be assigned the same address for writing two identical output tapes for checking purposes.

The 702-705 character code is used for recording information on the tapes. Automatic translation from the 702-705 code to and from the 650 code is provided. Except for the restrictions imposed on records by the fixed length numeric words and 60 word buffer of the 650, the characteristics of the magnetic tape records are identical with records recorded by the 702-705. Both horizontal and vertical redundancy checking are provided as well as the speed, density, and capacity which characterize the 727 tape drive.

Nine new operation codes are provided to control the magnetic tape functions. They are as follows:

03 RC (Read Check)

The next record on the tape specified by the data address is read and a horizontal and vertical redundancy check is made. Failure to pass these checks will cause an error indication. Since this operation does not require reading the tape record into the buffer, the contents of the buffer are not disturbed.

04 R (Read)

The next record on the tape specified by the data address is read into the buffer, the first word entering the word of the buffer to which the buffer ring had been set by the previous program. Succeeding tape words will read into succeeding buffer words until word 9059 has been filled with the last tape word. All data read by this instruction must be pure numeric data. The buffer ring will be left set at 9000.

The lowest order digit of each tape word has a zone indication attached to it which is translated to be the sign of the word read into the buffer. A "12" zone is translated to plus, and an "11" zone is translated to minus.

The number of words read from the tape must be exactly the number required to fill the buffer from the position to which it is set to the end of the buffer. More or less than this number of words will cause an error stop.

Thus it is possible to have numeric tape records of 1, 2, 3, etc. up to 60 words in length. On a 2400 tape this means that from 36,000 one word records (360,000 digits) to 7680 sixty word records(4,608,000 digits) can be recorded.

Examples

```
0100:  27  9000  0101  Set buffer ring at 9000
0101:  04  8010  0102  Read 60 numeric words from tape #1
                       into buffer words 9000-9059.
(More or less than 60 words will cause an error stop)
The buffer ring will be left set at 9000.

0100:  27  9045  0101  Set buffer ring at 9045
0101:  04  8010  0102  Read 15 numeric words from tape #1
                       into buffer words 9045-9059.
(More or less than 15 words will cause an error stop).
The buffer ring will be left set at 9000.
```

05 RA (Read Alphanumeric)

Since the 650 is a strictly numeric machine, alphanumeric and special characters cannot be read directly into it. In order to read alphabetic and special characters into the 650 from cards it is necessary to convert these characters to a two digit numeric code which can be stored within the 650. Similarly, when alphabetic and special characters are encountered on tape, it is necessary to convert them to a two digit code to enter them in the 650. Some indication must be given to the 650 whenever such a character is encountered so that it might know that this character must be converted to the two digit code. It is not feasible to give such an indication for each character read when the possibility of encountering such characters exists. A more economical way of accomplishing this is by giving an indication for a group of characters. Thus if an alphanumeric group is encountered, all characters within that group are converted to the two digit code.

An alphanumeric tape record used by the 650 has a special form. It must be either 10, 20, 30, 40, 50, or 60 words in length. Such a record consists of from 1 to 6 blocks, each block containing ten words. Within each block the tenth word is set aside as the control word. This word indicates which of the remaining nine words are numeric and which are alphanumeric. An eight in a particular digit position of the control word indicates that the corresponding word contains alphanumeric data. The absence of an eight indicates that the corresponding word contains pure numeric data.

When a RA instruction is given, the buffer ring must have previously been set to the beginning of a block (9000, 9010, etc., to 9050). Therefore, the length of an alphanumeric tape record is not variable by word as is a pure numeric record, but must be some multiple of ten words in length.

The data address of the RA instruction specifies the tape drive to be activated. The first ten digits of the tape record are read into the tenth word of the block to which the buffer ring had previously been set. Digit ten of the control word is ignored. Digit one is examined and if it is not an eight the next ten digits on the tape are read into the first word of the block. If it is an eight, however, the next five characters are converted to ten digits and entered into the first word of the block. Similarly, digit two is examined, etc., through digit nine. At this point the next control word is read from the tape into its position and analyzed in the same way for each of the words in its block. This process is repeated for each block until the last block has been read into 9050-9059.

The sign representation for words indicated by the control word as being numeric is the same as the case where a pure numeric record is read from tape. A word indicated by the control word as being alphanumeric will always appear as a positive word in the buffer.

As with numeric records, the number of characters read from the tape must be exactly enough to fill the buffer from the position to which it is set through 9059. More or less than enough will cause an error stop. The buffer will be left set at 9000 after this operation.

Examples:

Assume the tape record consists of four numeric records followed by five alphanumeric records. This would be recorded on the tape as follows:

0888880000nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnaaaaaaaaaaaaaaaaaaaaaaaaa

Control Word ← (Direction of Tape Motion) | Four Numeric Records of 10 digits each | Five Alphanumeric Records of five characters each.

A program to read this record is as follows:

```
0100:  27  9050  0101  Set buffer ring to 9050
0101:  05  8010  0102  Read tape record into words 9050-9059
```

The actual record on tape consists of 50 numeric characters and 25 alphanumeric characters. The 25 alphanumeric characters will be converted to 50 numeric characters making a total of 100 numeric characters read into the ten buffer words.

If a block is purely numeric it will consist of 100 numeric characters of which nine are used for control. If a block is all alphanumeric it consists of the ten numeric digits in the control word and 45 alphanumeric characters. The 45 alphanumeric characters will be converted to 90 numeric digits making a total of 100 digits for the block.

Numeric and alphanumeric words do not have to be assigned to any particular positions in a block but they can be arbitrarily interspersed. Thus word 1, 2, and 6 can be numeric while 3, 4, 5, 7, 8, and 9 are alphanumeric.

A record containing the maximum number of alphanumeric characters would be recorded as 6 control words and 54 words of alphanumeric data. Thus there would be 60 numeric digits and 270 alphanumeric characters recorded on the tape. When entered into the buffer the 270 alphanumeric characters would be converted to 540 numeric digits and the total record length would then be 600 digits. About 12,000 such records could be recorded on a reel of tape, or about 3,240,000 alphanumeric characters plus 72,000 control digits.

06 W (Write)

The W instruction is exactly analogous to the R instruction except that in this case a numeric record is written on the tape specified by the data address. Writing starts at the word to which the buffer ring had been set and continues until the end of the buffer. Thus records of from one to sixty words in length may be written depending upon where the buffer ring is set. The buffer ring will be left set at 9000 after this operation.

07 WA (Write Alphanumeric)

The WA instruction is exactly analogous to the RA instruction except that in this case an alphanumeric record is written on the tape specified by the data address. The buffer ring must have been previously set to the beginning of a block (9000, 9010, etc. or 9050). The tenth word of each block is a control word and specifies which of the remaining nine words are numeric and which are alphanumeric. As in the RA an eight indicates that the corresponding word is alphanumeric and no eight indicates numeric data. If alphanumeric data is indicated the ten digits for that word are "compressed" to five alphanumeric characters for recording on the tape. The tenth or control word of the block is written as the first word on the tape, the remaining nine words written in sequence following the control word.

54 BRNEF (Branch No End of File)

An end of file condition is caused by occurrence of one of the following:

1. Sensing a tape mark during reading (03, 04, 05).
2. Sensing reflective end of tape marker during writing (06, 07)

Occurrence of either of these conditions will turn on the Input/Output Indicator for that tape unit and also turn on an end of file indicator in the 650. This end of file indicator can then be interrogated by the BRNEF instruction to see if an end of file condition has been reached on any of the tapes.

If the end of file indicator is not on when it is interrogated by the BRNEF instruction the next instruction will be taken from the location specified by the data address. If the end of file indicator is on when it is interrogated by the BRNEF instruction the next instruction will be taken from the location specified by the instruction address and the end of file indicator, and the Input/Output indicator for that tape unit will be turned off.

Since there is only one end of file indicator which may be turned on by any tape unit it is necessary to interrogate it after a tape unit is used and before the next tape read or write instruction is given if there is the possibility of an end of file condition occurring.

If the reflective end of tape spot causes an end of file condition during writing, the record currently being written will be fully recorded. A tape mark should then be written to indicate the end of the file for reading. Successive write instructions may be given however to record another file on the same tape if desired. Care must be taken to assure that such a procedure will not cause the tape to be completely unwound from its reel.

55 RWD (Rewind)

The tape unit specified by the data address is rewound to the "Load Point". The load point is determined by a reflective spot placed at the beginning of the tape.

56 WTM (Write Tape Mark)

A tape mark is written on the tape specified by the data address.

57 BSP (Back Space)

The tape specified by the data address is backspaced one record.

During reading the information being read is checked for both horizontal and vertical redundancy. During writing the echo pulses are similarly checked. Failure to pass these checks will cause an error indication. All tape operations except the BRNEF instruction require that the data address be one of 8010-8015. Any other data address used with these instructions will cause a storage selection error.

Tape reading and writing speeds are 10 milliseconds for accelerate and decelerate time and .067 milliseconds time to read or write one character. Thus it would take 10 + 60 X .067 + 270 X .067 = 32.11 milliseconds to read or write a record containing 60 control digits and 270 alphanumeric characters.

If a tape read or write operation is given succeeding program steps (not requiring a buffer reference or another tape operation) will be executed in parallel with the read or write operation. Any succeeding instruction which requires a reference to the buffer or is another tape operation will cause an interlock so that the program will stop until the tape read or write operation is complete before continuing. Thus there could be 50 milliseconds of useful computation time in parallel with the reading of a 600 digit tape record.

IBM TYPE 650 HIGH SPEED STORAGE ATTACHMENT

Dura W. Sweeney and George R. Trimble, Jr.
International Business Machines Corporation

The High Speed Storage attachment for the 650 is contained in the Type 653 unit. It consists of 60 words, each of 10 decimal digits and an algebraic sign, of magnetic core storage. Because of the extremely low access time characteristic of magnetic core storage, it provides a source from which data and instructions may be made available. Since the access time for each of the 60 words is zero, the only time required for reference to a word in the High Speed Storage Unit is the 96 microseconds necessary to transfer the word to the arithmetic or control unit.

The High Speed Storage (HSS) is also used as the buffer for the Magnetic Tape Attachment for the 650. For this reason the terms HSS and buffer are used indiscriminately throughout.

Each of the 60 words has a four digit address associated with it by means of which the program can make reference to these words as sources of data or instructions or as destinations of results. These addresses are 9000 through 9059. Thus instructions may be executed from the High Speed Storage Unit, (HSSU), data may be operated upon from it, or intermediate and final results may be stored in it. The speed at which the 650 executes a program can be significantly increased by judicious use of the HSSU and will greatly reduce the need for the technique of optimum programming. Referring to the Sequence Chart in the article on Optimum Programming in Technical Newsletter #8 the effects of the HSSU on instruction execution times can be easily determined. Wherever a "Search for Data" or "Search for Next Instruction" segment is indicated, simply eliminate that segment if a word in the HSSU is referred to. For example, all of the "add" type instructions (10, 11, 15, 16, 17, 18, 60, 61, 65, 66, 67, 68) require only 8 word times or 768 microseconds to execute if everything is in the HSSU. The Branch Minus (46) requires either 288 or 384 microseconds. Similarly, times for each of the other operations can be determined.

In order to make efficient use of the HSS some means for transferring blocks of information between the HSSU and the drum must be provided. Four new instructions are included for this purpose. They provide an efficient method of transferring data from one set of locations to another and in addition permit a certain amount of editing of records by deletion of words either at the beginning or end of the record.

Magnetic cores are static memory devices to which a pulse must be applied to cause them to read in or out. During a block transfer it is necessary to pulse the words in the desired block consecutively causing them to read in or out in sequence. The device which accomplishes this is called a "Ring". Once the ring has been set to activate a particular word of HSS it will automatically advance to activate the next word, then the next, etc. until the complete transfer has been effected. The buffer ring actually consists of two rings, a ten position word ring and a six position block ring. The simultaneous position of both of these rings determine which of the 60 words the buffer ring is set to. For this reason the buffer can be considered as logically consisting of six-ten word blocks.

The following instruction allows the programmer to "Set" the buffer ring at any desired word so that the block transfers can be initiated starting with any of the 60 words.

27 SET (Set Buffer Ring)

The data address of the SET instruction must be one of the addresses 9000 through 9059. The buffer ring will then be set at the corresponding HSS word position. A data address other than 9000-9059 will cause a storage selection error to be indicated.

In addition, any instruction which refers to a word in HSS either for datum or an instruction will cause the buffer ring to be left set at that position.

Examples:

0560: 27 9023 0561 The buffer ring will be set at position 9023. The next instruction will be taken from location 0561. (Note that the instruction 27 9023 9012 would leave the buffer ring at 9012.)

0560: 15 9010 0561 The word in 9010 will be added into the lower accumulator and the buffer ring will be left set at position 9010.

The block transfer instructions are as follows:

08 LBB (Load Buffer Block)

The data address of the LBB instruction must be one of the drum addresses, 0000 through 1999. It specifies the location of the first word on the drum to be transferred to HSS. The first word of HSS transferred into is determined by where the buffer ring is set. Successive words on the drum are then transferred into successive words in HSS until one of the following occurs:

1. The end of a buffer block is reached. The six buffer blocks are 9000-9009, 9010-9019, 9020-9029, 9030-9039, 9040-9049, 9050-9059.

2. The end of a drum band is reached.

Words in HSS not transferred into will not be affected by this operation. The buffer ring will be left set at the last word position transferred into plus one. If the last word transferred into is 9059, however, the buffer ring will be left set at 9000. If the data address of the LBB instruction is not 0000-1999 a storage selection error will be indicated.

Examples:

```
0100:  27  9000  0101  Set buffer ring at 9000.
0101:  08  0500  0102  Transfer 0500-0509 to 9000-9009.
The buffer ring will be left set at 9010.

0100:  27  9016  0101  Set buffer ring at 9016.
0101:  08  0500  0102  Transfer 0500-0503 to 9016-9019.
The buffer ring will be left set at 9020.

0100:  27  9056  0101  Set buffer ring at 9056.
0101:  08  0500  0102  Transfer 0500-0503 to 9056-9059.
The buffer ring will be left set at 9000.

0100:  27  9000  0101  Set buffer at 9000.
0101:  08  0547  0102  Transfer 0547-0549 to 9000-9002.
The buffer ring will be left set at 9003.
```

In each of these examples the buffer ring was set by means of the SET instruction. This will not always be necessary in practice. The following examples illustrate other ways of setting the buffer ring.

```
0100:  27  9000  0101  Set buffer ring at 9000.
0101:  08  0500  0102  Transfer 0500-0509 to 9000-9009.
0102:  08  0510  0103  Transfer 0510-0519 to 9010-9019.
```

In this example the second LBB instruction makes use of the fact that the buffer ring has been left set at 9010 after the first LBB instruction. The buffer will be left set at 9020 after the second LBB instruction.

```
0100:  69  9000  0101  Load Distributor from 9000.
0101:  08  0500  0102  Transfer 0500-0509 to 9000-9009.
```

Since the Load Distributor instruction leaves the buffer ring set at 9000 it is not necessary to use the SET instruction prior to the LBB instruction.

28 STBB (Store Buffer Block)

The STBB instruction is exactly analogous to the LBB instruction except that in this case data is transferred from HSS to the drum instead of from the drum to HSS. Otherwise, all of the rules and conditions described under the LBB instruction still apply.

09 LB (Load Buffer)

The LB instruction is similar to the LBB instruction with one exception. The transfer of data continues until one of the following occurs:

1. The end of the buffer is reached.
2. The end of a drum band is reached.

Thus while the LBB instruction is a 1 to 10 word block transfer the LB instruction can cause as many as 50 words to be transferred from the drum to HSS.

Examples:

```
0100:  27  9000  0101  Set buffer ring at 9000.
0101:  09  0500  0102  Transfer 0500-0549 to 9000-9049.
The buffer ring will be left set at 9050.

0100:  27  9015  0101  Set buffer ring at 9015.
0101:  09  0500  0102  Transfer 0500-0544 to 9015-9059.
The buffer ring will be left set at 9000.

0100:  27  9000  0101  Set buffer ring at 9000.
0101:  09  0523  0102  Transfer 0523-0549 to 9000-9026.
The buffer ring will be left set at 9027.
```

29 STB (Store Buffer)

The STB instruction is exactly analogous to the LB instruction except that in this case data is transferred from HSS to the drum. Otherwise, all of the rules and conditions described under the LB instruction still apply.

Two other important features of the HSSU are the ability to perform TLU on the HSS and the ability to use one of the addresses 9000 through 9059 as the data address of the valid 70 codes which control the operation of the Type 533 and Type 407.

If the TLU operation has a 9000-9059 data address the following rules apply:

1. The TLU operation will start at the word in HSS specified by the data address. (Note that the only time the TLU operation will start at 9000 is if the data address is 9000.) For example if the instruction 0100: 84 9036 0101 is given, the TLU operation will start at 9036 and may search from 9036 to 9059. The information stored from 9000-9035 will not be searched.

2. The TLU operation considers the contents of the distributor and that section of HSS to be searched as positive.

3. The address of the word whose contents are greater than or equal to the contents of the distributor (both considered positive) is inserted in the data address positions of the lower accumulator.

4. The buffer rings are left setting at the last word searched plus one.

   For example, if the instruction 0100: 84 9036 0101 is given and the address of the number found is 9043, then the address, 9043, will be inserted in the lower accumulator and the buffer rings will be left set at 9044.
   If the instruction 0100: 84 9036 0101 is given and the address of the number found is 9059, then the address, 9059, will be inserted in the lower accumulator and the buffer rings will be left set at 9000.

5. If no number is found a storage selection indication will be made.

6. Although the TLU operation may begin at any point in HSS, the actual operation will not start until the home pulse (address 0000, 0050, etc.) is passed during a drum revolution.
   For example:

Location	Instruction	TLU Operation Starts
0547	84 9036 0548	in 3 word times.
0561	84 9010 0562	in 39 word times.
0598	84 9022 0599	in 49 word times.

If any of the valid 70 codes (70, 71, 75, 76, 77) has a 9000-9059 data address a load buffer block or store buffer block transfer will be initiated starting at the 9000-9059 data address and at word one of the read, punch, or print buffer and terminating at the end of the buffer block. The information in the read, punch, or print buffer will be transferred directly to or from the HSS without going through general storage on the drum.

For example:

0100:	70	9000	0101	Transfer 10 words of read buffer to 9000-9009
0100:	71	9036	0101	Transfer 4 word (9036-9039) to punch buffer words 1-4, words 5-10 are blank.
0100:	70	9012	9000	Transfer words 1-8 of read buffer to 9012-9019. Take 9000 as location of next instruction if non-load card. Take 9012 as location of next instruction if a load card.

There is a validity check made upon all information read out of HSS. Therefore, there will be an additional validity check performed on all information in the arithmetic or control unit if it is brought from the HSS, and there will be a validity check on all information transferred to or from the HSS

a. from or to the tape units
b. from or to general storage on the drum
c. from or to the read, punch or print buffers.

The time required for all transfer operations between the drum and the HSSU is as follows:

1. A minimum of 3 word times between the location of the instruction and the data address.

2. A minimum of (2+n) word times between the data address and the location of the next instruction. (n is the number of words transferred.)

LIST OF SUBROUTINES USED BY 650 CUSTOMERS

Bell Aircraft Corporation (Dr. M. Robinson)
- Read in - punch out
- Floating decimal interpretive routines
- Floating decimal: sin, cos, e^x, ln, log, arctan
- Matrix operations

Bell Telephone Laboratories (Mr. R. W. Hamming)
- Solution of $y'' - Ay' - Ay = B + 2x(t) + 2x'(t)$ in the form $\int_0^t e^{-\alpha t} x(t)dt$

Boeing Airplane Company (Mr. M. O. Post)
- Floating decimal arithmetic
- Fixed and floating decimal: sin, cos arcsin, arccos, arctan, log, ln, e^x
- Block move
- Load 7 words/card
- Punch 7 words/card
- Load floating point data without excess 50
- Punch floating point without excess 50
- Relocate instructions
- Trace
- Conversion of floating point fo fixed
- Curve fitting by least squares
- A^x

Carbide and Carbon Chemicals Company (Mr. J. E. Rowe)
- 2-3 dimensional Fourier synthesis
- Check-change routine
- Tracing routine
- Input-output routines
- Trimble's interpretive floating point routine extended to include sin, cos, log, exp. etc.

Chance Vought Aircraft, Incorporated (Mr. A. R. Mandelin)
- SAM I
- SAM II
- SAM III - for double precision arithmetic
- Trigonometric functions (RAND approx.)
- Translation
- Block entry
- Block punch-out (1/card through 7/card)
- Block punch-out with word change
- Matrix operations
- Danielowsky (real and imaginery)
- Mode shape removal routine

Detroit Edison Company (Mr. T. H. Lee)
- Complex arithmetic interpretive routine
- Complex matrix inversion
- Cube root
- Sinh, cosh, inverse sinh, inverse cosh
- Conversion of polar coordinates to Cartesian coordinates
- Dump routine which permits reloading by L1

Douglas Aircraft Corporation (Mr. R. E. Ruthrauff)
- Fixed point input (2 types) and output (3 types)
- Floating point input (2 types) and output (3 types)
- Fixed point: square root, sin, cos, arcsin, arccos, arctan, K^x (K=10, e), log, ln
- Floating point interpretive: add, subtract, multiply, divide, square root, multiple operations
- Matrix fixed and floating point

input	input
add, sub	add, sub
multiply	multiply
transpose	transpose
	inversion
	output

- Assembly

Eglin Air Force Base (Mr. H. L. Adams)
- Linear interpolation
- e^x $(-36 < x < 8)$
- ln (1+x) $(0 < x < 1)$
- Arctan A/B $-9999999999 < A < 9999999999$
  " $< B <$ "
- In planning (5 decimal digit values): trigonometric functions, logarithms, exponentials

General Dynamics Corporation (Mr. H. P. O'Neill)
- 10-digit floating decimal conversion routine
- 12x12 symmetrical matrix augmented by 7 vectors

General Electric Company (Miss S. G. Fleming)
- Floating point interpretive routine using Trimble's codes, plus sin, cos, ln, arctan
- Machine language: ln x $(10^{-9} \leqq x < 10^{10})$, e^x $(-20.7 x \leqq 23.0)$, sin x, cos x $(|x| < 10^{10})$
- Least squares curve fitting
- Automonitor for machine language
- Automonitor for floating point interpretive routines
- Punch 8/card
- Block transfer

Clear drum to zero from locations A through B

General Electric Company (Mr. W. H. Root)
- General purpose floating point
- Correlation matrix generation
- Analysis of variance (expected by January 1956)

General Electric Company (Mr. G. W. Hobbs)
- Fixed point routines (floating point in progress but incomplete): input-output, square root, sin, cos, arctan, log, exponential, double precision
- Automonitoring
- Tracing
- Dump block storage
- Clear drum

International Business Machines Corporation (Mr. J. A. Painter)
- Reset entire drum to any desired number
- Card conversion - 701 octal to 701 decimal
- Floating decimal complex arithmetic
- Real and complex roots of algebraic equations (use Lin's method - if fails, use Newton's method)
- La Place Transformation

International Business Machines Corporation (Mr. B. R. Faden)
- Load 1/card and 5/card
- Punch-out all non-minus-zero locations
- Punch-out 5/card in form for loading
- Drum clearing - clear to minus zero
- Tracing
- Statistical
- Square root

Knolls Atomic Power Laboratory (Mr. D. B. MacMillan)
- Loading routine requiring serially numbered cards and specially identified final card
- Punching routine to prepare the above
- Automonitoring routine

Lockheed Aircraft Corporation (Mr. R. W. Bemer)
- General purpose floating decimal system
- Tchebysheff 5th order polynomial through $<$ 50 unequally spaced points
- Utility routines
- Fourier analysis, 1000 points, Filon's method

McDonnell Aircraft Corporation (Mr. T. M. Bellan)
- Fixed and floating decimal arithmetic: square root, sin, cos, ln, e^x, arctan
- Complex arithmetic in floating decimal: $A+B \to K$, $AxB \to K$, $A/B \to K$, $K \pm AxB \to K$,
  $K \to C$, $C \to K$ Transfers (K is the complex accumulator)

Redstone Arsenal (Mr. P. W. Sage)
- Interpretive subroutines for sin, cos, arctan, log
- 10-digit fixed decimal
- 10-digit floating decimal
- 8-digit floating decimal (also, subroutine using 3 index registers)

United States Steel Corporation (Mr. C. W. Zahler)
- Fixed point: square root, cube root, $\sqrt{a^2+b^2}$, $\sqrt{a^2-b^2}$, e^x, log, sin, cos, tan, sinh, cosh, arcsin, arctan
- Fixed point simplex method linear program, maximum size $(m+1)(m+2) \leq 1650$

University of Wisconsin (Dr. A. W. Wymore)
- Dual (single-address) 8-digit floating decimal system including all usual transcendental functions, Steiffel-Hestenes linear systems solution
- Dual (single-address) 18-digit floating decimal system, basic operations only. (Transcendental functions available later this year.)
- All possible simple correlations among 10 5-digit signed variables, mean and standard deviation of each variable. (Factor analysis and standard analysis of variance routine available later this year.)

Westinghouse Electric Corporation (Mr. M. Middleton)
- Interpretive routine(code number in parenthesis): transfer (00), add (01), sub (02), multiply (03), divide (04), e^x (05), square root (06), ln (07), sin (08), cos (09), arctan (10), branch on zero (11), branch on minus (12)
- Fixed and floating point trace

LIST OF TYPICAL 650 APPLICATIONS

Mathematics

Algebraic equations - real and complex coefficients
Applied probability functions
Complex polynomials
Eigenvalues
Extrapolated Liebmann iteration on partial differential equation
Fourier analyses
Generation of tables of specialized functions
Linear programming
Matrix calculations
Minimization of functions of two variables
Ordinary differential equations
Random number generation
Random walk
Simultaneous linear and nonlinear equations
Simultaneous linear and nonlinear differential equations

Statistics

Analysis of variance
Auto-correlation and power spectrum
Climatological statistical analysis
Least squares curve fitting
Multiple correlation
Multiple bivariate frequency distribution tables of weather elements
Quality control
Standard deviations and means

Physics

Atomic power studies
Gamma ray attenuation
Neutron absorption breakdown
Nuclear calculations - Kron's method
Upper atmosphere research studies
X-ray crystal structure analysis

Aircraft Industry

Aeroelastic studies
Aircraft body and duct design
Armament systems evaluation
Bombing systems evaluation
Compressible flow studies
Data reduction - telemetered, theodolite, wind tunnel
Drag chute calculations
Engine cooling
Engine performance calculations
Fire control pursuit course solutions
Flutter and vibration analyses
Flight trajectory calculations
Fuel cell pressure analysis
Guidance problems
Guided missile optmization studies
Heating studies
High-speed instrumentation conversion
Load calculations
Lofting
Mach senser frequency response
Nozzle design calculations
Optical system design
Power plant calculations
Radar equipment design
Radar detection probabilities
Radar echo studies
Radar parameters optimization
Radio interference
Radome studies
Servomechanism calculations
Shears and moments calculations
Sound pressure analysis
Standard airplane performance calculations
Stress calculations
Wind tunnel balances computing

Chemical Engineering

Absorption analysis
Crude oil evaluation
Flash vaporization
Gas vapor cycle - performance coefficient
Liquid - vapor equilibrium calculations
Mass spectrometer analysis
Multi-source planar diffusion problems
Pilot diffusion cascade data analysis
Pipeline design, stress analysis
Platformer gas plant calculations
Refinery production analysis
Tankage studies

Electrical Engineering

Circuit design
Circuit breaker design
Motor and generator engineering studies:
- Core losses
- Critical shaft speeds
- Stability studies
- Transient studies

Power system design:
- Economic operation
- Loading and losses
- Stability studies
- Substation studies
- Transient studies

Transformer design

PARTICIPANTS

Adams, Henderson L., Chief, Machine Branch, Applied Mathematics Division, Directorate of Statistical Services, USAF, Eglin Air Force Base, Florida

Ahlin, Jack T., Applied Science Special Representative for Petroleum Industry, IBM Corporation, Houston, Texas

Alstad, Charles D., Chemical Engineer, The Dow Chemical Company, Computations Research Laboratory, Midland, Michigan

Battin, Richard H., Assistant Director, Instrumentation Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts

Bellan, Theodore M., Supervisor, Department of Applied Mathematics, McDonnell Aircraft Corporation, St. Louis, Missouri

Bemer, Robert W., Section Engineer, Mathematical Analysis Section, Lockheed Aircraft Corporation, Van Nuys, California

Bilo, Stephen J., Technologist - Flutter and Vibration, Fairchild Engine and Airplane Corporation, Hagerstown, Maryland

Bosak, Robert, Group Engineer, Mathematical Analysis Department, Lockheed Aircraft Corporation, Marietta, Georgia

Brokate, Klaus, IBM Deutschland GMBH, Germany

Canfield, Donald B., Programmer, Bethlehem Steel Company, Bethlehem, Pennsylvania

Clippinger, Richard F., Chief of Computing Services Department, Datamatic Corporation, Waltham, Massachusetts

Clotar, Gill, American Optical Company, Worcester, Massachusetts

Coffin, Edward W., Acting Manager, IBM Washington Data Processing Center, Washington, D. C.

Cohen, Marshall B., Junior Mathematician, Cornell Aeronautical Laboratory, Incorporated, Buffalo, New York

Comerford, Emma E., Senior Programmer, Datamatic Corporation, Waltham, Massachusetts

D'Arcy, Donald F., Head, Computation and Analysis Section, Carrier Corporation, Syracuse, New York

De Carlo, Charles R., Director, Applied Science Division, IBM Corporation, New York, New York

DeSantis, Richard A., Mathematical Engineer, Marquardt Aircraft Company, Van Nuys, California

Doll, G. L., Applied Science Representative, IBM Corporation, Chicago, Illinois

Drenick, William J., Mathematical Analyst, Weapons Systems Development Laboratories, Hughes Aircraft Company, Culver City, California

English, Julius C., Physicist, Computations Group, Savannah River Laboratory, E. I. duPont de Nemours and Company, Augusta, Georgia

Evans, Howard T., Physicist, U. S. Department of Interior, Geological Survey, Washington, D. C.

Faden, B. R., Manager, IBM Data Processing Center, Los Angeles, California

Fain, Charles G., (A/2c), Programmer, Directorate of Statistical Services, USAF, Eglin Air Force Base, Florida

Flanagan, Joseph, Applied Science Representative, IBM Corporation, Cambridge, Massachusetts

Fleming, Sarah G., Assistant in Charge of Digital Computers, Analytical Engineering, General Electric Company, Schenectady, New York

Fogel, Gerald D., Supervisor, Automatic Computing Facility, Grumman Aircraft Engineering Corporation, Bethpage, New York

Fritz, W. Barkley, Senior Engineer, Air Arm Division, Westinghouse Electric Corporation, Baltimore, Maryland

Fullerton, Herbert P., Project Leader, General Electric Switchgear, Philadelphia, Pennsylvania
Galvin, John C., Applied Science Division, IBM Corporation, New York, New York
Garrett, John E., Section Chief, Mathematical Statistics Section, Olin-Mathieson Chemical Corporation, New Haven, Connecticut
Graham, Jack N., (Jr.), Mathematical Analysis Branch, Machine Computation Section, USAF, Directorate of Intelligence, Washington, D. C.
Green, Thomas H., Research Engineer, Shell Oil Company, Houston, Texas
Greenberg, Sheldon,R., Mathematician, Collins Radio Company, Cedar Rapids, Iowa
Groth, Valbert J., Staff Engineer, The Standard Oil Company of Indiana, Whiting, Indiana
Hafner, Ralph, Head, Numerical Analysis Branch, U. S. Naval Ordnance Plant, Indianapolis, Indiana
Hamming, Richard W., Member of Technical Staff, Bell Telephone Laboratories, Chatham, New Jersey
Harris, William P., Computing Analyst, North American Aviation, Incorporated, Columbus, Ohio
Heising, W. P., IBM Data Processing Center, New York, New York
Hobbs, George W., Engineer, Aeronautic and Ordnance Systems, General Electric Company, Schenectady, New York
Horner, John T., Supervisor, Engineering Calculations, Allison Division, General Motors Corporation, Indianapolis, Indiana
Horton, T. R., Applied Science Division, IBM Corporation, Asheville, North Carolina
Hunter, G. Truman, Assistant Director, Applied Science Division, IBM Corporation, New York, New York
Kantner, Harold H., Supervisor, Mathematical Services Section, Armour Research Foundation, Chicago, Illinois
Koll, R. T., Applied Science Division, IBM Corporation, New York, New York
Krider, Leroy D., Mathematician, Naval Ordnance Laboratory, Silver Spring, Maryland
Lee, Tsai H., Engineer, Systems, The Detroit Edison Company, Detroit, Michigan
Lesser, Richard C., Director of Cornell Computing Center, Cornell University, Ithaca, New York
Luke, John W., Field Manager, Applied Science Division, IBM Corporation, Los Angeles, California
MacMillan, Donald B., Mathematician, Knolls Atomic Power Laboratory, General Electric Company, Schenectady, New York
Mandelin, Allan R., Computational Systems Engineer, Chance Vought Aircraft, Incorporated, Dallas, Texas
Maso, Essor, Mathematical Analyst, Weapons Systems Development Laboratories, Hughes Aircraft Company, Culver City, California
Merrick, Elsie V., Group Engineer, The Standard Oil Company of Ohio, Cleveland, Ohio
Middleton, Marshall, Senior Mathematician, Analytical Section, Westinghouse Electric Corporation, East Pittsburgh, Pennsylvania
Oakford, Robert V., Lecturer - Industrial Engineering, Radio Laboratory, Stanford University, Stanford, California
O'Neill, Henry P., Supervisor, Computer Facility, Electric Boat Division, General Dynamics Corporation, Groton, Connecticut
Painter, James A., Scientific Computation Laboratory, IBM Corporation, Endicott, New York
Parks, John, Supervisor of Statistical Analysis, Trans World Airlines, Incorporated, Kansas City, Missouri

Peiser, Alfred M., Head of Electronic Computing, M. W. Kellogg Company, Jersey City, New Jersey
Perry, Olney R., Engineer, General Electric Company ANP Project, Idaho Falls, Idaho
Poley, Stanley, IBM Data Processing Center, New York, New York
Post, Malcolm O., Research Engineer, Boeing Airplane Company, Seattle, Washington
Reid, Eugene B., Senior Mathematician, Standard Oil Company of California, San Francisco, California
Remilen, Charles H., Computer, Industrial and Scientific Computing Section, Eastman Kodak Company, Rochester, New York
Rind, Rene L., IBM Corporation, France
Robinson, Mark, Programmer, IBM Unit, Dynamics Engineering, Bell Aircraft Corporation, Niagara Falls, New York
Root, William H., Project Engineer, General Engineering Laboratory, General Electric Company, Schenectady, New York
Rosett, Frank, Research Engineer, Analytical and Computing Group, Vickers, Incorporated, Detroit, Michigan
Ross, Louis L., Assistant to Stress Analysis Head, The Babcock and Wilcox Company, Barberton, Ohio
Rowe, James E., Senior Mathematician, Carbide and Carbon Chemicals Company, Oak Ridge, Tennessee
Ruthrauff, Robert E., Manager of Computing Department, Douglas Aircraft Company, Incorporated, Tulsa, Oklahoma
Sage, Paul W., Mathematician, Redstone Arsenal, Huntsville, Alabama
Schacknow, Arnold B., Supervisor, Engineering Computing Section, Republic Aviation Corporation, Farmingdale, New York
Schricker, Otto, (Jr.), Chemical Engineer, Process Research Division, Esso Research and Engineering Company, Linden, New Jersey
Sewell, George V., Test Engineer, IBM Testing Laboratory, Endicott, New York
Shepherd, Elmer F., Technician, John Hancock Mutual Life Insurance Company, Boston, Massachusetts
Shreve, Darrell R., Research Mathematician, Research Division, The Carter Oil Company, Tulsa, Oklahoma
Smith, Robert L., (Jr.), Statistical Supervisor, Texas Agricultural Experiment Station, Agricultural and Mechanical College, College Station, Texas
Somerall, Leon H., Chief, Reduction Branch, USAF Weather Service, Asheville, North Carolina
Sweeney, Dura W., Mathematical Planning Group, IBM Corporation, Endicott, New York
Swift, C. W., IBM Data Processing Center, New York, New York
Thomsen, D. L., Applied Science Representative, IBM Corporation, Philadelphia, Pennsylvania
Trimble, George R., (Jr.), Mathematical Planning Group, IBM Corporation, Endicott, New York
Williams, Cleo B., Mathematician, Military Physics Research Laboratory, University of Texas, Austin, Texas
Wrubel, Marshal H., Associate Professor of Astronomy, Indiana University, Bloomington, Indiana
Wymore, A. Wayne, Project Supervisor, Numerical Analysis Laboratory, University of Wisconsin, Madison, Wisconsin
Zahler, Charles W., American Bridge Division, United States Steel Corporation, Pittsburgh, Pennsylvania

INTERNATIONAL BUSINESS MACHINES CORPORATION
590 Madison Avenue, New York 22, N. Y.
Form No. 34-6711-0-5M-P
Litho in U.S.A.