U. S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Institute for Numerical Analysis
Los Angeles, California

MANUAL OF THE SWAC COMPUTING SYSTEM

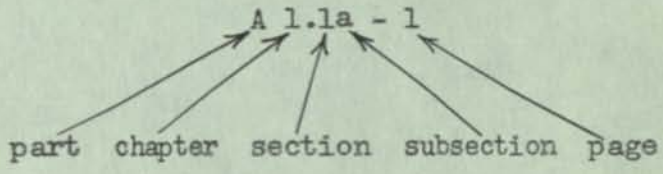
The preparation of this paper was sponsored (in part) by the Office of Scientific Research, USAF, and the Office of Naval Research

PREFACE

This manual has been prepared by the Mathematical Services Unit of the Institute for Numerical Analysis, to assemble the information needed to code for, operate, and test, the SWAC. Much of this information has appeared previously in the Logical Manual for SWAC, the SWAC Coding Guide, and various memoranda.

Details are given concerning the commands and logical design, and examples of coding done for the machine are included. In addition, there is some general information relating to automatic digital computers.

The report is issued in looseleaf form. Replacements of existing pages will be made from time to time, and additional information will be distributed as more codes are written, or modifications are made in the machine. For this reason the flexible numbering system described below has been adopted.



The manual is divided into several parts, and each of these into chapters, sections, and subsections. For example, the number B 2.2b-1 is interpreted as:

Part B.	Description of the Computing System
Chapter 2.	Physical Layout
Section 2.	Console and Main SWAC - description
Subsection b-1	Operating Console, page one

The complicated numbering system has the advantage of enabling new information to be added to the report without reorganizing the existing material. An issuing date is given in the upper left corner of each page. Thus there will be no doubt as to the most recent page in the event of replacement.

Sections marked with an asterisk in the Table of Contents are those planned but not included at the time of the first distribution. The Table of Contents will be augmented as the need arises.

Suggestions for additional material suitable for inclusion in the manual are welcomed.

PART A. INTRODUCTION

PART B. DESCRIPTION OF COMPUTING SYSTEM

PART C. CODING

PART D. OPERATING AND TESTING SWAC

PART E. NON-MATHEMATICAL CODES

PART E. NON-MATHEMATICAL CODES

PART F. MATHEMATICAL CODES

1. General Information Concerning SWAC

2. Binary Arithmetic

a. The National Bureau of Standards Western Automatic Computer uses the binary number system. This is a system in which the digits of a number are the coefficients of powers of the base 2, in the same manner as the digits in the familiar decimal system are the coefficients of powers of the base 10. In the decimal system the digits 0 through 9 are used; the binary system consists only of the digits 0 and 1.

The machine "recognizes" at each pulse time, either a pulse or no pulse. It is natural, therefore, to interpret the pulse, or absence of a pulse, as a 1 or a 0 respectively. Binary numbers equivalent to nine decimal digits can be represented in the machine. Instructions are stored the same way as numbers.

The use of the binary system makes possible the simplification of an electronic computer which has many advantages, one of which is the possibility of attaining greater speed in computations.

b. Examples of binary numbers:

Decimal		Binary
0	=	0
1	=	1
2	=	10
10	=	1010
11	=	1011
13	=	1101
181	=	10110101

$$10110101 \rightarrow 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

which means

$$181 \rightarrow 1 \times 10^2 + 8 \times 10^1 + 1 \times 10^0$$

1. General Information Concerning SWAC

c. Addition on binary scale:

Addition on the binary scale becomes a much simpler operation than in the decimal scale. The addition table in this scale consists of:

	0	1
0	0	1
1	1	10

Binary

0 + 0 = 0

0 + 1 = 1

1 + 1 = 10

Compare it to the addition table of the decimal scale:

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

Assume you want to add 13 and 14.

$$(13) = \frac{11000}{1101} \quad \begin{matrix} C \\ A \end{matrix}$$

$$\begin{matrix} (14) = 1110 \\ (27) = 11011 \end{matrix} \quad \begin{matrix} B \\ D \end{matrix}$$

In the course of adding A and B a "carry" digit is produced (3rd column from right); this carry is represented by C. From the viewpoint of the machine the digits of the numbers will arrive in the M register

1. General Information Concerning SWAC

simultaneously, but the least significant (right-hand ends in the illustration) must be added first. Thus, the machine adds the two numbers, proceeding column by column from right to left as a person would do. That is, the addition signal must be applied to the more significant columns somewhat later than to the less significant columns in order to have coincidence with the carry arriving from the less significant columns.

Whether the number is negative or positive is indicated by a digit to the left of the number. 0 means that the number is positive; 1 means that the number is negative. Thus, 0 000 101 = +5 1 000 101 = -5.

1. General Information Concerning SWAC

3. Notation Used in This Report

a. Command words are represented by α, β, γ, δ, F. That is, these letters denote subsections of the command word.

b. The following abbreviations are used in relation to parts of the computer:

M	M Register)	
A	A Register)	Arithmetic Unit
R	R Register)	
C	C Register)	
P	P Counter)	
S	S Register)	Control Unit
ε	ε Counter)	
0	0 Register	Input-Output Unit
m	Cathode Ray Tube Memory)	
CRT	Cathode Ray Tube)	Memory Unit

c. The following abbreviations of terms are used:

Cl	Clear (cf. e)
CO 1	Corrective 1 (cf. f)
K	Complement (cf. f)
RO 1	Round-off 1
w	Word, in machine sense means numerical information, alphabetic data, or instructions written in a form (binary digits) acceptable to the machine.
wr	With respect
→	Replaces (cf. g)

d. (X) Symbol used to denote the word (see definition of "word" above) stored in location X. For example, (M) represents the word stored in the M register; similarly (α) is used to denote the word stored in address α.

e. Cl(X) Indicates that information in unit X of the computer is cleared, that is, unit X is made to register, or remember, + 0. For example, Cl(M) indicates that the M register is cleared.

1. General Information Concerning SWAC

f. K digits of (M) wr 1's	Means to complement the digits contained in the word in the M register with respect to ones. That is, the ones are interchanged for zeros and vice versa. Sign digit is not changed.
K digits of (M) wr 2^n	Means complementation with respect to ones (as above) and a "CO 1" (corrective one) is then added to the least significant digit of the complemented number.
g. (X) → (Y)	Indicates word contained in unit X replaces the word contained in unit Y of the computer. For example, (α) → (M) means that the word in address α replaces the word in the M register.
X → Y	Similar meaning to above. For example, $\gamma \rightarrow \mathcal{E}$ means the number γ replaces the number stored in the $\mathcal{E}$ counter.
X + 1 → X	Indicates that number X is increased by unity. For example, $\mathcal{E} + 1 \rightarrow \mathcal{E}$ means the number stored in the $\mathcal{E}$ counter has one added to it.
h. SWAC addresses	References to memory addresses are made either in base 16 (hexadecimal) or base 10 (decimal) notation. Three digits, e.g. 025, specify a decimal number; two, e.g. 19, specify a base 16 number. In ambiguous cases the context will usually leave no doubt as to which is meant.

1. General Information Concerning SWAC

4. Design Principle

The main objective in designing the SWAC was to keep it as simple as possible and yet obtain a good computing machine. This objective is still adhered to. Improvements and changes which require substantial expenditures of time to construct have to be very worthwhile to be incorporated in the design.

On the other hand future improvements have been constantly kept in mind. The high speed memory is designed in a manner to make possible its expansion from 256 to 512, or even 1024, words at some later date. In a similar manner the intermediate speed magnetic drum memory was added after the main body of SWAC was a functioning computer. Future plans call for addition of a magnetic tape memory.

At the time of this writing the computer has facilities for reading information from punched cards, and transmitting information to punched cards. The speed of reading is 2400 words per minute; punching is accomplished at 1200 words per minute.

A keyboard allows for the insertion of information by hand, and a typewriter is used for printed output of information. Plans for the immediate future call for the addition of some form of high speed printing device to supplement, or replace, the typewriter.

1. General Information Concerning SWAC

5. Machine Language

Information is stored in the high speed memory of the SWAC as pulse or charge combinations on the faces of the cathode ray tubes. These pulses or charge combinations are called words. Therefore, when we speak of words in relation to computing machines we are not thinking of words in the ordinary sense, but rather we are using the term to designate the language which the machine works with and understands.

As the SWAC is a binary machine the words it uses are composed of binary digits. In a sense the SWAC's entire vocabulary consists of 0's and 1's. Therefore, every instruction and bit of information given to it must be written as a combination of binary digits in order that it may be understood by the machine. These binary numbers are stored upon the faces of the cathode ray tubes in the form of dots and dashes.

Words in the machine language are composed of a standard number of binary digits. This length is determined by the number of dots and dashes (charge patterns) which can be stored upon the face of one cathode ray tube. This number may be 256, 512, or 1024. In each case a different number of digits is required to specify an address, or location, in the memory, thus altering the size of the word. For example, if 256 charge patterns are stored on the face of each tube, then a 37 binary digit word is the proper length. If 512 are stored, 41 binary digits are required in each word; for 1024, the length of the word becomes 45 binary digits.

The length of the words used by the SWAC at present is 37 binary digits.

As the SWAC is a parallel machine, a single digit of the word is stored upon the face of a particular cathode ray tube. Accordingly, the number of cathode ray tubes in the memory is determined by the length of the word. As

1. General Information Concerning SWAC

the word for the SWAC is 37 binary digits long, a bank of 37 cathode ray tubes makes up the high speed memory.

Words in the machine sense may represent 1) numerical information called numbers, 2) alphabetic information, or 3) instructions to the machine. Instructions are of three types: commands, control words, and code words. These various types of information and instructions are discussed in greater detail under the specific sections devoted to them in this chapter.

Careful allotment is made of the 37 binary digit positions which compose each word of the language understood by the SWAC. The table at the end of this section shows how this space is allotted, depending upon the type of information to be given to the machine. It should be noted that numbers may be given to the machine either by using the base 16 method or the coded decimal method. A comparison of these two methods is given in the section on Numerical Information (Numbers). In both methods the digit in the zero position is used to specify the sign of the number, leaving the remaining 36 digits for numerical information.

Allotments are given for commands but not for control words or code words. This is due to the fact that no stated allotment of positions can be made for control words or code words. In the commands four addresses must be specified in each word. These four addresses generally consist of two operands, address of the result, and the address of the next command. It takes eight, nine, or ten digits to specify each address in the memory according to whether 256, 512, or 1024 words are stored in the high speed memory. Thus, in the word used by the SWAC, 32 digits are necessary to specify positions. Of the five remaining binary digits one is used to specify the sign of the number. Three digits define the eight basic command

1. General Information Concerning SWAC

words: Addition, Subtraction, Multiplication with Round-Off, Exact Multiplication, Compare, Extract, Input, and Output. One digit defines modifications of five of these eight basic command words. These five modified command words are referred to as the Special Commands and consist of Special Addition, Special Subtraction, Special Multiplication with Round-Off, Special Compare, and Initial Input.

Each of the four addresses of the command, α, β, γ, δ, can be written in any one of three ways:

1) As a group of 8 binary digits.
2) As a pair of base 16 numbers, each of which represents four of the binary digits of 1).
3) As three decimal numbers (maximum value 255) which represent the decimal equivalent of the eight binary digits of 1). This representation is the one usually used in coding.

On the Console SWAC displays commands in binary form, in groups of 4 binary digits. These are interpreted in pairs of base 16 digits which can be translated into decimal numbers less than 255 for comparison with written codes.

Table showing allotment of 37 binary digit positions which compose each word:

Binary Digits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
Base 16 Binary Numbers	S	1st Base 16 Digit				2nd Base 16 Digit				3rd Base 16 Digit				4th Base 16 Digit				5th Base 16 Digit				6th Base 16 Digit				7th Base 16 Digit				8th Base 16 Digit				9th Base 16 Digit			
Coded Decimal Numbers	S	1st Dec. Digit				2nd Dec. Digit				3rd Dec. Digit				4th Dec. Digit				5th Dec. Digit				6th Dec. Digit				7th Dec. Digit				8th Dec. Digit				9th Decimal Digit			
Command	0	α (Address; Usually Operand)								β (Address; Usually Operand)								γ (Address; Usually Result)								δ (Various Uses)								F (Function) Defines 8 Basic Commands			Mod. of 5
Console Digits	0	α 1	α 2	α 3	α 4	α 6	α 7	α 8	α 9	β 1	β 2	β 3	β 4	β 6	β 7	β 8	β 9	γ 1	γ 2	γ 3	γ 4	γ 6	γ 7	γ 8	γ 9	δ 1	δ 2	δ 3	δ 4	δ 6	δ 7	δ 8	δ 9	F1	F2	F3	F4

1. General Information Concerning SWAC

6. Numerical Information (Numbers)

"Words" in the machine sense are groups of binary digits representing either numbers or instructions. Instructions are written as, and look like, numbers. The difference lies in the way the machine interprets the information. Since instructions are written as numbers the machine can perform arithmetic operations on them. The power of the SWAC depends on its ability to alter instructions supplied to it.

The word used by the SWAC is 37 binary digits long. One binary position is used for the sign, leaving 36 binary positions available for numerical information. These 36 binary digits are roughly equivalent to 10.8 decimal digits.

As 36 is divisible by 4, it is convenient to express binary words in 9 groups of four binary digits each. This base 16 (or hexadecimal) notation is used by some coders, and by the operators at the SWAC console. The output typewriter of the SWAC is arranged to type out a single symbol for each of the possible 16 combinations of 4 binary digits. Any combination of 4 binary digits transmitted to the typewriter by SWAC will activate one key and produce the appropriate symbol. Nine such symbols, and a sign, therefore represent the binary word stored in SWAC.

A coded decimal representation of numbers may be made. In this case each group of four binary digits represents any one of the decimal digits from 0 to 9. Nine such groups placed in juxtaposition may be typed out to represent a decimal number. They obviously do not represent the binary equivalent of the decimal number. In order to translate the original binary number into coded decimal, a routine must be used to enable SWAC to perform this operation of "conversion". In a similar manner translation from coded decimal to binary may be accomplished.

1. General Information Concerning SWAC

These translations may be made when reading cards from the collator or when punching cards. In the first case an image of the holes in the card is stored in 10 cells of the SWAC memory, and the routine will translate these scattered digits into a binary number. In the case of punching, the translation puts the binary number into the form of a card image in ten cells, with binary "ones" representing the places where the holes will be punched in a card to produce the decimal answer desired. Obviously the "decimal word" as stored in SWAC does not, in any direct sense, represent the values of the 10 binary words which make it up.

The table on the following page illustrates the difference between the base 16 and the coded decimal representation of numbers. For a further discussion of decimal numbers on cards, see the section on Coding Conversion Routines.

As an example of the relationship between binary, and coded decimal, we may express the number 978 in binary. To do this, multiply the number by .5 (i.e., divide by 2), and tally a "0" each time the fractional part is zero, and a "1" whenever it is one-half. Discard the fraction and repeat. The development of the first three digits from the right is shown:

	Tally
978 x .5 = 489.0	0
489 x .5 = 244.5	1
244 x .5 = 122.0	0
	010

The full number is: 0011 1101 0010 divided into groups of 4 digits for convenience. The equivalent hexadecimal number is: 3 x 2. If we wish to express 978 in coded decimal form, however, we read, for each decimal digit, the group of four binary digits which represents it:

9	7	8
1001	0111	1000

1. General Information Concerning SWAC

It can be seen that the binary word we have just written is not the same as the true binary representation of 978, which was written previously.

Table showing difference between base 16 and coded decimal representation of numbers:

Decimal Number	Base 16 (Hexadecimal) Representation	Coded Decimal Representation	Typewriter Character Used	
			Base 16 (Hexadec.) Rep.	Coded Dec. Rep.
0	0000	0000 0000	0	0
1	0001	0000 0001	1	1
2	0010	0000 0010	2	2
3	0011	0000 0011	3	3
4	0100	0000 0100	4	4
5	0101	0000 0101	5	5
6	0110	0000 0110	6	6
7	0111	0000 0111	7	7
8	1000	0000 1000	8	8
9	1001	0000 1001	9	9
10	1010	0001 0000	u	10
11	1011	0001 0001	v	11
12	1100	0001 0010	w	12
13	1101	0001 0011	x	13
14	1110	0001 0100	y	14
15	1111	0001 0101	z	15

1. General Information Concerning SWAC

One method of representing numerical information in the machine is the so-called floating method in which the decimal point is not specified as a position someplace in the number but is specified by a certain number of digits. In effect, in a binary system, a number is represented by its significant figures times 2^k, where k is an exponent determining the position of the binary point. When this floating method is used in the decimal system, the number is represented by its significant decimal digits and a decimal exponent. In other words, the number is of the form $a \times 10^b$. The a and b in this representation are placed in one word in the machine, which is able to handle such numbers by means of routines which produce the results of any desired set of computational processes. This device increases the range of numbers that the machine can handle automatically at the price of losing a few significant figures. Naturally, if the number of significant figures is extremely important, then the exponent can be placed in one word position and the significant figures in another. Thus, an increase of range can be obtained at the cost of using up extra memory space.

A computation is performed most efficiently if the binary numbers are interpreted as having their binary point either in front of all the significant figures or at the end of all the significant figures. That is, the numbers should either lie between plus and minus 1, or be integers. The multiply commands in SWAC operate on this assumption. However, if it is desired that the binary point be located someplace in the middle of the word, this can be done on the computer at the price of shifting any products that might be computed.

Numbers are stored in the machine as signs and true values. Arithmetic processes can be carried out with complements but no complementary numbers are stored in the memory.

1. General Information Concerning SWAC

7. Numerical Information (Machine Instructions)

One type of information, which words in the machine sense may represent, is instructions. Obviously, there must be some way of directing the machine what to do next after each operation. In the SWAC the instructions are given to the machine in the same manner as the numerical and alphabetical information. Thus, the activity of the machine is determined by words it has stored in its high speed memory. This is known as automatic sequencing.

The word "instructions" is used to designate the general class of words which has to do with instructing the machine what to do in the course of a computation. Instructions may be divided into three classes: 1) commands, 2) control words, and 3) code words.

Commands are words that tell the machine a) exactly what operation to carry out, defining the arithmetic function, storage of operands and result, and b) what it is to do next. Commands are explicitly understood and obeyed by the machine. For use with the SWAC there are thirteen types of commands. These can be used to perform eight basic operations: addition, subtraction, multiplication with round-off, exact multiplication, compare, extract, input, and output. For five of these operations (addition, subtraction, multiplication with round-off, compare, and input) there are five modified special commands. All of these commands are described in detail in another section of this report.

Control words (also called dummies) are words or numbers which are not explicitly used in the computation and are not obeyed explicitly as commands are obeyed. They generally control the course of the computation or enter into the arithmetic-like operations which are performed upon commands. They

1. General Information Concerning SWAC

may contain numbers or parameters that determine when the machine is to cease carrying out a certain routine and change to a new one. Or, they may enable the machine to split up the words and extract a portion of them, in this way making use of only a part of a particular word. Following is an example of a control word. Assume that the number 1 is stored in some word position or address in the memory, and is made use of in counting the operations carried out in the course of doing a computation. In this way, the number 1 may actually serve as a control word in the logical arrangement of carrying out the computation as well as having a numerical value in the computation.

Code words are the third category of instruction words. This group of words is also variously designated as abbreviated code instructions[1], shorthand commands, abbreviated commands, and quasi-commands[2]. In general they consist of coded information which the computer is able to split up and make use of, so as to carry out a whole sequence of operations upon being given this one coded command. In other words, a code word is not an explicit command that the machine obeys, but it is an instruction from which the machine may pick out a certain routine of operations and carry it through. It is a compact representation in one word of several parameters or addresses necessary to call into action other sequences of instructions, usually called subroutines. Each subroutine extracts its

1) The term "abbreviated code instruction" was developed at National Physical Laboratory, England, in a group headed by Dr. A. M. Turing, of which H. D. Huskey was a member.

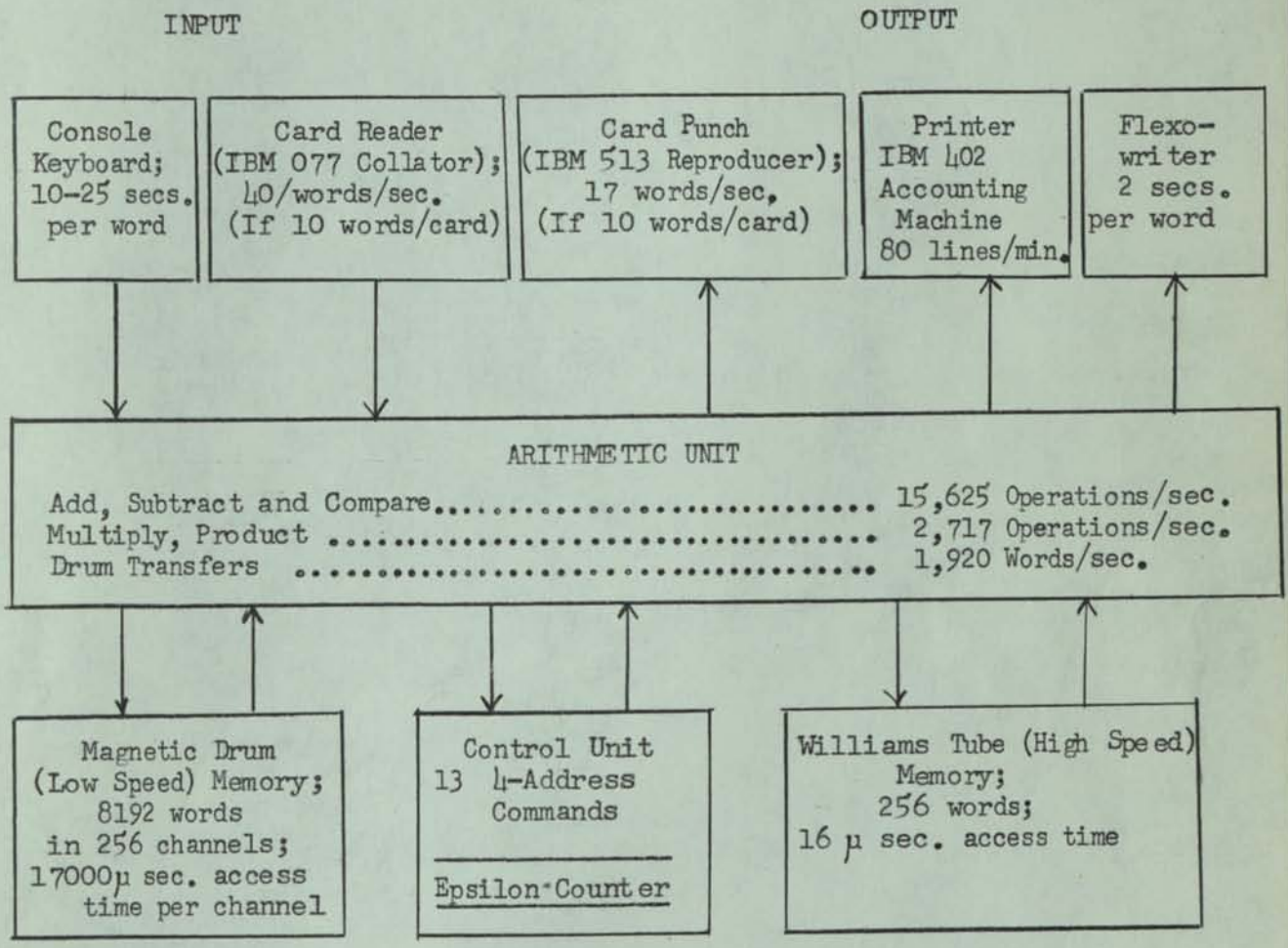
2) The term "quasi-command" was suggested by E. F. Moore, University of Illinois.

1. General Information Concerning SWAC

various parameters from this one code word. For example, one may specify a scalar multiplication with only one code word. The appropriate subroutine in the calculator extracts and properly places the various parameters from the code word. These parameters must specify a) the common factor (that is, specify its address in the high speed memory), and b) the location of the elements of the vector (say, by specifying the address of the first element and the number of terms in the vector), and c) where the result is to be placed. Thus, a code word is an instruction which specifies in one word a whole sequence of events for the calculator.

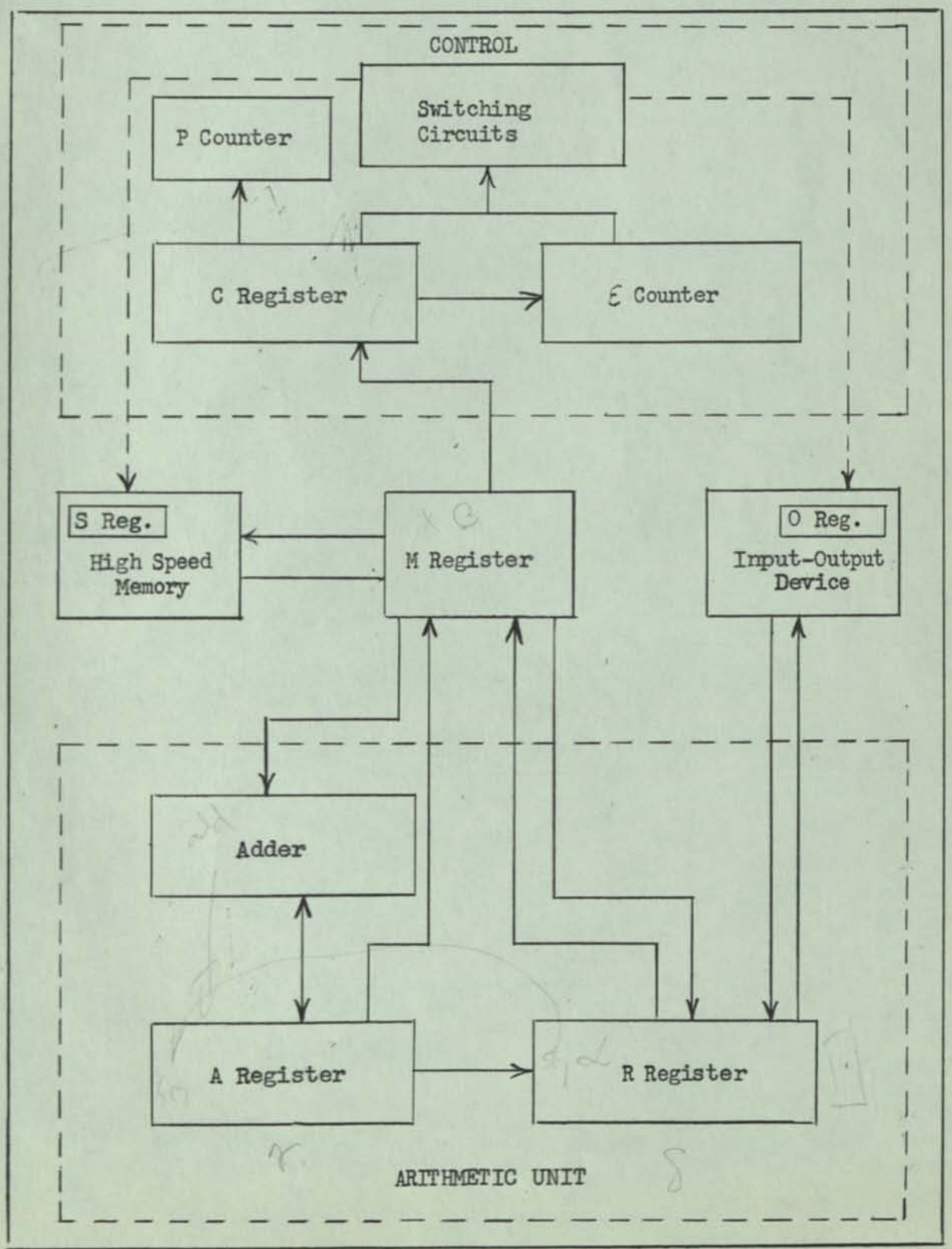
2. Beginners' Guide to SWAC

1. Simplified Chart of SWAC



2. Beginners' Guide to SWAC

1. Simplified Block Diagram of SWAC



2. Beginners' Guide to SWAC

1. Simplified Block Diagram of SWAC (explanation)

A Register Accumulator-type Register. The A Register temporarily holds the results of addition and subtraction operations, and along with R Register, temporarily holds the results of multiplication and extract commands.

C Register Control Register. Receives "next" command from the memory via the M Register, and "remembers" it while the pertinent operations are carried out. Can receive commands directly from R Register via the R→C switch on the keyboard.

M Register One word memory. Acts as central station for transfer of commands and numerical information in the machine. For example, in process of obeying a command normally information is transferred from the memory into M Register and from there into arithmetic circuits involving A and R Registers. The results are transferred back to the memory via the M Register. Commands are transferred from the memory to the Control Register via M Register. Input-output information is transferred between the memory and the R Register via M Register.

O Register Input-output Register. This register determines which particular input-output device is to transmit information to, or receive information from, the memory. The actual transfer of information is via the M and R Registers. One position on this register can be used for halt operations. For example, a command calling for input from a non-existent input-output device automatically causes the machine to stop.

P Counter Product Counter. This counter counts out the various steps of a multiplication operation. It also determines the number of positions which numbers must be shifted in the extract operation.

R Register Multiplier Register. Besides holding the partial results of multiplication and extract operations it holds the multiplier during the multiplication process. The multiplier is shifted out of R Register, digit by digit, as partial products are shifted into it. R Register is also an intermediate station in the transfer of information between memory and input-output devices.

S Register Selection Register. This is a one address register which controls the selection of words from the memory. Successive addresses are transferred to it, either from the Control Register or from $\mathcal{E}$ Counter.

$\mathcal{E}$ Counter The $\mathcal{E}$ Counter is a one address memory which is capable of having "1" added to it. It normally specifies the address from which the next command which the machine is to obey should come.

2. Beginners' Guide to SWAC

2. Description of SWAC

The "National Bureau of Standards Western Automatic Computer," usually referred to as SWAC, consists basically of nine interconnected units (see simplified chart of SWAC A 2.1-0).

The high speed (Williams Tube) memory (H.S.M.) serves to remember or store numbers as well as commands which the computer is to obey. These numbers and commands, referred to as words, consist of a sign digit and 36 binary digits (usually called bits) each of which is either a 0 or 1 (see A 1.2 for description of binary number system). The memory physically consists of a bank of 37 cathode ray tubes (CRT). On the face of each tube is a 16 by 16 array of dots and dashes which may appear as in fig. 1.

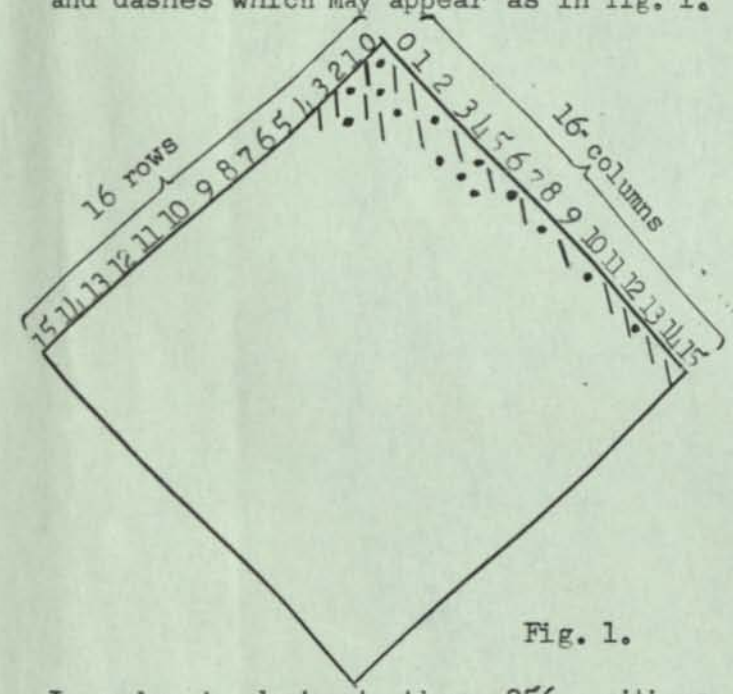


Fig. 1.

Each dash represents the digit 1 and each dot the digit 0. By referring to the digits in the ith row and jth column of all 37 CRT's simultaneously we obtain a set of 37 of these binary digits representing one word in the high speed memory. Since there are 16 rows and 16 columns to choose from we have a total of 16 X 16 = 256 words. In order to designate these 256 positions, or cells as they are normally called, we have assigned each of them a number by which we can refer to it. This number is called the address of that particular cell. An address may be written as a 3 digit decimal number from 000 to 255. Cell 000 is referred to as the origin. The three digits serve to indicate that the number is in decimal notation. The computer can either read or store a word in any one of these cells

2. Beginners' Guide to SWAC

in about .000016 sec. (i.e., the "access time" is about 16 microseconds, written as 16μ sec.).

The drum memory consists of a cylinder about 8" in diameter and 10" long rotating on its axis at approximately 3600 rpm. The surface of the drum is coated with a magnetic material and information (i.e., words) is recorded on this in the form of minute magnets much in the same way a wire or tape recorder works. This drum has sufficient capacity to "remember" 8192 words of 37 binary digits each. These are arranged into 256 groups, called channels, of 32 words each. The words on any channel may be transferred from the drum to the high speed memory (READING FROM DRUM) or back (WRITING ON DRUM) in blocks of 8, 16 or 32 words. (In addition, a block of 2 consecutive channels may be transferred at one time.) Each of these transfer processes takes no more than .017 seconds (in the case of fewer than 64 words, less), which is a very short time indeed, but is about 1000 times as long as the access time for the high speed memory. (For details see B 6.1).

The arithmetic unit has the ability to add, subtract or multiply two numbers as well as to perform certain logical (i.e., non-arithmetical) operations. This unit does all the numerical calculations performed by the computer.

The input equipment is the means of getting the machine commands and the data for a problem into the H.S.M. The primary input device is an IBM (type 077) collator which reads (properly punched) standard IBM cards, interpreting a punched hole as the digit 1, at the rate of 240 cards per minute, and transfers the information (i.e., words) into the high speed memory. On the IBM card 10 rows and 37 columns are usually used (although any number from 1 to 11 rows may be used), each row representing one word of 37 bits. (For details see B 3.2.)

2. Beginners' Guide to SWAC

The other input device currently in use is the console keyboard, which is a set of 7 push buttons and 1 springloaded toggle switch on the console, by means of which the operator can also introduce information into the SWAC. This is, of course, a much slower input device than the IBM collator.

The output equipment provides the method of getting answers out of the machine. The primary output device is the IBM (type 513) Reproducing punch. By using this, words may be punched on IBM cards at the rate of 100 cards per minute (with up to 24 words per card; usually 10 words per card are used, so that the cards can be used easily for input purposes). A Flexowriter controlled by the computer provides a means of typing out information at a rate of about ½ word per second (for details see B 4). A printer (IBM 402 Accounting Machine) is also available for direct SWAC output.

The control unit serves as the nerve center of the SWAC. Its purpose is to coordinate the actions of all the other units. This unit, by examining words in the high speed memory, obtains commands telling it how all the units of the machine should be made to operate.

To solve a problem on SWAC we prepare a set of numbers and commands which are placed in the High Speed Memory. The drum may, of course, have previously been loaded with other words which can be called into H.S.M. when needed. The control unit then looks at the commands one by one and is thus told what to do with the numbers (sometimes commands, too) in the high speed memory. These words are operated upon in accordance with the commands and the results are stored again in the high speed memory. When the final answers to the problem are achieved they are then either typed out or punched out on IBM cards, or printed on an IBM tabulator sheet.

2. Beginners' Guide to SWAC

3. Notes on the Elements of Coding for SWAC

a. Meaning of SWAC Commands

Commands are given to SWAC as words in the high speed memory (H.S.M.). At the beginning of a problem the control unit is "told" from what cell to get its first command. After obeying a command, SWAC either gets its next command from the cell immediately following that occupied by the command just obeyed, or from some other cell (as specified by the command just obeyed).

Words (representing either commands or numbers*) are stored in the H.S.M. (or the drum) and consist of a set of 37 binary digits (bits). The sign (the leftmost bit) is normally + (stored as a 0) for a command, but even if it is negative (stored as a 1) it has no effect on the performance of the command, as this bit is ignored by SWAC when a word is used as a command. The remaining 36 bits actually represent the command; they are broken into 5 groups, 4 of 8 bits each called the α, β, γ, δ parts and 1 of 4 bits called the F part of the word. Consider, e.g., the following word, which has been separated into the above 5 parts:

	α	β	γ	δ	F
±	00011011	01101001	00110111	11010101	0101
	8 bits	8 bits	8 bits	8 bits	4 bits

Each 8 bit portion can represent a number from 0 to $2^8 - 1 = 255$ and consequently can be the address of any cell in the H.S.M. The first 4 parts of a command can thus refer to any 4 cells of the H.S.M. The last 4 bits, the F(function)

N.B. *When we think of a SWAC word as a number N, we usually think of N as having its (binary) point at the left of all its 36 bits so that $|N| < 1$. This is a convenient way of thinking about the situation, especially when problems of overflow and the operation of multiplication are being considered, so we shall adopt this convention throughout these notes.

portion, can represent a number from 0 to $2^4 - 1 = 15$. The value of this F portion determines which operation command SWAC is to obey. We shall probably find it convenient to think of the decimal equivalent of each of the above 5 parts of a SWAC word; this is called the DECIMAL COMMAND FORM (D.C.F.) of a word (e.g., D.C.F. of above example is 027, 105, 055, 213, 05).

The "language" which SWAC can "understand" consists of 13 different operation commands (summarized on C 1.2-1). Each operation command is assigned a decimal number from 00 to 15 (or their hexadecimal equivalents, 0 to Z) by which it is identified.

Three of these numbers (03, 13, 15) are normally not used to denote SWAC operations commands, even though they would be interpreted AT PRESENT by the SWAC control mechanism as respectively equivalent to (02, 12, 14). However, since it is planned to make slightly different uses of them at some future date, it is not advisable to use them in writing codes. When SWAC is told by its control unit to obey a command it first looks at the F portion of the command to find out which of the 13 operations commands is involved. Then SWAC, in most cases, operates on the numbers found in those cells specified by the α and β parts of the command in accordance with the operation specified by F, and places the result in that cell of H.S.M. specified by the γ part of the command. The δ portion of a command serves different purposes in different commands.

Suppose, for example, the control unit finds an 05 in the F portion of a word. Then SWAC copies the contents of the cell specified by α, adds to it the contents of the cell specified by β, and puts the result into the cell specified by γ. Then the control unit takes the next command from the cell specified by δ.

It is now convenient to introduce a notation which makes the description of the commands simpler.

2. Beginners' Guide to SWAC

Let the number a, $000 \leq a \leq 255$, be the address of some cell of the high speed memory; then the contents of cell number a is denoted by (a). To say then, for example, that cell 100 contains the number $\frac{1}{2}$ we need only write $(100) = \frac{1}{2}$. Another useful notation is "→", by which we mean that the quantity to the left of "→" replaces the quantity on the right. Thus (a) → (b) means that the contents of cell a replaces the contents of cell b; i.e., the previous contents of cell b are destroyed because the word in cell a is put into cell b. In SWAC, the cells of the H.S.M. are non-accumulative; i.e., "writing" a word in any cell destroys what was originally in that cell. The 05 (special add) command described above may therefore be described by:

* $(\alpha) + (\beta) \rightarrow (\gamma)$, and obey next command in cell δ.

There is another place in SWAC where an 8 bit binary number is stored. This place is called the "ϵ register" or command counter. The function of the ϵ register is to indicate to the control unit of SWAC the cell from which it should get its next command. The control unit of SWAC then always does the following sequence of operations.

1. Obtain the command in the cell designated by the ϵ register.
2. Obey that command (note that this changes the ϵ register also).
3. Obtain the command in the cell designated by the ϵ register.
4. Obey that command (note that this changes the ϵ register also).
5. Obtain the command, etc.

E.g., in obeying the 05 (SPECIAL ADD) command the SWAC performs the operation * $(\alpha) + (\beta) \rightarrow (\gamma)$ and then puts the number δ into the ϵ register; i.e.,

---

*Actually, you may get overflow (O.F.) in which case you get other results. REMEMBER THESE ARE #'S $-1 \leq \# \leq +1$ The true situation is as follows: If $|(\alpha) + (\beta)| < 1$, then $(\alpha) + (\beta) \rightarrow (\gamma)$. If $|(\alpha) + (\beta)| \geq 1$, then $(\alpha) + (\beta) - 1 \rightarrow (\gamma)$ if $(\alpha) + (\beta) > 1$ or $(\alpha) + (\beta) + 1 \rightarrow (\gamma)$ if $(\alpha) + (\beta) < -1$.

2. Beginners' Guide to SWAC

$\delta \rightarrow \epsilon$. Thus, after doing step 2 of the above sequence the next command would be that which was specified by the δ of the command just obeyed.

Now suppose $|(\alpha) + (\beta)| \geq 1$. This is then a number longer than can fit into a single SWAC cell. With the 05 command the integral part of the sum is lost (i.e., we have overflow) and only the sign and fractional part is stored in cell γ. If this were the case in a numerical calculation we would lose part of the sum, and incorrect answers would result. Consequently we must provide some way of detecting overflow. This is done by the ordinary ADD command, F = 04, which again performs the same addition as the 05 command above, but then one of the following two events will take place:

(a) If there is an overflow, $\delta \rightarrow \epsilon$ so that the next command will come from cell δ.

(b) If there is no overflow, $\epsilon + 1 \rightarrow \epsilon$, so that the subsequent command will be the next command sequentially in the memory.

The subtract commands are similar to the add commands:

F = 06 (SUBTRACT): $^{*}(\alpha) - (\beta) \rightarrow (\gamma)$ { a) $\delta \rightarrow \epsilon$ if there is overflow; b) $\epsilon + 1 \rightarrow \epsilon$ if there is NO overflow.

F = 07 (SPECIAL SUBTRACT): $^{*}(\alpha) - (\beta) \rightarrow (\gamma)$, $\delta \rightarrow \epsilon$ in all cases.

When two numbers in SWAC are multiplied, the point (called a binary point rather than a decimal point) is considered to be at the left (as mentioned earlier), so that all numbers are less than 1 in magnitude. Consequently the product can not exceed 1 in absolute value and there is no possibility of an overflow. The product of two 36 bit numbers is of course

---

*Changes similar to those in case of 04 and 05 in these statements must be made here in case of overflow.

2. Beginners' Guide to SWAC

72 bits long. In order to take care of this, the answer can either be rounded off to 36 bits or be put into 2 cells.

There are two multiply commands which give rounded products:

F = 10 (MULTIPLY):

rounded off $(\alpha)(\beta) \rightarrow (\gamma)$, $\epsilon + 1 \rightarrow \epsilon$

F = 11 SPECIAL MULTIPLY:

rounded off $(\alpha)(\beta) \rightarrow (\gamma)$, $\delta \rightarrow \epsilon$.

Information is obtained from SWAC by means of the output command (F = 02). For this command the number in the δ portion of the word selects the output device and the mode in which it is to operate. If δ = 000, the typewriter is selected and SWAC types out the contents of the cell specified by the α part of the command. Under normal circumstances the typed word appears as a sign and 9 hexadecimal digits (i.e., the equivalent of 36 bits) in reverse order, viz. the least significant digits first. Other numbers in the δ part of the command select other output units as follows: (see C 2.5e for full list)

δ	output device selected
000	Typewriter
080	IBM punch
112	Drum
096	Breakpoint .

As is indicated by the above, the operation of transferring information from the high speed memory onto the drum (i.e., writing on the drum) is also considered as an output operation. (Similarly the operation of transferring information from the drum to the high speed memory (i.e., reading from the drum) is called an input operation.) In addition, the breakpoint "output"

2. Beginners' Guide to SWAC

command specified by δ = 096 causes the display of (α) and (β) in the form of neon lights on SWAC's console. For typewriter output (δ = 000) the β and γ parts of the command usually are not used. For some other output methods these do have a meaning. For details see C 2.5.

2. Beginners' Guide to SWAC

b. Preparation of a simple routine

Suppose, as an initial example, we wish to have SWAC compute the exact values of n^2 for n = 1 (1)50 and type out n and n^2. (Note: the notation here is a standard one used to describe mathematical tables. A symbol n = a(h)b means that n varies from a to b in steps of h.) One way of doing this would be to suppose we have the numbers 1 to 50 stored somewhere in SWAC, and then obey the following sequence of 150 commands:

compute	1^2
type	1
type	1^2
compute	2^2
type	2
type	2^2
.	
.	
.	
compute	50^2
type	50
type	50^2

This procedure would take 3 commands for each integer n, thus a total of 3 x 50 = 150 commands. The commands would occupy 150 cells in SWAC and the integers 1 to 50 would occupy another 50 cells.

To prepare such a procedure for SWAC we would use a coding sheet like that on page C 2.1a-7. On these forms there is a column labeled "Cell No.", in which is placed the number of the cell into which the word on that line of the sheet is to go. Columns labeled α, β, γ, δ, and F are provided for the 5 separate parts of each word, a "±" column for the sign of the word,

and a column for remarks. Additional columns have meanings and uses we need not concern ourselves with at the moment; for this see D 1.3.

The commands in a code are usually written in the decimal command form, as explained above, the first 4 parts being written as 3-digit decimal numbers and the F part as a 2-digit decimal number.

To code the above procedure we must first consider the possible range of all numbers in the computation, since (at least for multiplication purpose) all numbers in SWAC cells are thought of as less than 1 in magnitude. We must therefore "scale" all numbers so that their magnitude falls inside the range -1 to +1. For this problem n goes from 1 to 50; if we scaled the numbers n by multiplying each of them by 1/51, all the numbers n would be less than 1. However, with this choice the numbers would not be exact in SWAC since 1/51 could not be expressed exactly in 36 bits. The most convenient choice for a scale factor is usually an integral power of 2. For this problem we could scale by a factor 2^{-k} with $k \geq 6$, for $n^{-k} \leq 1/64$ and all numbers would be less than one. If n is stored as $n \cdot 2^{-k}$, then upon computing n^2 with a multiply command we get $(n2^{-k})(n2^{-k}) = n^2 2^{-2k}$. If this result is to be computed exactly by means of a command whose F = 10 or 11 the product must be entirely in the first 36 bits. This means we must have $2^{-2k} \geq 2^{-36}$ so $k \leq 18$. Thus $6 \leq k \leq 18$; let's choose k = 16 for this problem (the choice of either k = 8 or 16 would have certain advantages because we would thus create a "one in α" ($1_\alpha \equiv 2^{-8}$) or a "one in β" ($1_\beta \equiv 2^{-16}$) either of which is useful in other connections).

2. Beginners' Guide to SWAC

Now suppose the numbers $1 \cdot 2^{-16}$, $2 \cdot 2^{-16}$, $3 \cdot 2^{-16}$, $\cdots$, $50 \cdot 2^{-16}$ are stored in the respective cells 206, 207, $\cdots$, 255 of the H.S.M. The code for the above then could be written out as follows:

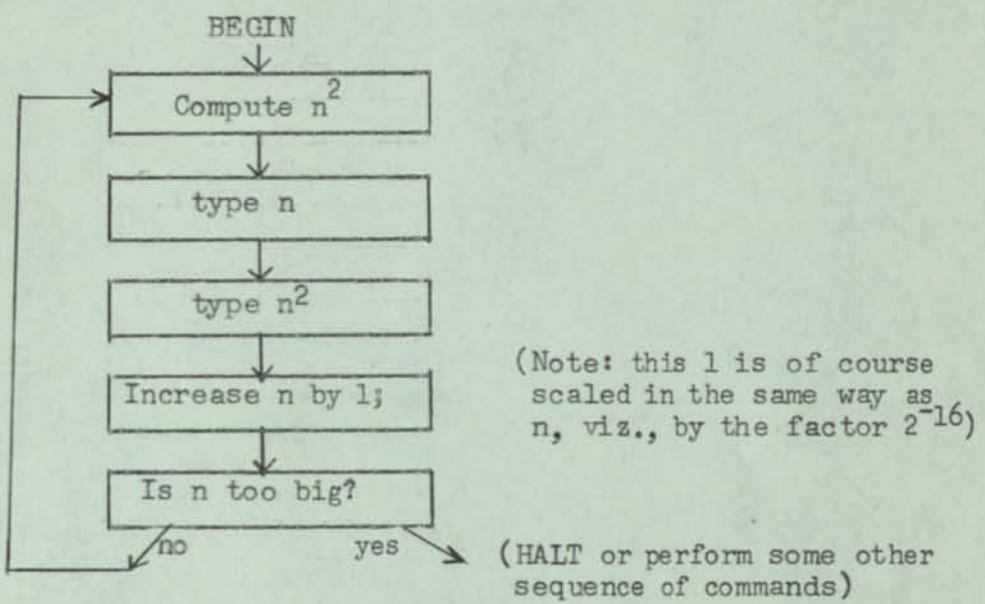
cell no.	α	β	γ	δ	F	Remarks
000	206	206	150		10	$1^2 \cdot 2^{-32} \rightarrow (150)$
001	206			000	02	type $1 \cdot 2^{-16}$
002	150			000	02	type $1^2 \cdot 2^{-32}$
003	207	207	150		10	$2^{-32} \cdot 2^2 \rightarrow (150)$
004	207			000	02	type $2 \cdot 2^{-16}$
005	150			000	02	type $2^2 \cdot 2^{-32}$
.	.	.	.		.	.
.	.	.	.		.	.
.	.	.	.		.	.
147	255	255	150		10	$2^{-32} \cdot 50^2 \rightarrow (150)$
148	255			000	02	type $50 \cdot 2^{-16}$
149	150			000	02	type $50^2 \cdot 2^{-32}$

The blanks above are irrelevant; usually they would be taken as 000, to save punching time. This method is quite long and wastes a great deal of storage. A cyclic procedure will do the same job, while at the same time requiring much less storage (not to say effort in writing and punching the commands).

In order to accomplish our task we write, in the form of a "FLOW CHART," the steps which we wish SWAC to take in sequence and which, in fact, SWAC has done in the above setup. Such a diagram is the following:

2. Beginners' Guide to SWAC

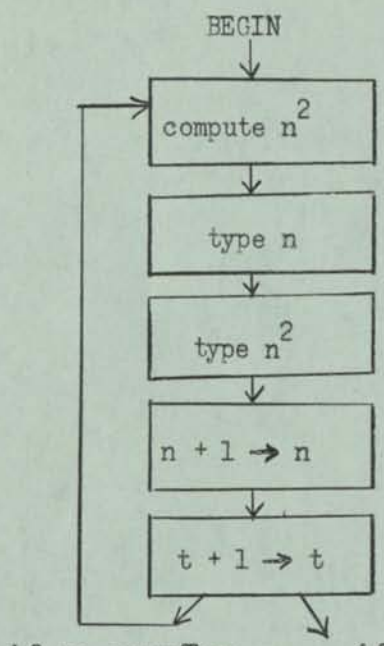
Flow Chart



n is initially taken to be 1 when SWAC begins the above. Then on each cycle we will compute n^2, type n and n^2 and repeat until n exceeds 50, when we stop. The checking to see if n is too big is a method of causing SWAC to go through the above loop 50 times. One method of doing this is to store a positive number t which is increased by a constant K each time through the loop. If t is set initially at a value t_o such that $t_o + 50K = 1$ while $t_o + 49K < 1$, and if this increasing is done by an (04) ADD command, then at this last (50th) step there will be an overflow so that SWAC follows a different sequence of commands. (Such an operation of "counting" the number of times through a loop is called tallying.) We may actually economize on storage space by scaling so that we may use here the same constant ($K = 2^{-16}$) as was used to "modify" n. If we do this, we may rewrite the flow chart as follows (each operation box now represents exactly one SWAC command):

2. Beginners' Guide to SWAC

Flow Chart



This "routine" apparently requires:

For Commands: 5 cells (for the 5 operations above)

Constants : 1 (must store $1 \cdot 2^{-16}$ to increase n and t)

Variables : 3 $\{n, n^2, t$

However, we will actually need one other command since, after the branch point add command which causes t + 1 → t, the usual (if $n \leq 50$) sequence of events is to go on to the next cell (i.e., $\epsilon + 1 \rightarrow \epsilon$). Therefore we need a vacuous command following the command t + 1 → t to cause SWAC to go back to the beginning of the loop (i.e., to the command which computes n^2). The code now might look as follows:

2. Beginners' Guide to SWAC

	cell no.	α	β	γ	δ	F	Remarks
COMMANDS	000	008	008	007		10	$n^2 \rightarrow (007)$
	001	008			000	02	type n
	002	007			000	02	type n^2
	003	008	009	008		04	$n + 1 \rightarrow n$
	004	006	009	006	(exit)	04	$t + 1 \rightarrow t$
	005	007	007	007	000	07	return to 000.
VARIABLES	006	[255	206	000	000	00]	t
	007						n^2
	008	[000	001	000	000	00]	n
CONSTANT	009	000	001	000	000	00	2^{-16}

NOTE: The word "exit" in δ of command 004 represents the address of the next command to be obeyed after this routine is completed.

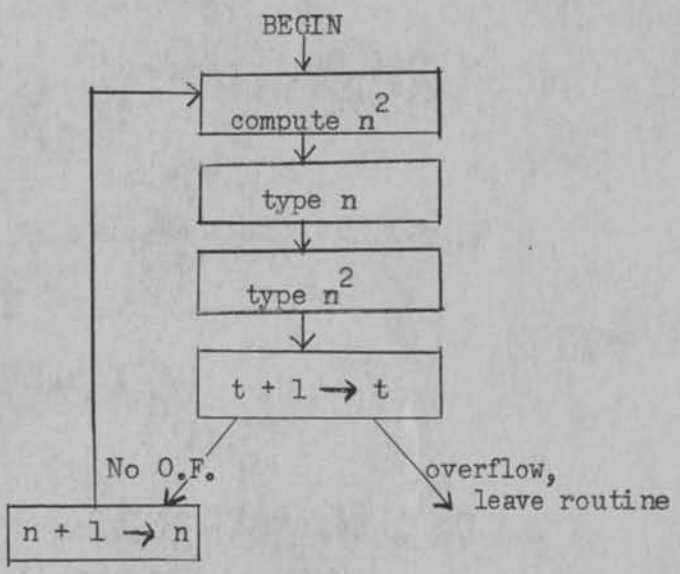
The first 4 commands compute n^2, type n and n^2, and "modify" n. The 5th command (in cell 004) is the tallying command. It increases the number in cell 006 by 1 at each loop until it has been increased 50 times, whence it will overflow and SWAC will take its next command from the cell whose address is placed in the position marked "(exit)". The command in 005 is a special command whose sole purpose is to cause SWAC to go back and repeat commands 000 to 004. The α, β, and γ parts of this command have to be filled in some (usually vacuous) fashion and the numbers 007, 007, 007 have been used, as this will give (007)-(007) $\rightarrow$ (007) so that 007 is cleared. This will not cause any trouble as we are finished by this time with the contents of 007.

2. Beginners' Guide to SWAC

The brackets surrounding (006) and (008) in the above routine are used to indicate that the words in question represent the initial values of variables.

By rearranging the sequence of operations the above-mentioned vacuous command can be eliminated (the same thing could also be accomplished by other devices). This can be done with the following flow chart, e.g., (actually, the commands could be arranged in other ways, also):

Flow Chart



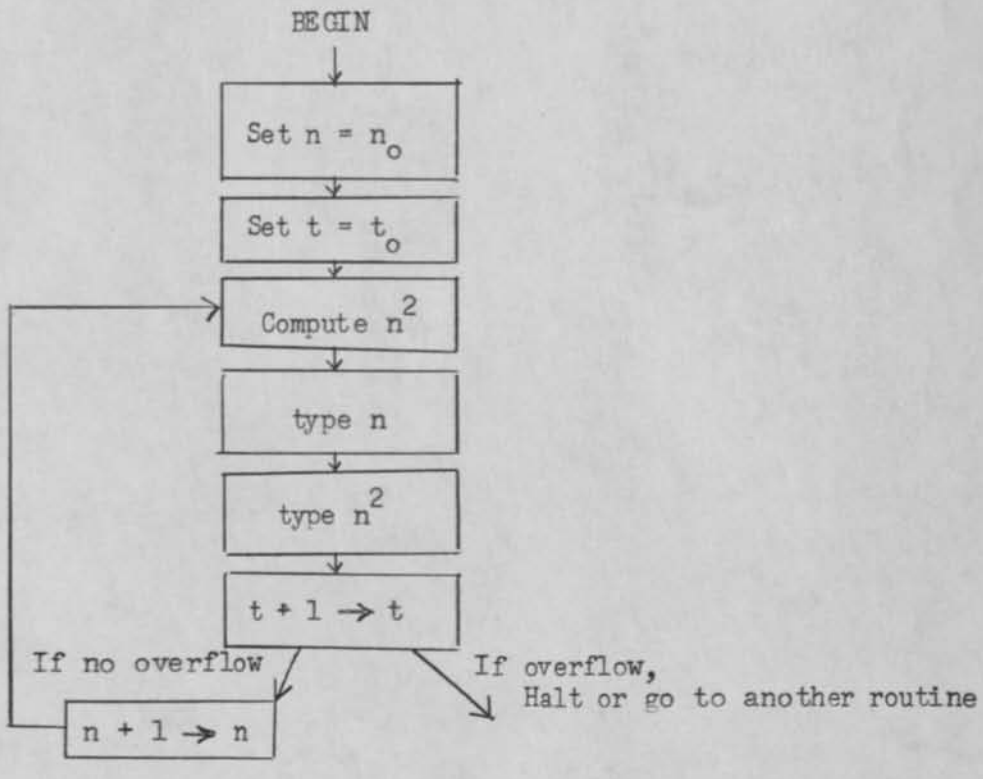
NOTE: Here we would use a special add command for n + 1 → n, which would have the effect of transferring control to the"compute n^2" command).

The routine as outlined above is a "single shot" affair in the sense that it can be used only once. After going through it, the contents of the cell containing t_o and the initial number $n_o = 1$ have been altered so that if we tried to use this routine again it would not function correctly. Usually,

in an actual problem there will be several of these smaller routines and they will be used over and over again. This means that provision must be made so that this routine could be reused. This is done by including several extra commands at the beginning which set the variables n and t at their correct initial values n_o and t_o. Even when a routine is to be used only once it is a good idea to do this so that, when code checking, the routine can be used over if desired.

If this example is reformulated so that it computes squares of the integers $n_o(1)(n_o + K)$ and resets itself if used again, the coding would be done as follows:

Flow Chart



2. Beginners' Guide to SWAC

Here we need: 7 commands

4 constants { n_o, t_o, 2^{-16}, 0 } (to increase n and t) (needed for setting n_o and t_o)

3 variable cells { n, n^2, t }

The code might then be:

	Cell no.	α	β	γ	δ	F	Remarks
Commands	000	009	010	011		04	$n_o \rightarrow$ (011) (Set n to its initial value
	001	008	010	013		04	$t_o \rightarrow$ (013) (Set t to its initial value)
	002	011	011	012		10	$n^2 \rightarrow$ (012) (Compute n^2)
	003	011			000	02	type n
	004	012			000	02	type n^2
	005	013	007	013	010	04	$t + 1 \rightarrow t$, exit to 010 if overflow
	006	011	007	011	002	05	$n + 1 \rightarrow n$, 002 $\rightarrow \in$ (go back and compute n^2 again).
Constants	007		001				2^{-16}
	008						t_o
	009						n_o
	010						0
Working Space*	011						n
	012						n^2
	013						t

*The advantage of putting working space last is that this avoids punching cards for these cells.

In this example it was not obvious from the flow diagram how many cells would be needed (e.g., it did not indicate the "necessity" of a zero for setting n_o and t_o, although you actually could use 1 with n_o-1 and t_o-1). If this is the case we may use a form of what is called relative coding. Rather than assign fixed addresses to commands, variables and constants, we may assign commands to cells with addresses C + 000, C + 001, C + 002, ••• , constants to cells K + 000, K + 001, ••• , and variables to cells V + 000, V + 001, ••• , e.g. Then, after completing the code, we can count the number of cells used by each category and choose the values of C, K and V so that these blocks (commands, constants, working space) lie in different portions of the memory.

Other SWAC Commands:

COMPARE: (08) $(\alpha) - (\beta) \rightarrow (\gamma)^*$ $\begin{cases} \epsilon + 1 \rightarrow \epsilon & \text{if result} < 0 \\ \delta \rightarrow \epsilon & \text{if result} \geq 0 \end{cases}$

SPECIAL COMPARE: (09) $|(\alpha)| - |(\beta)| \rightarrow (\gamma)$ $\begin{cases} \epsilon + 1 \rightarrow \epsilon & \text{if result} < 0 \\ \delta \rightarrow \epsilon & \text{if result} \geq 0 \end{cases}$

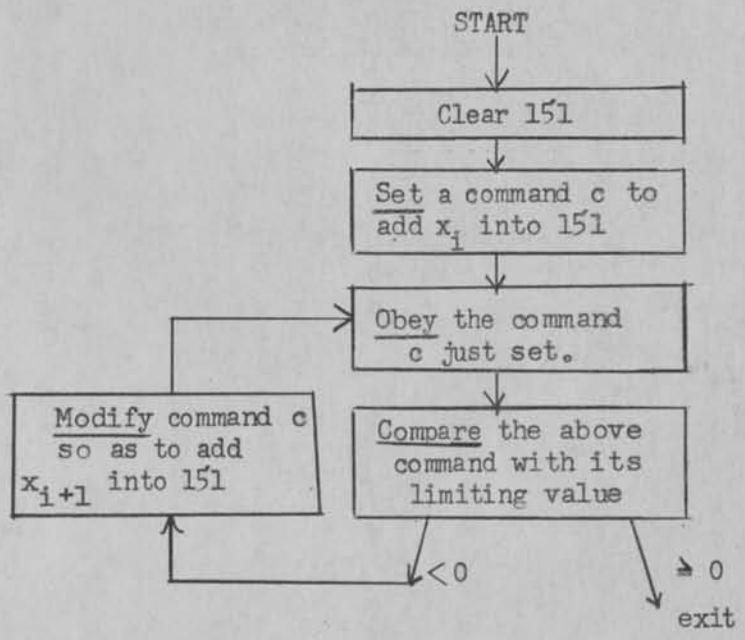
These also serve as branch point commands in that SWAC follows two different courses of events depending on whether the difference ≥ 0 or < 0.

One use of these commands is to tally. Suppose we have numbers x_i, i = 1, ••• , 50 stored in cells 101 to 150 and we wish to find their sum (ignoring any overflow) and place it in cell 151. The following flow diagram shows one way of doing this:

---

*See footnote, page A 2.3a-3.

2. Beginners' Guide to SWAC



The idea here (which is a basic device for performing something a specified number of times) is to use the same command over and over to add x_i into 151, each time increasing the appropriate address so that the next x_i will be added.* When all the x_i have been added, the command will reach some limiting value, at which time we wish to stop. This state is determined by comparing the modified command at each stage with its limiting value--when the 2 are equal (compare command gives zero result) the computation goes out of the loop. The code might take the following form:

---

*It is worth noting that this particular order (viz. SET the command, OBEY the command, COMPARE command with its limit, MODIFY the command if less than its limit) of commands can always be used to do a single thing a specified number of times. Other orders of the 4 operations, SET, OBEY, COMPARE and MODIFY are also possible, but it is well always to use a particular order in coding such an elementary loop in order to avoid confusion.

2. Beginners' Guide to SWAC

Cell no.	α	β	γ	δ	F	Remarks
000	151	151	151		06	(151) - (151) = 0 → (151) (clears 151)
001	004	005	008	008	05	Puts command in (008) and obeys it, since 008 → ϵ.
002	008	006	009	exit	08	Compare add command with its limiting value.
003	008	007	008	008	05	Modify command and obey it.
004	101	151	151	002	05	DUMMY to set (008)
005			Zero			
006	150	151	151	002	05	COMPARATOR (limiting value of add command).
007	001					MODIFIER (to change add command)
008	[the command to add x_i into 151 is stored here]					
009	Wastebasket — to dispose of useless results.					

The command that actually forms the sum is placed in cell 008 and obeyed there. We need cell 009 as a "Wastebasket" (most problems require one for depositing trash) to put the useless result of the compare command which is in cell 002.

Other commands available in SWAC are the following:

F = 12 (PRODUCT) — (Unlike the other multiply commands this keeps the full 72 bit product):

Most significant part of $(\alpha)(\beta) \rightarrow (\gamma)$
Least significant part of $(\alpha)(\beta) \rightarrow (\delta)$
} then $\epsilon + 1 \rightarrow \epsilon$

The correct sign of $(\alpha)(\beta)$ appears in both (γ) and (δ); however, in case either one of these parts of the product is zero, caution must be exercised

in trying to use its sign for later calculations.

Some uses are:

1) To get a more accurate product.

2) To clear two cells with 1 command.

e.g., to clear cells 024 and 232 assuming zero to be in cell 047.

α	β	γ	δ	F
047	047	024	232	12

Then $(\alpha)(\beta) = 0 \cdot 0 = 0$ gives 0 in both 024 and 232.

3) To clear one cell and set another cell with a certain constant.

E.G., clear 045 and set 169 with a number which appears in cell 124. We need a 1_F or 2^{-36} to act as one factor; assume this to be in 241.

Then

α	β	γ	δ	F
124	241	045	169	12

gives $(124)(241) = (124) \cdot 2^{-36}$.

The most significant part, $0 \rightarrow (045)$

The least significant part, $(124) \rightarrow (169)$.

F = 14 (EXTRACT)

(α) is called the EXTRACTOR. It determines which digits of (β) are to be extracted [i.e., preserved] and which are to be destroyed.

(β) is called the EXTRACTEE. It is the word upon which the extract operations are performed.

(γ) result

(δ) amount of shift if any.

2. Beginners' Guide to SWAC

The extract operation first forms a number which has 0's in digit positions corresponding to 1's in the extractor (α), and the same digits as (β) in those positions corresponding to 0's in (α). The result is shifted left δ binary places if $0 \leq \delta \leq 036$, or right $\delta - 064$ binary places if $064 \leq \delta \leq 100$, and rounded off. The result is placed in γ. It should be noted that the sign digit can be either extracted or destroyed also.

E.g., suppose the command is

α	β	γ	δ	F
026	123	240	070	14

where (026) = -. 0101 1011, 1010 1111, 0000 1100, 1100 0000, 1100

(123 = -. 1100 1111, 0000 1000, 1110 0101, 0101 1110 0011

After the extract but before the shift we get

+. 1000 0100, 0000 0000, 1110 0001, 0001 1110, 0011,

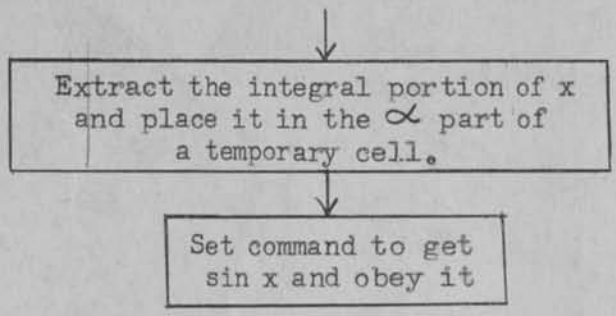
which upon being shifted right 070 - 064 = 6 binary places, and rounding, goes into 240 thus:

+. 0000 0010, 0001 0000, 0000 0011, 1000 0100, 1000

→ (240)

The amount of shift is determined by the last 6 bits of δ, and whether it is right ($\delta_2 = 1$) or left ($\delta_2 = 0$) is determined by the second digit of δ. An example of the use of the extract command is the following: Suppose we have a table of sin x, $x = 0(1)45^\circ$, stored in cells 100 to 145. Then suppose we want to look up sin x where x is given in degrees in the scaled form $2^{-16}x$. A flow chart to do this would be

2. Beginners' Guide to SWAC



The coding would be as follows (assume 002 initially to have the value $2^{-16}x$).

	dell	α	β	γ	δ	F	Remarks
Start→	000	005	002	002	008	14	extract [x] → (002).
	001	004	002	002	002	05	set command to get sin x and obey it.
	002	[	$2^{-16}x$			]	temporary (initially x)
	003	000	000	000	000	00	zero storage
	004	[100]	003	002	(exit)	05	dummy
	005	255	000	255	255	15	β EXTRACTOR

The result sin x is then left in cell (002).

Here we have taken the integral part of the number x and added it to 100 to form the address of the correct quantity in the table. EXTRACT is also used for getting at information which is closely packed (i.e., more than 1 datum per word).

00 INITIAL INPUT: This causes SWAC to read one word from the input device and transfer it to cell ϵ, then advance ϵ counter by 1, i.e.,

(input device) → (ϵ) and $\epsilon + 1 \rightarrow \epsilon$

01 INPUT: (input device) → (α) and $\epsilon + 1 \rightarrow \epsilon$

An input device (collator, keyboard or drum) is selected by δ and the word is transferred to address α.

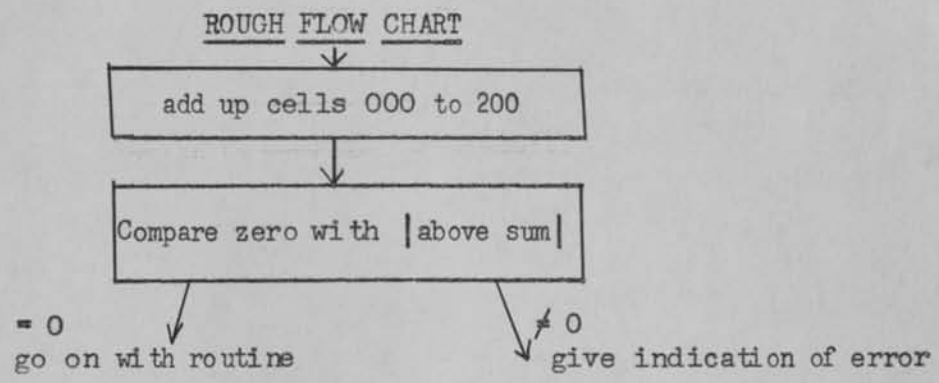
These commands may be used to HALT SWAC by calling for input from the keyboard or from the collator, provided the collator motor is not idling. Reference to the drum memory is also accomplished with the 01 COMMAND. For a complete description see section B6 of the SWAC Manual.

The above commands complete the list of 13 commands available on SWAC. For a further breakdown (into "periods") of each of these commands see the section on "Command Analysis" (part D 1.2) in the SWAC Manual.

2. Beginners' Guide to SWAC

c. Miscellaneous Techniques and Devices

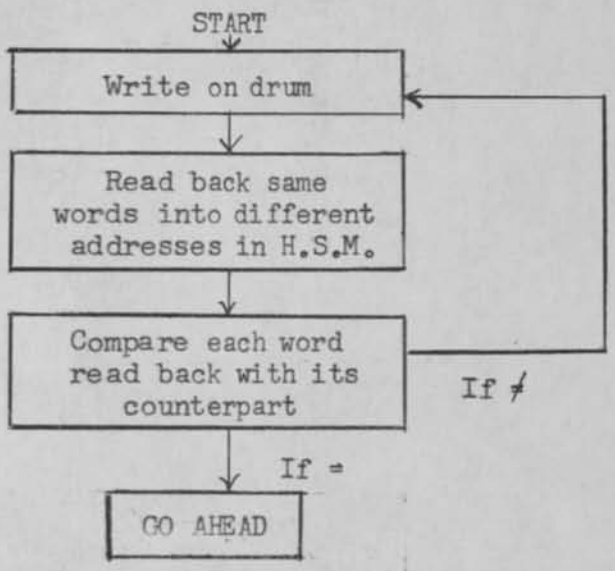
1. Sum (and difference) checks: In the course of a long calculation, the possibility of machine failures is increased. One way of detecting these and sometimes correcting them is the use of sum checks. A sum check consists of adding up the contents of a large number of cells, neglecting any overflow, and comparing this with the predetermined correct value of the sum. Then if any word in the sum has been changed the sum will not check. Of course such a check cannot include cells which contain variables unless these are set to some fixed value each time the sum check is made. This is one reason for putting all constants and instructions in one block and keeping the temporaries (working space) separate. For example, suppose that the instructions and constants are in cells 000 to 199 and that cells 201 to 255 contain variables. We assume that the negative of the correct sum of cells 000 to 199 has been calculated and stored in 200. A check may then be done as follows:



Sum or difference checks may also be used in connection with either reading from or writing on the drum, although the parity check digit feature of the new drum largely obviates the necessity for a sum check in connection with reading from the drum. The use of a difference check in connection with drum loading (writing on the drum) might be made as follows:

2. Beginners' Guide to SWAC

ROUGH FLOW CHART



2. Arithmetic checks can be included at various points in a routine to make sure the routine is working correctly. For example we might be using routines to compute sin x and cos x. These could be checked by computing $\sin^2 x + \cos^2 x$ and seeing if the result is acceptably close to 1.

A problem may also be done two different ways and/or on two different days; comparison of results can be done either by sight or with the use of auxiliary IBM equipment.

3. Breakpoint output. The command:

α β 000 096 02

will display the contents of cells α and β in the R and M registers respectively and halt the computer (if breakpoint switch is on) so that the operator can visually determine whether some intermediate result is correct. For details see C 2.5d.

2. Beginners' Guide to SWAC

4. Breakpoint digit. The $\beta 5$ cathode ray tube provides a 38th bit in all words of H.S.M. If the digit on this CRT is a 1, when SWAC tries to obey the word in that cell as a command it will halt. This can be used to stop a routine when it gets to some specified point. The breakpoint digit may be made 1 by punching column 79 of the corresponding word on the SWACARD. For details see D1.3.

2. Beginners' Guide to SWAC

5. Remarks on input coding and output coding. (Reference: Sections C 2.5a,c, D 1.5 - D 1.7) When it is desired, during the course of a SWAC computation, to feed information through the collator, we may use either the initial input (00) or the input (01) commands. Normally 00 is used when loading a program into the machine initially, while 01 is used for later inputs.

To transfer a word from an IBM card into SWAC via the collator, it is necessary to have a word pulse (WP) punch in that row of column 5 on the IBM card which contains the word in question. The word may be in any of the rows 9 through 1 and X (a special jack plug in the collator plugboard allows the use of the 0 row as well; rows 9 - 1 and X constitute the "standard" input via collator).

Speaking in a somewhat non-technical fashion, a word pulse is a signal to SWAC to a) store[1] the contents of the row in question in the memory; b) advance the epsilon counter by one (i.e., $\epsilon + 1 \rightarrow \epsilon$). Furthermore, the presence of a word on a given row of a SWAC card means that the collator brushes will "read" this word even if there is no word pulse punch on that row, and superimpose this word on the contents of the R register of SWAC. If, in addition, there is a word pulse punch on this or a subsequent row, the contents of the R register will then be transferred to the high speed memory. Thus, if one wishes not to have a word read into SWAC, one must delete not only the word pulse on the row but the word as well.

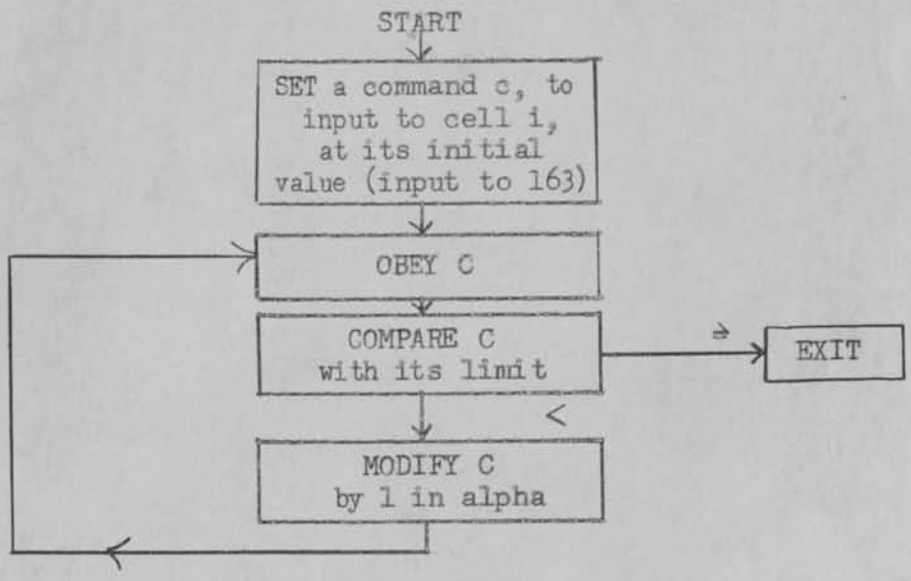
---

[1] The contents of the row in question are stored in address alpha when F of the command being obeyed is 01, or in address epsilon when F of the command being obeyed is 00.

2. Beginners' Guide to SWAC

Since the collator feeds cards at the approximate rate of 240 cards per minute, there is some small length of time between the reading (by collator brushes) of one row on the card and the next row. Actually, there is at least one millisecond for SWAC to perform computations while the collator brushes are between rows of the card.[2] This allows about 16 addition times for SWAC to do computing between the rows. This feature means, e.g., that we can compare an input command with its limit, modify the command and then obey this (next) input command.

For example, if we wish, in the middle of a computation to read 20 words from cards into cells 163 through 182, we can do it as follows:



[2] There is approximately $\frac{1}{4}$ second or 250 milliseconds (m secs.) of reading time per card which (because of the space between cards) is equivalent to about 16 rows per card. Thus we have about 16 milliseconds for each "row cycle." Some small fraction of this is free between holes, which may be off center if punched on different or poorly gauged machines. A conservative guess is that

2. Beginners' Guide to SWAC

Since the cycle in the above routine requires only 2 addition times between input commands, it does not interfere with the operation of the collator. The code for the above problem might be as follows:

	Cell No.	α	β	γ	δ	F	REMARKS
COMMANDS AND WORKING SPACE	105	036	017	106		04	SET C
	106	[163			000	01]	OBEY C HERE
	107	106	037	waste-basket	EXIT	08	ARE WE THROUGH?
	108	106	020	106	106	05	MODIFY C AND OBEY AGAIN
CONSTANTS	020	001					MODIFIER
	017						ZERO
	036	163			000	01	DUMMY
	037	182			000	01	COMPARATOR

It is, as mentioned above, also possible under special circumstances to use the initial input command (F = 00) during computation. To carry this out in the previous example: first, clear cells 163 through 182; second, place the command to be obeyed after the input operation in cell 183; third, set epsilon at 163,

Coding for punch output (cf. C 2.5c and E 1.7) is very similar to coding for collator input, in that SWAC computation can take place between card rows. The punching rate is approximately 100 cards per minute.

---

about 1 millisecond is free in every case; however, experiments by K. E. Ralston and J. L. Selfridge showed that, with cards punched on our IBM 513 Reproducer, at least 40 add times of free time existed between rows.

2. Beginners' Guide to SWAC

These are the flexible characteristics in punchout: (a) word pulses may be punched on any combination of rows of the cards to be punched, according to the manner in which the reproducer plug board has been wired; (b) from one to twenty-four words may be punched on a card, again depending to some extent on the plug board being used.

We must remember, however, that input via collator is limited to the eleven rows 9 through 0 and X, one word per row. Thus it is not possible to get as many of the twenty-four words on one card back into SWAC via collator as one wishes.

Coding for direct tabulator output is very similar to output coding for the punch. Coding for typewriter output in any of its many ramifications is a very detailed proposition. (cf. C 2.5d and D 1.8.)

2. Beginners' Guide to SWAC

d. Putting the Problem on SWAC

The many details which actually are involved in the running of a problem are exemplified by the following.

Suppose we think of the routine coded for computing n^2, n = 1(1)50. In putting the problem on SWAC we must then go through the following steps in the indicated order:

1. Analyze the problem and prepare the code on special SWAC coding sheets, giving suitable descriptions in the "Remarks" column so that we will later be able to understand what the code is supposed to do.
2. Prepare a decimal deck. Most routines are written with commands in the decimal command form that we have used in previous examples. Data may also be left in decimal form. If this is the case we next want to punch our program onto a set of IBM cards, known as DECARDS. There are a number of specific FORMATS for these decards, depending on what kind of a program is written. A typical such format is the following:

Column	
1-3	card number
4	blank
5	word pulses (all rows punched)
6	blank
7-9	α
10-12	β
13, 14, 27	γ
16-20	problem number
28-30	δ
31,32	F
34	blank if word is +, x-punch if word is -.

2. Beginners' Guide to SWAC

Such a deck is most conveniently prepared on the IBM 024 punch with the aid of a master card. The personnel in the IBM room will gladly assist anyone in doing this.

3. List the DECARDS on the 402 accounting machine. Proofread this list against the original written routine. One person should read the list aloud and another person should follow the original.

4. Correct any mistakes.

5. Convert the DECARDS to SWACARDS on SWAC by use of the appropriate conversion routine. If one wishes to be conservative, do this twice on two different days and compare the results to make sure of the correctness. A few of the conversion routines check the conversion, however, and a few even furnish a device for checking the punching of the answers.

6. Supply the converted (BINARY) deck with any special punches necessary, such as $\delta \rightarrow \in$ or Breakpoint, and any unneeded punches removed. If tape has been used to "remove" any punches, such cards should be reproduced immediately so as to eliminate the possibility of their jamming the collator input to SWAC.

7. To put the problem on SWAC

   a) Clear (i.e., "brainwash") the memory by pressing the cl. m button.

   b) Clear $\in$ by pressing cl $\in$ button.

   c) Set SWAC on CONTINUOUS and press OOPB once to get to period 3. Check to see that $\in$ counter is clear.

   d) Put SWACARDS followed by 2 blank cards in collator and press collator start button.

2. Beginners' Guide to SWAC

The routine is now in the memory. If the ϵ counter is at its correct starting value we need only press OOPB to start; if not, we may change ϵ by going to period 7 on COMMAND and "typing" in a suitable instruction to R, then transferring it to C. The $\delta \rightarrow \epsilon$ feature may also be used in some instances.

8. At this point when the code operates incorrectly we wish we had done something else earlier. That is, we should have:

Prepared a complete set of instructions for running the code. It should contain all indications that tell us whether the code is running correctly, such as listing any typeouts, breakpoint displays, etc. in binary, hexadecimal (hex.) or whatever form they will come from SWAC. If something does go wrong, we should have a list of all intermediate calculations in hex. so we can check the calculation step by step as results appear on the various console neons. It is futile to sit at the SWAC console and try to check a routine when one doesn't have written out (as it is awfully easy to forget) exactly what is to happen. The myriad flashing neons of SWAC will only add to the confusion of a beginner.

9. Remember that in every stage of the process:

HASTE MAKES WASTE.

2. Beginners' Guide to SWAC

e. Miscellaneous Remarks

There are certain aspects of coding which probably do not belong in the notes on the elements of coding, at least not in detail. Some of these aspects are mentioned briefly in this paragraph, with appropriate references.

1. SUBROUTINES. When the coder is faced with the necessity of doing some "elementary" operation such as division, square root, sine x etc., as part of a larger problem, he must write a subroutine to accomplish this. This subroutine will usually involve an iterative process. This is true in the case of such things as division, extraction of roots and the like. Various devices for doing these exist and many of these subroutines have been constructed already and are available in our library of subroutines (available in the Numerical Analysis Research library).

When the coder is faced with the problem of computing the sine of x or some other transcendental function, for example, he may choose to do this by using a finite approximating series for the function in question.

The above illustrates the important fact of life that coding for SWAC amounts to reducing everything to a finite set of operations which may later be repeated at will.

2. DRUM. In any problem of reasonable size the coder must store material on the magnetic drum. As mentioned above, this is done by writing an "output" command whose delta part (112) specifies output from the high speed memory to the drum (i.e., "writing" on the drum).

Since it takes approximately 1000 times as long to gain access to a word on the drum as to one in the high speed memory, it is customary (in problems where computing time is an important element) to put the less

2. Beginners' Guide to SWAC

frequently used material on the drum. For example, the elements of a matrix which are used either a row or a column at a time might be put on the drum while the program which performs operations on the rows may be in the high speed memory. Sometimes, of course, it becomes necessary because of the magnitude of the problem to store part of the instructions themselves on the drum in addition to constants like the elements of a matrix.

As implied by the example in A 2.3c, it is desirable, if one wishes to be conservative, to check the correctness of material written on the drum when any writing is done. For this purpose, a drum writing subroutine similar to the one outlined there can be used.

Sometimes, because the problem in question is so large, it is necessary to "close-pack" information on the drum in the sense that every word of certain channels on the drum is used for storage. This would be true, for example, if one had a matrix with 400 rows and 20 columns in which case one could store the various rows of the matrix consecutively on channels of the drum, first row occupying the first 20 positions of channel 0, the next row occupying the last 12 positions of channel 0 and the first 8 positions of channel 1, etc.

As a specific example of this, in one problem it was necessary to store certain numbers $p(a, \alpha)$, $1 \leq a \leq 5$, $1 \leq \alpha \leq 9$, there being more than 1000 of them, on the drum. It was necessary to locate a random one of these numbers, there being $(a + 1)(\alpha + 1) - 1$ of each of these for a given pair (a, α). It proved to be possible to locate the first of the $p(a, \alpha)$ by means of a formula. In fact, the first $p(a, \alpha)$ was on channel 016 + N and became word 095 + K in the high speed memory where

2. Beginners' Guide to SWAC

$$N = \left[\frac{G^* + 1}{32}\right]$$

$$K = 32\left\{\frac{g^* + 1}{32}\right\} \quad ,$$

where $[x]$, $\{x\}$ respectively denote the integral part and fractional part of x, and where

$$g^*(a, \alpha) = f^*(a, \alpha) + \left[\frac{f^*}{31}\right] \quad ,$$

$$f^*(a, \alpha) = 9(a - 1)(3a + 5) + \frac{(\alpha - 1)(a\alpha + 2a + \alpha)}{2} \quad .$$

The order of loading of a SWAC problem which uses the drum probably would be as follows: a) clear the drum (this is desirable if you wish to count on the drum having zeroes anywhere), b) load the drum-loading routine in the high speed memory, c) using the above drum-loading routine, load the drum with what you want on it initially, d) load the main routine in the high speed memory, e) take off! For details about the drum, see paragraphs D 1.4 and D 6.1.

3. INTERPRETATION SCHEMES. It is possible, by loading an appropriate program into the high speed memory of SWAC, to cause SWAC to behave like a different machine than it really is. For example, it is possible to cause SWAC to behave as though it were a high speed card programmed calculator. In this and similar instances, the programmer would be able to write his code in a way entirely different from the usual way in which one writes codes for SWAC. In effect the "language" which SWAC "understands" when so loaded is different from the ordinary "language."

2. Beginners' Guide to SWAC

A program which, when loaded into the high speed memory of SWAC, causes SWAC to behave in this fashion so far as the coder is concerned is usually referred to as an interpretation routine. Many such interpretation routines exist. Some of them are called SWACPEC, Floating SWACPEC, floating SWAC, and THE interpretation routine (code number 00300). The latter is described very well in C 2.3a and E 2.1 of the SWAC manual.

The usual effect of using one of these interpretation routines is that one has an easier job of coding to do, at the expense of much slower computing. This is sometimes desirable and the interpretation routines definitely do have a place in the scheme of things. However, any detailed descriptions of them probably do not belong in a set of notes on the elements of coding.

4. RELATIVE CODING. As mentioned on page A 2.3b-10, it is sometimes unfeasible to assign fixed addresses to commands, variables and constants on an a priori basis. In this case, a form of relative coding is used and one simple form of relative coding was described on the above referenced page.

The kind of relative coding (alphabetical relative coding) described on the aforementioned page required, probably, that the ultimate assignment of absolute numerical addresses be done by a human being. It is easily possible to avoid this operation by the device of using numerical relative coding, in conjunction with what is described as dictionary cards, for the ultimate translation of the program to a form in which each word is expressed in the usual 37 bit form. This kind of operation is described in some detail in paragraph C 2.2.

2. Beginners' Guide to SWAC

5. CODE CHECKING. One of the big problems that a coder is faced with is the problem of checking his code once it is written to determine whether or not it is correct. Various devices exist for trying to determine the answer to this question.

One of the "conservative" things one might do is to perform a hand computation paralleling at least one cycle of an iterative process, the computation of which SWAC is called upon to perform. The program on SWAC can then be operated one command at a time and the result verified by checking with the hand computed result. This quite obviously is a time consuming operation.

At the other end of the spectrum of ideas for code checking is the following: The coder loads his problem into SWAC, pushes the appropriate buttons hoping that the problem will run the way it is supposed to do. If it does run, fine. If it does not (i.e., if the program is detoured by virtue of SWAC's having gone into a loop which was unexpected), then the machine is stopped, the contents of the memory and, if necessary, of the drum are punched and listed, and one retires to his cubby hole to find out whether or not he knows what the difficulty is.

There should be some kind of compromise between the two extremes mentioned above. Although there is no absolutely foolproof device for determining whether or not the programming prepared for SWAC will solve the problem, there are certain systematic devices for eliminating some of the "bookkeeping" mistakes from the code. Probably among the best of these is the device invented by Dr. C. B. Tompkins involving the use of and listing of various properties of "elementary sequences." An elementary sequence is

2. Beginners' Guide to SWAC

a set of commands each one of which except the first has an unique predecessor, and each one of which except the last has an unique successor. Any program can be broken down into a finite number of elementary sequences. After the program has been written down, it is broken up into elementary sequences which are then serially numbered. For each such elementary sequence, one then lists the serial number of the sequences(s) from which entry to this sequence is gained, as well as the serial number(s) of the "exit" sequence(s). In addition, one lists the criterion for determining which elementary sequence the exit is made to. Furthermore, the commands which are set by each elementary sequence and the indices which are set are listed specifically. In addition, remarks can be put down for each elementary sequence. This is done in a systematic way and blank forms for doing this are available. As stated above, this device does not guarantee that you get all the "bugs" out of the program, but it helps.

In addition, it is desirable to maintain a listing of the variable cells with specific symbols, a statement of the position of each variable in the word, as well as other pertinent information about each variable. Again, this can be done in a fairly systematic fashion.

6. STRATEGY FOR CODING. In analyzing a substantially sized problem, for computational purposes, it is suggested that one start by drawing a very rough flow diagram of the operation which it is planned to carry out. In this rough flow diagram, each "operation box" might represent a complex of operations. For example, one operation box might represent the operation square root of x, another the operation: solve a system of 10 linear

2. Beginners' Guide to SWAC

algebraic equations in 10 unknowns, etc. After establishing this rough flow diagram in the format mentioned one can then explore the problem of accomplishing each of the complex operations. For each complex operation, for example, one can draw a detailed flow chart in which each operation box represents one SWAC operation. Ultimately, then, a program for the whole problem is broken down into a set of SWAC commands for each of which one can write its coded form. Of course, after one has a good deal of experience with writing programs for SWAC, it probably is not necessary to make such a detailed breakdown of each complex of operations as mentioned above. For instance, if the method for accomplishing certain complex operations is already known, one need go no further in drawing the detailed flow chart. Ultimately, of course, the whole program is made up of a finite number of elementary sequences of commands as defined by Tompkins.

7. SUMMARY. In this set of notes, which have to do with the elements of coding, many items necessarily have to be either omitted entirely or touched on only briefly. Such things as the myriad switches, buttons, flashing lights, etc., at the console have no place here (see article B 2). The same is true of details about operating any of the auxiliary machines associated with SWAC. No attempt has been made here to refer to the period breakdown of any commands (see article D 1.2). Such things as test routines (article D 2), delta to epsilon transfer (articles B 3.1, B 3.2, D 1.3b), spill-over and read around ratio, delay, floating point operations, multiple precision, etc., probably do not belong in these notes and have therefore only been mentioned here.

2. Beginners' Guide to SWAC

It is hoped that these notes on the elements of coding will provide at least a guide to the matter of coding for SWAC and that with these notes, anyone with a problem for the machine can, by searching out the appropriate references and talking to "experts," accomplish his objectives in a reasonable length of time.

2. Beginners' Guide to SWAC

f. Exercises.

1. The non-negative distinct quantities $x_i (i = 1, \cdots, 50)$ are in addresses $115 + i$; determine the largest x_i, store it in address 115 and its subscript in 114.
2. Given nos. x_i in address $i (i = 101(1)150)$, compute Σx_i (ACCOUNTING for overflow and put the result in cells 151 and 152.
3. Compute and punch the approximate values of y for $x = 0(.001)1$ if $y' = -y$ and $y = 1$ when $x = 0$ and if you approximate y' at $x = x_k$ by

$$\frac{y_{k+1} - y_k}{x_{k+1} - x_k} .$$

4. Given a proper fraction N in SWAC. Convert it to decimal form and type it out with the digits in correct order.
5. Devise a code to compute (using Newton's method) the reciprocal of the integer N, which is in SWAC and is scaled by 2^{-k}.
6. Devise a code to compute ab correct to 72 bits if a and b each have 72 bits (double precision multiplication).
7. Compute and punch a table of nos. τ_n for $n = 1(1)100$, given that $\tau_n[\text{Ln } \tau_n - 1] = n + \frac{1}{8}$.

2. Beginners' Guide to SWAC

g. A Short Bibliography

1. Annals, Harvard Computation Laboratory, 27, 1951, Aiken, H. H. (ed.), Synthesis of Electronic Computing and Control Circuits. (Serial)

2. Booth, A. D. and Booth, K. H. V., Automatic Digital Calculators, London, 1953.

3. Burks, A., Goldstine, H. H., and von Neumann, J., Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, Princeton, 1946. (Serial)

4. Engineering Research Associates, Staff, High Speed Computing Devices, New York, 1950. (QA 75.5 E6)*

5. Hartree, D. R., Calculating Instruments and Machines, Cambridge, 1949. (QA 75 H3)

6. University of Illinois, Digital Computer Programming Notes and Reports.

7. Wilkes, M. V., Wheeler, D. J., and Gill, S., The Preparation of Programs for an Electronic Digital Computer, Cambridge, Mass., 1951. (QA 75 5W)

---

*The numbers in parenthesis are the call numbers in the Numerical Analysis Research library.

1. Power Control

1. Power Cabinets

The SWAC Power Cabinets are located in the Electronics Laboratory, to the South of the room containing SWAC. On the East end of the large cabinet are located the power controls which enable the operator to turn the machine on or off. A smaller cabinet to the West holds the drum power controls.

The SWAC controls are shown in the diagram at the end of this section. The voltage control knob, the voltage meter which shows output from and input to the alternator, and a phase switch, are located at the top of the control panel on the Power Cabinet. Beneath the voltage control is a toggle switch for the control transformer. A switch to the right of this is unused at present. The three meters below these switches show filament supply voltages to the memory, arithmetic and control sections. A Powerstat directly underneath each meter controls the appropriate supply. Under the center Powerstat is a double button ON-OFF switch with pilot light. This is the alternator switch, which closes the circuits that start the motor-generator set in the generator shed.

On the drum Supply Cabinet are the controls shown in the figure at the end of this section. [The various meters and indicators will be mentioned in detail only if they apply to the operation of the SWAC, and not to the engineering exclusively.]

1. Power Control

2. Turning SWAC ON [Numbers in ( ) refer to list of switches, see section on Operating Switches.]

a. Press the ON button of the alternator switch (19) on the Power Cabinet. The red pilot light should go on. The meter at the top of the panel will slowly move to a value of approximately 232 volts. Read the top scale. If the value needs changing, move the voltage control knob (21) to adjust the voltage. The phase switch (22) should be set to phase 1, "alternator", when this adjustment is made. The correct voltage is: $232\ ^{+5}_{-1}$ volts.

b. Throw the control transformer switch (20) UP.

c. BE SURE THAT FILAMENT POWERSTATS ARE TURNED COMPLETELY OFF. They should be twisted counterclockwise as far as they will go.

d. At the Console, reset the fuse block lights by depressing the reset lever, located in the center of the cluster of fuse lights. Push the AC control switch (14) ON button. A sharp click will be heard from the Power Cabinet as various relays pick up; also the hum of the fans in the roof. The air conditioning compressor should also have started.

e. Returning to the power cabinet switch panel, turn the three SWAC Powerstats (23) clockwise slowly until each associated meter reads one volt. Slowly increase each filament voltage in turn in one volt steps until the proper voltage is reached. The correct voltage is marked on each meter. Note that the filaments in the main SWAC cabinet are now on, and their light should be visible through the glass doors. Note in the Log the time of AC ON.

f. If the Drum is to be used, turn on the drum AC at this point. See the instructions for turning on the drum power, Section on Power Control, Drum.

1. Power Control

g. WAIT 15 MINUTES before turning on the DC. Always give the filaments ample time to warm up. Push the DC ON button on the Console (15). If there are no blown fuses, various lights on the Console will now be on. Push the clear memory switch to reset the period lights.

h. Push the upper button behind the right-hand panel (16) to turn on the high voltage. A raster should appear in the lower monitor tube, and a bright spot, the origin, in the upper.

i. WAIT 5 MINUTES before trying to run any routines. Then proceed with the memory tests, as outlined under testing. Note in the Log the time of turning on SWAC DC.

1. Power Control

3. Turning SWAC OFF

a. Slowly push the DC OFF button (15) on the Console. When it is depressed about 1/16 to 1/8 inch, the high voltage will turn off, and the rasters in the memory will disappear. Then as the button is pushed further the DC supplies to the rest of SWAC (including the Drum, if ON) will be cut off. The pilot lights should go out.

b. Turn the SWAC Powerstats on the Power Cabinet counterclockwise to zero. Do this smoothly but not too rapidly. Also turn down drum Powerstat.

c. At the Console push the AC OFF button on the AC control switch. The air conditioning fans and compressor will turn off. At this time note in the Log the time of turning machine off.

d. Turn off the control transformer by means of the toggle on the Power Cabinet.

e. Turn off the drum AC on drum Power Cabinet at this time. See procedure for turning off Drum, Section on Power Control, Drum.

f. Push alternator OFF switch. The pilot light should go out.

1. Power Control

4. Drum

a. Instructions for turning Drum ON (Step f, section on Power Control, Turning SWAC ON)

1. Make sure that the drum Powerstat on the drum Supply Cabinet is turned counterclockwise to the OFF position. Push the filament ON button (24) and turn up the drum Powerstat slowly until phase 3 reads 113 volts on the meter.

2. The delay light will turn on in approximately two minutes. The DC may now be turned on either from the drum Supply Cabinet or from the Console (27), AFTER the SWAC DC has been turned on. If a blown fuse has been replaced in the drum circuits, the reset button (28) on the drum supply must be pushed before drum DC can be applied.

3. Push the lower button (17) of the drum reset switch, behind the right-hand panel of the Console. Period ring and epsilon should advance. This resets the address selector circuits of the Drum.

b. Instructions for turning Drum OFF

1. Drum DC will go OFF when SWAC DC is turned OFF. It may be turned off independently by means of the drum DC switch (27) on the Console or the drum Power Supply Cabinet.

2. Turn down drum Powerstat (counterclockwise) to turn off the drum filaments.

3. Push the filament OFF button. All pilots will go out except the red light marked AC POWER. This will turn off when the alternator is turned off.

2. Physical Layout

1. Operating Switches

The following list summarizes the most important operating switches on the SWAC Console or in the main cabinet. Reference can be made to this list whenever these switches are mentioned in other sections. There is no particularly logical order to this list.

(1) Clear Memory. Located on the platform of the Operating Console. A microswitch which, when pushed, clears all CRT's (i.e., reads zeros into all cells of every CRT.) It also sets the period light to period 7. If more than one period light is on (as may be the case when DC is first turned on) it turns off all period lights except 7.

(2) Clear Epsilon. A microswitch located on the platform of the Operating Console. Sets the epsilon counter to zero. It may be operated simultaneously with the clear memory switch.

(3) OOPB. A microswitch on the platform of the Operating Console. Supplies a "Start Function Table" pulse to the SWAC when pushed. The subsequent operation of SWAC depends on the setting of the mode-of-operation switch, and on the contents of the memory.

(4) OMPB. A microswitch on the platform of the Operating Console. Supplies pulse to allow one shift and add of a multiplication or a product, or one shift of an extract.

(5) NO RI. A spring return toggle switch on the right-hand panel of Operating Console. It inhibits the transfer of R→M on tape input, to permit correction of incoming numbers by hand. (Gives incorrect operation on collator input.)

2. Physical Layout

(6) Clear α10. A spring return toggle switch next to the NO RI switch. Clears the α10 CRT at any time during SWAC operation. α10 CRT stores a "one" in a cell position when a word has been read into that position of the memory, on input or during operation.

(7) Delay. A snap toggle switch and pilot light next to the clear α10 switch. Used in conjunction with mode-of-operation switch. With MO switch on COM, and delay ON, the SWAC will operate at reduced speed. The time delay available ranges from 64 μsecs to 512 μsecs per command. It is chosen by the setting of three toggle switches located inside the front left-hand doors of the SWAC cabinet. Table 1 shows how these switches are set for different delay:

Table 1 (U = up, D = down)

Delay (Switch)	Top	Middle	Lower
64 μsec.	U	U	U
128	U	U	D
192	U	D	U
256	U	D	D
320	D	U	U
384	D	U	D
448	D	D	U
512	D	D	D

(8) Breakpoint. A double throw toggle switch and a pilot light on the right-hand panel alongside the delay switch. It has snap action for on-off positions, and a spring return for clear-on positions. The breakpoint switch controls the operation of the β5 CRT. A full description of the breakpoint is given elsewhere in this report. The

2. Physical Layout

breakpoint light (indicating a digit in the $\beta 5$ CRT) is located in the row of neons marked "F" under the S register neons on the right-hand panel.

(9) Converted Output. Two three position switches in the main SWAC cabinet. One (a toggle switch) is located on the lower left chassis of the left-hand section (control). The other is on the patch panel, reached through the door in the East end of the SWAC. These switches must both be set to normal, or both to converted, otherwise incorrect action of SWAC will result. The center position of both switches is never used. The setting of the switches is indicated by red and amber neon lights on the right-hand panel of the Operating Console, marked CONV-NORM. Note that the neon indicators for $\delta 9$, F_1, F_2, F_3, and F_4 of the R register are connected to the corresponding positions of the A register when the switches are set to CONV. Therefore information stored in R will always show zeros in these positions, although the flipflops are storing the information correctly. If these indicators in the A register are on, then those in R will be on also.

(10) Patch Panel. In the East end of the SWAC cabinet are located two sets of hubs joined by a set of plug wires. The arrangement of these wires determines the format of the typed output of SWAC. The format is usually set for 9 base sixteen digits, and sign, followed by a "tab" impulse and "stop" impulse. Up to 12 digits may be typed, by a suitable rearrangement of the wires.

2. Physical Layout

(11) Origin. A snap toggle switch located in the right-hand section of the SWAC cabinet, on the fourth chassis from the right, bottom row. It shifts the position of the 000 cell on the raster by 8 rows. That is, the origin is either in the lower left corner of the raster, or 8 rows above, as viewed on the Operating Console monitor scopes.

(12) Memory Monitor Selector. The choice of a CRT to be displayed by the memory monitor is made by means of two dial switches on the center panel of the Console. The left-hand switch chooses one of the major groups α, β, γ, δ, F, or the sign. The right-hand switch chooses the particular CRT in the group. It is inoperative when the sign digit is displayed. The center position, between 4 and 6, is operative for α(α5: overflow digit CRT), β(β5: breakpoint digit CRT), and γ(γ5: CRT for γ4 is in the γ5 position in the memory rack). The γ4 position is inoperative. The position corresponding to "0", counterclockwise from "1", is operative on α(α10: read-in digit CRT). Positions 6-9 are inoperative for F.

(13) Mode-of-Operation. (MO switch) Located directly beneath the lower monitor CRT on the center panel of the Console, this selector switch provides four modes of operation:

a. Continuous (CONT)
b. Command (COM)
c. Period (PER)
d. Multiply (MULT)

When set on command, and with the delay switch ON, reduced speed operation of SWAC is possible.

2. Physical Layout

(14) AC Control. An On-Off push button switch and pilot light on the left-hand panel of the Console. This switch turns on the AC supply for filaments in the SWAC, as well as the fans which draw air through the SWAC cabinet. When turned off it shuts off all AC and DC supplies.

(15) DC Control. Two On-Off push button switches and pilot lights on the left panel of the Console. These DC switches supply direct current plate, screen and bias supplies to the SWAC and the Drum. Note, however, that turning SWAC DC off, also turns off the drum DC, and that the drum DC cannot be turned on unless SWAC DC is on also. However, when SWAC DC is turned on, drum DC must be switched on independently.

(16) High Voltage. A microswitch behind the upper left corner of the right-hand panel on the Console. It is the upper of the two switches located there. After DC is turned on, this switch must be pressed to supply high voltage to the CRT's. Until this is done no rasters show on the faces of the tubes.

(17) Drum Reset. Directly under the high voltage switch behind the right-hand panel. Serves to reset the counting register of the Drum after drum DC is turned on. May also be used to step OOPB rapidly on command at the rate of 60 times per second. This can only be utilized if no drum operations occur in the routine being obeyed.

(18) Letters-Numbers. A toggle switch on the second chassis from the left, bottom row, in the left-hand (control) section of the main SWAC. Allows the printing of letters or numbers on a typewriter output command which has 20 or 40 (base 16) coded in the δ part of the command (see section on Coding Procedures).

2. Physical Layout

(19) Alternator. An ON-OFF push button switch on the East end of the Power Cabinet. Turns on and off the motor-generator set which supplies the three phase AC for all SWAC operation.

(20) Control Transformer. A toggle switch on the East end of the Power Cabinet. It supplies a single phase of AC to the SWAC. The Flexowriter equipment, in particular, is on this line.

(21) Voltage Control. A control knob for adjustment of the output voltage of the motor-generator set.

(22) Phase. A six position control knob on the East end of the Power Cabinet which feeds the voltages from any of the three input phases or the three output phases of the motor-generator set to a voltmeter located beside it, for monitoring purposes.

(23) SWAC Powerstats. Three control knobs on the East end of the Power Cabinet which serve to adjust the filament supply voltages to their proper values. These controls are always turned off (completely counterclockwise) before SWAC AC is turned OFF, and not turned on until after SWAC AC is turned ON.

(24) Drum AC. An ON-OFF push button switch on the drum supply cabinet which supplies voltage to the drum motor, and in conjunction with the drum Powerstat, to the remainder of the drum circuitry.

(25) Drum Powerstat. A control knob on the drum supply cabinet which serves to adjust the line voltage to the transformers supplying drum AC. A meter located alongside monitors the voltage.

(26) Drum Phase. Allows monitoring of the three phases of input voltage to the Drum. It is a three position control knob on the drum power supply cabinet.

2. Physical Layout.

(27) Drum DC. A push button ON-OFF switch on the drum supply cabinet. It has parallel circuitry with the drum DC switch on the Console.

(28) Fuse Reset. A lever in the cluster of fuse lights on the left-hand panel of the Console, and a push button on the drum power supply cabinet. They reset the alarm circuits associated with SWAC and Drum, respectively. Used after a blown fuse is replaced.

2. Physical Layout

2. Console and Main SWAC - description

a. Physical Features

The SWAC room contains the SWAC memory, arithmetic, and control units (all in one cabinet), the drum, a collator, a card punch, an output typewriter, input tape unit, and the operating console.

In the laboratory, just outside the SWAC room, are located the power supply cabinet and the drum power supply. Each of these cabinets has mounted on it AC controls for turning the SWAC and Drum power on and off.

In addition to this equipment there are two power sheds external to the building. One houses the motor-generator (MG) set which supplies 3-phase 220V AC to the SWAC, and the other houses transformers, rectifiers, control relays, and associated circuitry.

Access to the main SWAC is through doors in the sides and ends of the cabinet. The center doors facing the console cover the memory units. The two pairs of doors on either side open on the control chassis, and the doors at the rear provide access to the arithmetic units. The product counter is at the West end of the cabinet, and the patch panel on the East end.

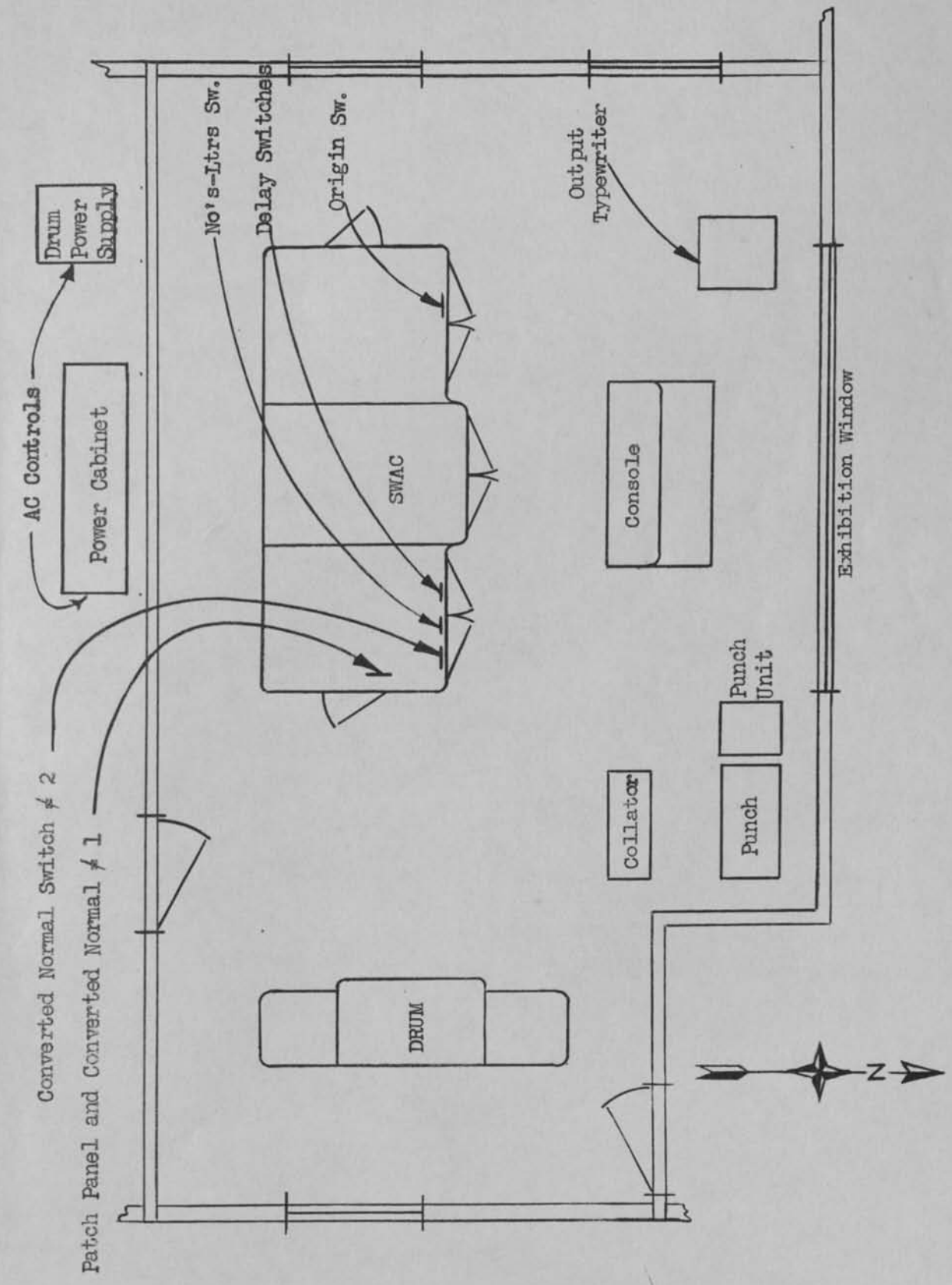
The operating switches and indicator lights are mounted on panels set on the top of an office desk. This operating console also houses some control circuitry, which is reached for repair and maintenance by opening the left-and right-hand panels.

2. Physical Layout

Flanking the console on the East side are the punch output unit, an IBM reproducer coupled to it, and an IBM collator. West of the console are the tape input and Flexowriter output units. Each of these pieces of equipment has its complement of switches mounted on it.

The drum cabinet is situated at the East side of the SWAC. It contains the drum, and all the circuitry for drum operation, except the power supply.

2. Physical Layout



2. Physical Layout

2. Console and Main SWAC - Description

b. Operating Console

The diagram at the end of this section shows the layout of the operating console panels. At the upper left are the fuse indicator lights. "Control" and "AC Line" are ON if SWAC is on. If any other light is on, it indicates a blown fuse. The DC cannot be turned on unless all the fuse lights are out. The lever at the center serves to reset the indicator lights. The drum DC switch is located alongside the fuse indicators. At the lower left corner of the panel are the AC and DC ON-OFF switches of the SWAC.

The M, A and R registers are located on the left panel. Each digit of each register is shown by one neon light. In addition, both M and A have overflow indicators. Reading from top to bottom and left to right on any register we have: sign, overflow, α_1, α_2, α_3, α_4, α_6, α_7, α_8, α_9, β_1, ... β_9, γ_1, ... γ_9, δ_1,... δ_9, 8 unused indicators, F_1,... F_4.

The right-hand panel holds the indicators for the C register, ε counter, S register, breakpoint digit, F or operation register, RI-RO lights (always lit), normal-converted output signal, product counter, and Input-Output register.

The four top rows of the C register indicators represent α, β, γ, δ of the command being obeyed. The two rows just beneath these represent the ε counter and the S register. All rows display addresses in binary form with the least significant digit on the right.

2. Physical Layout

The operation register, labelled F, indicates the function part of the command in the C register. It is located beneath the S register neons. The row of neons labelled PC (for product counter) is used by the engineers when trouble-shooting in the event of multiply, product, or extract failures. With MO switch on MULT, these neons indicate the number of shifts which have occurred in the product counter during one of these operations.

The Input-Output register displays the 2nd, 3rd, and 4th digits of the delta of any input or output command in the C register, thereby indicating which input or output device has been activated by the command.

Along the bottom of the right-hand panel are four toggle switches: NO RI, Clear α IO, Delay, and Breakpoint. Each of these switches, as well as the breakpoint neon, and the normal-converted output signal, is discussed in the section on Operating Switches.

On the extreme right of the right-hand panel are time meters for AC, DC, and running time. A voltmeter with selector switches enables the operator to check voltages at various points inside SWAC. To the left of this meter is a temperature meter (not operative at present) and a red temperature warning light. This flashes on and off if for any reason the temperature inside the main SWAC or drum cabinets exceeds a safe level.

The central panel of the Console holds two cathode ray tube (CRT) monitors. The upper, or operation, monitor shows the raster of spots representing the location of all commands currently being obeyed by the SWAC. The lower, or memory, monitor shows the raster of spots present in any one of the 40 CRT's which make up the SWAC memory. Selection switches

2. Physical Layout

are provided for easy choice of CRT. The left selector chooses the sign CRT, or any major group: α, β, γ, δ, F. The right selector chooses the particular CRT in the group. Each monitor tube has independent focusing and intensity controls.

Directly under the memory monitor is the "mode-of-operation" (MO) switch. It has four settings:

1. Continuous (CONT)
2. Command (and run "delayed") (COM)
3. Period (PER)
4. Multiply (MULT)

On a small platform in front of the monitor CRT's are the one operation push button (OOPB), the one multiplication push button (OMPB), a clear memory switch, and a clear epsilon switch (ε CLEAR m). Eight neon indicators (the period lights) show the stage at which the ring counter (which counts the periods of each command) is set. Normally only one of these is lit at one time; the one that is on indicates which period is about to be obeyed. If the mode-of-operation switch is set to (PER) the machine will perform one period operation for each push of the OOPB button, obeying step by step the command that is at that time in the C register. This mode-of-operation is used primarily for checking the various registers of the SWAC.

In multiplication, product or extract all the additions and shifts which take place occur, with one push of the OOPB switch, in period 4. These may be broken down into individual add and shift steps by putting the mode-of-operation switch on MULT in period 4 and operating with the OMPB switch. The result of each operation of OMPB is displayed in the A and R registers.

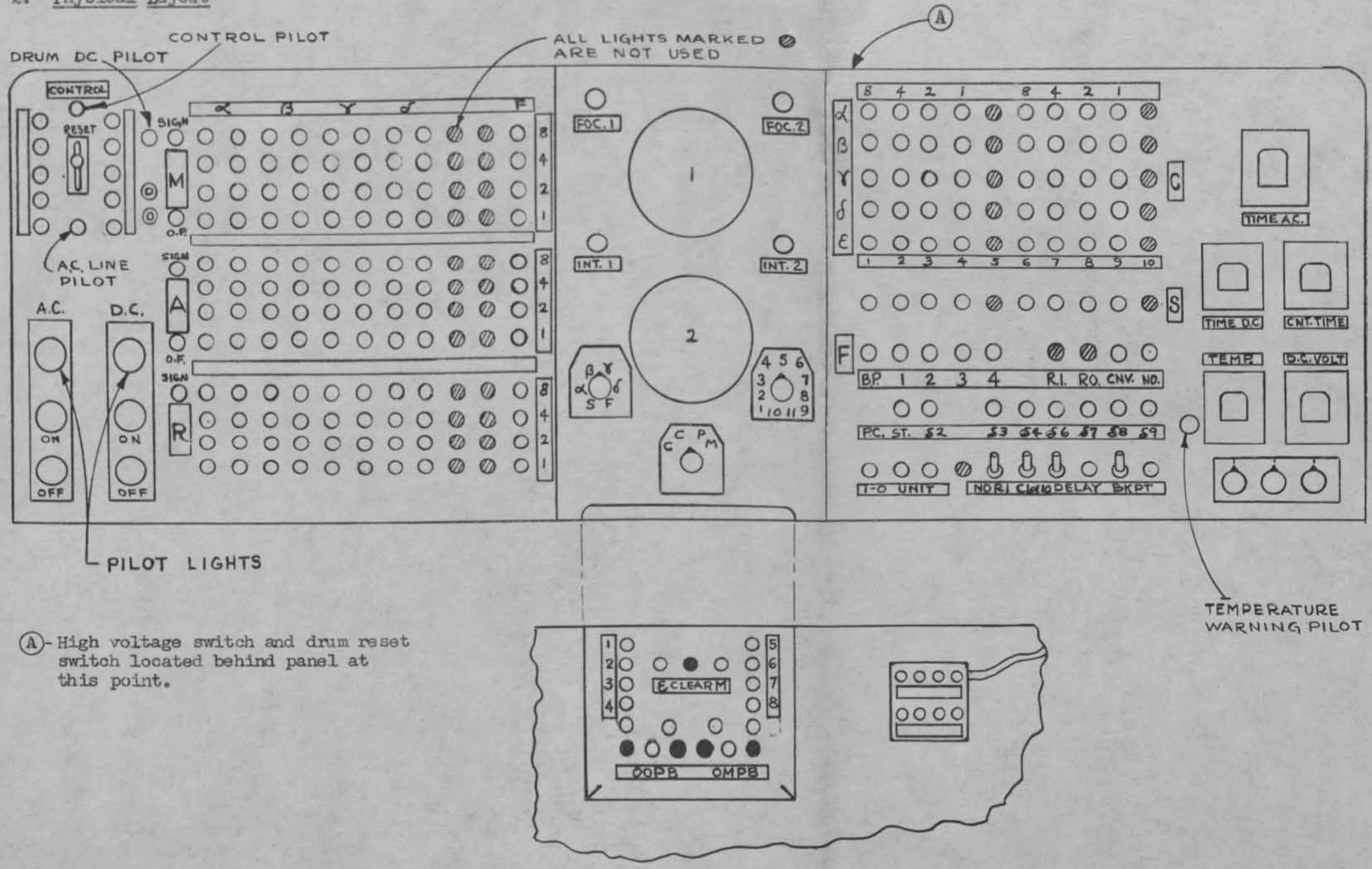
2. Physical Layout

Note that due to the AC coupling of the circuits in the arithmetic units, the carry pulses are not present on PER or MULT operation. The computer will not do correct addition on PER, nor correct multiplication on MULT, for this reason.

Four additional neons on the platform indicate the operation of four flipflop circuits in the "Gating A" chassis, and "OOPB Control" chassis. These are for the use of the engineers.

A keyboard stands on the Console next to the OOPB button platform. It is connected to the Console by a flexible cable. Keyboard layout and operation are discussed in the section on Input Devices.

2. Physical Layout



(A)- High voltage switch and drum reset switch located behind panel at this point.

2. Physical Layout

2. Console and Main SWAC - Description

c. Main SWAC

The center section of the SWAC cabinet contains forty cathode ray tubes (CRT's). Each tube with its attendant circuitry is mounted on a demountable, plug-in chassis, interchangeable with any of the others. The CRT's are labelled according to the position they hold in the memory.

Each CRT holds 256 digits in a raster of 16 x 16 spots. These spots represent all the digits of a given position in the SWAC "word". The forty CRT's are allocated as follows:

36: binary digits ($\alpha_1 \cdots \alpha_4, \alpha_6 \cdots \alpha_9, \beta_1 \cdots\cdots, F_1 \cdots F_4$)

1: sign digit

1: overflow digit (α_5)

1: breakpoint digit (β_5)

1: input reference digit (α_{10})

Thus any cell in the memory can be specified completely only by referring to the correct spot in the raster of every CRT.

Centrally located in the memory unit is a monitor scope with a pickup lead which may be clipped onto the output signal lead of any of the CRT's. Adjustment of a CRT is done by an engineer using this monitor to observe the signal from the CRT.

On the right of the memory units are housed the deflection amplifiers, regeneration circuits, and C register chassis. (The C register chassis in the bottom row, fifth from the right, holds the origin switch.)

2. Physical Layout

On the left of the memory units are the control chassis, the period ring (flipflops determining the periods of a command), and the typewriter ring (flipflops determining the positions of typed digits). In this section are the delay switches (on the Command-Delay chassis) and one of the normal-converted switches.

Also, on Control G, are found a series of hubs marked with the operations: output, extract, product, mutliplication, compare, subtract, add. A plug-in wire leads from any one of these to a volume control at the bottom of the Input-Output D, from which point a signal is sent to the loudspeaker on the wall behind the cabinet. This is the audio circuit which is used as a check on proper running of routines, and, less seriously, as a source of music obtained by properly coded routines.

Input-Output D chassis also holds the Letters-Numbers switch. Input-Output C chassis (bottom row, extreme left) has on it one of the two switches which must be operated to change SWAC from normal to converted output or back again. (See the Output section of this manual.)

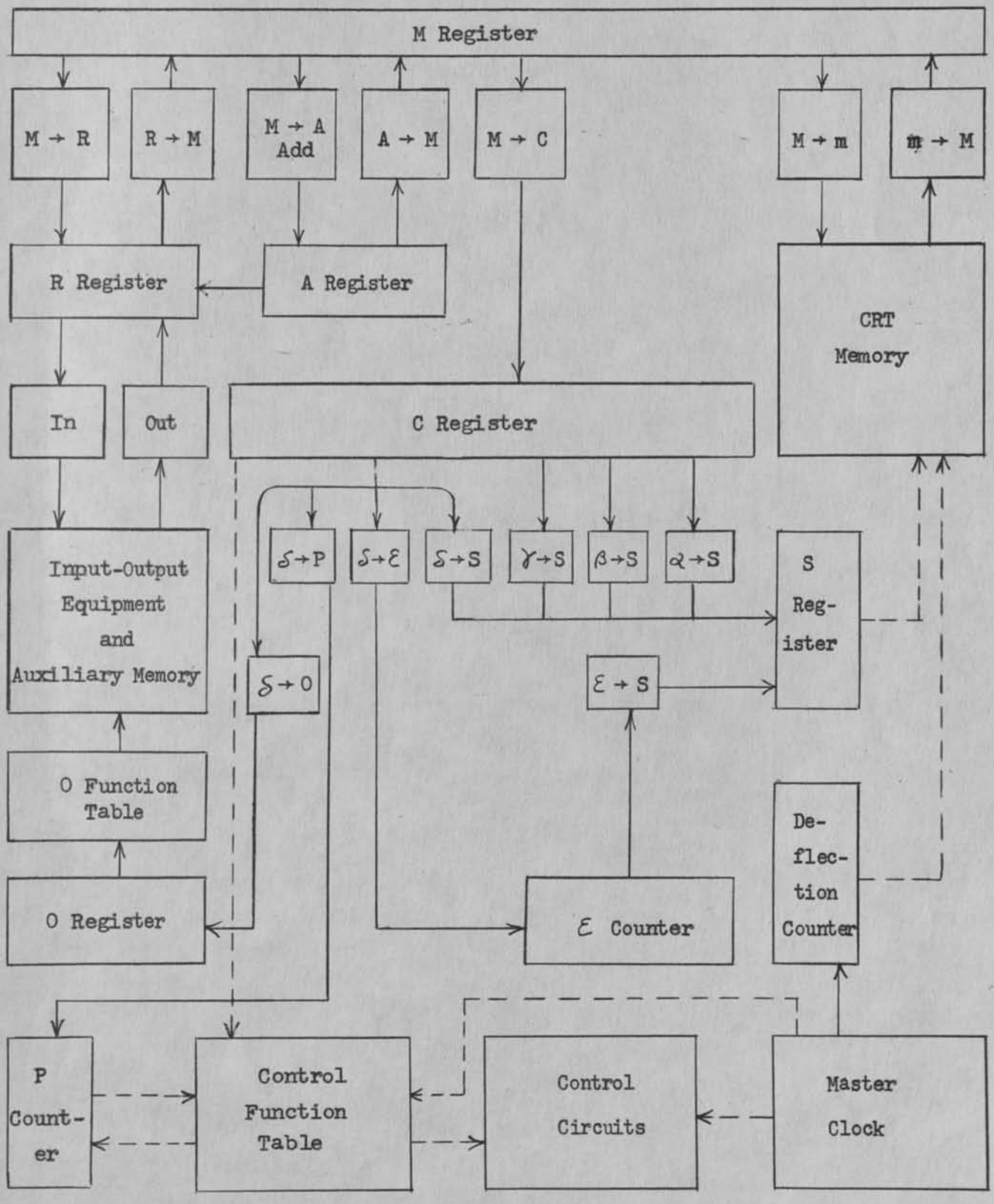
The arithmetic section, at the rear of the cabinet, contains one A chassis, and one M-R chassis for each digit of the SWAC "word". All of these chassis are of the plug type, and may be replaced by spares when found to be defective.

Near the floor level, in the center rank of these chassis, is a thermostat connected to the temperature warning light on the Console. It is in the path of the incoming refrigerated air, and gives warning if the air temperature goes too high. This is usually due to a failure of one of the compressors of the air conditioning unit.

2. Physical Layout

2. Console and Main SWAC - description.

d. Detailed Block Diagram

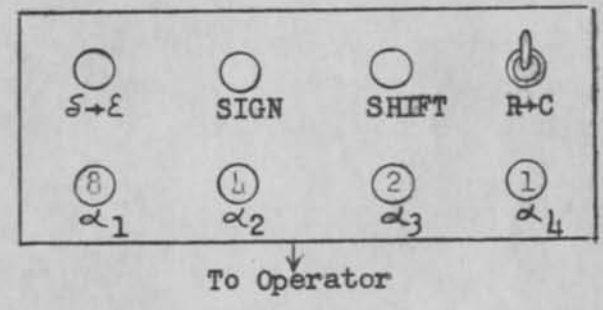


3. Input Devices

1. Console Keyboard

a. On February 3, 1954, the input Flexowriter and the output punched tape unit on SWAC were discontinued. Temporarily the code 10 in delta which signified punch tape output will be inactive.

b. A keyboard is installed to replace the input typewriter. It consists of a set of seven push button switches like the 00PB, and one spring loaded toggle switch like the clear alpha 10 switch. The keyboard arrangement is as follows:



The standard input command calls for input from this board instead of the typewriter

c. Operation with the standard input command in the origin. (01 00 00 10 j). With SWAC on command, and in period 3, each operation of one of the alpha switches or the sign switch sets one of the corresponding digits in the R register. Thus any base sixteen (hexadecimal) number will be represented by some combination of the four alpha switches. Each time the shift switch is depressed the number in the R register will shift four binary digits to the right. This is exactly the operation of the previous "0" key on the typewriter. The alpha switches can be operated simultaneously in any combination. Note that the sign digit is

3. Input Devices

not shifted, and cannot be cleared out after being set, except by operating the OOPB. The word in R register is transferred to the memory in the usual fashion by pushing OOPB.

d. The delta to epsilon switch operates in the same way as the ";" key on the typewriter. It sets the epsilon counter by transferring the contents of delta of the C register to the epsilon counter.

e. Operation of the R→C toggle. (The SWAC must be on COMMAND.) After a word has been put into the R register, it may be transferred to the C register directly, without any reference to the high speed memory. To do this, place the SWAC on period 7 of any command, and put a word into R register via the keyboard. HOLD the R→C toggle in the ON position, and operate OOPB. The word in R will replace whatever was in the C register. If the word transferred from R to C was an input command, SWAC will halt on period 3 after OOPB is operated. If the word was any other command, it will be obeyed, and SWAC will halt in period 7 again. Epsilon need not be cleared before inserting an R→C transfer. If the command inserted is a "special," then epsilon will be set by the command after it is obeyed, unless it is followed by another R→C transfer. If the command is not a "special," the value of epsilon will not change after the R→C command is obeyed. Thus epsilon will stand at 00 if it was cleared before the R→C.

3. Input Devices

f. Comments on the use of R→C.

It will no longer be necessary to restrict the contents of cells 00 and 01 to their present values. Input to any cell of the high speed memory, or transfer to any value of epsilon may be made at any time by R→C transfer of the appropriate command. This is true regardless of what the C register may contain before the R→C.

If R = 0's, then R→C will clear the C register. SWAC will halt on period 3 calling for input to the epsilon displayed in the C register. The contents of any cell in the high speed memory may be cleared or modified in this manner. If a routine is in a loop, it will be possible to escape, since the R→C will supersede any setting of epsilon made by any command in the high speed memory.

Any number of R→C transfers may be made in succession. Since the operation in SWAC is actually R→M→C, the device should prove useful in detecting SWAC failures in R, M, and C registers.

3. Input Devices

2. Collator

Am IBM Type 077 collator is used for card input of information to the SWAC. Only the feed mechanism is used; the brushes are wired through the plugboard to special cables connecting to the SWAC. The read-in action is a parallel one; all the holes in a single row of the card are read simultaneously, (as "ones") and the impulses generated set the R register flip-flops. At the same time, a "word pulse" (a hole in column 5 of the standard input card), in the same row as the information, signals a "Start Function Table" pulse. Then the word in R is transferred to the high speed memory. SWAC then proceeds to ask for another input, or begins computing if the card is completely read. Any number of words (rows) from 1 to 10 may be read in per card. Each word read in must be accompanied by the word pulse. If any word is not, it will remain in the R register and be superimposed on the following word.

The collator cams are set in such a way that the rows 9 through 1, and X are read. The cards are fed 9's edge first into the machine, and the words will be read in order from the bottom of the card. Any information punched in the 0 row or the Y row will be ignored, whether or not there are word pulses present. Identification in binary form is often stored in these rows for visual checks.

There are two feeds on a collator, and the wiring of the boards may be arranged to use either one. Appropriate wiring diagrams are available in the operator's manual in the SWAC Console. The reader is urged to read the IBM Manual of Instruction for the 077 collator for details of the feed

3. Input Devices

mechanism. In general, one input board is used, employing the standard format of the SWACARD. It is possible to vary this format of course, but it is usually better to utilize the standard card form for both input and output.

The operating switches of the collator are five in number. The on-off switch should be turned on several seconds before it is desired to operate the collator; a time delay circuit is incorporated to delay starting. Above the on-off switch are four buttons which control feeding. Counting from the right they are: Start, Stop, Run-out, and Error Reset. The start button will initiate card feeding in either primary or secondary feed. (The secondary feed is the upper one). The stop button will halt the motor drive without turning off the main power supply of the collator. The Run-out button will empty the feeds of the machine, when all cards must be removed. We are not concerned with Error Reset.

The Start switch will not initiate card feed unless the SWAC is calling for card input, but it will in any case start the drive motor. This will continue to idle if there are cards in the feed hopper, and if the SWAC at any time calls for card input the drive shaft will be engaged, and card feeding will start. Thus if SWAC calls for card input intermittently, between runs of computing, the cards will feed when called for, if the drive motor is idling. Also, if the SWAC begins computing immediately after an initial read-in, the motor will continue to idle. It is perfectly safe to start and stop the motor while the SWAC is computing. The motor drive will not idle if the feed hopper is empty, or if the reading stations are empty.

3. Input Devices

The usual practice when operating the collator is to place blank cards after the deck being read in (two in the secondary, three in the primary feed). Thus all the information cards may be run out into the stacker of the machine, leaving the blanks in the feed rolls. It is never necessary to empty the feed completely since one can always place blanks in the hopper to retrieve data cards which are still in the feed rolls. If the Run-out switch is used, then the SWAC will receive a word of Z's (37 binary ones) which will go into the origin as the last card is read out of the collator. It will be necessary to clear epsilon and the memory after this.

In addition to the information on a row of the card, and the word pulse, there are available a delta to epsilon pulse, and a breakpoint pulse. The breakpoint digit is described in the section on Operating Procedures. A hole may be punched on any row except O, and Y, of a SWACARD (in column 79 of the standard format) and if the breakpoint switch is ON, a digit will be stored in the appropriate address of the beta 5 CRT. These digits will not store if the breakpoint switch is off.

The delta to epsilon pulse punch is punched in column 4 of the standard SWACARD. It causes a clear epsilon, then a transfer of delta to epsilon, when it is read. The word on the same line with the delta to epsilon punch is put into the delta of the command which stood in the C register at the time of reading the word. (This will usually be zero on initial input.) It is important that column 4 of all cards read by the collator be kept blank except for delta to epsilon punches. Any column 4 punch except a O or Y punch will be treated by the SWAC as a delta to epsilon pulse.

4. Output Devices

1. Typewriter

The output commands with 00, 20, 40 in delta operate the typewriter of the SWAC on which the daily log is kept. This typewriter stands on a small console adjacent to the SWAC Console. It is an IBM electric typewriter equipped with solenoids under the keys. Impulses from the SWAC operate these solenoids which in turn operate the type bars and the various shift and spacing controls. The format of each word typed out is determined by the patch panel inside the East door of the SWAC cabinet. The number of words typed per line can be adjusted by arrangement of the tab stops at the rear of the typewriter. Ordinarily the word typed has 9 digits and sign, and six words are typed per line.

The patch panel (See fig.) contains two double rows of hubs which are coupled together to give the format desired. The left-hand column of hubs is connected to the typewriter ring flipflops; the right-hand column provides impulses for the various digits to be typed, spaces, and decimal points (periods). The figure shows the wiring for a nine digit output, with sign preceding, a decimal place after the second digit, and a space after the sixth digit. The topmost hub on the left is deliberately left blank. In the absence of a delay circuit it is necessary to provide sufficient time for carriage return when the typewriter reaches the end of a line. If this is not done the first character of the next line will print "on the fly" as the carriage is returning.

The word being typed out is stored in the R register and as each digit

4. Output Devices

prints, the word is shifted to the right four binary places. The digit appearing in F is the next digit to be typed. Thus the word is typed out in the reverse order from that in which it stood in the register. This condition exists on so-called NORMAL output. The NORM-CONV switches will be on NORM, and the NORM pilot light on the Console will be on.

On converted output a multiplication takes place each time a digit is typed. The coder must provide a multiplier equal to one-half the base to which the word is being converted; this multiplier is not to exceed one base 16 digit. It is stored in the M register in the F position. Multiplication of the word in R by the word in M results in a digit in the F position of the A register, and a remainder in the R register. The digit in A is typed out, and the remainder in R is multiplied by the digit in M. This process is repeated until the patch panel wiring signals a stop. In this output the word is typed out in direct order. By wiring the patch panel for eleven digit output full decimal accuracy can be obtained.

The hub labelled TAB on the patch panel provides an impulse that spaces the typewriter carriage to the next tab stop. The STOP hub halts the typing, resets the typewriter ring, and provides a Start Function Table pulse which signals SWAC to proceed with computation.

The above description is applicable to the 00 and 20 forms of output. The coding of 20 causes a limited alphabet or digits to be typed out. The choice is made by means of the Numbers-Letters switch. This is usually set on LTRS position and digits are obtained by using the 00 output code. Note

4. Output Devices

that this switch, as well as the patch panel and the NORM-CONV switches, are set at the beginning of a problem routine, and not changed during its operation.

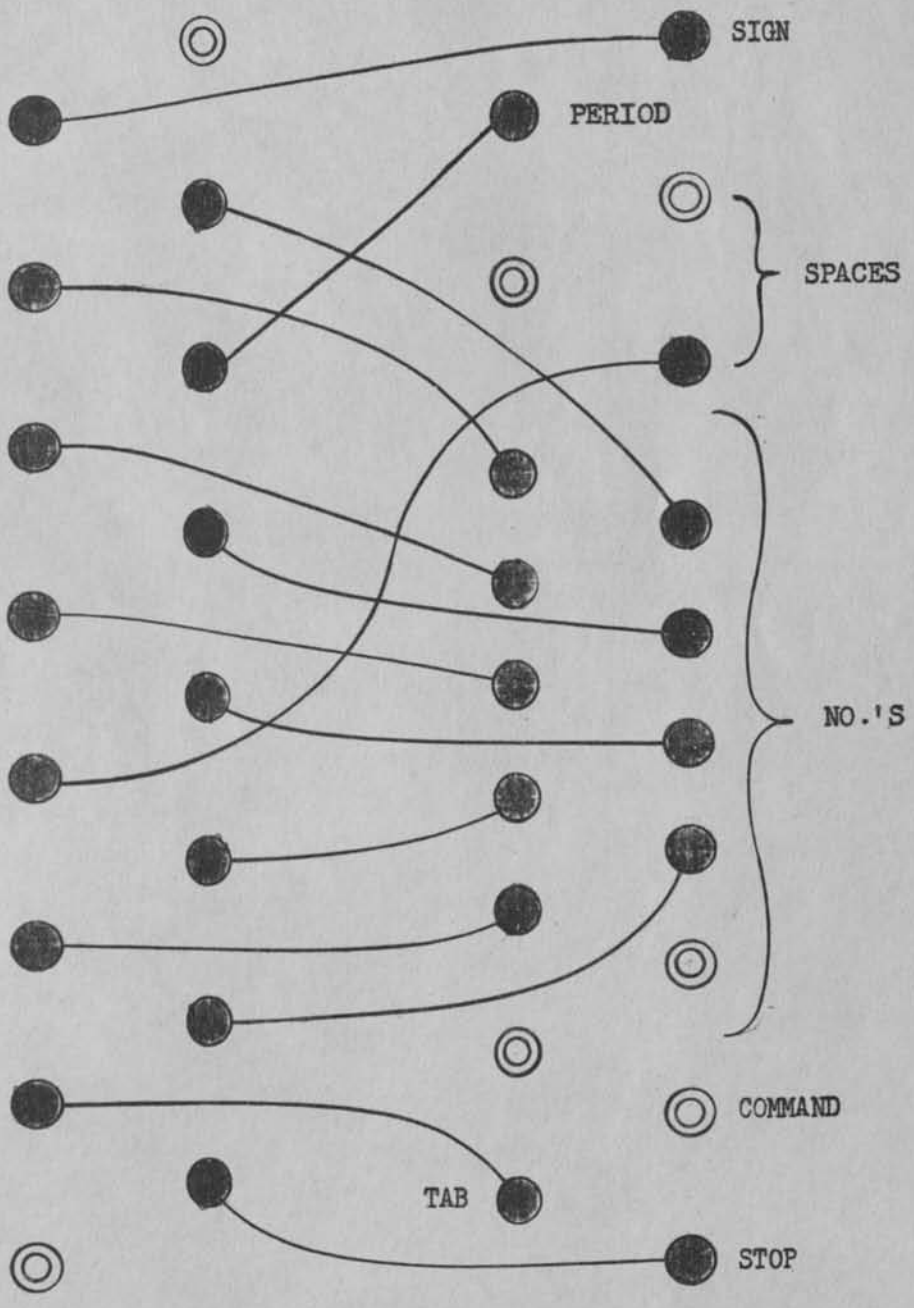
The coding for typewriter output which has 40 in delta of the command allows a full alphabet to be typed out. Only one character is typed for each output command. The typewriter ring is not activated in this form of output, therefore all spaces, carriage returns, and tabs must be coded as separate output commands. Output should be NORM, otherwise only spaces will be produced on the typewriter.

Mechanically the typewriter is rather delicate, so care should be exercised that it is treated gently. When feeding paper into the carriage see that the pressure rolls on the platen are released, and remain that way. The pins on the platen will guide the paper. The pressure roll control handle is on the left-hand end of the carriage and should be pulled toward the operator to the released position. The paper will jam if it is fed through the platen with pressure rolls on and the pins engaged in the holes at the edges of the paper.

See section on Operating Procedures for instructions on keeping the SWAC log.

4. Output Devices

PATCH PANEL



4. Output Devices

2. Reproducer Punch

The card output from the SWAC is punched in a Type 513 reproducer which is connected as a Summary Punch to the SWAC. A Punch Unit, consisting of a set of 75 thyratrons, receives signals from the R and M registers during the punch cycle, and translates these into signals to the punch magnets of the reproducer. A receptacle in the side of the Punch Unit receives the "snake head" at the end of the reproducer cable. The Punch Unit is connected to SWAC via wall sockets located behind the unit.

The control switches on the Punch Unit are two push buttons, labelled "Start" and "Stop". Two pilot lights, for AC and DC power indication, should turn on when the start switch is operated. On the 513 reproducer the ON-OFF switch is at the extreme right side of the machine. When the 513 is on, the green pilot light under the chip basket should be on. If it is not, and the motor of the reproducer is running, the generator circuits are at fault.

Under the "read" side of the machine is a series of switches which control the various feeds of the 513. (For a more complete description of the 513 and its controls the reader is urged to consult the Operating Manual of the 513 Reproducer, which is available in the IBM room.) The switches are, reading from left to right: a triple switch, labelled REPRODUCE; a GANG PUNCH switch; X-MASTER, X-DETAIL switch; and a single toggle switch (labelled S.P.) which is used to control feeding of cards and is not standard on a 513. For normal Summary Punch operation all the toggle switches should be UP. In this position the punch is ready to receive information from the SWAC. The S.P. toggle serves to disconnect

4. Output Devices

the emitter of the punch from the SWAC in the DOWN position, and allows the operator to manipulate cards in the feed of the machine without affecting the routine in the SWAC.

The plugboard determines the format of the card which will be punched and also contains some control wiring which will determine the number of rows per card. This circuitry is displayed on a plugboard diagram which is kept in the Operators' Manual in the SWAC Console. Each position of the R and M registers has a hub assigned on the plugboard, and wires from these hubs to the punch magnets determine the punching arrangement. There is also a series of jackplugs each of which connects the emitter of the punch with a specific row of the card. As few as one, or as many as twelve, rows may be punched.

Contrary to usual practice, the cards in the 513 are fed 9 edge FIRST. This is done to aid coding. It must therefore be remembered when a plugboard is wired, that all column numbers given on the board are actually the complements with respect to 81 of the desired columns, and that all row labels on the board are complements of the desired rows. Thus column 1 on the board will be column 80 on the card and column 20 on the board will be column 61 on the card, etc. In like manner, the 9 row, so labelled on the board, will be the Y row on the card. Standard plugboards are available for 36 digit output (R register only) and 72 digit output (both M and R). In general, it will not be necessary for the operator to do any wiring except the changing of the emitter jackplugs.

4. Output Devices

There follows a description of the operation during a punch cycle. We assume that ten words are to be punched in the card, so that the results can be fed back into the collator.

An output command appears in the C register. This causes the punch to start a card feed. At the same time the contents of the R and M registers set the thyratrons in the punch unit. When the card reaches the 9 row, the emitter of the 513 sends current to the punch unit where it is distributed by the thyratrons to the cables returning to the plugboard of the 513. The magnets then receive impulses according to the board wiring, and for each "1" in the R (or M) register a hole will be punched. At this time the SWAC receives a Start Function Table pulse from the punch and will resume computing while the card travels from the 9 row position to the 8 row. At 8 row time the SWAC must have another output command waiting in the C register. The whole cycle of operations will be repeated. When the card reaches the "0" row, there will be no punch cycle, since we have assumed that the jackplug for the 0 row has been left out. The X row will be punched, but not the Y row. After this cycle the card feed will halt, unless the SWAC has another output command ready. Notice that it is impossible to stop the card after the feed has begun, until the card is completely through the punching station. Thus the number of jackplugs wired in the board must be a factor of the number of outputs coded in the routine.

When the punching operation is completed for a routine, the last card punched and the one preceding it are still in the feed of the punch. To

4. Output Devices

remove these the operator throws the right-hand (S.P.) toggle switch DOWN and then uses the card feed button to run the cards out of the machine. If the blank cards in the hopper are left there it will be necessary to push the Stop button after two cards have fed, to halt the feed. When the S.P. toggle is UP, only one card feeds each time the feed button is pushed. If the toggle is left up and the 513 feed emptied, the SWAC will receive one Start Function Table pulse for each jackplug on the board. If a routine is standing in the memory it may be spoiled by this.

To load the machine it is not necessary to have cards in the feed when SWAC calls for output. The hopper may be filled and the start button pushed after the SWAC has halted for an output command. If the feed is empty the punch will not start itself. Under these circumstances the first card through will not be punched. Thus a gang punch card may be placed in front of the blank deck and retrieved for later use.

All reproducing and gang punching functions of the 513 are operative while the machine is summary punching. Prenumbered decks may be used on the read side, to reproduce serial numbers on summary punched cards, and job number identification may be gang punched from the punch brushes. The 513 operates as an ordinary reproducer when the snake head is coupled back into the right side receptacle of the 513. The S.P. toggle switch should be DOWN in this case.

6. Magnetic Drum

1. General Description

The magnetic drum is a cylinder coated with a magnetic material. It is rotated continuously at 3600 rpm. It is used to store binary information, which it receives from and transmits to the high speed memory of the SWAC. Each binary digit is represented by the magnetic state of a small spot of the surface. One polarity represents a zero, the other represents a one. Around the circumference of the drum are bands of these spots, each band containing the 36 binary digits and the sign for each of 32 numbers. Such a band of spots (called a channel) is written and read by a single read-write head under which it rotates. There are ~~128~~ 256 channels, giving a capacity of ~~4096~~ 8192 words to the drum. Normally information is stored on the drum in blocks of 32 words (referred to as a full channel), but it is also possible to store 16 word (half channel) and 8 word (quarter channel) blocks.

Transmission of information between the drum and the high speed storage of the SWAC is initiated by the Input-Output circuitry. To go from high speed storage to drum (writing) is an output command (F = 2), and to go from drum to high speed storage (reading) is an input command (F = 1). Note that all operations are performed with respect to the high speed memory of SWAC. "112" in the delta address of the command is necessary to designate a drum operation. (Written 70 in base 16.)

If the SWAC is considered to have 16 lines of high speed storage, each with 16 positions, then reading and writing operations of the drum are

6. Magnetic Drum

always done with respect to an adjacent pair of these lines. The lines may be numbered from 000 to 015. Then the line pair referred to by a drum command is always an even numbered line and an odd numbered line, in that order (i.e., lines 000 and 001, 002 and 003, 004 and 005, etc.)

The alpha address of the read or write command designates the line pair referred to in the high speed memory by specifying the address of the first word of the first (even numbered) line of the pair. These designations are 000, 032, 064, ... , 224. For example, the 32 positions from 032 to 063 will be referred to by a drum read or write command if the alpha of that command is 032*. No modification of the command in alpha is necessary to get half or quarter channel operations.

The beta address of the command is used to tell whether a full, half, or quarter channel block is desired; and further, which half or quarter channel block. If 000 is put in beta, the full channel block (32 words) will be used. If 004 or 006 is put in beta, a half channel block (16 words) will be used, with 004 indicating the first or even numbered line and 006 indicating the odd numbered line. If 012, 013, 014, or 015 is put in beta, a quarter channel block (8 words) will be used, with 012 indicating the first 8 words of the even numbered line and 013 the last 8, and with 014 indicating the first 8 words of the odd numbered line and 015 the last 8.

The gamma address of the command designates the channel of the drum which will be used. The channels are numbered beginning with 016 and

* Strictly speaking, if n ($\leq$ 255) is the number in the alpha address, then the first word of the line pair designated will be in the address denoted by the largest integer contained in n/32 times 032.

6. Magnetic Drum

ending with 143. If any of the numbers 000 through 015 is used with a read command (1 in F), the cells designated by the α-β of the command will be cleared. If these numbers are used with a write command (2 in F), nothing will happen. If the number in γ is greater than 143, then noise will appear in the high speed memory in the positions designated by α-β of a read command. Nothing will happen with a write command if the γ address is greater than 143.

There is a one-to-one correspondence between the drum positions 000 through 031 and high speed memory positions 2k + 000 through 2k + 031 (2k = 000(032)224). Special care must be taken to note, when half channel blocks are used, that 000 to 015 corresponds to 2k + 000 through 2k + 015 only, and 016 to 031 corresponds to 2k + 016 through 2k + 031. The same restriction applies to quarter channel blocks. (c.f. Figure 1) This one-to-one correspondence is accomplished by means of "clock" pulses on the drum. These enable writing or reading to begin as soon as the command is given, and control is turned over to the drum. Thus there is no waiting period for the drum to reach position 000 to start operation. For example, suppose the command in the C register is a drum read command which is to read a full channel into the high speed memory, and at the time that the drum takes control the reading head is at channel position 025. Then reading begins with address 2k + 025, proceeds through 2k + 031, and then continues from 2k + 000 through 2k + 024. When the drum reaches this point it will have completed a full revolution and control will be restored to the SWAC.

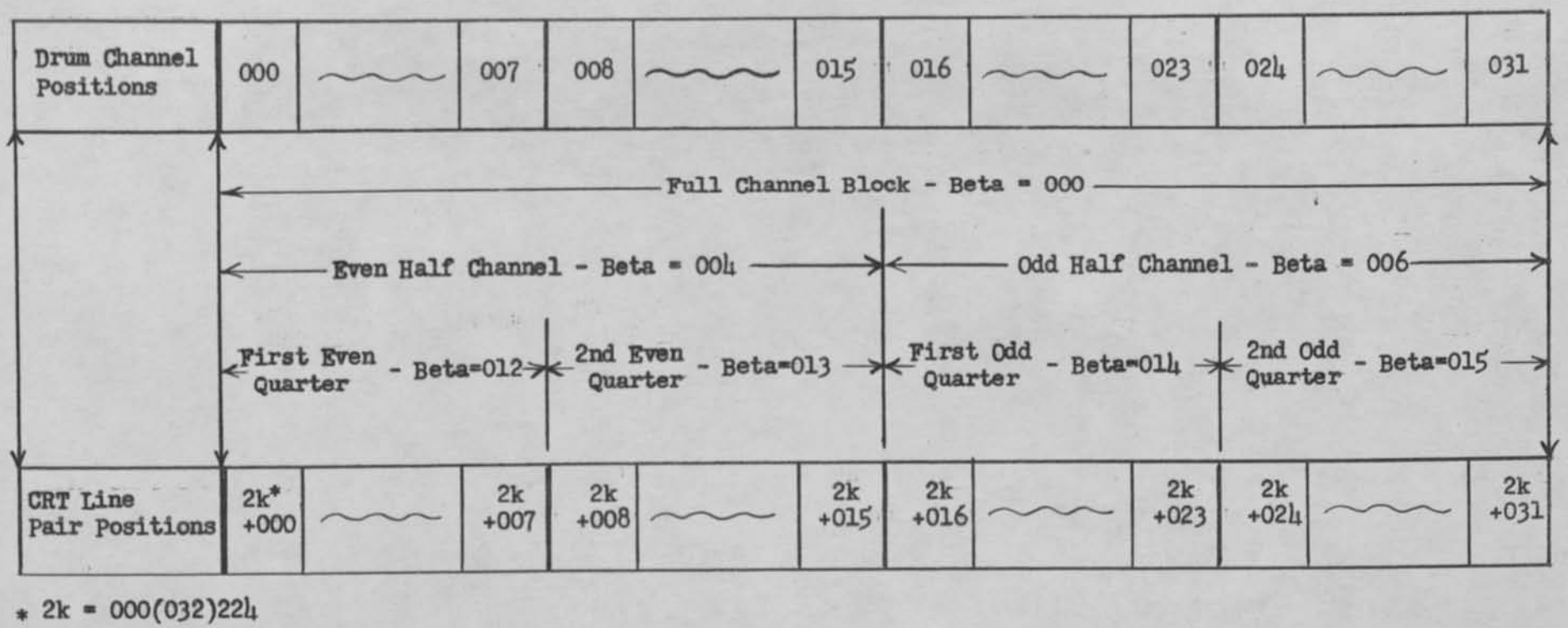
It takes .017 seconds to complete a read or write command regardless of the number of words being transmitted. Half channel and quarter channel

6. Magnetic Drum

operations are controlled by inhibiting the reading or writing pulses during parts of the drum revolution, but a full revolution must be made under all circumstances.

The read or write commands may be performed while running the computer on COMMAND or PERIOD, as well as on CONTINUOUS. The drum command (112 in delta) initiates a STOP SWAC and a START DRUM pulse at the beginning of period 2 on read or period 4 on write. At the completion of the transfer the SWAC takes over control again and advances to the next period.

Figure 2 gives a summary of the decimal drum command. Figure 3 gives the distribution of the binary digits in a drum command. Where a digit is labelled "ignored", it is immaterial whether a "1" or "0" is present in that position.



* 2k = 000(032)224

Figure 1. Relationship of Drum Channel to CRT Line Pair.

6. Magnetic Drum

α	β	γ	δ	F
Specifies CRT Line Pair Range of Numbers: 000(032)224	Specifies No. of Words Transferred When β is 000 32 words 004) 16 words 006) 012) 013) 8 words 014) 015)	Specifies Drum Channel Range of Numbers: $j<016$ (Write=nothing Read = zeros $j>143$ (Write=nothing (Read = noise	For Drum Commands δ is 112	Operation Symbol Write= 02 Read = 01

Figure 2. Decimal Drum Command

α									β								γ								δ								F			
1	2	3	4	6	7	8	9		1	2	3	4	6	7	8	9	1	2	3	4	6	7	8	9	1	2	3	4	6	7	8	9	1	2	3	4
CRT Line Pair				These Digits Ignored					Fraction of Channel								Drum Channel								Ignored		1's Specify Drum			These Digits Ignored							Operation Symbol			

Figure 3. Distribution of Binary Digits

1. Introduction

Determining the proper numerical method to be used in solving a particular problem on a computing machine is called programming. This includes drawing up a sequence of arithmetic operations for the particular calculation and choosing methods, determining numbers of significant figures, deciding on checking procedures, etc., and possibly laying out a diagrammatic chart indicating the flow of the computational process. Thus, to program a problem is to transform it from its original mathematical representation into the form mentioned above.

The process of transforming this algebraic or flow sheet formulation of the problem into a symbolism which a particular computing machine is able to handle is called coding. The set of instructions for doing a problem, or a particular phase of a problem, is called a routine.

In most problems it is necessary to make use of subroutines. A subroutine is "a routine within a routine," which is used to perform operations which are common to many problems. For example, conversion of numerical data from decimal to binary representation is usually necessary, since the computer operates in the binary system and data are ordinarily given in decimal representation. It is possible, of course, to include in the routine for each problem all the commands necessary to perform this conversion. In general, however, it is more efficient to have available a subroutine to take care of this decimal-binary conversion. Then, whenever conversion is necessary, this subroutine can be read into the high speed memory and be used to convert as many numbers as required.

The set of generalized instructions, in which subroutines are referred to but not included, for doing a particular problem is called the main routine for that problem.

1. Introduction

Three types of instruction words are used in routines and subroutines.

First, there are the thirteen types of commands. Commands represent the elementary operations which the machine can perform, such as addition, subtraction, etc. This type of instruction word is explicitly obeyed.

Control words are a second type of instruction word. These words are not actually used in the operation, but are used to control the course of the computation. Examples of control words are numbers to be used as extractors, numbers which may be added to given commands for modification, parameters to be used as bounds or to signal the end of an iteration procedure, and so forth. In the coding examples which follow, a control word is usually referred to as a dummy.

Code words are the third type of instruction word. These are instructions that are not obeyed explicitly, but define the operation to be performed and its parameters. Code words consist of a compact representation in one word of several parameters which are needed to specify the operation of a subroutine, including an entry device to transfer the control to that subroutine. Each subroutine extracts its various parameters from this one code word, making it possible, for example, to specify a vector multiplication with only one code word.

An interpretation routine is used when the main routine contains code words. The interpretation routine keeps track of the place in the main routine, causes segments of the main routine to be brought into the high speed memory from the magnetic drum, inspects each successive instruction in the main routine to see if it is a command (in which case it is obeyed), or extracts an entry (the address of the command of a subroutine which should be obeyed first) if it is a code word, and instructs the computer to go to the appropriate subroutine.

1. Introduction

1. General Discussion of the Commands

Commands are frequently called command words or orders. The SWAC has thirteen types of commands in its vocabulary. Each of these is 37 binary digits long and may be represented in the form α, β, γ, δ, F.

F stands for function and has the following thirteen values (and sixteen symbols):

Addition (4)

Special Addition (5)

Subtraction (6)

Special Subtraction (7)

Multiplication with Round-Off (u)

Special Multiplication with Round-Off (v)

Exact Multiplication (w) or (x)

Compare (8)

Special Compare (9)

Extract (y) or (z)

Initial Input (o)

Input (1)

Output (2) or (3)

It is F that determines which one of the thirteen of the above listed functions is involved in the present computation.

α, β, γ, δ generally represent addresses, that is, the position of the words, in the memory. More specifically, α and β are usually the

1. Introduction

addresses of operands (numbers which serve as input to operation). γ is generally the address of the result.

δ has various uses. Normally, the machine obeys commands coming from successive addresses in the memory as determined by a so-called fifth address, ε, described later in this section. However, in the case of the Special Addition, Special Subtraction, and Special Multiplication with Round-Off Commands, the next command is determined by δ. In this way the machine may transfer the source of control with each command. When operating in this manner, any arbitrary pattern of commands in the memory may be obeyed. Naturally, these special commands may be interspersed in any desired manner among the other commands. δ is also the address of the next command in case of overflow, or exceeding of capacity of the machine, in the Addition and Subtraction Commands. In the Exact Multiplication Command it is the address of the least significant part of the product. In case of the Compare Command it is the address of the next command if the difference is non-negative; in Special Compare it is the address of the next command if difference of the absolute values is non-negative. The direction and number of places to shift are determined by δ in the Extract Command. In the Input Commands δ selects the particular input device to be used and the type of input; in Output it selects the particular output device and the type of output. Thus, in the SWAC the uses of δ (the fourth address) are many and diversified. (In the EDVAC the fourth address is always used to specify the next command whether it is in order or not.)

1. Introduction

A fifth address is denoted by $\mathcal{E}$. This address is stored by a binary counter known as the $\mathcal{E}$ counter in the Control Unit. Each time a command is obeyed the number in $\mathcal{E}$ is increased by unity; in this manner, the machine normally obeys a sequence of commands coming from successive address positions in the memory throughout the operation. $\mathcal{E}$ is the address of the next command in the Addition and Subtraction Commands, except in the case of overflow as noted above. $\mathcal{E}$ is always the address of the next command in Multiplication with Round-Off, Multiplication Exact, Extract, Initial Input, Input, and Output. In Compare it is the address of the next command if the difference is negative; in Special Compare it is the address of the next command if the difference of the absolute values is negative.

As originally designed the SWAC was to understand eight different types of commands. Consequently, F then had eight values. Since then, the number of types of commands has been extended by the following device: The eight commands are still interpreted by an eight-output function table (that is, a crystal matrix in the machine), but the fourth digit in the function designation controls certain operations relative to the reading-in of the next command. This in effect increases the number of types of commands from eight to thirteen.

Thus, we have in the case of Addition, Subtraction, Multiplication with Round-Off (not Multiplication Exact), Compare, and Input two types of operation. These are designated by Addition (4) and Special Addition (5), Subtraction (6) and Special Subtraction (7), Multiplication (u) and Special

1. Introduction

Multiplication (v), Compare (8) and Special Compare (9), Initial Input (o) and Input (1). The fourth digit is not operative in the case of output, multiplication exact, or extract. In other words, either 2 or 3 in F means output, either w or x means multiplication exact, and either y or z means extract.

In Special Addition, Special Subtraction, and Special Multiplication with Round-Off, the next command is determined unconditionally by the address δ of the present command as already noted. In other words, these special commands are similar to the four-address command systems used in such computers as the EDVAC. They represent a means of transferring of control from one point in the memory to some other point in the memory. Therefore, in the SWAC, there is no Transfer of Control Command (sometimes called Unconditional Transfer Command) since this operation is automatically taken care of by these three special commands. (Transfer of Control can also be accomplished by either the Compare Command, or by the Addition and Subtraction Commands if it is certain that capacity will be exceeded.)

These Special Commands in Addition, Subtraction, and Multiplication with Round-Off were fairly easy to attain. Their addition was made possible simply by using the extra digit in the F representation to control the read-in of the address from which the next command is to be obtained. It should be remembered, of course, that by using these special commands the facility of detecting overflow is lost. Therefore, if there is a question of overflow, then the coder must use the normal Addition and Subtraction Commands in order

1. Introduction

to be able to detect it. However, there are many cases in the course of performing a computation where one is simply counting, or carrying out operations in which the size of the numbers involved are known, so that there is no question of overflow.

The exceed capacity circuits used in the Addition, Subtraction and Compare Commands operate as follows: There is one extra digit position in the registers of the arithmetic unit. This extra digit position has no input or output to the rest of the machine except to the CRT in position $\alpha 5$. By having the carry-over go into this extra digit position and remain stored there during the execution of the particular command, two numbers can be added together whose sum would normally exceed the capacity of any storage space in the machine. In the Addition and Subtraction Commands a carry-over into this extra digit position means that capacity has been exceeded.* In the Compare Command it signifies that capacity has been exceeded in the actual operation, but in this case it is immaterial as the sign of the result will be correct whether capacity has been exceeded or not.

In the case of the two Compare Commands, Special Compare differs from Compare in that the signs of the numbers are disregarded in the former.

Initial Input is the command used when the problem is initially given to the machine; Input is used to put additional information into the machine after it is already in operation.

Two principles have been followed in deciding upon the previously mentioned thirteen commands. First, that there should be as few commands as

* Note that on command mode of operation, the F4 neon in the C register will be lighted in the event of overflow. This will make the operation look like Special Add or Special Subtract.

1. Introduction

possible so as to simplify the electronic circuitry, thus carrying out the design principle of the machine. (Actually, the electronic function table which interprets these commands has only eight positions.) The second principle is that the commands should be as general as possible, thus eliminating the need for a greater number of commands. For example, the Extract Command (logical product) allows the use of any factor whatsoever, and the elections in case of overflow are completely general. Similarly, there is no Shift Command. This operation can be done by using the Extract Command. No separate Halt Command is necessary, as stopping the machine can be accomplished at will simply by specifying within δ of the Input Command the keyboard as the input device.

In the SWAC, a general purpose computer, no specific commands exist for many operations which the machine must be able to perform. Naturally, the machine must be able to do operations such as division, floating addition and subtraction, floating multiplication and division, and drum-to-memory sequence transfers. To have attempted to build in circuitry to enable the machine to perform such tasks with a single command would have made the machine so cumbersome from the circuitry angle that it would have taken very long to construct. Very likely, if constructed, it would have been practically impossible to maintain in operation. Therefore, arithmetic operations such as the above, too complicated or elaborate to be performed by one of the thirteen types of commands contained in the SWAC's vocabulary, must be initiated by a sequence of commands called a routine or subroutine. For

1. Introduction

example, division can be done by a routine consisting of repeated subtraction. The whole process of division, which may amount to as much as one-hundred operations, can be completely determined by approximately fifteen commands. The various commands used in the routine may be repeated over and over again operating each time on different numbers. Similarly, all other complicated operations can be accomplished by routines of standard instructions which can be stored in the memory. The subject of routines and subroutines is discussed more fully in recent papers by Harry D. Huskey[1], and in other sections of this manual.

The SWAC in executing each of the thirteen commands contained in its vocabulary has the process divided into eight steps or periods. This is because the machine goes through eight different electronic stages in executing the command.

In general, during Steps 1 and 3 the operands that are involved in the particular command are read into the arithmetic unit. In most cases the actual operation is carried out in Steps 2 and 4. Usually in Step 5 the result of the operation is placed back into the memory. The next command which the machine is to obey is generally obtained by Steps 6, 7, and 8.

---

1) H. D. Huskey, Semi-Automatic Instruction on the Zephyr, Proceedings of a Second Symposium on Large Scale Digital Calculating Machinery, Annals of the Computation Laboratory of Harvard University, Vol. XXVI, Harvard University Press (1951).

H. D. Huskey, Characteristics of the Institute for Numerical Analysis Computer, Mathematical Tables and Other Aids to Computation, Vol. IV, No. 30, (April, 1950).

1. Introduction

In the case of Extract, Exact Multiplication, Multiplication with Round-Off, and Special Multiplication with Round-Off, the machine stops in Step 4 and goes through a routine. This routine consists of the repeated additions, and/or the repeated shifts, that must be carried out in these commands. In the case of the Initial Input and the Input Commands, the machine halts in Step 2, and waits for the auxiliary input or output device to carry out its portion of the operation. In the Output Command this procedure takes place in Step 4.

1. Introduction

2. Meanings of the Addresses for the Commands used by the SWAC.

COMMAND	α	β	γ	δ	F
ADD	ADDRESS OF AUGEND	ADDRESS OF ADDEND	ADDRESS OF SUM	ADDRESS OF NEXT COMMAND IF OVERFLOW	4
SPECIAL ADD	ADDRESS OF AUGEND	ADDRESS OF ADDEND	ADDRESS OF SUM	ADDRESS OF NEXT COMMAND	5
SUBTRACT	ADDRESS OF MINUEND	ADDRESS OF SUBTRAHEND	ADDRESS OF DIFFERENCE	ADDRESS OF NEXT COMMAND IF OVERFLOW	6
SPECIAL SUBTRACT	ADDRESS OF MINUEND	ADDRESS OF SUBTRAHEND	ADDRESS OF DIFFERENCE	ADDRESS OF NEXT COMMAND	7
MULTIPLY	ADDRESS OF MULTIPLIER	ADDRESS OF MULTIPLICAND	ADDRESS OF PRODUCT ROUNDED-OFF	NOT USED	u
SPECIAL MULTIPLY	ADDRESS OF MULTIPLIER	ADDRESS OF MULTIPLICAND	ADDRESS OF PRODUCT ROUNDED-OFF	ADDRESS OF NEXT COMMAND	v
PRODUCT	ADDRESS OF MULTIPLIER	ADDRESS OF MULTIPLICAND	ADDRESS OF MOST SIG-NIFICANT PART OF PRODUCT	ADDRESS OF LEAST SIGNIFICANT PART OF PRODUCT	w or x
COMPARE	ADDRESS OF MINUEND	ADDRESS OF SUBTRAHEND	ADDRESS OF DIFFERENCE*	ADDRESS OF NEXT COMMAND IF DIFFERENCE IS NON-NEGATIVE	8
SPECIAL COMPARE	ADDRESS OF MINUEND	ADDRESS OF SUBTRAHEND	ADDRESS OF DIFFERENCE OF ABSOLUTE VALUES	ADDRESS OF NEXT COMMAND IF DIFFERENCE OF ABSOLUTE VALUES IS NON-NEGATIVE	9
EXTRACT	ADDRESS OF EXTRACTOR (DETERMINES DIGITS TO BE EXTRACTED)	ADDRESS OF EXTRACTEE	ADDRESS OF EXTRACTED AND SHIFTED RESULT	IF SECOND BIN. DIG. OF δ IS: 0-SHIFT LEFT 1-SHIFT RIGHT OTHER DIGITS TELL NUMBER OF PLACES TO SHIFT	y or z
INITIAL INPUT	(INCOMING IN-FORMATION GOES TO ADDRESS ϵ)	NOT USED	NOT USED	SELECTS INPUT DEVICE AND TYPE OF INPUT	0
INPUT	ADDRESS OF INCOMING INFORMATION	PORTION OF DRUM CHANNEL	CHANNEL ADDRESS ON DRUM	SELECTS INPUT DEVICE AND TYPE OF INPUT	1
OUTPUT	ADDRESS OF OUTGOING INFORMATION	PORTION OF DRUM CHANNEL**	CHANNEL ADDRESS ON DRUM	SELECTS OUTPUT DEVICE AND TYPE OF OUTPUT	2 or 3

*Overflow gives same word in γ as in 4 and 6 Commands **Address of ½ base, for converted output

1. Introduction

3. Summary of Command Operations

Command	Result	Next Command
Add (4)	$(\gamma) = (\alpha) + (\beta)$.*	(ε) if no overflow, (δ) if overflow.
Special Add (5)	"	(δ).
Subtract (6)	$(\gamma) = (\alpha) - (\beta)$.*	(ε) if no overflow, (δ) if overflow.
Special Subtract (7)	"	(δ).
Multiply (u)	(γ) = most significant half of $(\alpha) \cdot (\beta)$ rounded-off.	(ε).
Special Multiply (v)	"	(δ).
Product (w or x)	(γ) and $(\delta) = (\alpha) \cdot (\beta)$, most significant half in (γ). [Sign digit in both.]	(ε).
Compare (8)	$(\gamma) = (\alpha) - (\beta)$.*	(δ) if $(\alpha) \geq (\beta)$, (ε) if otherwise.
Special Compare (9)	$(\gamma) = \lvert(\alpha)\rvert - \lvert(\beta)\rvert$.	(δ) if $\lvert(\alpha)\rvert \geq \lvert(\beta)\rvert$, (ε) if otherwise.
Extract (y or z)	(γ) = result of following: Digits of (β) made 0 if corresponding digits of (α) = 1, and resulting number is shifted. (Right shift if second binary digit of δ = 1; left shift if second binary digit of δ = 0. Amount of shift = no. represented by last 6 binary digits of δ.)	(ε).
Initial Input (0) (except Drum)	(Input device) → (ε).**	(ε).
Input (1)(except Drum)	(Input device) → (α).**	(ε).
Output (2 or 3) "	(α) → (Output device).**	(ε).
Input (0 or 1) (δ = 112)	32 words → CRT memory.***	(ε).
Output (2 or 3)(δ = 112)	32 words → Drum channel.***	(ε).

* Most significant digit lost in case of overflow.
** Particular Input-output device in 11, 12, and 13 determined by $\delta 2$, $\delta 3$, and $\delta 4$.
*** See section on Magnetic Drum.

1. Introduction

4. Timing Chart for SWAC Operations

OPERATION	F CODE	TIME (in milliseconds)
Add	4,5	.064
Subtract	6,7	.064
Compare	8,9	.064
Multiply	u,v,w,x	.384
Extract	y,z	.128-.384*
Input Tape (per word)	0,1	Not Used
Input Cards (per word)	0,1	25.
Output Type (per word)	2,3	~2000.
Output Tape (per word)	2,3	Not Used
Output Cards (per word)	2,3	60.
Input from Drum	0,1	17.**
Output to Drum	2,3	17.**

* The time is a function of the amount of shift.

If r = number of binary digits of right shift, and
ℓ = number of binary digits of left shift,

then

$$r = 36 - \ell$$

and the time $t = .128 + .064 \left[\frac{r+2}{8}\right]$ milliseconds.

If no shift is desired a zero right shift should be specified.

** This time is for 32 words or any fraction thereof.

NOTE: The times given for card input and card output assume continuous feeding of the cards, and 10 words per card. If the feed is halted for computation, add 600 ms for each halt of the output punch, and 250 ms for each halt of the collator. Note further, that computing time begins after the last word pulse is read on a card. Therefore, it may actually be faster, in some cases, to put fewer words on a card, in order to permit computing while the upper end of the card is passing through the collator. The feed will then be continuous, but the time per word will be increased somewhat over that given in the table.

2. Procedures

1. General Procedures

a. Introduction

These introductory remarks are intended to summarize coding procedures now in use. The reader is referred to other sections of the manual for more detailed accounts of particular processes. It is assumed that the coder is familiar with the meanings and symbols of SWAC language, as described elsewhere in the manual. In particular the command chart and allocation of addresses should be studied.

Assume that a problem has been stated in arithmetic terms, that is, so that each operation can be translated to SWAC language without further mathematical analysis. The problem so stated may allow of direct translation. More usually, the coder will have to draw up a flow chart, or at least rearrange the equations of the problem in some sequence to facilitate coding. Sometimes a map of the memory, showing the contents of various cells, is drawn up. These decisions are usually left to the individual coder.

If a flow chart is decided upon, it may consist of only a few blocks, each containing an instruction which actually represents many commands in SWAC, or it may be so detailed that each command in the routine occupies one block. Some coders write their routines section by section, building up the flow chart concurrently.

Another decision to be made is the type of coding to be used: relative, or absolute, or card programmed. In the latter case SWACPEC or Floating SWACPEC will be chosen, and the instructions in the appropriate section of the manual followed. Whether relative or absolute coding is chosen depends to some extent on the complexity of the problem

2. Procedures

to be solved. If the routine will be complicated, with many subroutines, and much drum storage, relative coding will enable the coder to keep track of his "bookkeeping", prevent many errors, and facilitate detection of errors.

A broad outline should be made of the code, assigning storage for main routine, constants, temporaries, intermediate results, subroutines, and final answers. Rearrangement of material inside the memory is easy now, so the final form of the routine should be decided upon at this stage.

When making the assignments, store the constants either directly before or after the commands, and assign temporaries (initial value equal zero) in a group apart from the rest. This group of words may be omitted from the input cards since zero values need not be read into the machine if the $\mathcal{S} \rightarrow \mathcal{E}$ transfer device is used. Grouping the temporaries also simplifies memory checks, since temporaries are not included.

Modified commands, and variable tallies should be "prestored". That is, a constant equal to the initial value of the command or tally is stored, and this is used to replace the last value of the modified word before the loop containing it is re-entered. By properly choosing the location of the commands which prestore, the coder can assure himself of entry points for recalculation during code checking, or in the event of trouble halts. The operator must know the meaning of the various error halts and typeouts, what he may inspect to determine the trouble, and where he can enter the routine to re-compute or to continue. Even if a cycle of commands is used only once in a routine, the coder should prepare a list of prestorable commands for his own use in code checking. The initial values can be typed in from the keyboard when necessary.

2. Procedures

Since SWAC has no built-in checks, include coded checks in the routine. Memory sums, summation of drum channel storage, halts for unexpected overflows, and mathematical checks on the arithmetic are all coded. One may wish to have a computation repeated after a failure occurs; in this case an alternating tally will permit a recalculation, and then a halt in the event of a repetition of the error. The remarks column on the coding sheet should indicate the cause for each failure.

Many add and subtract commands of a routine can never be expected to have overflow. There are others, however, in which overflow may possibly occur, indicating error. The size of a factor may exceed that expected by the coder, or there may be some function computed by a series which causes overflow when the terms are summed. The first suggestion that comes to mind is to let the delta of those commands be 000; then epsilon goes to 000 upon any overflow. Inspection of the α5 CRT will show the location of the address which received the overflowed answer.

Alternatively, if there are enough storage cells available after assigning addresses to the main routine, store several halt commands, 001 000 000 016 01, or breakpoints, one for each possible overflow. Code the addresses of the halts in the delta portions of the add and subtract commands which may overflow. In the remarks column of the halt commands indicate which overflow causes each halt; the operator of the SWAC can then type on the log sheet which function had an error, for later investigation by the coder.

The coder should prepare step-by-step instructions for running the routine. They should describe the amount of memory filled by card initial read-in, the new data to be inserted in the collator, the frequency of memory checks, etc. This is particularly important if the routine is of a "standby"

2. Procedures

nature, and will be run by several operators.

It has been standard practice in SWAC coding to assign to cells 000 and 001 the following:

Cell No.	α	β	γ	δ	F	Remarks
000	001	000	000	016	01	Keyboard input to 001.
001	(Temporary; initially zero)					

This permits the operator to type any desired command into 001, and cause SWAC to obey it immediately. The R → C transfer on the keyboard makes this assignment less necessary, since any command may be inserted directly into the C register and obeyed.

We suggest the use of 000 for zero storage. This has two conveniences: it permits the coder to write "000" when he means value = zero, and it still provides an error halt in the event of overflow in a command containing 000 in δ. (Take care that your routine does not require the collator to be idling, for the origin cell will not be a halt in this case.) The operator will still have the option of clearing epsilon manually anywhere in the routine and returning to the origin. A command may be typed into 000, and obeyed there.

All data in SWAC are treated by the machine as numbers between -1 and +1. Therefore the coder must think of data in the same way, and assign scale factors accordingly. It is possible to store powers of 10, both positive and negative, and scale data in the machine by products. This is clumsy and a fruitful source of errors. It is much better to scale by powers of 2. Keep a record of the scale factors assigned to the result of each operation in the remarks column of the coding sheets.

2. Procedures

Example: (all coding and data in base 16)

Compute: $a_i b_i$ where $a_{i+1} = a_i b_i + 7$, and $b_{i+1} = b_i + 2$, until b_i reaches a given limit.

Given: a_o = 1.69y4; maximum known to be a_i = 7.yz49

b_o = 4.y496; maximum known to be b_i = y.4697

Store: $2^{-3}a_o$ = .2x3w80000 and $2^{-3}a_i$ = .zxy920000

$2^{-4}b_o$ = .4y4960000 and $2^{-4}b_i$ = .y46970000

Cell No.	α	β	γ	δ	F	Remarks
43	56	59	57	44	u	$2^{-7}a_ib_i \rightarrow (57)$
44	81	57	58	04	y	$2^{-3}a_ib_i \rightarrow (58)$
45	58	80	56	46	4	$2^{-3}(a_ib_i+7) \rightarrow (56)$
46	59	82	59	47	4	$2^{-4}(b_i+2) \rightarrow (59)$
47	83	59	86	43	8	Is (83) < (59)?
48	(Command to be obeyed after (59) has reached its limit)					
⋮						
56						$a_i \cdot 2^{-3}$
57						$a_ib_i \cdot 2^{-7}$
58						$a_ib_i \cdot 2^{-3}$
59						$b_i \cdot 2^{-4}$
⋮						
80	y0	00	00	00	0	$7 \cdot 2^{-3}$
81	00	00	00	00	0	Zero storage.
82	20	00	00	00	0	$2 \cdot 2^{-4}$
83	y4	69	60	00	0	Limit of $b_i \cdot 2^{-4}$
86	(Temporary storage)					Hash.

2. Procedures

Note that cell 44 contains an extract command in which the shift serves to multiply $a_i b_i \cdot 2^{-7}$ by the 4th power of 2. Remember that a left shift of 1 multiplies a number by 2, a right shift of 1 divides by 2, in the binary system.

A left shift using the extract command has a serious disadvantage: power of detecting overflow is lost. An alternative device, which takes up more memory space, but does allow detection of overflow in the event that the size of the numbers involved is not well known, is to double the number in question a predetermined number of times. Indicate an error, or enter a rescaling routine, if overflow occurs during the doubling process.

If the data in a problem cover too wide a range of values, so that significant digits will be lost in fixed point operations, use the floating point subroutines. These treat a value as a number q: $\frac{1}{2} \leq q < 1$, accompanied by an appropriate power of 2. (See the section on floating point subroutines.)

A diagram of the "SWAC Coding Sheet" is given on the following page.

2. Procedures

SWAC CODING SHEET Page

Coded By: Date: Problem:

Hex Cell	Cell No.	Card No.	S	α	β	γ	δ	F	CD M	+	-	BK PT	REMARKS

2. Procedures

1. General Procedures

b. Memory Check

Memory Check is a device to notify operator of loss or pickup of numbers in the memory. Summing the memory after routine read-in (or after computation of 1st value) checks immediately for failure to read in a word, or to store it, properly. Intermittent summing in the same manner during computation checks for loss or pickup during operation. One more memory check after the last computation will verify accuracy of the routine during final computations.

For example, assign addresses to storage in the following order:

1) Commands.

2) Constants and dummy commands.

3) Temporary storage and (if possible) modified commands.

Assign the last memory location in 2) to the given memory sum. The first time the routine is checked out on the SWAC this sum is entered in the memory as zero. Assign one cell in 3) to the memory sum as computed at regular intervals by the machine.

Whenever a memory check is to be made, the routine enters a short sequence of commands which add 1) and 2) as stored in the memory. The summing routine stops just before the cell containing the given memory sum. It then compares the sum of 1) and 2) as given with the sum computed by the machine and stored in a cell of 3). If they do not agree, indication of memory failure should be given by the machine.

The coder may choose to have the routine type the difference between the two sums and then halt. Possibly an analysis of the difference will

2. Procedures

indicate the failure. If the difference is large, the only recourse may be to read the routine in again. Hence, it is important to code so that computation may be continued from any point.

The first time the routine is checked out, the memory check causes a failure since the given sum has been entered as zero. The difference between the two sums is in this case the correct sum. On another day the routine is again read into the machine, this time using as the memory sum the type-out obtained on the first trial. If there is no failure, that sum computed by the machine is entered permanently in the routine as the given memory sum, and so used in subsequent computing runs of the routine.

If the given memory sum is stored negatively it can be included in the summing. Then the machine sum should be compared with zero to determine accuracy of the memory. In the event it is not zero, the machine sum can be typed out for analysis of the error.

When a problem requires the use of subroutines the assignment of storage would be the same as described above, with the addition of 4): subroutine commands and constants. In this case, the summing cycle would add the contents of 1) and 2), skip over the contents of 3), and add 4). Then the routine would compare that sum with the given memory sum, as above.

The routine must take into account the few modifiable commands and temporary storage cells which must necessarily be stored in 1), 2), and 4). Since they are not always of the same value, they could cause the memory sums to disagree. If permissible, those cells should be restored to zero before being included in the machine sum, or they may be omitted.

2. Procedures

A memory sum should have a constant value; barring that, it should vary in a predictable fashion.

NOTE: In adding the memory, be sure to use a special add, thereby ignoring overflow of the memory sum.

2. Procedures

1. General Procedures

c. Modified Commands

One of the more powerful features of the coding for a machine like SWAC is the possibility of changing commands which are already in the memory. This is called "modifying" and it treats the commands like numbers, operating on them to produce new commands, which can be obeyed in turn. It is often possible to save considerable memory space by ingenious use of modification. The following paragraphs offer some suggestions on modifying commands.

The most straightforward way to code a cycle of, say, six operations that are to be performed eight times is to code in 48 successive cells the eight groups of six commands. However, it may take fewer cells to use the same group of six commands, stored in six cells, eight times over. This requires modifying one or more of the references of the six commands depending on the computation, each time the cycle is performed. It also necessitates a tally count up to eight, and a command to leave the cycle when completed.

In some instances, the amount of shift called for in an extract command varies with the numbers being used. The routine stores a dummy extract command with delta equal to zero or an appropriate constant. This dummy is combined with a computed delta value and used each time the extract is to be obeyed.

Instead of storing thirteen output commands to type out the contents of memory addresses from 49 to 5v, modify one output command and use it repeatedly. Begin with (73) = 49 00 00 00 2. Modify it in alpha, compare

2. Procedures

it with a limiting dummy, and use it until (5v) has been typed out.

Cell No.	α	β	γ	δ	F	Remarks
73	[49]*	00	00	00	2	(k) out
74	92	73	73	75	4	k + 1 → k
75	93	73	94	73	8	Through?
76	(Command to be obeyed after (5v) has been typed out)					
92	01	00	00	00	0	2^{-8}
93	5v	00	00	00	2	Limit for (73)
94	(Temporary storage; "waste basket")					

This same method can also be used for a summing routine, such as a memory check.

Another example of a modified command is one used in table look-up. Suppose the logarithms of the integers from 1 to 1x (base 16) were stored in addresses 60 to 7W. A dummy (5z 89 v9 36 4) would be stored in (88). To obtain (log y), y would be added in alpha to the dummy; the resulting command would then be obeyed as in the following example:

To find the logarithm of the integer part of x, number in temporary storage address v5.

Cell No.	α	β	γ	δ	F	Remarks
33	87	v5	v6	04	y	Extract and shift integer part of x to y = alpha of (v6).
34	v6	88	35	35	4	Combine (v6) with dummy and place it in (35).
35	[5z]*	89	v9	36	4	Command to transfer log y to (v9). This command may be put into the memory initially as zero; it is always prestored by (34).
36	(Command to be obeyed after obtaining the log y in (v9))					

* [ ] = portion to be modified.

2. Procedures

Cell No.	α	β	γ	δ	F	Remarks
87	00	0z	zz	zz	z	Extractor.
88	5z	89	v9	36	4	Dummy for (35).
89	00	00	00	00	0	

v5 ($2^{-12}x$; known to be $1 \leq x \leq 1x$)

v6 (Temporary storage; contains $y = 2^{-8}$ times the integer part of x)

v9 (Temporary storage; contains log y)

Before re-entering a cycle containing a modified command it is necessary to restore that command to its original state. If it could be guaranteed that the command had been modified a given number (n) times, then the amount of modification times n could be subtracted from the command to restore it. However, it is very possible that the modification was done less than n times (for instance, in checking out the routine); therefore, the policy of pre-storing a command before entering a computing cycle will insure its accuracy. [Note that the example of table look-up on the preceding page does not require such pre-storing, but that the type-out example first given requires pre-storing of (73) = 49 00 00 00 2.]

Whenever it is possible in the coding, store a modified command in that portion of the memory designated for temporary storage. This eliminates skipping that address when adding commands and constants for memory check, or subtracting that command from the sum after completion of summing. In the table look-up example, the command obeyed in (35) could be stored and obeyed in (v8) among the temporary storage cells as in the following:

2. Procedures

Cell No.	α	β	γ	δ	F	Remarks
33	87	v5	v6	04	y	Int. part of x → alpha of (v6).
34	v6	88	v8	v8	5	Combine (v6) with dummy and → (v8).
36	(Command to be obeyed after obtaining the log y in (v8))					
87	00	0z	zz	zz	z	Extractor.
88	5z	89	v9	36	5	Dummy for (v8). [(89) is zero].
v5	(2^{-12}x; known to be $1 \leq x \leq 1x$)					
v6	(Temporary storage; contains $y = 2^{-8}$ times the integer part of x)					
v8	[5z]	89	v9	36	5	Log y → (v9); epsilon returns to 36.
v9	(Temporary storage; contains log y)					

In the type-out example given, since (73) cannot be a special command (specifying the next epsilon) such storing in the temporary portion of the memory is not possible.

2. Procedures

1. General Procedures

d. Tallies

These are three ways of tallying to govern the number of times a particular cycle is followed:

a) A constant may be added to a temporary storage cell until the accumulation causes an overflow. [Example 1]

b) A limiting constant may be compared with unity (the result of the comparison -- or subtraction -- being returned to the constant storage). In this compare tally the constant is reduced by unity; at the same time the direction of routine is governed by the sign of the difference. [Examples 2 and 3]

c) A limiting dummy command (A) may be compared with a modified command (B) until (B) has been modified up to the value of (A). [Example 4]

Tallies to cause one operation (C) to occur alternately with another operation (D) are termed alternating tallies. Either the compare commands [Examples 6 and 7] or the overflow signal of an add command [Example 5] may be used to accomplish this alternation.

Example 1:

Adding until overflow occurs

(As long as (38) does not signal an overflow, the routine will go on to (39). When there is overflow, then routine goes to (4z). If m is properly chosen ($m \geq \frac{1}{n+1}$), (47) is left clear and need not be restored before using this tally again. For example, if n = zz, let (97) = 01 00 00 00 0. If n = z, let m = 10, instead of 01.)

Cell No.	α	β	γ	δ	F
38	97	47	47	4z	4
39	(Command to be obeyed n times)				
47	(Initially zero; successively (in alpha) m, 2m, 3m, ..., 0 with overflow signal)				
4z	(Command to be obeyed once, only after (39) has been obeyed n times)				
97	(m)	00	00	00	0

2. Procedures

Example 2:

Using compare command

(The results of the compares in (23) will be positive, or 0, n times; hence, (26) will be obeyed n times before the result of the compare in (23) will become negative. (3u) would have to be reset before using the tally again; it is left with -1 in its storage.)

Cell No.	α	β	γ	δ	F
23	3u	79	3u	26	8
24	(Command to be obeyed once, only after (26) has been obeyed n times)				
26	(Command to be obeyed n times)				
3u	(Initially n; successively n-1, n-2, ..., 0, -1)				
79	("One" in position corresponding to the units position of n in (3u))				

Example 3:

Using special compare command

(The result of the first compare will be negative and (57) will be obeyed. The next time (56) is obeyed, the absolute value of (67) is used, and the result is again negative. Hence, (57) is obeyed until the compare of (56) results in (67) being 0, i.e., positive; then (60) is obeyed. Before being used again, the tally in (67) must be restored from 0 to n + 1.)

56	v9	67	67	60	9
57	(Command to be obeyed n times)				
60	(Command to be obeyed once, only after (57) has been obeyed n times)				
67	(Initially n+1; successively n, n-1, n-2, ..., all negative; finally, 0)				
v9	("One" in position corresponding to the units position of n+1 in (67))				

Example 4:

Comparing a modified command

(Before (u5) is obeyed it is modified by (u4) to read 49 in alpha. As long as the alpha of (u5) is $\leq$ 75, the result of the compare in (u6) is positive, or 0 and epsilon returns to (u4). After (76) has been typed out, (u6) has a negative result, and epsilon continues to (u7). Before entering this cycle again, the routine must restore (u5) with alpha = 48.)

u4	u5	wx	u5	u5	4
u5	[48]	00	00	00	2
u6	ux	u5	yz	u4	8
u7	(Command to be obeyed after the memory addresses from 49 thru 76 have been typed out)				
ux	75	00	00	00	2
wx	01	00	00	00	0
yz	(Temporary storage; "waste basket")				

2. Procedures

Example 5:

Alternating by add command

(Accumulating $\frac{1}{2}$ in alpha of (67) causes alternately no overflow and then overflow. Upon overflow, (67) is left zero; if (24) is obeyed an even number of times therefore, the cycle can be entered again without resetting (67) to zero.)

Cell No.	α	β	γ	δ	F
24	67	42	67	49	4
25	(Command to be obeyed the first and all odd times)				
42	80	00	00	00	0
49	(Command to be obeyed the second and all even times)				
67	(Initially zero; alternately 80 00 00 00 0 and zero (with overflow signal))				

Example 6:

Alternating by compare command

(The result of the first compare in (92) is -k in (wx). Since the absolute value of (wx) is used for the second obeying of (92), that result is 0 in (wx). Therefore, the results of the compare alternate from negative to zero (positive). If (92) is obeyed an even number of times, (wx) need not be restored to 0 before re-entering the cycle.)

92	wx	w4	wx	uv	9
93	(Command to be obeyed the first and all odd times)				
uv	(Command to be obeyed the second and all even times)				
w4	(Any constant, k, used otherwise in routine)				
wx	(Initially zero; alternately -k and zero)				

Example 7:

Alternating of compare command

(Result of the first time obeying (2u) is that (2u) is stored negatively; routine goes to (2v). Since the machine ignores sign in obeying commands, the next time (2u) is obeyed as if it were positive; that result is a +(2u), and epsilon goes to (49). Hence, the results alternate from negative to positive, and (2u) is alternately stored negatively and positively. If obeyed an even number of times, (2u) is left stored positively; it need not be restored before re-entering cycle.)

2u	60	2u	2u	49	8
	(Initially positive; alternately negative and positive)				
2v	(Command to be obeyed the first and all odd times)				
49	(Command to be obeyed the second and all even times)				
60	00	00	00	00	0

NOTE: Of the three alternator methods of tallying, examples 5 and 6 require five cells of storage each. Example 7, by having (2u) modify itself instead of another cell, requires only four cells.

2. Procedures

2. Relative (Symbolic) Coding

Relative (symbolic) coding is a method of preparing problems for solution on high speed computers in which symbols are used to represent storage positions. There are many forms of this type of coding in existence, and the one which is used depends on the coder. However, to mechanize the process of translating the relative code to the binary notation used in SWAC, the method herein described has been devised.

In the SWAC system, the storage is divided into regions each of which is assigned any decimal number from 00 to 95. Each element of a region is also assigned a number starting with 00. There may never be more than 100 elements in a region. Thus each element of the program is uniquely determined by a 4 decimal digit number, the first two digits being the region number, the last two the position in the region.

A simple example of this division could be:

Region No.	Use
03	Constants
04	Commands
05	Temporaries
etc.	

Then the number 0328 would refer to the 29th constant or 0421 would refer to the 22nd command.

Relative coding is especially useful for programs which have many subroutines. Each of these subroutines could be coded as a separate routine with a different region number. Then they could be checked out separately before the full routine is assembled.

The code is prepared for key punching in the normal four address system, each address containing four decimal digits. The ordinary SWAC coding sheets

2. Procedures

will be used. A command would look as follows:

Cell No.	α	β	γ	δ	F
0420	0016	0505	0504	0435	07

When the code is completed, positions in the memory are assigned to the regions. The address of the first element in each region is used to make up a dictionary. If there are 85 commands, 20 constants and 8 temporaries, the following assignments could be made:

Region No.	Storage	Use
03	002-021	Constants
04	022-106	Commands
05	107-114	Temporaries

Then the dictionary would be:

03 - 002

04 - 022

05 - 107

etc.

The coder must fix regions 00, 01, and 02 for absolute coding by always starting the dictionary in the following way:

00 - 000

01 - 100

02 - 200

If this is done, every relative address less than 0256 will remain the same when converted to absolute address form. This is particularly helpful if one wishes to code constants (or other numbers) into a form which will be read by the 00212 (relative conversion) routine. For example, suppose one

2. Procedures

wishes to put .01 in cell 0321, i.e., the 22nd position in region 03. Its absolute address form is as follows (obtained by 4 multiplications by 256 and one by 16):

Cell No.	α	β	γ	δ	F
0321	002	143	092	040	15

The easiest way to determine the relative addresses corresponding to 002, 143, etc. is to leave room for them in regions 00, 01, and 02 by starting the dictionary card as above. Then the relative form of this number is:

0321 0002 0143 0092 0040 15

This leaves regions 03 to 95 available to be used by the coder in the usual way.

The relative routine is punched in the IBM room one word per card and listed on the tabulator for proofreading. The card layout is:

Columns	Information
1-3	Card No.
5	Word Pulses (X, 1-9)
8, 10-11	α
12-13	β(1st two digits)
16-20	Problem No.
27-28	β(last two digits)
29-30, 32-33	γ
34, 47, 49-50	δ
51-52	F
53	Sign (X: - ; 0 or blank: +)
9, 14, 31, and 48	Punched with automatic zero punch

At the same time the dictionary cards are prepared. These cards are numbered starting from 000. Each card contains twelve 3-digit fields,

2. Procedures

which are the first addresses of twelve regions. The first field on each card is the first address of the region whose number equals the card number times twelve. For example, card 000 contains regions 00 through 11; card 001 contains regions 12 through 23, etc. The dictionary card layout is as follows:

Columns	Information
1-3	Card No.
5	Word Pulses (X, 1-9)
7-9	Position of region 12n (n = card no.)
10-12	" " " 12n + 1
13-14, 27	" " " 12n + 2
28-30	" " " 12n + 3
31-33	" " " 12n + 4
34, 47-48	" " " 12n + 5
49-51	" " " 12n + 6
52-54	" " " 12n + 7
67-69	" " " 12n + 8
70-72	" " " 12n + 9
73-75	" " " 12n + 10
76-78	" " " 12n + 11
16-20	Problem No.

When the cards have been prepared, listed, and proofread, they are sent to the SWAC for conversion. Routine 00212 is used to convert the relative coding. For each relative card read in, a card containing both the relative word and its decimal equivalent is punched out. When ten relative cards have been input, and the corresponding relative-decimal punched, a card containing the ten binary words is punched.

This routine is divided into two main parts. Part I converts the dictionary to binary and stores it. Four dictionary locations are stored

2. Procedures

per address in cells 224 to 224+K, where K = the largest integer in n/4 (n = the largest region number). The dictionary cards are placed immediately following the dictionary routine (first 6 cards). When the routine has read in it halts calling for keyboard input. The number of dictionary cards is typed into F of the R register. OOPB is pushed; the dictionary cards are read in, converted, stored, and the memory from 000 through 223 is cleared. The routine then halts in the origin calling for initial input of Part II.

Part II is the conversion part of the routine. It is read in immediately following Part I. SWAC halts in 002 asking for keyboard input of the initial serial number of the binary cards. This is typed into F of the R register, in hexadecimal notation. In 003 SWAC again halts calling for keyboard input of the initial serial number of the relative-decimal cards. This is also typed into F in hexadecimal notation. The routine then goes to the command calling for collator input of the relative deck.

The routine feeds a relative card and then punches a card which has both the relative code and its decimal equivalent punched on it. The relative portion is in the same fields as the input card. The decimal equivalent is punched in:

Columns	Information
67-69	α
70-72	β
73-75	γ
76-78	δ

After 10 relative cards have been read in and 10 relative-decimal cards have been punched out, a binary card with 10 binary commands on it is punched out. This card has also a 1 punched in column 80, and a binary serial number punched in the Y-row.

2. Procedures

The input to the collator is the normal input using the regular 36-digit input board. The output uses the normal 36-digit output board with all gang punch hubs wired, i.e., 12 words are punched out. Also, a wire must be plugged from "Comparing Magnets from Comparing Brushes", column 76, to the punch magnet column 1. This produces the 1 in column 80 on the binary card.

There must be a multiple of 10 relative cards, or blank word pulse cards must be added to make a multiple of 10.

IT MUST BE NOTED THAT THIS ROUTINE DOES NOT CONVERT NUMBERS, ONLY COMMANDS, OR NUMBERS CODED AS COMMANDS!!!

To separate the binary cards from the relative-decimal cards, sort the cards on column 80. The binary cards will then fall into the 1 pocket, and the relative-decimal cards into the discard pocket.

After the binary cards are separated from the relative-decimal cards, the latter may be listed on the IBM tabulator. A 405 board has been wired to do this.

A summary of the process now being used for SWAC relative coding is as follows:

I. Code is prepared on regular four-address coding sheets with 4 digits per address and two for F.

II. The relative cards are keypunched, one word per card, and listed on the tabulator for proofreading. The dictionary cards are prepared at the same time.

III. Routine 00212 is used to convert the relative code to decimal and to binary. The binary cards have a one punch in column 80.

IV. The result deck is sorted on column 80 to separate the binary cards from the relative-decimal cards. The latter are then listed on the tabulator.

V. The binary cards are used as input to the SWAC.

2. Procedures

3. Interpretative Coding

a. Subroutines

Not all mathematical operations necessary to complete a problem are available in the thirteen basic commands of SWAC. The most frequently used of these complicated operations (logarithm, exponential, trigonometric functions, division, etc.) have been coded as subroutines for inclusion in main routines. There is a short cycle of commands, called the Interpretation Routine, designed for inclusion in any main routine, which provides a standard and uniform method of entering and leaving subroutines.

Regardless of the number of commands (k) required for each subroutine, they have all been coded to be stored in memory addresses from (zz - k + 1) to zz. They are all less than 128 cells in length. If a main routine refers to two or more subroutines, the coder assigns storage to them depending only on the number of cells used by each. (Theoretical example: store the logarithm subroutine from 9u to x3, and the square root subroutine from x4 to y6. However, the coding sheets for these subroutines would be for cells w6 to zz and yx to zz, respectively.)

When the routines are read into SWAC, the subroutines are read in from tape or cards as originally coded; they are read into the memory addresses assigned as in the example above. There is another small routine, the Preparatory Routine, which inspects each word of the subroutines. It modifies the "cross-reference" addresses according to the

2. Procedures

parameters determined by the coder. (In the logarithm example, the parameter = 2w = zz - x3, the difference between the addresses for which the subroutine was coded and the addresses in which it is to be used. In the square root, that difference = 19 = zz - y6.)

After the main routine and subroutines have been read into the memory, the coding should first cause the Preparatory Routine to modify the subroutines. Some problems require so much storage that the space allotted to the Preparatory Routine must be used for other commands; since the Preparatory Routine is used only at the start, to modify subroutines, no harm is done by storing another subroutine in the addresses formerly occupied by the Preparatory Routine (z2 to zz).

In the example above, using the logarithm and square root subroutines, a third routine could be stored from y7 to zz (for example, the sine subroutine). This routine would also be read into SWAC as coded, but need not be modified as are the two others. After the two subroutines have been modified by the Preparatory Routine, the main routine should enter a cycle of commands which read the third subroutine into addresses y7 to zz from cards.

Example:

Cell No.	α	β	γ	δ	F
((38) modifies (37) and epsilon goes	37	[y7]	00	00	00	1
to 37. When the modification of (37) causes an overflow, epsilon goes to 3u.	38	37	78	37	3u	4
Overflow occurs only after (37) has caused read-in to (y7) through (zz).)	39	79	79	79	37	7

2. Procedures

Example: (cont.)

Cell No.	α	β	γ	δ	F	
3u	(Command to be obeyed after the last word of the sine subroutine has been read into (zz))					
78	01	00	00	00	0	2^{-8}
79	00	00	00	00	0	Zero storage.

The main routine has a code word stored for each subroutine that is to be entered. The code word is not a command nor a numerical constant, but contains the addresses and parameters needed by the subroutine for computing. For example, the square root code word might contain the following addresses.

Sign	α	β	γ	δ	F
-	Address of n	Address of $\sqrt{n}$	Number of whole integers of $\sqrt{n}$	Address of next command	Number of this code word

Every code word is stored negatively. The Interpretation Routine, distinguishes code words from commands by sign, and always places the current code word in memory address 16. The Interpretation Routine also extracts the various addresses given in the code word. It uses these addresses to store the pertinent values in certain temporary storage cells. Each subroutine is coded to find the necessary values in those same cells.

In the use of subroutines, the constants needed by the subroutines are stored in specific memory cells, and all subroutines refer to these same memory cells. This group of constants in the assigned positions is

2. Procedures

called the "pool" of constants.

Memory allocation is as follows:

The cells from 1w through 24 contain specific "pool" constants. Cells from 03 through 09 are set aside for temporary storage used by subroutines. Cells 0u through 1v are occupied by the Interpretation Routine mentioned above. The Preparatory Routine is stored in cells z2 through zz. The Preparatory Routine is stored cells z2 through zz. This leaves cells 25 through z1 for allotment by the coder to his main routine and subroutine storage. By carefully scanning the subroutine coding, the coder may possibly find that not all the "pool" constants and the temporaries in addresses 03 to 09 and 1w to 24 are used by his particular subroutines. In that case, he may assign any unused storage from 03 to 09 and 1w to 24 for constants or storage in the main routine.

The cover page of each subroutine has a description of the operation it performs, a list of the pool and temporary storage cells it uses, the number of cells its commands and constants require for storage, the number of cells to be modified by the Preparatory Routine, and its code word. This should be all the coder needs to plan his use of subroutines. He need not copy the subroutines to the pages of his coding sheets; an indication of where he plans to store them in the memory is sufficient. In fact, he should avoid copying subroutines as a precaution against copying errors.

There is a library of IBM cards on file, containing the subroutines for card input to the SWAC. By use of the IBM reproducer a coder can

2. Procedures

obtain a duplicate copy of any subroutine on cards. These he incorporates into the cards containing his main routine. Reference should be made to other parts of this manual when a specific subroutine is to be used in a code.

Most subroutines and the Interpretation Routine are now available in relative code form. If the other parts of a coder's routine are in relative also, this form of the subroutine should be used. The Preparatory Routine is not needed in this case.

2. Procedures

5. Input and Output

a. Input Coding

When a routine is to be read into the machine an IBM collator is used to insert the information. On initial input, SWAC inspects successive cells in the memory and replaces the zeros with words from the cards as long as it finds both F = 0 in the memory, and cards in the collator. There is a "word pulse" on each row of the card containing information. This word pulse signals SWAC to store the contents of that row on the card, and to increase epsilon.

When F is 1, SWAC inputs to the memory address specified by the alpha of the input command. Here again, the word pulse signals SWAC to store the contents of the row on the card, and to increase epsilon. However, the next command is not an input command, in general. The coder will usually modify the input command, compare to see whether a complete card has been read, and return to the input command before the next row on the card is reached.

There is ample computing time between rows to accomplish this. Note that SWAC is powerless to halt the card feed except at the end of each card. Therefore all the word pulses on a given card must be used to transfer information. That is, if there are 10 word pulses on a card, then the input command must be obeyed 10 times, with suitable modification between rows. If the SWAC is computing and word pulses are read in at random, incorrect results are produced.

If a row on a card is to be omitted, it is not sufficient to remove the word pulse from that row. Read-in to the R register will occur from the row, and the digits will remain there until another word pulse is received. If the following row has digits to be read in, these will be

2. Procedures

superimposed on the previous contents of R, and the resulting word will be incorrect. Therefore, when eliminating word pulse punches, the coder must be sure that the rows involved are blank.

The following is a typical example of an input routine:

Example 1. To read into SWAC 20 binary words, into cells 193 through 212.

Cell No.	α	β	γ	δ	F	Remarks
025	053	001	026	000	04	Prestore the input command (001 is the zero cell).
026	[193	000	000	000	01]	Input.
027	026	054	026	000	04	Modify the input command.
028	055	026	017	026	08	Have 20 words been read in? If not, return to cell 026 (017 is temporary storage).
029	Command to be obeyed after input is completed.					
053	193	000	000	000	01	Dummy for input.
054	001	000	000	000	00	2^{-8} · modifier.
055	212	000	000	000	01	Upper limit for compare command.

This routine takes 7 cells in the high speed memory.

The coder may choose to read in with Initial Input instead of Input (F = 0, instead of F = 1). He must then clear any cells which will be used for storing the data read off the cards. Previous to the acquisition of the drum, it was necessary to clear the cells by some repeated clear command. However, the coder may now call for input to the CRT memory from any channel number less than 016. Zeros will be read into any 8, 16, or 32 words in the high speed memory designated by the coder. Care must be taken that the drum command does not clear out other essential data in the routine.

2. Procedures

With this type of read-in, the data are stored in consecutive cells in the memory, unless commands are interspersed. These commands must not take more time to perform than exists between successive rows on the cards, or the machine will get into difficulties. Here, also, the coder must make sure that the number of empty cells he has assigned for storage is an exact multiple of the number of word pulses per card, so that halting of the input will occur only at the end of a card feed cycle.

Example 2. To read into SWAC 20 binary words into cells 193 through 212, using Initial Input.

Cell No.	α	β	γ	δ	F	Remarks
025	192	004	015	112	01	Clear 16 cells, from 192 thru 207 (note that 192 is cleared also).
026	192	014	015	112	01	Clear 8 cells, from 208 thru 215 (note that information in 213-215 will be destroyed).
027	053	001	213	193	05	Put a command in 213 which will return control to cell 028.
028	Command to be obeyed after input is completed.					
053	001	001	001	028	07	Dummy for command in 213.

This routine takes 4 cells in the memory, and requires the drum to be operating.

A study of the second example leads to the following comments.

A coder will in general try to begin his input in the first cell cleared by the drum. This will eliminate the awkward gap created by clearing cell 192 in example 2. Although the second example takes only 4 cells, it must have a cell reserved at the end of the read-in section, to transfer control back to the routine. This cell may be a temporary, or it may be permanently assigned. Since 026 cannot be made a special (in the second example), 027

2. Procedures

must be coded. A rearrangement of the code such that the two drum clear commands immediately precede the read-in would obviously save more space. Recoding will possibly result in the following code:

Example 3. To read into SWAC 20 binary words into cells 192 thru 211, using Initial Input.

Cell No.	α	β	γ	δ	F	Remarks
190	192	004	015	112	01	Clear 16 cells, from 192 thru 207.
191	192	014	016	112	01	Read in, from channel 016, zeros into cells 208 thru 211, and four commands in 212 thru 215.
192	Begin read-in on Initial Input.					
⋮						
211	End read-in.					
212	Command to be obeyed after input is completed.					These commands come from channel 016.
213						
214						
215						

To obviate the necessity of assigning a channel of the drum for the commands in the excess cells, the coder could plan to read his information in multiples of 8 rather than of 10.

2. Procedures

5. Input and Output

b. Typewriter Output

When the F of the command being obeyed is 2 the SWAC executes an output. The α and β portions of an output command determine the cells whose contents will be sent to an output device. The delta portion of the command determines which output device will receive the information. The beta portion determines the cell whose contents are displayed in the M register on the console during the output. The R register displays the contents of the alpha address.

If delta is 00, the SWAC types the contents of the cell referred to in alpha. (If delta is 10, there is an output to punched tape. This output has been discontinued.) Normally the typewriter format has eleven characters:

positive number: space, 9 integers, tab character, "stop"

negative number: minus, 9 integers, tab character, "stop"

The nine digits are typed or punched in reverse order, the most significant at the extreme right.

During typewriter output, the tab character activates the tab key of the SWAC typewriter. The "stop" does not appear on type-out.

The tab has no significance if read back into the SWAC from tape; its purpose is to operate the tab key of the auxiliary tape listing typewriter, enabling the operator to list answers in columns. The "stop" character merely lists on that typewriter as "/". During read-in to the

2. Procedures

SWAC it signals the end of one complete word and causes the increase of epsilon.

Typewriter output can be set by the operator for "converted" output by a switch; it is possible thus to have values expressed to base 10 or 16. With this kind of output the digits are typed in the correct order, the most significant at the extreme left. The operator can also obtain a word of eleven digits in "converted" output. Besides notifying the operator that he desires "converted" output, the coder must also code an appropriate cell in the beta of every "00" and "20" type-out. For decimal output, he must have beta of the command refer to a cell containing $5 \cdot 2^{-36}$; for hex, he refers to a cell containing $8 \cdot 2^{-36}$. It should be noted that 11 decimal digits are roughly equivalent to 9 hex digits.

If the coder is interested in only part of a word (for example, three hex digits) the operator can cause the tab and "stop" characters to follow directly after the first three integers, thus skipping the other integers. Since a word leaves storage in reverse order during "normal" output, the three integers obtained in this example would be those stored in delta and F. Note that these are the three least significant integers. On "converted" output the digits would be the three most significant integers, two in alpha, and one in beta. The type output of a nine integer word requires two seconds.

It is also possible to change the format of each word. In addition to the integers, tab, and "stop", there are available three space characters and a period. Following the coder's instructions, the operator can intersperse

2. Procedures

the other characters among the digits of an output word.

Normally the output typewriter is set to type six words per line; if the outputs come in groups of two, three, or six words no change need be made. According to the coder's instructions, the operator can cause the typing of fewer words per line to accommodate groups of four or five words. In all the above variations of output the coder must bear in mind that any change in the output for type will effect all words.

20 Output

If the delta of an output command is 20, SWAC types nine letters representing the nine digits in the cell referred to by alpha. There is a limited alphabet of 15 letters and a space. (Cf. paragraph on Letters, this section, for corresponding digits and letters.) As an example of its use, suppose the coder were interested in knowing only the fact that f(x) had exceeded a limit. He might not have the routine halt in the event of that overflow. However, he might code the routine to type a reminder to re-check the function at x, and then to continue computing with x + 1.

The following commands would type such a reminder:

Cell No.	α	β	γ	δ	F	Remarks
32	7x	00	00	20	2	Type "re do".
33	4z	00	00	00	2	Type x at time of overflow.
34		(Continue with rest of computing routine)				
⋮						
4z		Temporary storage assigned to x.				
⋮						
7x	xx	29	xy	3x	x	"Re do" in reverse order, with space characters inserted.

2. Procedures

NOTE: The cell 7x could contain xx od xe rx x on the coding sheets and the same letters would be typed from SWAC. (The "x" code is a space.) If the SWAC has been set for "converted" output, it would be possible to use "20" letter output, providing the beta of the output command refers to a cell containing $8 \cdot 2^{-36}$.

30 Output

If the delta of an output command is 30, dummy output takes place. This command is rarly used, for it causes only a momentary halt in the routine.

40 Output

A delta of 40 in an output command causes SWAC to type one letter or one number representing the last five binary digits in the cell referred to by alpha. The tab and carriage return keys of the output typewriter are not automatically activated as in other type-outs. They must be coded in "alpha" cells just as are the letters. If a routine includes any "40" outputs there must be instructions to the operator to set the "40" output switch either to "numbers" or "letters". (See table at end of this section.) Though only one character is typed for each output command, there is available a complete alphabet.

If SWAC has been set for "converted" output (instead of "normal"), a space will be typed out.

2. Procedures

Using 20 in $\mathcal{S}$ in the Output Command makes it possible to type out using a limited alphabet. That is, numbers in the computer will be printed out as letters according to the following table:

Letters (20 Output)

Letter	No.	Letter	No.	Letter	No.	Letter	No.
a	4	e	y	m	u	r	3
b	5	f	z	n	v	s	6
c	8	i	0	o	2	t	7
d	9	j	1	p	w	space	x

2. Procedures

"40" Output (single character) Table

If (α) contains: α β γ δ F	SWAC will type: Switch Setting Numbers	Letters
00 00 00 00 0	Nothing	Space
01	Backspace	1
02	Nothing	.
03	Nothing	Carriage Return
04	½	Shift Up
05	'	Shift Down
06	Nothing	g
07	Nothing	h
08	-	tab
09	/	k
0u	,	u
0v	;	v
0w	Nothing	w
0x	Nothing	x
0y	Nothing	y
0z	Nothing	z
10	0	i
11	1	j
12	2	o
13	3	r
14	4	a
15	5	b
16	6	s
17	7	t
18	8	c
19	9	d
1u	u	m
1v	v	n
1w	w	p
1x	x	Space
1y	y	e
1z	z	f

2. Procedures

5. Input and Output

c. Punched Card Output

When the delta of an output command is 50, the SWAC causes the numbers in the cells referred to by α and β to be punched on an IBM card. A card may receive 1 to 24 words in binary representation*. The time required for punching allows SWAC to obey two or three commands between the punching of two successive lines (or words). The coder may choose to store only one output command, and modify the α and β of the command, or he may write the successive output commands explicitly.

The coder is free to specify which rows on the card he desires to have punched. Such specification may not be changed during the course of the computation. Remember that, if the cards are fed back into the SWAC, the "0" and "Y" rows will not be read by the collator brushes. Computing may be done by the SWAC while the card is passing over rows not being punched. Computing time of approximately 20 milliseconds is available between two punchings separated by one blank row.

The cards punched from output may be read back into SWAC as data, by the collator. During the punching, a word pulse accompanies each line or word of punching; during subsequent input the word pulse is the signal to SWAC of the completion of one word, causing epsilon to increase. If any of the words punched out are not to be read back into SWAC, the coder must anticipate this at the time of punch-out by requesting the

---

*Whether SWAC is on "Normal" or "Converted" Output is immaterial.

2. Procedure

operator to inhibit the WP punch where necessary. Any or all of the word pulse punches may be eliminated.

Eliminating the word pulse does not eliminate read-in to the R register. The digits on a row without a word pulse will be set in R and remain there until a Start Function Table pulse is received. This may be a word pulse later on the card, or a pulse from SWAC. If SWAC is computing and a row of digits is read into R incorrect results are sure to be produced. Therefore, when eliminating word pulse punches, the coder must in general be sure that the rows affected are blank.

Routines have been coded which will call for collator input of cards carrying binary values, and punch out cards of the decimal equivalents. This enables a coder to have the punched results of a computation translated to decimal and then listed on IBM equipment. There is also a routine to translate data punched in decimal form to cards in binary notation; the binary data is then ready to be called for as input during a computation routine.

2. Procedures

5. Input and Output

d. Breakpoint Output

A breakpoint output command is designated by 60 in delta. No output device is activated; breakpoint is a halt, accompanied by the display of two values on the Console of SWAC. The contents of the cells referred to in the alpha and beta portions of the output command show in the R and M registers, respectively, for comparison with hand-computed values.* This is most useful for checking out coding on SWAC; after a routine is running the operator may set a switch to bypass the breakpoint halt.

Note that although the breakpoint is not operative with the switch "off", an appreciable delay is introduced as each breakpoint is passed. If there are many breakpoints of this type, an alternate routine, omitting them, should be used during the computation.

* Whether SWAC is set for "normal" or "converted" output is immaterial in "60" outputs.

2. Procedures

5. Input and Output

e. Summary of Input and Output Codes

Input Command: F is 0 or 1

Output Command: F is 2

Delta			
Hex.	Dec.	Input Unit	Output Unit
00	000	Collator	Typewriter (nine digits)
10	016	Keyboard	(Inactive)
20	032	Collator	Typewriter (nine letters)
30	048	Keyboard	Dummy
* 40	064	Collator	Typewriter (one character)
50	080	Keyboard	IBM Card Punch
60	096	Collator	Breakpoint
70	112	Drum	Drum

* Does not operate correctly on converted output.

2. Procedures

9. Writing up Codes for SWAC Manual

No code which any one else might want to use should be regarded as finished until a description of it has been inserted into this Manual in Part E or Part F. Preparation of such a description is the responsibility of any person making a SWAC code.

Whenever possible, code descriptions should follow this outline:

1) Number of code.

2) Who wrote it and/or who knows the code well. Date of composition.

3) Title.

4) Summary. [What parameters there are, how accurate, etc.]

5) Preparation of input data. [E.g., how to punch matrices.]

6) Operating procedure:

   a. Output [converted or not; output board and jack plugs].

   b. Switches [breakpoint, delay].

   c. Drum. [Used or not; not even to clear cells ?]

   d. Input. [Special collator board? Description of what cards are in the input deck.]

   e. Machine operation. [Expected inputs, outputs, halts, etc.]

   f. Expected duration [as function of parameters, if known].

   g. Failure halts [and what do to].

   h. Final program halt.

   i. How to deal with output. [Listing on IBM, follow with another SWAC code, etc.]

2. Procedures

7) Mathematical method used. [Brief survey and/or references to published material or other descriptions in our library.]

8) Outline of what cells and what drum channels are used for what. [Where this information looks important].

9) Flow chart. [Either reproduce it or state its location in the building.]

10) Code. [Either reproduce it or state its location in the building. Remarks are usually desirable.]

10. Standard Categories in Code Library (stored in binary card files)

Routine numbers	Reference in SWAC Manual	Category
00000 - 00099	Part D	Memory Tests
00100 - 00199	Part D	Arithmetic tests
00200 - 00299	Part E	Conversion routines Even-numbered routines -- decimal to binary. Odd-numbered routines -- binary to decimal. Miscellaneous (odd-numbered and even-numbered routines).
00300 - 00499	Parts C and E	Subroutines 00300 - 00309 Interpretive codes 00310 - 00319 Division 00320 - 00329 Square root and square root with division 00330 - 00339 Logarithms 00340 - 00349 Trigonometric functions 00350 - 00359 Exponential functions 00360 - 00369 Floating point and complex operations 00370 - 00379 Multiple precision operations 00380 - 00389 Memory output routines 00400 - 00409 Data-handling routines
00500 - 00599	Part F	Matrix codes 00510.0 - 00519.9 Data handling and conversion 00520.0 - 00529.9 Inverting matrices 00530.0 - 00539.9 Solving linear systems 00540.0 - 00549.9 Computing eigenvalues 00540.0 - 00544.9 Symmetric matrices 00545.0 - 00549.9 Nonsymmetric matrices 00550.0 - 00559.9 Solving linear inequalities 00560.0 - 00569.9 Solving linear games 00570.0 - 00579.9 Solving nonlinear algebraic systems

10. Standard Categories in Code Library (stored in binary card files)

Routine numbers	Reference in SWAC Manual	Category
00600 - 00699	Part F	Ordinary and Partial Differential Equations Routines
		00600 - 00649 Ordinary Differential Equations
		00650 - 00699 Partial Differential Equations
		00650 - 00659 Initial Value Problems
		00660 - 00669 Boundary Value Problems
		00670 - 00679 Eigenvalue Problems
		00680 - 00699 Miscellaneous
00700 - 00799	Part E	Drum codes
		Tests, loading, punching, clearing, etc.

2. Procedures

3. Interpretive coding

b. SWAC Subroutines Available in NAR Library*

Routine number	Title (unless otherwise noted, routines are single precision, fixed point, designed for use with 00300)	Approximate ave.time (millisec.)	Number of cells	Coded by:	Date
00300	Interpretation routine, used with codes 00310 through 00369 (single precision).	1.5	37	R.Lipkis, R.Horgan	1/50
00302	Interpretation routine, used with code 00370 (double precision).		33	L.Rickard, S. Marks	4/53
00303	Interpretation routine, used with code 00371 (triple precision).		38	L.Rickard, S. Marks	4/53
00305	Preparatory routine. Modifies cross-references within a subroutine; made obsolete by use of card input, and relative coding form of subroutines.		14	R.Lipkis	1/50
00310	Division, any size quotient. Coder may specify scaling of quotient.	12	19	R. Lipkis	2/50
00311	Division, quotient ≤ 1, absolute value.	28	13	L.Straus	7/52
00320	Square root, result rounded in last binary digit.	25	15	R.Bozak	1/51
00321	Square root, not rounded. Assumes operand > 0.	23	8	L.Straus	7/52
00322	Square root and division. Obtains absolute value of quotient. Assumes denominator $\neq 0$.	25	12	L.Straus	10/52

*Available in card files: binary, relative-decimal, and absolute decimal form.

2. Procedures

Routine number	Title (unless otherwise noted, routines are single precision, fixed point, designed for use with 00300)	Approximate ave. time (millisec.)	Number of cells	Coded by:	Date
00323	Square root and division. Obtains quotient with proper sign. Halts if denom. = 0.	23	17	L. Straus	10/52
00324	Square root.	11	27	T. Southard	6/55
00330	$\log_e N$. Variable t in $2^{-t}\log_e N$; stores $N \times 2^{-r}$, $r > 1$.	23	34	L. Straus	8/52
00331	$\log_e N$. Fixed t = 6 in $2^{-t}\log_e N$; stores $N \times 2^{-r}$, $r > 1$.	22	30	L. Straus	8/52
00332	$\log_e N$. Obtains $2^{-6}\log_e N$; $N < 1$; stores $N \times 2^{0}$.	29	17	L. Straus	7/53
00340	Sine and cosine.	7	32	L. Cutler, R. Bozak	9/52
00341	Sine and cosine, floating point.	9	46	L. Cutler, R. Bozak	7/51
00350	e^{-x}.	7	28	S. Marks	10/51
00351	$a^{\pm x}$. (Subroutine includes 00322 division and square root.)		35	L. Straus	7/53
00352	e^{-x}, floating point.		32	M. Melkanoff	8/53
00360	Floating operations: add subtract multiply divide	4 4 2 6	45	R. Lipkis, L. Cutler	8/52
00361	Floating operations: add subtract multiply divide sq. root	4 4 2 6 8	63	L. Cutler, R. Horgan	9/52

2. Procedures

Routine number	Title (unless otherwise noted, routines are single precision, fixed point, designed for use with 00300)		Approximate ave. time (millisec.)	Number of Cells	Coded by:	Date
00362	Complex operations:	add	1	50	R. Lipkis, L. Cutler	9/52
		subtract	1			
		multiply	2			
		divide	36			
00363	Complex operations:	add	1	68	L. Cutler, R. Horgan	9/52
		subtract	1			
		multiply	2			
		divide	36			
		$\sqrt{x^2 + y^2}$	26			
		$\sqrt{x + iy}$	77			
00370	Double precision operations:			64**	S. Marks, L. Rickard	4/53
		add	2			
		subtract	2			
		multiply	4			
00371	Triple precision operations: (add, subtract, multiply)			94**	S. Marks, L. Rickard	4/53
00372	Double precision operations:			102	R. Lipkis, S. Marks	1/51
		add	2			
		subtract	2			
		multiply	4			
		divide	100			
	Uses special interpretive routine; see code.					

**Includes interpretive code.

3. Punching

1. Identification of Punchouts

The standard 36-digit output board has two 8-binary-digit identification fields: cols. 37-44 punch the α part of the M register and cols. 57-64 punch the β part of the M register.

The coding of punch commands to make use of this feature is as follows:

1) If the address of the punch command is put in the β part of the punch command, cols. 37-44 will show the address from which the word was punched and cols. 57-64 will show the address of the punch command.

2) If the address of the zero storage cell is put in the β part of the punch command, these fields will be blank.

3) If 000 is put in the β part of the punch command, col. 44 will be punched in each row, but all other cols. will be blank, assuming the command 001 000 000 016 01 is in cell 000.

NOTE: It is not possible to gang punch information in columns 37-44, or 57-64, on the standard 36-digit board. Columns 15 thru 26 are available, as well as columns 35, 36, 45, 46, 55, 56, 65, 66 and 80.

1. Operating Procedures

1. General Procedures

a. Summary

These general operating procedures are issued in order to systematize the many conflicting operations which must be done on the SWAC.

The reader must realize that several different types of operations, and many groups of individuals, require time on the machine. Some of the operations are: (1) Research problems for NAR staff members, (2) Research problems for UCLA staff members, (3) Machine maintenance and testing, and (4) Engineering modifications.

A committee has been set up to serve as a clearing house for schedules, priorities, and general administrative details of SWAC operation.

The following operating procedures are issued by the committee to serve as a general guide. They are subject to alteration by the SWAC Supervisor.

1. All departments using SWAC are expected to do their own coding and operating. Instruction in both is available from NAR.
2. A qualified operator must be present in the SWAC console room at all times when the machine is in operation, except when the engineers have taken charge of the machine. A list of individuals qualified to operate SWAC is posted on the SWAC bulletin board. Anyone may learn to operate SWAC and qualify for operation. To qualify for operator a person must be familiar with the console panels of SWAC, must know how to

1. Operating Procedures

turn the machine on and off, must be able to test the machine, and must be able to operate the IBM equipment attached to it. See the SWAC Supervisor for details. Under no circumstances is SWAC to be operated by an unauthorized operator without specific permission from the SWAC Supervisor.

3. Requests for SWAC time should be made on forms posted on the SWAC bulletin board. Any individual whose codes are ready to be checked or run will sign the list and give dates and times when he will be available, problem number, and estimated time required to check the code or compute. Estimates of time wanted are to be made in units of tenth hours. Time will be assigned according to the order in which requests appear on these sheets, subject to priorities and previous commitments of time. Backlog problems are usually available to fill in unused or unwanted time, but each coder should make an effort to estimate accurately the time he needs. The SWAC has a scheduled warm-up time of $\frac{1}{2}$ hour, and testing time of 1 hour. Engineering usually takes $1\frac{1}{2}$ hours. This leaves 13 hours of each 16 hour double shift for computing.

4. The SWAC Supervisor will make a daily schedule not more than 24 hours in advance from the posted lists of requests for SWAC time. In the event of breakdown, the schedules may be shifted or rearranged. Each operator is expected to be available when SWAC is ready for his problem.

1. Operating Procedures

5. The scheduled operation of SWAC covers the period from 0800 to 1700 Monday through Friday. Operation outside this period is on a voluntary basis, but a schedule of nominal assignments of swing shift time is posted once a week on the bulletin board. Changes in this schedule may be made by informal agreements among the qualified operators. In general it will be advisable for at least two operators to share a swing or graveyard shift and relieve each other occasionally.

6. University rules of safety require the presence of two people in any building where there is electrical equipment. Therefore, during swing and graveyard shifts, at least two persons, one of whom must be an operator, must be present in the SWAC room, electrical laboratory, or the IBM room at all times when SWAC is in operation. It is not sufficient that the second person be in the library, or his office. Unless he is within close call of the SWAC operator the whole purpose of the rules is defeated. If an operator is unable to be present at the console, and an alternate operator is not available, the machine must be turned off.

7. It is the responsibility of swing and graveyard shift operators to lock up securely when they are through. See that all power supplies, irons, test equipment, and air conditioners are off, that the stockroom door is locked and the key in its proper place in the SWAC console, and that the three entrances to the

1. Operating Procedures

SWAC rooms are secured. Include a checkup to see that IBM equipment is turned off, including the hand punch at the SWAC console, and the one in the laboratory. Two check lists are posted, one on the wall next to the SWAC bulletin board, and one over the telephone in the laboratory. The person locking up is requested to initial both these sheets after inspecting the items listed.

8. If routine maintenance is performed during the swing shift, one person in addition to the engineer must remain near SWAC during the maintenance period. Even though the machine is off, the test equipment is not.

9. If breakdowns occur after 1700, or on weekends, and one of the engineers is not available, the machine should be turned off. Operators should not attempt to do any engineering. Note on the log the symptoms of any failures as an aid to diagnosis on the following day.

10. Space around the SWAC console is limited. This area should be occupied only by those concerned with the problem at hand. Staff members are requested to keep this space clear so those operating SWAC can have ample room. The operator on duty has the right to enforce this rule. Operators will make necessary exceptions to this rule when visitors are present.

11. Operation of the IBM equipment not directly connected with SWAC is under the same rules as those stated here, wherever applicable. Rules 2, 6, and 7, in particular, must be followed.

1. Operating Procedures

1. General Procedures

b. SWAC Log

1. A continuous record of all operations on SWAC must be maintained on the output typewriter. This record is used to make up cost accounts as well as statistical breakdowns of SWAC time. The following paragraphs are intended as a guide to enable operators to keep the log in a systematic and accurate manner:

2. The categories of time which may be recorded on the SWAC log have been reduced in number from twelve to ten:

A. ROUTINE MAINTENANCE: Maintenance of chassis by the engineers.

B. OUT OF ORDER: Machine not functioning, and engineers in charge.

C. IDLE: Power on, but machine not being operated. Operator has abandoned the machine and has been unsuccessful in finding a user.

D. TESTING: All component or circuit testing.

E. IMPROVING ENGINEERING FACILITIES: Modification, change, or experiment on any equipment. Work on collator, punch, or tabulator comes under this category. Engineers are in charge.

F. IMPROVING MATHEMATICAL FACILITIES: Code checking or testing of routines of a general nature not specifically intended for any particular problem. This includes sub-routines, conversion routines, and test routines. Computing Committee is in charge.

1. Operating Procedures

G. DEMONSTRATION: Exhibition of machine to any visitors whenever this involves the use of the machine to the exclusion of other work. Such demonstrations must be arranged in advance with the Chairman of the Computing Committee.

H. CODE CHECKING: Problems whose codes are being checked out.

I. COMPUTING: Problems in which results are being produced, or conversions of codes or results.

J. OFF: Power off.

3. The time of beginning of each operation is written as a four digit number: 0000 to 2400, i.e., on a 24-hour basis. This time is recorded to the nearest multiple of six minutes. The clock in the SWAC room is marked off at six minute intervals to assist in correct recording of the time.

4. The record for each entry is to be written with the time of beginning at the left margin. On the same line, in capital letters, type one of the nine categories listed in 2. above. On the line beneath list the problem number and initials or name of the operator. A sample of the log might look as follows:

0955 COMPUTING
809-00 (K. N. Trueblood)

1137 CODE CHECKING
6500 (Wilson)

1225 IMPROVING ENGINEERING FACILITIES
(F.H.H.)

(Engineer will note type of work done.)

1. Operating procedures

1400 (F.H.H.)	TESTING
00100	(Name of test, followed by SWAC typeouts, if any.)

5. The SWAC day on the log begins at 0800. Please start a new page at 0800. This is particularly important if the machine is run through the graveyard shift.

6. The log is typed in duplicate. The carbon copy may be removed after typing and filed by the operator, if desired. A new page may be started for any person wishing it. Always leave the original copy on the typewriter.

7. Details of all difficulties encountered, and any evidences of erratic behavior, should be noted on the log, together with the time of the occurrence. Do not be too brief! The more information the log contains, the more useful it will be. Please be accurate in noting problem and test routine numbers, and the time.

8. If the machine is suspected of malfunction during CODE CHECKING or COMPUTING the operator must call an engineer if one is available. In the absence of an engineer, the burden of proof as to malfunction of SWAC is on the operator and he may not change his category of COMPUTING or CODE CHECKING to any other except IDLE. If he chooses to test the machine, and successfully traces erratic behavior, then his time will be charged to the appropriate category when the log is made up. If he does not wish to run the risk of finding no trouble, then he should turn the machine over to some one else or leave it idle.

1. Operating procedures

9. In no case is an operator to attempt to repair the machine without the advice of an engineer. This is particularly applicable to all graveyard and weekend operation. The risk of damage to the machine or the operator is too great to make the gain of a few hours computing time worth while, even supposing that a repair could be made. This prohibition does not extend to clearing jammed cards from the IBM equipment. It does specifically prevent an operator from replacing any component or chassis inside the SWAC, the Drum, or the Power Cabinets.

1. Operating Procedures

2. Command Analysis

a. Addition and Special Addition (Time 64 μsec.)

Following are the steps which take place in Addition (4) and Special Addition (5):

1. Cl(M), Cl(A,R), (α) → (M)

2. a) K digits of (M) wr 1's — [The digits are complemented with respect to 1's depending on sign. If sign is positive the digits are not complemented; if sign is negative the 1's and 0's are interchanged. Thus, 1 000 101 (-5 in decimal notation) is changed to 1 111 010. Note that sign is not changed.]

b) (M) + (A) + CO 1(?) → (A) — (Corrective 1 is only inserted if (M) was complemented. It is inserted in carry input circuit of the units stage.)

3. Cl(M), (β) → (M)

4. a) Repeat Step 2a — (Complemented with respect to 1's depending on sign, as in Step 2a.)

b) Repeat Step 2b — (Sum now stored in A register may be complementary number.)

c) K digits of (A) wr 2^n; Cl(M) — (Complement if (A) is a complementary number.)

5. a) (A) → (M)
b) (M) → (γ)

6. $\mathcal{E}$ + 1 → $\mathcal{E}$, Cl(M)

If Capacity Not Exceeded and 4.	If Otherwise (Capacity Exceeded, and/or 5.)
7. Cl(M), ($\mathcal{E}$) → (M)	7. Cl(M), (δ) → (M), Cl($\mathcal{E}$), δ → $\mathcal{E}$
8. Cl(C), (M) → (C), Cl(M), Cl(O)	

1. Operating Procedures

Example of Addition:

Augend: (α) = 0 000 101 (+5)

Addend: (β) = 1 001 011 (-11)

	M Register	A Register
End of 1.	0 000 101	0 000 000
" " 2 a.	0 000 101	0 000 000
" " 2 b.	0 000 101	0 000 101
" " 3.	1 001 011	0 000 101
" " 4 a.	1 110 100	0 000 101
" " 4 b.	1 110 100	1 111 010
" " 4 c.	0 000 000	1 000 110
" " 5 a.	1 000 110	1 000 110
" " 5 b.	1 000 110	1 000 110
" " 6.	0 000 000	1 000 110
" " 7.	Next Command	1 000 110
" " 8.	0 000 000	1 000 110

1. Operating Procedures

2. Command Analysis

b. Subtraction and Special Subtraction (Time 64 μsec.)

Following are the steps which take place in Subtraction (6) and Special Subtraction (7):

1. Cl(M), Cl(A,R), (α) → (M)

2. a) K digits of (M) wr 1's — [The digits are complemented with respect to 1's depending on sign. If sign is positive the digits are not complemented; if sign is negative the 1's and 0's are interchanged. Thus, 1 000 101 (-5 in decimal notation) is changed to 1 111 010. Note that sign is not changed.]

b) (M) + (A) + CO 1(?) → (A) — (Corrective 1 is only inserted if (M) was complemented. It is inserted in carry input circuit of the units stage.)

3. Cl(M), (β) → (M)

4. a) K digits of (M) wr 1's — (Complementation depends upon sign and subtraction operation. That is, complementation and change of sign takes place if number is positive. If negative, do not complement but change sign.)

b) Repeat Step 2b — (Sum now stored in A register may be a complementary number.)

c) K digits of (A) wr 2^n; Cl(M) — (Complement if (A) is a complementary number.)

5. a) (A) → (M)
b) (M) → (γ)

6. $\mathcal{E}$ + 1 → $\mathcal{E}$, Cl(M)

If Capacity Not Exceeded and 6.	If Otherwise (Capacity Exceeded, and/or 7)
7. Cl(M), (ε) → (M)	7. Cl(M), ($\mathcal{E}$) → (M), Cl($\mathcal{E}$), δ → $\mathcal{E}$
8. Cl(C), (M) → (C), Cl(M), Cl(0)	

Note: With exception of Step 4a all operations are the same as in Addition.

1. Operating Procedures

Example of Subtraction:

Minuend: (α) = 0 000 101 (+5)

Subtrahend: (β) = 1 001 011 (-11)

	M Register	A Register
End of 1.	0 000 101	0 000 000
" " 2 a.	0 000 101	0 000 000
" " 2 b.	0 000 101	0 000 101
" " 3.	1 001 011	0 000 101
" " 4 a.	0 001 011	0 000 101
" " 4 b.	0 001 011	0 010 000
" " 4 c.	0 000 000	0 010 000
" " 5 a.	0 010 000	0 010 000
" " 5 b.	0 010 000	0 010 000
" " 6.	0 000 000	0 010 000
" " 7.	Next Command	0 010 000
" " 8.	0 000 000	0 010 000

1. Operating Procedures

2. Command Analysis

c. Multiplication Round-Off and Special Multiplication Round-Off (Time 384 μsec.)

Following are the steps which take place in Multiplication Round-Off (u) and Special Multiplication Round-Off (v):

1. Cl(M), Cl(A,R), (α) → (M)
2. (M) → (R)
3. Cl(M), (β) → (M)
4. a) Sign (M) added to sign (R), Cl sign (M)
   b) Multiply — [Shift and add. That is, shift (A) and (R), excluding sign digits, one position to right. Inspect digit in extra position at right of R register. If it is 1, add multiplicand into the A register. Shift again. Machine does as many additions as there are 1's in the multiplier, and one shift more than the number of digits, excluding sign digit, in the multiplier. Process always ends with shift.]
   c) Cl(M), sign (R) → sign (A), add RO 1 — [Round-off answer by adding 1 to the least significant digit of (A) if the most significant digit of (R) is a 1.]
   d) (A) → (M)
5. (M) → (γ)
6. $\varepsilon + 1 \rightarrow \varepsilon$, Cl(M)

If u.	If v. (Special)
7. Cl(M), (ε) → (M)	7. Cl(M), (δ) → (M), Cl(ε), $\delta \rightarrow \varepsilon$
8. Cl(C), (M) → (C), Cl(M), Cl(O)	

Note: In Multiplication digits are always stored in true form, not complemented form as is sometimes done in Addition.

1. Operating Procedures

Example of Multiplication Round-Off:

Multiplier: (α) 1 101 101 (-45); Multiplicand: (β) 0 000 101 (+5)

	M Register	A Register	R Register
End of 1.	1 101 101	0 000 000	0 000 000 0
" " 2.	1 101 101	0 000 000	1 101 101 0
" " 3.	0 000 101	0 000 000	1 101 101 0
" " 4 a. 4 b.	0 000 101	0 000 000	1 101 101 0
End of 1st Shift	0 000 101	0 000 000	1 010 110 1
" " 1st Addition	0 000 101	0 000 101	1 010 110 1
" " 2nd Shift	0 000 101	0 000 010	1 101 011 0
	0 000 101	0 000 010	1 101 011 0
" " 3rd Shift	0 000 101	0 000 001	1 010 101 1
" " 2nd Addition	0 000 101	0 000 110	1 010 101 1
" " 4th Shift	0 000 101	0 000 011	1 001 010 1
" " 3rd Addition	0 000 101	0 001 000	1 001 010 1
" " 5th Shift	0 000 101	0 000 100	1 000 101 0
	0 000 101	0 000 100	1 000 101 0
" " 6th Shift	0 000 101	0 000 010	1 000 010 1
" " 4th Addition	0 000 101	0 000 111	1 000 010 1
" " 7th Shift	0 000 101	0 000 011	1 100 001 0
End of 4 c.	0 000 000	1 000 100	1 100 001 0
" " 4 d.	1 000 100	1 000 100	1 100 001 0
" " 5.	1 000 100	1 000 100	1 100 001 0
" " 6.	0 000 100	1 000 100	1 100 001 0
" " 7.	Next Command	1 000 100	1 100 001 0
" " 8.	0 000 000	1 000 100	1 100 001 0

1. Operating Procedures

2. Command Analysis

d. Multiplication Exact [Product] (Time 384 μsec.)

Following are the steps which take place in Multiplication Exact (w):

1. Cl(M), Cl(A,R), (α) → (M)
2. (M) → (R)
3. Cl(M), (β) → (M)
4. a) Sign (M) added to sign (R), Cl sign (M)

   b) Multiply — [Shift and add. That is, shift (A) and (R), excluding sign digits, one position to right. Inspect digit in extra position at right of R register. If it is a 1, add multiplicand into the A register. Shift again. Machine does as many additions as there are 1's in the multiplier, and one shift more than the number of digits, excluding sign digit, in the multiplier. Process always ends with shift.]

   c) Cl(M), sign (R) → sign (A)

   d) (R) → (M)

   e) (M) → (δ)

   f) Cl(M), (A) → (M)
5. (M) → (γ)
6. $\varepsilon + 1 \rightarrow \varepsilon$, Cl(M)
7. Cl(M), (ε) → (M)
8. Cl(C), (M) → (C), Cl(M), Cl(O)

Note: The operations to Step 4c are the same for Multiplication Round-Off and Multiplication Exact.

1. Operating Procedures

Multiplier: (α) = 1 101 101 (-45); Multiplicand: (β) = 0 000 101 (+5)

	M Register	A Register	R Register
End of 1.	1 101 101	0 000 000	0 000 000 0
" " 2.	1 101 101	0 000 000	1 101 101 0
" " 3.	0 000 101	0 000 000	1 101 101 0
" " 4 a. 4 b.	0 000 101	0 000 000	1 101 101 0
End of 1st Shift	0 000 101	0 000 000	1 010 110 1
" " 1st Addition	0 000 101	0 000 101	1 010 110 1
" " 2nd Shift	0 000 101	0 000 010	1 101 011 0
	0 000 101	0 000 010	1 101 011 0
" " 3rd Shift	0 000 101	0 000 001	1 010 101 1
" " 2nd Addition	0 000 101	0 000 110	1 010 101 1
" " 4th Shift	0 000 101	0 000 011	1 001 010 1
" " 3rd Addition	0 000 101	0 001 000	1 001 010 1
" " 5th Shift	0 000 101	0 000 100	1 000 101 0
	0 000 101	0 000 100	1 000 101 0
" " 6th Shift	0 000 101	0 000 010	1 000 010 1
" " 4th Addition	0 000 101	0 000 111	1 000 010 1
" " 7th Shift	0 000 101	0 000 011	1 100 001 0
End of 4 c.	0 000 000	1 000 011	1 100 001 1
" " 4 d.	1 100 001	1 000 011	1 100 001 1
" " 4 e.	1 100 001	1 000 011	1 100 001 0
" " 4 f.	1 000 011	1 000 011	1 100 001 0
" " 5.	1 000 011	1 000 011	1 100 001 0
" " 6.	0 000 000	1 000 011	1 100 001 0
" " 7.	Next Command	1 000 011	1 100 001 0
" " 8.	0 000 000	1 000 011	1 100 001 0

1. Operating Procedures

2. Command Analysis

e. Compare and Special Compare (Time 64 μsec.)

Following are the steps which take place in Compare (8) and Special Compare (9):

	If 8.*	If 9 (Special).*
1.	Cl(M), Cl(A,R), (α) → (M)	Cl(M), Cl(A,R), (α) → (M) excepting sign digit.
2.	a) K digits of (M) wr 1's. (The digits are complemented with respect to 1's depending on sign. If sign is positive, the digits are not complemented; if sign is negative, the 1's and 0's are interchanged except for sign digit, the same as in Addition and Subtraction.)	
	b) (M) + (A) + CO 1(?) → (A) (Corrective 1 is only inserted if (M) was complemented.)	
3.	Cl(M), (β) → (M).	Cl(M), (β) → (M) excepting sign digit.
4.	a) K digits of (M) wr 1's. (Complementation depends upon sign and subtraction operation. That is, complementation and change of sign takes place if (M) is positive. If negative, do not complement, but change sign.)	
	b) Repeat Step 2b.	
	c) K digits of (A) wr 2^n; Cl(M). (Complement if (A) is a complementary number. Note sign remains unchanged.)	
5.	a) (A) → (M)	
	b) (M) → (γ)	
	If (A) is negative.	If (A) is non-negative.
6.	ε + 1 → ε, Cl(M)	ε + 1 → ε, Cl(M)
7.	Cl(M), (ε) → (M)	Cl(M), (δ) → (M), Cl(ε), δ → ε
8.	Cl(C), (M) → (C), Cl(M), Cl(0)	

*Note: The only difference between Compare and Special Compare is that in Special Compare the sign digit is not transferred in Steps 1 and 3.

1. Operating Procedures

Example of Compare:

Minuend: (α) = 1 000 101 (-5)
Subtrahend: (β) = 0 000 110 (+6)
5th Address: ε = 01 001 001

	M Register	A Register	ε Counter
End of 1.	1 000 101	0 000 000	01 001 001
" " 2 a.	1 111 010	0 000 000	01 001 001
" " 2 b.	1 111 010	1 111 011	01 001 001
" " 3.	0 000 110	1 111 011	01 001 001
" " 4 a.	1 111 001	1 111 011	01 001 001
" " 4 b.	1 111 001	1 110 101	01 001 001
" " 4 c.	0 000 000	1 001 011	01 001 001
" " 5 a.	1 001 011	1 001 011	01 001 001
" " 5 b.	1 001 011	1 001 011 (negative)	01 001 001
" " 6.	0 001 000	1 001 011	01 001 010
" " 7.	Next Command	1 001 011	01 001 010
" " 8.	0 000 000	1 001 011	01 001 010

Note: In Special Compare the minuend and subtrahend are treated as positive numbers regardless of whether their sign digits are positive or negative. Hence, in Special Compare the above example would still be valid if (α) = +5 and/or (β) = -6.

1. Operating Procedures

2. Command Analysis

f. Extract (Time: 0 to 5 right shifts = 128 μsec.
6 to 13 " " = 192 μsec.
14 to 21 " " = 256 μsec.
22 to 29 " " = 320 μsec.
30 to 36 " " = 384 μsec.)

Following are the steps which take place in Extract (y):

1. Cl(M), Cl(A,R), $(\alpha) \rightarrow$ (M)

2. (M) $\rightarrow$ (R)

3. Cl(M), $(\mathcal{S}) \rightarrow$ (M)

4. a) K digits of (M) wr 1's, $\mathcal{S} \rightarrow$ PC* (Independent of sign; sign is also changed.)

   b) Superimpose (R) on (M) [Where 0's in (R), (M) unchanged; otherwise, (M) changed into 1's.]

   c) K digits of (M) wr 1's, Cl(A), Cl(R) (Independent of sign; sign is also changed.)

   d) Sign (M) $\rightarrow$ sign (R), Cl sign (M), (M) $\rightarrow$ (A)

   e) Shift A right a number of times as determined by $\mathcal{S}$. If $\mathcal{S}_2 = 1$, number of shifts = number represented by last 6 digits of $\mathcal{S}$. If $\mathcal{S}_2 = 0$, number of shifts = number of digits in the word minus the number represented by the last 6 digits of $\mathcal{S}$. [See g) below.] (First digit of $\mathcal{S}$ always = 0. Shifts greater than number of digits in a word cause machine to operate improperly. Note that the sign digit is not shifted.)

   f) Cl(M), sign (R) $\rightarrow$ sign (A), add RO 1 (Round-off same as in Multiplication Round-Off.)

   g) Shifted number $\rightarrow$ (M) (Number transferred from A register if $\mathcal{S}_2 = 1$ (right shift); from R register if $\mathcal{S}_2 = 0$ (left shift).]

5. (M) $\rightarrow (\mathcal{Y})$

6. $\mathcal{E} + 1 \rightarrow \mathcal{E}$, Cl(M)

7. Cl(M), $(\mathcal{E}) \rightarrow$ (M)

8. Cl(C), (M) $\rightarrow$ (C), Cl(M), Cl(0)

*Note: $\mathcal{S}_2 - \mathcal{S}_4$, $\mathcal{S}_6 - \mathcal{S}_9$, are transferred to the Product Counter.

1. Operating Procedures

Example of Extract:

Extractor: (α) = 0 001 100 (+12)
Extractee: (β) = 1 001 011 (-11)
4th Address: δ = 00 000 101 (Last 6 digits = 5)

	M Register	A Register	R Register
End of 1.	0 001 100	0 000 000	0 000 000
" " 2.	0 001 100	0 000 000	0 001 100
" " 3.	1 001 011	0 000 000	0 001 100
" " 4 a.	0 110 100	0 000 000	0 001 100
" " 4 b.	0 111 100	0 000 000	0 001 100
" " 4 c.	1 000 011	0 000 000	0 000 000
" " 4 d.	0 000 011	0 000 011	1 000 000
" " 4 e.	0 000 011	0 000 001	1 100 000
" " 4 f.	0 000 000	1 000 010	1 100 000
" " 4 g.	1 100 000	1 000 010	1 100 000
" " 5.	1 100 000	1 000 010	1 100 000
" " 6.	0 000 000	1 000 010	1 100 000
" " 7.	Next Command	1 000 010	1 100 000
" " 8.	0 000 000	1 000 010	1 100 000

1. Operating Procedures

2. Command Analysis

g. Initial Input and Input

Following are the steps which take place in Initial Input (0) and Input (1):

1. a) $\delta \rightarrow 0$. (0 register determines input device.)

   b) Cl(A,R), Cl(M)

2. a) Computer halts.

   b) Input Device energized.

   c) Information in input device transferred into R register. (Either automatically as from card reader equipment or magnetic drum, or manually as from keyboard.)

   d) When all information is in R register, input device is de-energized, and computer starts again. (For Drum input, device de-energized after 32 words are transferred.)

3. (R) → (M)

4. - - - - - - - - - -

If 0 (Initial Input)	If 1 (Machine already in operation)
5. (M) → ($\mathcal{E}$)	5. (M) → (α)
6. $\mathcal{E} + 1 \rightarrow \mathcal{E}$, Cl(M)	
7. Cl(M), ($\mathcal{E}$) → (M)	
8. Cl(C), (M) → (C), Cl(M), Cl(0)	

1. Operating Procedures

2. Command Analysis

h. Output

Following are the steps which take place in Output (2):

1. a) $\mathcal{S} \rightarrow 0$. (0 register determines output device.)

   b) Cl(M), Cl(A,R), $(\mathcal{A}) \rightarrow$ (M)

2. (M) $\rightarrow$ (R)

3. Cl(M), $(\mathcal{B}) \rightarrow$ (M)

4. a) Computer halts.

   b) Output device energized.

   c) One word in R register automatically transferred to output device. (On punch card output, one word each in M and R registers transferred to card punch.)

   d) Output device de-energized, and computer starts again. (On Drum output, de-energized after 32 words have been transferred.)

5. - - - - - - - - - - -

6. $\mathcal{E} + 1 \rightarrow \mathcal{E}$, Cl(M)

7. Cl(M), $(\mathcal{E}) \rightarrow$ (M)

8. Cl(C), (M) $\rightarrow$ (C), Cl(M), Cl(0)

1. Operating Procedures

3. Special Operations

a. Breakpoint Digit

Use of the Breakpoint Digit will enable SWAC to halt in routines wherever desired without the use of a command to produce the halt. The Breakpoint Digit may be considered a part of the function (F) of a command and is stored in $\beta 5$ CRT. A Breakpoint Digit will cause the machine to halt after the command has been read out of the memory, but before the command is executed. The halt will occur during Period 8. Therefore, the machine will halt with the Period 1 light on, the command in the C register, and the Breakpoint neon (located on the console) indicating the presence of this digit in the command. To proceed in the routine the operator must press OOPB.

The Breakpoint Digit must not be used with input or output commands which use IBM equipment nor with commands obeyed during a card cycle.

Input to the Breakpoint CRT can be had from the collator by using column 79 of IBM cards.

Insertion of a Breakpoint Digit into an address is done by reading into the machine and obeying the following command:

α	β	γ	δ	F
[ ]	000	000	096	01

1. Operating Procedures

Removal of a Breakpoint Digit from an address is done by reading into the machine and obeying the following command:

α	β	γ	δ	F
[ ]	000	000	096	02

where,

α holds the address
β and γ are not used
δ specifies Breakpoint (096) type input or output
F specifies the command (input [insert] or output [remove]).

For the above, read-in is performed via the keyboard.

A switch on the console marked "Breakpoint" controls the use of this extra digit in the command. When the Breakpoint Digit is to be used the switch must be in the center or ON position. When the switch is down in the OFF position the use of the Breakpoint Digit is inhibited and read-in from the collator to the Breakpoint Digit is inhibited, but insertion or removal of a Breakpoint Digit via the keyboard is not affected. When this switch is pushed up to its momentary contact position marked "Clear" the Breakpoint CRT will clear. This CRT unit will also clear when the rest of the memory is cleared.

Breakpoint Digits will halt the machine only if they are part of a command. If a command containing a Breakpoint Digit is acted upon as a number (as in modifying commands) the machine will not halt. If a number (not a command) contains a Breakpoint Digit and is mistaken for a command the machine will halt. Although the Breakpoint Digit is stored in $\beta 5$, it does not affect and is not affected by arithmetic operations.

A Breakpoint output command which is placed in a routine will also act as a halt for the computer if the Breakpoint switch is in the ON

1. Operating Procedures

position and will display what is held in addresses α and β. If the switch is in the OFF position the machine will not halt. The command will be obeyed as far as removal of Breakpoint Digits in address α is concerned regardless of the position of the Breakpoint switch.

There is no provision for storing Breakpoint Digits on the magnetic drum.

1. Operating Procedures

3. Special Operations

b. Delta to Epsilon

Any time that either a column 4 punch is read from a card or the $\delta \rightarrow \mathcal{E}$ key on the keyboard is operated, there will be both a "clear epsilon" [cl($\mathcal{E}$)] and a "delta to epsilon transfer" ($\delta \rightarrow \mathcal{E}$). This is independent of whatever command is being obeyed at the time.

If the above takes place in Initial Input (0) this will clear during Period (Step) 3 and δ (presumable zero) will be transferred; then in Period 6, $\mathcal{E}$ will be increased by unity ($\mathcal{E} + 1 \rightarrow \mathcal{E}$). Thus, the next command will come from address "one".

Therefore, while operating on "command" the $\delta \rightarrow \mathcal{E}$ key on the input keyboard can be used (a) to simulate some of the effects of overflow or (b) to force a "compare" to go to δ.

1. Operating Procedures

4. General Drum Operation

A complete description of the drum and how it operates is given in the section on Magnetic Drum. Testing the Drum is covered in the section on Testing. What follows covers a few additional details for the operator.

After the machine is tested in the morning, the drum DC is turned on. This switching is delayed so that the operator will not have to turn drum DC on and off repeatedly when SWAC DC is turned on and off during adjustments. The drum test is made after all other parts of the machine have been tested to the operator's satisfaction. When drum DC is turned on, the reset button inside the right-hand panel must be pushed momentarily to reset the address counter of the Drum. Note that turning on drum DC will set the S register arbitrarily. Therefore, the machine should not contain a routine at the time the drum DC is turned on.

The drum AC is not controlled on the console. The drum power cabinet is located in the Laboratory, next to the high voltage supply cabinet. Drum AC must be turned on independently when SWAC AC is turned on.

A delay light on the power cabinet indicates when the filaments in the drum circuits have warmed up sufficiently. (See Section on Power Control.)

The pilot light on the console will tell the operator whether the drum DC is on. If a drum read or write command is given with drum DC off, SWAC will halt. This command cannot be obeyed at this stage merely by pressing DC ON, due to the setting of the S register. It will be necessary to read the routine in again.

1. Operating Procedures

At the present writing the Drum operation is only tolerably satisfactory. Engineering modifications will continue to be made to improve it. After the Drum passes its daily test in the morning, secular changes will occur in the Drum itself and its reading heads, and it may be necessary at any time to test it again. The operator must note the performance of the machine with various routines and decide when any component needs testing. Don't make snap judgements! Be sure a part of the machine needs testing by observing the routines which, in themselves, often constitute the best test available.

Drum commands are not like other input or output commands. The drum takes control of the operation, and 32 words are transferred to or from the high speed memory, after which control is restored to the SWAC. Thus a drum command can be obeyed on COM, or PER, as well as CONT.

If on COM, the SWAC halts in period 7 after the transfer. On PER, SWAC halts in period 6 after an output and the last word read out is displayed in M. Due to the method of transfer this may be any one of the 32 words. On input the halt will be in period 4.

There is no such thing as initial input for a drum command. In other words, whether F = 00, or 01, the drum transfer command will be obeyed as though F = 01. Input will be to the group of cells specified by the α and β of the command.

1. Operating Procedures

5. Initial Read In

To prepare SWAC for initial read-in of numbers or commands the memory must be cleared and the epsilon counter must be set to zero. Clearing the memory will set the period light to 7. Pushing OOPB once will set period 3 light. Note that it is very easy to push OOPB more than once, therefore never fail to make a visual check that the epsilon counter is actually zero. The mode-of-operation switch must be set to continuous, (CONT) (or to COM, with the delay switch ON). (All neons in the M, A, and R registers, and the C register, should be out.) SWAC is now ready to receive information from keyboard or collator.

While the memory of the SWAC is ostensibly empty at this stage, the zeros in the C register, representing the command in the cell: $\mathcal{E}$ = 000, are being interpreted by the machine as a command to input information into cell 000. Therefore if the R register receives some information, and a "Start Function Table" pulse is supplied (by OOPB, or word pulse on the card, or stop signal on the tape), the word in R will go into cell 000 in the CRT's. The R register will be cleared. In period 6, $\mathcal{E}$ will be increased by 1, and in period 7 a new command, from cell 001, will be drawn into the C register from the CRT's. This command is, like its predecessor, all zeros. It is interpreted as a command to input information (via the R register) to cell 001. This process is repeated until the memory is filled. If now, epsilon is cleared either by carry-over from its highest value (255), by a clear epsilon signal from card (or tape), or manually, then the command in cell 000 of the CRT's, now no longer zeros, will be set up in the C register and obeyed.

1. Operating Procedures

6. Collator Input

Assuming SWAC is cleared as described in the preceding section, the operator may feed a routine on cards through the collator. The secondary feed of the collator is used at the present writing, but the operation of the primary feed is much the same.

The cards are "joggled" and smoothed, and the deck placed face down, nines edge (bottom of the card) leading, in the hopper of the secondary feed. Two blank cards (three in the primary) should follow the punched deck. The start button (right-hand side) is depressed for one card cycle, after which it may be released, and the cards will continue feeding automatically, as long as SWAC calls for input.

If the routine is arranged so that computing starts immediately after read in, the collator will stop feeding cards, but its drive shaft will continue to idle. This may be halted by depressing the stop button (next to start button) at any time during the computing. With this arrangement, one blank card remains in the hopper, and one card of the routine is still in the feed rolls of the collator, although it has passed the reading brushes. At some later stage, when SWAC is again clear the start button may be held down to feed this last blank in, and retrieve the last routine card.

When data are being fed through the collator, and the routine breaks down for any reason, the cards remaining in the collator feed may be removed by putting two blank cards in the hopper, and, with SWAC memory cleared, holding down the start button until the blanks are drawn into the feed rolls.

1. Operating Procedures

If the collator misses a feed, the card remaining in the rolls will be read, but no new card will be pulled in from the hopper. In this event, repair the bent (bottom) card in the hopper, and continue operation by pushing the run-out button. No error will occur in operation because of the gap in feed.

When data are being processed, the collator should be left on "idle," so that when the SWAC calls for input, cards will be fed. Unlike the punch, the collator motor is not started by the SWAC. If SWAC calls for input and the collator is not idling, the start button must be pushed to begin the feeding operation.

A word of caution is in order for those operators unfamiliar with IBM equipment. "Joggle" all cards carefully to smooth the edges of the decks, and make sure no out-of-line cards get into the hopper. A jam will result if this precaution is not observed.

1. Operating Procedures

7. Punch Output

To use the punch, the operator fills the punch hopper with blank cards, fed face down and 9 edge first. Note that this is contrary to usual IBM practice which specifies 12 edge first. With the summary punch switch DOWN, the operator holds the start button down long enough to feed two cards. The punch must be halted by depressing the stop button. The summary switch may be raised at this time. A master card for gang punching may precede the deck. A check should be made that the proper board is in the punch, and that any necessary emitter and gang punch wires have been added.

If the punch feed has been filled as described, the SWAC will initiate a punching cycle when it obeys a card-punch output command. The punch motor does not idle, as the collator motor does.

If reproducing is to be done during summary punching, the necessary wires must be inserted in the board, and the master deck fed into the read side of the reproducer in phase with the punch side. If a master gang punch card is used on the punch side, a blank should be put in front of the reproducing deck on the read side.

For a detailed description of the punch unit and the method by which punching is accomplished, see the Output Devices section.

During the running of a routine, the stacker of the punch should be emptied periodically. This can be done between cards feeds, when the machine is not punching. After a problem has been run, push the S.P. toggle switch down, and press the start button on the punch. After two card feed cycles stop the punch. This will feed the last two answer cards into the stacker. If it is desired to clear the punch feed entirely, lift the blank cards from the hopper, and hold the start button down for three cycles.

1. Operating Procedures

If it is desired to stop the SWAC while punching, either the punch or the SWAC must be halted. If the routine is not wanted, clear the memory. The punch will halt at the end of a card feed cycle. If the routine should be needed, halt the punch by pushing the stop button with one hand, and throwing the S.P. switch down at the same time with the other hand. SWAC will halt at the next output command, and the punch will halt at the end of the card feed cycle. The last card punched will be incorrect.

If a card fails to feed in the punch, SWAC will halt at the next punch out command. Confirm the fact that it is a feed failure by lifting out the deck of blanks from the hopper, and look at the throat of the punch. Normally a card should be partially in the feed. If this is not the case, then there has been a misfeed. Discard the two cards at the bottom of the blank deck. Push down the S.P. toggle switch, and clear out the card feed by holding down the start button. Load the hopper with cards, putting a gang punch master in front of the deck, if one was used originally. Feed two cards into the punch. Now lift the S.P. toggle switch, and the punch should start, obeying the SWAC punchout instruction.

Be sure that the above sequence of operations is followed explicitly, or there will be a good chance that punched information will be lost, or that the routine in the SWAC will be spoiled by word pulses fed back from the punch.

This operation is more complicated when the reproducing feature is being used; the additional precaution is to see that the pre-punched deck on the read side of the punch is in phase with the cards on the punch side.

1. Operating Procedures

The warning given under collator operation holds here also: "Joggle" cards well, and be sure there are no nicked edges in the deck. In addition, remember that the holes punched into cards create chips. Clean the chip hopper periodically, and preferably before the chips start spilling out.

1. Operating Procedures

8. Typewriter Output

When SWAC obeys a typewriter output command, it halts with period 5 light on, and energizes the typewriter. A typewriter ring consisting of a sequence of flipflops "counts" the characters typed out, according to the wiring of the patch panel. At the end of the cycle, a "stop" wired from the panel halts the typing, and supplies a "start function table" pulse which "tells" the SWAC to do period 5. If on CONT, the SWAC proceeds to the next command. If on COM, SWAC will halt in period 7. If on PER, the halt will be in period 6.

The mode-of-operation switch may be turned from CONT to COM or PER during a typeout. SWAC will halt when the typeout is complete. When the machine is being operated on COM, the operator may call a typeout command into the C register. If this happens, then time must be allowed for typing to be completed before OOPB is pushed again. Pushing OOPB too soon causes the R register to clear and part, or all, of the information typed out may be lost.

The digits being typed out may be seen in R as they shift to the right. (This is on normal output.) The M register meanwhile displays the contents of the address in α. If SWAC is on PER at the completion of typeout, this number is still displayed in M. The R register will be clear. On COM, both M and R registers will be clear after the typeout.

On converted output, the M register holds the multiplier which is equal to $\frac{1}{2}$ the base to which the output is being converted. The multiplication, which takes place on the remainder in the R register between successive digital typeouts, makes the number in R seem to shift to the left. At the same time, the last four binary digits of the A register flash as each

1. Operating Procedures

digit of the typeout is emitted by SWAC.

On single character output: 40 (base 16) or 064 (decimal) in $\mathcal{S}$, the typewriter ring is not energized. The single character will be emitted from the last five digits of R, on normal output. Converted output will not be used deliberately for single characters.

The letters-numbers switch is used for 20 or 40 output, and may be set beforehand to one or the other of its settings. No choice may be made in the coding between letters or numbers. In general, the switch is set on letters, and numbers are typed out on the 00 code.

The operator should see that the typewriter has paper in the rolls, and that the carriage is set properly for the format desired. Single, and double, spacings are available. Tab stops may be inserted at various places to vary the format, which is normally set at six words per line. The left margin stop may be moved in as much as three inches. The right margin stop must never be moved, as it controls the automatic carriage return.

The typewriter uses a two-fold perforated edge paper. The duplicate copy is available to the person running a problem, for his files. The paper is fed by pins on the ends of the platen, not by pressure rollers. Therefore, see that the pressure roll release at the left side of the carriage is always pulled toward the front of the typewriter in the fully released position. Conflict between the pressure rolls and the pin feed leads to torn perforations and, ultimately, jammed paper.

Instructions for keeping the log of the SWAC are posted on the bulletin board, and are included elsewhere in this manual.

1. Operating Procedures

1. General Procedures

a. Summary

These general operating procedures are issued in order to systematize the many conflicting operations which must be done on the SWAC.

The reader must realize that several different types of operations, and many groups of individuals, require time on the machine. Some of the operations are: (1) Research problems for NAR staff members, (2) Research problems for UCLA staff members, (3) Machine maintenance and testing, and (4) Engineering modifications.

A committee has been set up to serve as a clearing house for schedules, priorities, and general administrative details of SWAC operation.

The following operating procedures are issued by the committee to serve as a general guide. They are subject to alteration by the SWAC Supervisor.

1. All departments using SWAC are expected to do their own coding and operating. Instruction in both is available from NAR.
2. A qualified operator must be present in the SWAC console room at all times when the machine is in operation, except when the engineers have taken charge of the machine. A list of individuals qualified to operate SWAC is posted on the SWAC bulletin board. Anyone may learn to operate SWAC and qualify for operation. To qualify for operator a person must be familiar with the console panels of SWAC, must know how to

1. Operating Procedures

turn the machine on and off, must be able to test the machine, and must be able to operate the IBM equipment attached to it. See the SWAC Supervisor for details. Under no circumstances is SWAC to be operated by an unauthorized operator without specific permission from the SWAC Supervisor.

3. Requests for SWAC time should be made on forms posted on the SWAC bulletin board. Any individual whose codes are ready to be checked or run will sign the list and give dates and times when he will be available, problem number, and estimated time required to check the code or compute. Estimates of time wanted are to be made in units of tenth hours. Time will be assigned according to the order in which requests appear on these sheets, subject to priorities and previous commitments of time. Backlog problems are usually available to fill in unused or unwanted time, but each coder should make an effort to estimate accurately the time he needs. The SWAC has a scheduled warm-up time of $\frac{1}{2}$ hour, and testing time of 1 hour. Engineering usually takes $1\frac{1}{2}$ hours. This leaves 13 hours of each 16 hour double shift for computing.

4. The SWAC Supervisor will make a daily schedule not more than 24 hours in advance from the posted lists of requests for SWAC time. In the event of breakdown, the schedules may be shifted or rearranged. Each operator is expected to be available when SWAC is ready for his problem.

1. Operating Procedures

5. The scheduled operation of SWAC covers the period from 0800 to 1700 Monday through Friday. Operation outside this period is on a voluntary basis, but a schedule of nominal assignments of swing shift time is posted once a week on the bulletin board. Changes in this schedule may be made by informal agreements among the qualified operators. In general it will be advisable for at least two operators to share a swing or graveyard shift and relieve each other occasionally.

6. University rules of safety require the presence of two people in any building where there is electrical equipment. Therefore, during swing and graveyard shifts, at least two persons, one of whom must be an operator, must be present in the SWAC room, electrical laboratory, or the IBM room at all times when SWAC is in operation. It is not sufficient that the second person be in the library, or his office. Unless he is within close call of the SWAC operator the whole purpose of the rules is defeated. If an operator is unable to be present at the console, and an alternate operator is not available, the machine must be turned off.

7. It is the responsibility of swing and graveyard shift operators to lock up securely when they are through. See that all power supplies, irons, test equipment, and air conditioners are off, that the stockroom door is locked and the key in its proper place in the SWAC console, and that the three entrances to the

1. Operating Procedures

SWAC rooms are secured. Include a checkup to see that IBM equipment is turned off, including the hand punch at the SWAC console, and the one in the laboratory. Two check lists are posted, one on the wall next to the SWAC bulletin board, and one over the telephone in the laboratory. The person locking up is requested to initial both these sheets after inspecting the items listed.

8. If routine maintenance is performed during the swing shift, one person in addition to the engineer must remain near SWAC during the maintenance period. Even though the machine is off, the test equipment is not.

9. If breakdowns occur after 1700, or on weekends, and one of the engineers is not available, the machine should be turned off. Operators should not attempt to do any engineering. Note on the log the symptoms of any failures as an aid to diagnosis on the following day.

10. Space around the SWAC console is limited. This area should be occupied only by those concerned with the problem at hand. Staff members are requested to keep this space clear so those operating SWAC can have ample room. The operator on duty has the right to enforce this rule. Operators will make necessary exceptions to this rule when visitors are present.

11. Operation of the IBM equipment not directly connected with SWAC is under the same rules as those stated here, wherever applicable. Rules 2, 6, and 7, in particular, must be followed.

1. Operating Procedures

1. General Procedures

b. SWAC Log

1. A continuous record of all operations on SWAC must be maintained on the output typewriter. This record is used to make up cost accounts as well as statistical breakdowns of SWAC time. The following paragraphs are intended as a guide to enable operators to keep the log in a systematic and accurate manner:

2. The categories of time which may be recorded on the SWAC log have been reduced in number from twelve to ten::

A. ROUTINE MAINTENANCE: Maintenance of chassis by the engineers.

B. OUT OF ORDER: Machine not functioning, and engineers in charge.

C. IDLE: Power on, but machine not being operated. Operator has abandoned the machine and has been unsuccessful in finding a user.

D. TESTING: All component or circuit testing.

E. IMPROVING ENGINEERING FACILITIES: Modification, change, or experiment on any equipment. Work on collator, punch, or tabulator comes under this category. Engineers are in charge.

F. IMPROVING MATHEMATICAL FACILITIES: Code checking or testing of routines of a general nature not specifically intended for any particular problem. This includes sub-routines, conversion routines, and test routines. Computing Committee is in charge.

1. Operating Procedures

G. DEMONSTRATION: Exhibition of machine to any visitors whenever this involves the use of the machine to the exclusion of other work. Such demonstrations must be arranged in advance with the Chairman of the Computing Committee.

H. CODE CHECKING: Problems whose codes are being checked out.

I. COMPUTING: Problems in which results are being produced, or conversions of codes or results.

J. OFF: Power off.

3. The time of beginning of each operation is written as a four digit number: 0000 to 2400, i.e., on a 24-hour basis. This time is recorded to the nearest multiple of six minutes. The clock in the SWAC room is marked off at six minute intervals to assist in correct recording of the time.

4. The record for each entry is to be written with the time of beginning at the left margin. On the same line, in capital letters, type one of the nine categories listed in 2. above. On the line beneath list the problem number and initials or name of the operator. A sample of the log might look as follows:

0955 COMPUTING
809-00 (K. N. Trueblood)

1137 CODE CHECKING
6500 (Wilson)

1225 IMPROVING ENGINEERING FACILITIES
(F.H.H.)

(Engineer will note type of work done.)

1. Operating procedures

1400 (F.H.H.) TESTING

00100 (Name of test, followed by SWAC typeouts, if any.)

5. The SWAC day on the log begins at 0800. Please start a new page at 0800. This is particularly important if the machine is run through the graveyard shift.

6. The log is typed in duplicate. The carbon copy may be removed after typing and filed by the operator, if desired. A new page may be started for any person wishing it. Always leave the original copy on the typewriter.

7. Details of all difficulties encountered, and any evidences of erratic behavior, should be noted on the log, together with the time of the occurrence. Do not be too brief! The more information the log contains, the more useful it will be. Please be accurate in noting problem and test routine numbers, and the time.

8. If the machine is suspected of malfunction during CODE CHECKING or COMPUTING the operator must call an engineer if one is available. In the absence of an engineer, the burden of proof as to malfunction of SWAC is on the operator and he may not change his category of COMPUTING or CODE CHECKING to any other except IDLE. If he chooses to test the machine, and successfully traces erratic behavior, then his time will be charged to the appropriate category when the log is made up. If he does not wish to run the risk of finding no trouble, then he should turn the machine over to some one else or leave it idle.

1. Operating procedures

9. In no case is an operator to attempt to repair the machine without the advice of an engineer. This is particularly applicable to all graveyard and weekend operation. The risk of damage to the machine or the operator is too great to make the gain of a few hours computing time worth while, even supposing that a repair could be made. This prohibition does not extend to clearing jammed cards from the IBM equipment. It does specifically prevent an operator from replacing any component or chassis inside the SWAC, the Drum, or the Power Cabinets.

2. Testing

1. General Discussion

a. Memory

The cathode ray tubes of the memory store information in the form of dots (zeros) or dashes (ones) on the face of each tube. Each spot is actually a charged area.

A defect on the face of a tube may prevent the storage of information; this is known as a "flaw". If any one spot on the face of a tube is referred to repeatedly over a short time interval, the charge which builds up will tend to "spillover" and create a "one", or dash, in an adjacent "zero", or dot, position. Regeneration circuits are continually operating in the memory to maintain the status quo of information. Spillover trouble occurs if too many references are made to a spot between successive regenerations of the spots surrounding it.

It is necessary to test the memory for these two major types of fault. Suitable tests to determine the condition of the memory have been coded, and modifications are occasionally made in the direction of stronger tests. A list of the tests which SWAC must pass is issued whenever such a change is made.

In general, a "flaw" test (such as 0000.3: R Z/2) will be alternated with a spillover test (such as 0000.8: SO R) until both tests run without failure. The origin switch is alternated for each test, so that all parts of the memory will be checked. Each test includes an automatic transfer of the routine to cells 8 rows away, and the SWAC will

2. Testing

halt in a breakpoint after such a transfer. (Breakpoint switch should be ON for all memory and arithmetic tests.) The origin switch can be thrown at this point, and the OOPB button pressed. The SWAC will execute the routine in the new location, and transfer it back, at the completion of a run. Assuming the run was successful, the SWAC is now ready for another routine. The alternation of storage and spillover routines is necessary because adjustments of the CRT's on a spillover test tend to be just the opposite of the adjustments made on a storage test. All adjustments of CRT's are to be made by the engineer on duty, and not by the operator.

In general, failure to store in a particular spot of a specific CRT is a flaw on the face of the tube. Failure to store on a specific CRT with no particular spot occurring more than any other may be due to a low intensity signal. Spillover is usually caused by too high an intensity, or out-of-focus spots. See the section on trouble-shooting for further causes of failure in the memory of SWAC.

2. Testing

1. General Discussion

b. Arithmetic

After the memory is checked, and adjusted to good operating condition, the operator proceeds with arithmetic tests. These fall into three categories: 1) general, 2) diagnostic, and 3) specific operation. Unfortunately, no general test can ever check all parts of the SWAC completely. It will always be possible to code a routine which will cause SWAC to fail, even though it passes the test routines. On the other hand, a failure of a general test does not always mean that the machine is incapable of computing correctly. Any rules laid down concerning machine performance should take this into account.

The general test for SWAC (such as 00100) is a routine of all the operations SWAC can do, arranged so that a reasonably large array of numbers is included in each operation. If successfully passed, the SWAC will type "cest fini". If other type-outs occur, the operator will proceed with appropriate diagnostic tests in an attempt to trace down the failures. A full description of the diagnostic extract (00101), product (00102), add (00103) and subtract (00104) tests is available in the desk of the operating console.

If the failure occurs on a specific number or set of numbers, then one of the tests which repeats this number in a specific operation should be tried. At this stage the operator will have called the engineer and should follow his suggestions so that the trouble may be traced with an oscilloscope.

2. Testing

It must be stressed that successful checking out of the SWAC can best be done by an operator who is familiar with recent behavior of the machine during tests. A small portion of a test routine may fail persistently for no apparent cause. Considerable time could be wasted if this failure were traced to its source. If SWAC gives no trouble when running problem routines, then it is sometimes better to tolerate the failure of the test, even coding around it, if necessary, until more evidence builds up as to the source of trouble.

2. Testing

1. General Discussion

c. Drum

Since the Drum is an auxiliary memory device, it is logical that the principle failures to be expected would be pickups or losses on transfer of information. Therefore the drum test consists of a routine which writes a specified word or sequence of words on all channels of the Drum, and reads this information back, to compare it with the original material. Any failures are displayed in a breakpoint command which tells the operator which of the channels is at fault. (The breakpoint switch must be ON.) In the present state of development of the Drum, sporadic failures will occur with each running of the test. The operator will have to use judgement in deciding how much failure to tolerate. Here, experience is the best guide, and a knowledge of recent behavior of the SWAC and Drum is a valuable aid.

If any channel shows persistent failures during the test, an engineer can "clean" the channel electrically, after which the test should be repeated.

2. Testing

1. General Discussion

d. Punch

The Card Punch of the SWAC is an IBM Reproducer, model 513, whose cable connection to the SWAC is made through a Punch Unit, which serves to transmit signals from the R and M registers to the punch magnets of the 513 reproducer. Both units are tested simultaneously.

As in summary punching with any IBM equipment, loss of information is the primary cause of failure, and spurious (extra) information is of secondary importance. The tests coded for the punch call for the output of a fixed word onto punched cards, and a visual check of the cards by the operator will serve to identify erroneous digits, if any. Tests punching out one word per card, and one to ten words per card, are provided. The appropriate board, for 36-digit or 72-digit output, must be chosen.

If failures occur they may be caused by the 513 Reproducer, or the Punch Unit. The operator will have to determine which device is failing. If one column on the cards is persistently blank, for example, a loose wire on the board or a bent prong on the sub panel behind the board would be the first things to look for on the 513. Next the thyratron in the Punch Unit corresponding to the card column should be replaced, or interchanged with another. If this does not help matters, then an oscilloscope should be used to trace the signal, in case the failure is due to a break somewhere in the cables.

2. Testing

If a routine goes into a loop, types out erratically, or otherwise indicates failure, either during or after a punch cycle, check the number of jack plugs in the output board. If these are not consistent with the number of output commands coded in the routine, trouble can be expected. Note that due to the asynchronism of Punch and SWAC, a repeat run of the routine need not show the same failures as the first run. This has led to many false conclusions as to the source of the trouble.

2. Testing

1. General Discussion

e. Collator

The Collator input device requires very little attention.

Failure to read correctly has in most cases been traced to dirt and dust accumulated on the feed rolls and brushes. The test for the Collator consists of a set of 256 words, each equal in value to the cell into which it will be read. This value is repeated in α, β, γ, and δ of each cell. Definite patterns are formed in the monitor scope by this test, and failure to read correctly is easily noted by breaks in the pattern. This test also serves in the event of C register trouble.

A stack of word pulse cards, fed through the Collator continuously, is used by the engineers when inspecting the shape of the word pulse on an oscilloscope.

To clean the Collator secondary feed, open the top cover, release the catch holding down the brush assembly and raise it up. Open the plugboard to remove DC voltage from the brush roll. Blow out the accumulated dust and "wool", with a vacuum cleaner or pressure blower. If the roll looks burnt (particularly around column 5, the W.P. column) it may be polished with very fine emery cloth. The roll may be rotated automatically by holding down the hopper card levers in both primary and secondary feeds, and the start button. Leave the plugboard OPEN.

2. Testing

2. Summary of Test Routines

a. Current Test Routines for SWAC

I. MEMORY TESTS

00000	Memory R
00001	Memory RZ
00002	Spillover (85:1)
00003	Memory RZ/2
00004	Spillover (85:1 Double)
00006	Spillover (107:1)
00008	Spillover (107:1 Double)
00010	Read around

The SWAC is required to pass 00000, 00003, and 00008.

II. ARITHMETIC TESTS

00100	General Testing
00101	Diagnostic Extract
00102	Diagnostic Product
00103	Diagnostic Addition
00104	Diagnostic Subtract
00105	Subtract F
00106	Extract A
00107	Product F
00108	Add Pulse Test
00109	Extract Test
00110	Punch Test (1/card)
00111	Punch Test (1-10/card)
00112	Product ab
00113	Add ab
00114	Subtract ab
00115	Multiply ab
00116	Extract ab

The SWAC is required to pass 00100.

III. GENERAL TESTING: 00100

A. Test 00100 consists of three sections: 1) Space, 2) Extract, 3) Product, Multiply, and Subtract. Section 1 is completed first and then 2 and 3 are used in the cycle where the operand b: -Z's(11 11 11 11 1) +Z's is changed for each successful cycle. Operand a: -Z's(04 10 41 04 1) +Z's goes through its complete range for section 3 in each cycle of 2-3. At the end of the test, the SWAC types "cest fini" and halts in the origin.

2. Testing

B. Test 00100 ERROR typeouts and suggested tests for repeated failures.

1. Memory failure

1	2
"mem d is"	delta

Input 00100 again or return to memory adjustment.

2. Space test, i.e. $(-Z's + 2^{-k})\ 2^{-36}$

1	2	3	4
"space"	$(-Z's + 2^{-k})$	$(-Z's + 2^{-k})\ 2^{-36}$	delta

a. Bypass the space test section to check extract, multiply, product, and subtract. Clear epsilon and type into 01: 5w 50 30 80 b, and continue.

b. Add pulse test, 00108.

3. Extract

1	2	3
"ectract"	extractee	extractor

There are two extractors involved for this typeout; the first is indicated by typeout three, the other is the complement of typeout three.

a. Diagnostic Extract 00101
b. Extract A 00106
c. Extract Test 00109
d. Extract ab 00116
e. Product tests (see product)

4. Product and Multiply

1	2
"prod"	operand of beta
"mlt"	operand of beta

a. Diagnostic Product 00102
b. Product F 00107
c. Product ab 00112
d. Multiply ab 00115

5. Subtract

1	2
"sbt"	operand

6. a. Diagnostic Subtract 00104
b. Subtract F 00105
c. Subtract ab 00114
d. Diagnostic Add 00103
e. Add ab 00113

2. Testing

IV. Diagnostic Extract 00101

Typeouts:

SET 1

1	2	3
extractor	extractee	delta

SET 2

1	2	3	4	5	6
first	second	command	first	second	command

For a bona fide extract failure, there should be two sets of typeouts. Set 2 should follow Set 1, giving one or more groups of three typeouts which indicate the results of the first extract and then a latter check extract called the second. The third typeout of Set 2. With the typeouts of Set 1 and Set 2, the extract failure may be determined. There are 148 extracts as in 00100 for a given extractee. These are stored in the upper part of the memory and are summed to check for extract failures. If the Sum is incorrect, Set 1 typeouts occur giving the difference in typeout 3. Then each result is checked and for any discrepancy Set 2 is typed.

If there are repeated typeouts of Set 1 with no Set 2 typeouts, there are addition failures (rather than extract failures) which occur as the SWAC adds the 148 results of the extracts. Another possibility is a "solid" extract failure which occurs the same for each attempt. In either case test 00101 is non-useful. Use test 00106.

2. Testing

2. Summary of Test Routines

b. Use of Test Routines on SWAC

I. Memory Tests.

The following memory tests are in current use on SWAC:

1. Memory R: 0000.0
2. Memory $RZ/_2$: 0000.3
3. Spillover: 0000.8

When the machine has been turned on in the morning, Memory R is run through as a general check of the memory. This routine should also be used at any time that a quick memory check is desired. It is not necessary to use it during regular memory adjustments after the morning tests.

Further memory adjustment is accomplished by making complete runs of Spillover and Memory $RZ/_2$. Run each on both origins, and obtain good runs on both routines after any adjustment of CRT's or deflection amplifiers. A good run on Memory $RZ/_2$ contains no type-out on either origin. For Spillover the number of type-outs should, in general, not exceed two, and these should not be from the same cell or the same address. However, the operator may decide on the basis of his adjustments of the memory, and the general behavior of the machine, whether more type-outs constitute a passing test. Note particularly that all of these tests are run on continuous.

II. Arithmetic Tests.

The following arithmetic tests are in current use on the SWAC:

2. Testing

1. General Testing: 00100
2. Diagnostic Extract: 00101
3. Diagnostic Product: 00102
4. Diagnostic Add: 00103
5. Diagnostic Subtract: 00104

The several older tests, such as Extract A, Subtract F, and Product F, are available if the operator wishes to apply more tests to the machine. In general the five listed above will be all that are used.

In testing, run 00100 through the SWAC first. At the completion of the test, SWAC types "cest fini." This is a sufficient test of SWAC performance if no type-outs occur, but another run may be made on delay, if desired.

General test indicates five types of operational failure: gap test, extract, product, multiply, and subtract. If two or three typeouts of any type occur, another run may be made. If this run is successful, the machine is all right. There is a possibility that the indicated error is not of the type which actually occurred. For instance, since the extracts performed in 00100 are checked by summing, an add failure in this check will be indicated as an extract failure. Unfortunately, the diagnostic extract test 00101 is subject to the same difficulty. Therefore, caution must be used in interpreting type-outs on these tests, especially in regard to extract.

2. Testing

If the operator is reasonably sure that the indicated error has really occurred, then the diagnostic test for that type of failure is put through the SWAC. For either product or multiply failure, use the diagnostic product 00102; for subtract failure, use either diagnostic add or subtract. For gap test failure, use product test. This will indicate either no failure, or pinpoint the trouble. If a pair of numbers can be found which is failing, then one of the more simple routines (Product 1, Add 1, Subtract 1) may be used to allow the engineers to spot the trouble on an oscilloscope.

In the event that no error occurs on the diagnostic test chosen, the operator should run the remaining diagnostic tests, or try the general testing routine again.

It should be borne in mind that any change in the CRT or deflection amplifier sections necessitates a complete memory test, and any change in the product counter, A chassis or M chassis requires a run of General Test 00100. If a fuse blows, memory adjustment must be made when the D.C. is again turned on.

2. Testing

3. General Test Routine No. 00100

General Test 00100 is designed as a necessary and sufficient test of SWAC arithmetic performance. A perfect run has only one type-out, "cest fini", at the completion of the test. Other type-outs indicate types of operational failures: "space", extract, product, multiply, and subtract. "Add" is not included in this test, for this operation involves the same circuit components as the subtract operation. All references to commands and numbers are given in hexadecimal command form, since the information contained in this memorandum is given primarily to aid the SWAC operator.

General Description:

Test 00100 consists of three sections: 1) Space, 2) Extract, and 3) Product, Multiply, and Subtract. Section 1 is completed first and then Sections 2 and 3 are alternated in a cycle where the operand "b" is changed only after each successful cycle. [b: -Z's (11 11 11 11 1) +Z's]. [Z's: ZZ ZZ ZZ ZZ Z]. Operand "a" [a: -Z's (04 10 41 04 1) +Z's] goes through its complete range in Section 3 in each cycle of Sections 2 and 3. The test is designed so that one may change the increment of "b" (5v); [(5v): hexadecimal address 5v], but NOT the increment of "a". At the end of the test, SWAC types "cest fini" and halts in the origin. The test may be repeated by pushing OOPB (One Operation Push Button) twice.

2. Testing

Included in each Section 2 and 3 of 00100 is a "flicker" subroutine which consists of two commands: a 256 tally add command and a special multiply command of (-Z's) by (-Z's). The purpose of the flicker is to cause change of load on voltage supplies and thus bring out marginal failures. Flicker may be removed from Section 2 by changing (u0) from (33 43 33 u3 4) to (01 01 01 u3 7), and from Section 3 by changing (vy) from (33 41 33 uw 4) to (01 01 01 uw 7).

It must be remembered that this test is not diagnostic, and due to its complexity an indicated failure may be misleading, e.g., an indicated extract failure may be due to a subtract failure within the extract section.

Summary of Operations:

The following information summarizes the methods used to detect failures, the type-outs, and suggested tests for repeated failures.

1. Memory Sum Check.

The routine is checked by summing the memory at the start of the test and before each repetition of the test. The memory may be checked at any time by returning to the origin and going on to (02). Notice that any modification of the test will cause a memory type-out. The test may be continued by pushing 00PB after type-out. The negative memory sum in (62) may be adjusted for modifications of routine if the operator desires.

Type-outs:

1	2
"mem d is"	difference

Tests:

Input 00100 again or return to memory adjustment.

2. Testing

2. Space Test.

The space test consists of products $(-Z's + 2^{-k})(2^{-36})$ and "halts"; the halts are effected by flexowriter spaces. There are 512 products and 4 halts for each k. [k = 36(-1)1].

Type-outs:

1	2	3	4
"space"	$(-Z's + 2^{-k})$	$(-Z's + 2^{-k})(2^{-36})$ [least significant]	difference (3) - (2)

Tests:

1) Add Pulse Test 00108. Use 00108 in cooperation with an engineer.

2) See "Product and Multiply" tests.

3) To bypass the space test Section 1, clear epsilon, type into (01): 5w 50 30 80 b and continue.

3. Extract.

The extract section constitutes 90% of the test's running time, for a minimum (i.e., no extract failures) of 169, 756 extracts before the completion of the test.

Range of Variables:

Extractee "b": -Z's (11 11 11 11 1) +Z's

Extractor "q": 2^{-k}; k = 36(-1)1, ∞; $2^{-\infty} = 0$.

Extractor "p": Complement of q.

Shifts: All right and all left: 0(1)36.

For given values of b, q, and p all shifts are made and the 74 results using the q extractor and the 74 results for the p extractor are summed separately in Q and P, mod. 1 (i.e., overflow ignored). The check is $Q - P = b$, $b < 0$; and $Q + P = b$, $b \geq 0$. See SWAC Memorandum No. 7, "Diagnostic Extract 00101", for additional information.

2. Testing

Type-outs:

1	2	3
"ectract"	extractee "b"	extractor "q"

Notice that there are two extractors involved for error type-outs; the first is indicated by type-out (3), the other is the complement of type-out (3). Diagnostic extract type-outs (routine 00101) give the complete information on extract failures.

Tests:

1) Diagnostic Extract 00101
2) Extract A 00106
3) Extract Test 00109
4) Extract ab 00116
5) Extract ab (Flicker) 00128
6) Product and Multiply tests
7) Diagnostic Subtract 00104
8) Add Pulse Test 00108

4. Product and Multiply.

For a given cycle where "b" is held constant, "a" changes: -Z's (04 10 41 04 1) +Z's, a total of 127 products and 127 multiplications per cycle (31 cycles). The results of "producting" and "multiplying" (a)(b) are summed in six sums:

1) AB_p^+ in cell (28), the sum of the most significant part of product for $a \geq 0$.
2) ab^+ (2u), sum of least significant part of product for $a \geq 0$.
3) AB_m^+ (2v), sum of multiplication results for $a \geq 0$.
4) AB_p^- (29), $a < 0$.
5) ab^- (2v), $a < 0$.
6) AB_m^- (27), $a < 0$.

The carry cell (37) and round-off cell (36) are recorded to correct the above sums for checking against an independent calculation of the sum by

2. Testing

formula: $\sum (a)(b) = b \sum a = (AB, ab) \bmod. 1 = b(Z's)2^5$, where "a" ranges from 0(04 10 41 04 1) +Z's.

Type-outs:

1	2
"prod"	operand of beta "b"
"mtp"	operand of beta "b"

Tests:

1)	Diagnostic Product	00102
2)	Product F	00107
3)	Product ab	00112
4)	Multiply ab	00115
5)	Product ab (Flicker)	00124
6)	Multiply ab (Flicker)	00127

5. Subtract.

For each cycle "b" is held constant while "a" changes: -Z's (04 10 41 04 1) +Z's. The sum of the 127 subtractions $\sum (a - b)$ is = -127 b.

Type-outs:

1	2
"sbt"	operand of beta "b"

Tests:

1)	Diagnostic Subtract	00104
2)	Subtract F	00105
3)	Subtract ab	00114
4)	Diagnostic Add	00103
5)	Add ab	00113
6)	Add ab (Flicker)	00125
7)	Subtract ab (Flicker)	00126

6. Current SWAC Test Routines.

A table of current SWAC test routines is included for reference.

2. Testing

MEMORY TESTS

00000	Memory R	00010	Read-around (16 note)
00001	Memory RZ	00012	Square Spillover (107:1)
00002	Spillover (85:1)	00014	Square Spillover (85:1)
00003	Memory RZ/2	00016	Square Spillover (64:1)
00004	Spillover (85:1 Double)	00018	Square Spillover (43:1)
00006	Spillover (107:1)	00020	Spillover (160:1)
00008	Spillover (107:1 Double)		

ARITHMETIC TESTS

00100	General Test	00115	Multiply ab
00101	Diagnostic Extract	00116	Extract ab
00102	Diagnostic Product	00117	Deflection Test (Fast)
00103	Diagnostic Addition	00118	Deflection Test (Slow)
00104	Diagnostic Subtract	00119	Collator Input
00105	Subtract F	00120	C-Register (-Z's)
00106	Extract A	00121	C-Register (odd) δ and ε
00107	Product F	00122	C-Register (even) δ and ε
00108	Add Pulse Test	00123	C-Register δ and ε
00109	Extract Test	00124	Product ab (Flicker)
00110	Punch Test (1/card)	00125	Add ab (Flicker)
00111	Punch Test (1-10/card)	00126	Subtract ab (Flicker)
00112	Product ab	00127	Multiply ab (Flicker)
00113	Add ab	00128	Extract ab (Flicker)
00114	Subtract ab		

DRUM TESTS

00700	Diagnostic Drum
00701	General Drum
00702	Drum Clear

2. Testing

4. Diagnostic Extract Routine No. 00101

Diagnostic Extract Test 00101 is used to determine the extractor, extractee, and shift involved in an extract failure. This test is similar to the extract section of General Test 00100. A minimum (i.e., no failures) of 169,756 extracts are executed before the completion of the test.

All references to commands and numbers are given in hexadecimal command form, since the information contained in this section of the manual is given primarily for the SWAC operator.

Test 00101 consists of SWAC cards number 000 to 018 in four subsets: (1) Main Routine: cards 000 to 013, (2) Routine Sum Check: 011 to 013, (3) Restore Card: 014, and (4) Headings: 015 to 018. The usual card order is 015 - 018, 000 - 013 (omitting 14). Set (4) types out headings and leaves the memory clear with epsilon at (00) calling for set (1). The heading cards (4) may be omitted and the main routine (1) run in alone if the operator desires. The routine sum check (2) which is located in addresses (6y) to (86) is executed first after the main routine (1) goes in. These addresses are used for temporary cells; therefore, to run a routine sum check later in the test, the routine sum check may be restored by a return to (00) and input of sets (3) and (2) in this order, i.e., cards 014, 011 - 013.

2. Testing

Range of Variables:

Extractee "b": -Z's(11 11 11 11 1) +Z's. [Z's = ZZ ZZ ZZ ZZ Z]

Extractor "q": 2^{-k}; k = 36(-1)1, ∞; $2^{-\infty} = 0$.

Extractor "p": Complement of q.

Shifts: All right and all left: 0(1)36.

For given values of b, q, and p all shifts are made and the 148 results are stored in the upper part of the memory. Then the 74 results using the q extractor and the 74 results using the p extractor are summed separately in Q and P, mod. 1 (i.e., overflow ignored). The check is $Q - P = b$, $b < 0$; and $Q + P = b$, $b \geq 0$.

Type-outs:

For an extract failure there are two sets of type-outs:

SET 1

1	2	3
extractor "q"	extractee "b"	difference [(Q ± P) - b]

SET 2

1	2	3	4	5	6
first result	second result	command	first result	second result	command

The type-outs of Set 1 indicate extractor "q", extractee "b", and the difference between the sum of the results and "b". After this Set of type-outs, each extract is repeated and this result, called the "second", is checked against the "first" result. The type-outs of Set 2 are in groups of three giving the first and second results and the extract "command" whose result (the second) disagreed with the first result. The extract command from cell (33), (3u), (41), or (48) reads: [6u 69 15 n e] or [6v 69 15 n e]. The alpha of the command indicates the extractor involved; (6u): "q"; (6v): "p"; and the "n" indicates the shift involved.

2. Testing

If there are repeated type-outs of Set 1 with no Set 2 type-outs between, there are addition failures (rather than extract failures) which occur as the SWAC adds the 148 results of the extracts. In order to avoid the addition failures, a modified Test 00101 may be used where command (03) reads [63 64 69 00 4] instead of [63 02 69 00 6]. This causes the extractee "b" to range: -Y's(11 11 11 11 1) +Z's. If the addition failure situation still persists, clear (69) and enter the routine at command (04). This causes the extractee "b" in (69) to range: 0(11 11 11 11 1) +Z's, and avoids summing extract results of many binary one digits at the beginning of the test. If the addition failures continue to occur, use the Subtract and Add tests recommended in SWAC Memorandum No. 3. A list of extract tests is given elsewhere in this manual.

2. Testing

5. Memory Test Series 00000 to 00099

General Description:

Memory Test Series 00000 to 00099 consists of tests which check the storage and spillover characteristics of the SWAC Williams' tube memory. The tests serve to indicate which CRT units require adjustment or replacement. Since the memory tests make use of the arithmetic and control sections, a failure in one of these may be misinterpreted as a memory failure. In each test there is a shift subroutine which duplicates each address (n) in (n + 128). After the test is duplicated the routine halts in a breakpoint command so that the operator may "shift the origin." This shift subroutine allows the operator to check both "halves" of the memory without reading the test in again. Current memory tests are listed and classified as follows:

I. Memory (which test storage characteristics only)

A. Non-Flicker	B. Flicker
00000 Memory R	00001 Memory RZ
	00003 Memory RZ/2

II. Spillover (which primarily tests spillover characteristics, and secondarily tests storage characteristics)

A. "ZERO" Spillover (i.e. spillover in an electrostatic region of zeros)

2. Testing

1. Single Spillover

00002 Spillover (85:1 S)
00006 Spillover (107:1 S)
00020 Spillover (160:1 S)

2. Double Spillover

00004 Spillover (85:1 D)
00008 Spillover (107:1 D)

B. "ONE" (or Square) Spillover (spillover in an electrostatic memory predominantly of "ones")

00012 Square Spillover (107:1)
00014 Square Spillover (85:1)
00016 Square Spillover (64:1)
00018 Square Spillover (43:1)
00022 Square Spillover (160:1)

C. Read-around

00010 Read-around (16 note)

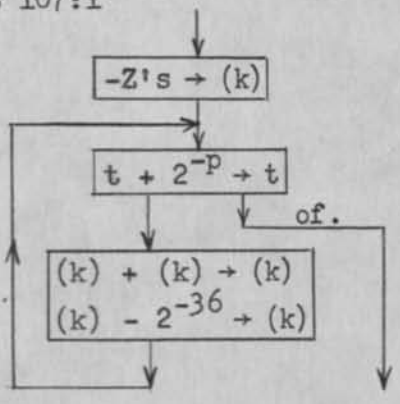
Subroutine Flow Diagrams:

The following flow diagrams are basic subroutines used in memory and spillover tests to diagnose storage and spillover properties. A list of definitions is given for the interpretation of the flow diagrams.

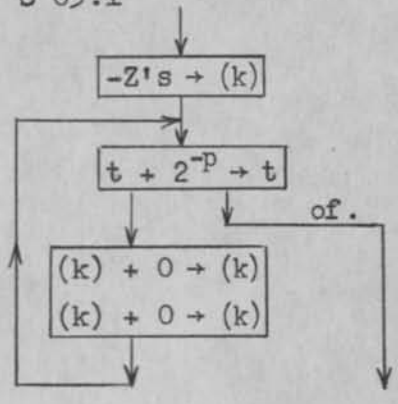
(k):	reference address of the test
(m) and (n):	temporary storage addresses
dotted line:	indicates intermediate commands
t:	tally
2^{-p}, 2^{-q}:	tally increments
-Z's:	-ZZ ZZ ZZ ZZ Z
-Z's → (k):	-Z's goes to address (k).
(k) →:	address (k) output by Flexowriter or by breakpoint command
S:	spillover ratio, the number of references to address (k) per one complete regeneration of the memory. S = 64s/a; where s: number of references to address (k) per subroutine loop; and a: number of add times per loop.
R:	read-around ratio, the number of references to addresses in a 3 by 3 memory region per one complete regeneration of the memory. R = 64r/a; where r: number of references to the 3 by 3 region per subroutine loop, and a: is the number of add times per loop.
of:	overflow

2. Testing

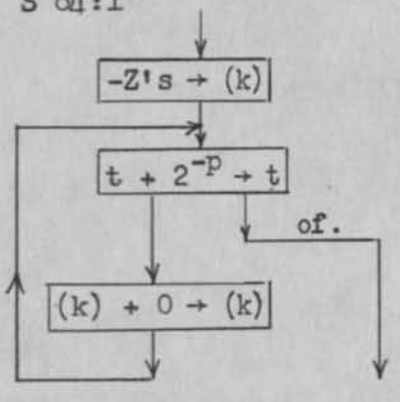
I. S 107:1



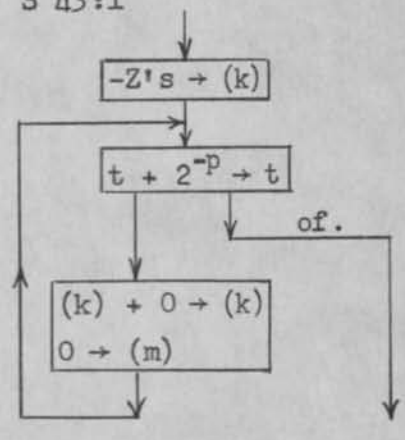
II. S 85:1



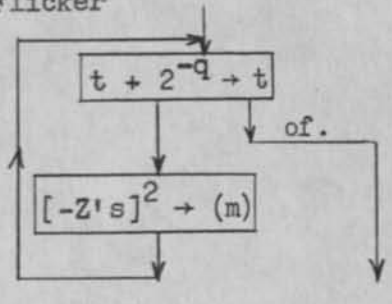
III. S 64:1



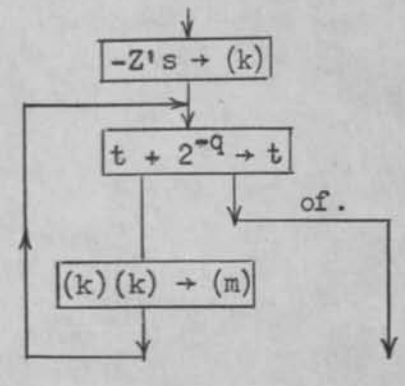
IV. S 43:1



V. Flicker

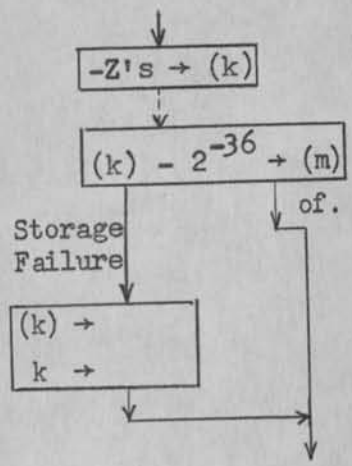


VI. Flicker (k)

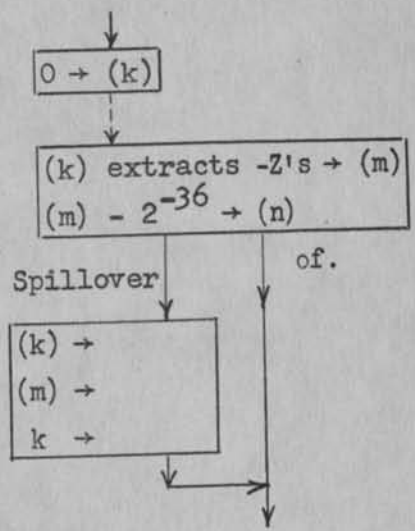


2. Testing

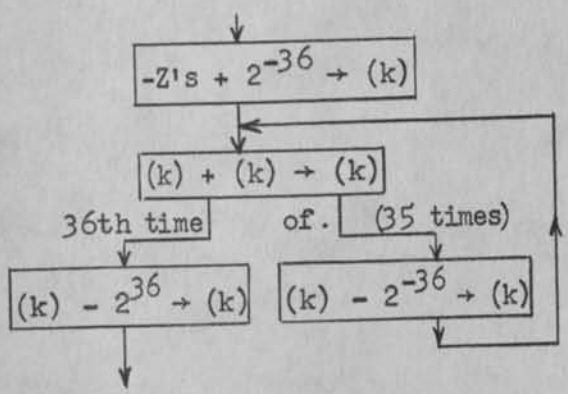
VII. Address Storage Check



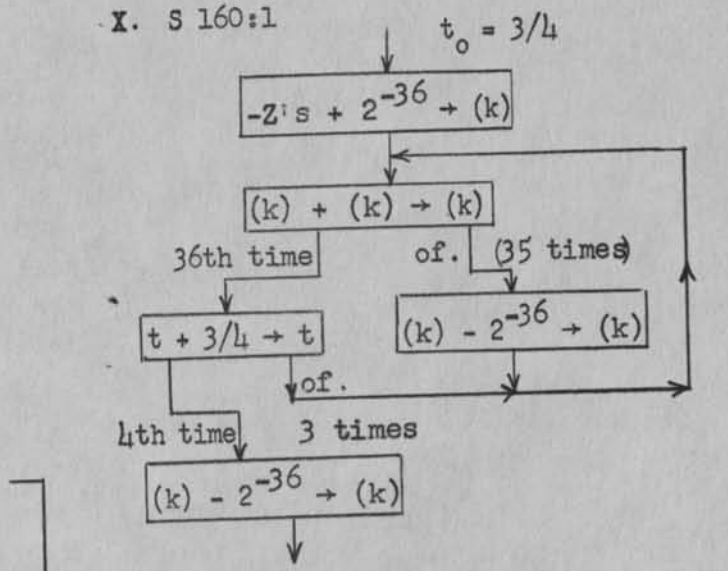
VIII. Address Spillover Check



IX. S 160:1

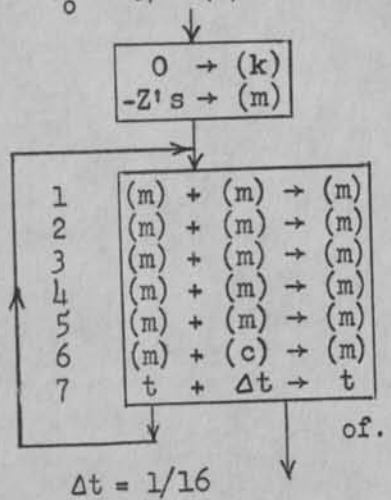


X. S 160:1



XI. R 24(24)216,219 max;

t_o = 15/16(1/16)0.



(c): (-00 00 00 01 Z)

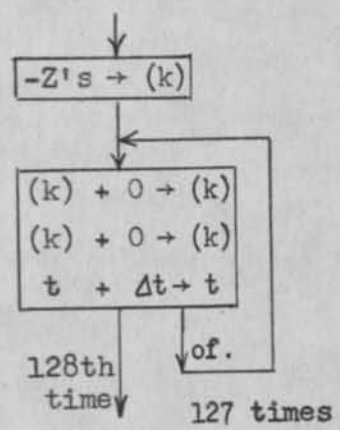
3 by 3 region of address (k)

2	4	m
6	k	3
5	7	1

[Position of commands relative to address (k)]

2. Testing

XII. S 85:1



Δt: 127/128

I. Memory Tests.

00000 Memory R

Memory R checks storage in addresses k = 40(1)ZZ using subroutines VII and XII. A storage failure is indicated by two type-outs: address (k) and an address having k in alpha.

00001 Memory RZ

Memory RZ adds subroutine V (Flicker) to Memory R. This subroutine is alternately used with the testing of storage in (k) to cause change of load on the voltage supplies, and thus bring out marginal failures.

00003 Memory RZ/2

Memory RZ/2 uses subroutine VI [Flicker (k)], II, and VII. The use of VI with alternate addresses of (k) gives the best check on "(k)'s" storage ability. Addresses k = 41(1)ZZ are checked for storage.

II. Spillover Tests.

A. ZERO Spillover.

These tests check for spillover in an electrostatic region of zeros. "ZERO" spillover test check address (k) for spillover and

2. Testing

storage in four steps: (1) Address (k), which is left zero, is checked by VIII (Address Spillover Check); (2) Subroutine I, II, III, IV, or IX is executed; (3) Address (k) is checked for storage by VII (Address Storage Check); (4) Zero goes to (k) and reference address k is increased by one, where the cycle (1) to (4) is repeated to check address (k + 1). Single spillover uses one reference address (k); whereas Double spillover uses two reference addresses alternately, (k_1) and (k_2).

00002 Spillover (85:1 S)
00004 Spillover (85:1 D)

Spillover 00002 and 00004 use subroutines II, VII, and VIII; where k = 40(1)ZZ in 00002 and k_1 = 70(1)ZZ, k_2 = ZZ(-1)70 in 00004. A spillover is indicated by two type-outs: an address having alpha equal to k and address (k). For a storage failure, the type-outs are in reverse order.

00006 Spillover (107:1 S)
00008 Spillover (107:1 D)

Spillover 00006 and 00008 use subroutines I, VII, and VIII, where k = 40(1)ZZ in 00006 and k_1 = 70(1)ZZ, k_2 = ZZ(-1)70 in 00008. It must be noted that for ratios of 107:1 and 160:1, the address of a storage failure is the only thing that can be diagnosed - NOT the CRT digit involved. Therefore in case of memory failure the word "memoire" is typed out rather than address (k).

00020 Spillover (106:1 S)

Spillover 00020 uses subroutines VII, VIII, and IX. This is a single spillover test with a very high spillover ratio of 160:1.

2. Testing

B. ONE (or Square) Spillover.

These tests check for spillover in an electrostatic memory which is predominantly "ones". Square spillover tests put -Z's in addresses 57(1)ZZ. Then a 3 by 3 square of "zeros" with center reference (k) is inserted, where k = 68(1)YY. Subroutine I, II, III, IV, or IX is used. Address (k) is checked for storage by VII; and addresses (k-17), (k-1), (k+1), and (k+15) are checked for spillover by VIII. Square spillover tests have proved to be the most rigorous.

00012 Square Spillover (107:1)
00014 Square Spillover (85:1)
00016 Square Spillover (64:1)
00018 Square Spillover (43:1)

These tests are similar except for the use of spillover subroutines I, II, III, and IV respectively. Subroutines VII, and VIII are used in each, but test 00012 cannot indicate the CRT digit store failure, therefore 00012 shows nothing (zeros) in the M and R registers for a storage failure and the address may be determined by observing (k) of the "square" in the memory. If a storage failure shows up consistently in test 00012, which cannot be detected by test 00003, it usually may be diagnosed by test 00014. In tests 00014, 00016, and 00018 a storage failure is indicated by displaying -Z's in the M and (k) in the R register.

For spillover the breakpoint of address spillover check VII gives (k) in R and (m) in M. For no spillover in (k), the result in (m) is -Z's and in (n) is zero with overflow. If the R register indicates no spillover and the M indicates spillover, we have a quasi-spillover effect due to the fact that the cell (k) read as a "one" but regenerated as a "zero". This is due to the difference in gating for reading and regeneration.

2. Testing

Nevertheless, it is a type of spillover and steps should be taken to remedy it. If quasi-spillover occurs too frequently in a given CRT, and CRT adjustment fails to correct it, the CRT unit will have to be replaced.

S register effect is usually first evidenced by quasi-spillover. This effect may be checked by clearing the memory and observing the base-line of a CRT in the CRT test monitor on extended sweep. S register effect is evidenced by little blips on the baseline. These can be moved by pushing 00PB while on period operation. The trouble lies in the CRT and NOT the S register. If these blips are too high or problem failure indicates trouble in a CRT having S register effect, the CRT must be replaced.

00022 Square Spillover (160:1)
[Not Coded]

If a harder test than 00012 is required, test 00022 can be coded by using subroutine X in place of IV in the square spillover 00012. Cells (k-17), (k-1), (k+1), and (k+15) would be tested for spillover and cell (k) for storage as in 00012.

C. Read-around. 00010 Read-around (16 note)

This test obeys seven add commands in a 3 by 3 square read-around address (k) using subroutines VIII and XI. The region of address (k) is referred to 24 times per loop (Diagram XI). Therefore a variable read-around [R = 24(24)216,219] is obtained by changing the number of cycles made in the loop reaching the limit of 219 for ten cycles. The number of cycles varies from 1 to 16, consequently the title "16 note" comes from the 16 notes as heard from the audio monitor. Only address (k) is checked for spillover by VIII; therefore spillover outside the 3 by 3 region is not detected. Read-around 00010 has proven to be ineffective in causing spillover, but is preserved for informational value.

2. Testing

6. Drum Tests 00700 and 00704

Drum Tests 00700 and 00704 are to be used when testing a particular channel of the drum. The information to be read to the drum is appended to the end of the routine.

Routine 00700 is used when continuous reading or writing is required; 00704, when an alternation of reading and writing is sought. Both routines halt in 02 asking for a keyboard input of the particular channel number into the gamma of cell 80. A brief description and the code of both routines follows.

00700: This routine will either continuously write or continuously read. To write send control to 03 and push 00PB. To read, send control to 07 and push 00PB. The routine consists of three cards numbered 1, 2, and 6. Put the 30 words of binary information to be read to the channel immediately following cards 2. Follow this with card 6.

The code for 00700:

Cell No.	α	β	γ	δ	F	Remarks
00	00	00	00	00	0	The origin
01	01	01	01	20	7	Transfer control to 20
02	80	00	00	10	1	Type in channel into γ
03	80	04	05	05	5	Prestore write command
04	20	00	00	70	2	Dummy for Drum write
05	20	00	[00]	70	2	Drum write
06	01	01	01	05	7	Return to 05
07	80	08	09	09	5	Prestore Read command
08	40	00	00	70	1	Dummy for Drum read
09	40	00	[00]	70	1	Drum read
0u	01	01	01	09	7	Return to 09

2. Testing

00704: This routine is used when alternate read and write commands are needed. It first writes the data on the channel and then reads it back again. No comparisons are made by the machine.

The code for 00704:

Cell No.	α	β	γ	δ	F	Remarks
00	00	00	00	00	0	The origin
01	01	01	01	20	7	Transfer control to 20
02	80	00	00	10	1	Type in channel into γ
03	80	04	05	07	5	Prestore Write command
04	20	00	00	70	2	Dummy for Drum write
05	20	00	[00]	70	2	Drum write
06	40	00	[00]	70	1	Drum read
07	80	08	06	05	5	Prestore Drum read command
08	40	00	00	70	1	Dummy for Drum read

3. Code Checking

1. Code Checking on SWAC (General)

To facilitate orderly code checking on SWAC, the following regulations have been set up.

1. After a code is hand-punched on cards, a listing will be returned to the coder. This listing must be proofread for punching and copying errors, and missing or extra cards.

2. After proofreading is completed the hand-punched cards can be converted on SWAC. If a code is in symbolic (relative) form, a deck of decimal cards and a binary deck will be produced by the SWAC. The decimal deck must be listed before code checking, and the listing must be on hand when the code is checked out. If the code was in absolute form the original decimal listing (corrected) shall be used during code checking. Under no circumstances shall a relative code be used during code checking on SWAC.

3. Additional aids to code checking are a flow chart, and the remarks on the hand written code.

4. A planned sequence of operations should be written out by the coder before checking out a code. This will save time on the SWAC. If a test problem is to be run, have answers ready in the form in which SWAC will produce them. Thus there will be no delay in checking results. Plan where breakpoints will be needed, and have them punched on the binary cards beforehand.

5. Take down as much information as you can while code checking. Note the position of epsilon when halts occur, the cells involved in operation loops and so on.

3. Code Checking

2. Modification of a Routine in Machine

After a routine is in the memory, it may be necessary to make changes in the commands. This is particularly true in code checking. To assist in making these changes, most coders use cell 000 for a keyboard input command: input to cell 001. Cell 001 itself is left empty or used only as a temporary storage.

The operator may return the SWAC to the origin, that is cell 000, in the following manner. With mode of operation switch on COM, and period 7 light on, the clear epsilon switch is pressed. If the command in the C register is not a special add, subtract, or multiply, nor a compare command, and if no overflow has occurred in the result in the M register, the address of the next command will be 000; otherwise, the delta to epsilon transfer will occur, regardless of the action of the clear epsilon switch. Use may be made of the R → C transfer to prevent the delta to epsilon transfer.

If 00PB is now pushed, the "input to 001" command will be in the C register, SWAC will be on period 3, and a command may be typed into the R register by hand. Any command so inserted may be obeyed by pushing 00PB twice if on COM, or once, if on CONT. In the latter case, the machine will go on to obey the command in cell 002 unless the word typed into 001 was an input command, a breakpoint, or a special.

Note that all input commands halt in period 3 to receive input, and output commands halt in period 5 during output.

3. Code Checking

Suppose now that a routine has been fed into the memory, and a memory check sum was computed which typed out. We wish to insert the negative of this memory sum into, say, cell xz (or 223, in decimal). The machine was halted after the typeout, and is in period 7, of some command. Clear epsilon switch is operated, and the input command in 000 brought into the C register by pushing 00PB. Now we type into R register the command xz 00 00 10 1, which calls for input to cell xz. This will be typed in as 1010000zx, since all numbers entered into R shift in from left to right.

Advance epsilon so this command stands in the C register, SWAC on period 3. Type in the negative memory sum. If an error is made in typing, begin over again, and the erroneous digits will be shifted out ahead of the correct ones. Note that the minus sign, once typed in, cannot be removed from the R register except by transfer to M.

Pushing 00PB once will serve to transfer this memory sum to the cell xz. It will remain in R, since no clear R occurs at this time. SWAC will now be on period 7, and epsilon may now be cleared. Push 00PB and again the command in 000 will appear in the C register. Suppose now that we wish to continue computation, starting in cell 15 (021 in decimal), but we would like to have 001 empty after we leave it. A special command of some kind will take us to cell 15, and we can insert zeros in cell 001 by the proper kind of instruction. If, therefore, we type in 01 01 01 15 7, a special subtract command, the answer, 000000000, will go into cell 001, erasing the command, and the delta to epsilon transfer will set epsilon to 15 (021 in decimal).

3. Code Checking

Note that if clear epsilon switch is operated when the SWAC is standing on period 3, COM operation, the command in the C register will be obeyed, when OOPB is pressed, as though it were in the origin. If this command is not a special, the next command in the C register will come from cell 001, otherwise it will come from the delta displayed in the C register.

3. Code Checking

4. A, M, and R Register Neons.

On COM mode of operation, the SWAC displays the answers to commands in the A and R registers on period 7. The A register contains the results of add, subtract, and compare commands. For a product command the most significant half of the result is displayed in the A register, the least significant in R. A multiply command will also display both halves of the result, but the most significant half is rounded and the least significant half is not stored in the memory.

In Extract the whole shifted result is displayed, and the operator should look at the C register to determine whether a left or right shift was called for, i.e., whether the contents of the R register or the A register have been stored.

Output type breakpoint displays the contents of address α in the R register and the contents of address β in the M register, on period 5. On period 7 only the α address is displayed.

A $\beta 5$ breakpoint halts the command affected with period 1 light ON. The answer to the previous command is still displayed in the A and R registers, although the command containing the breakpoint is displayed in the C register. This is a useful code checking device.

0. List, Non-Mathematical Codes (other than Conversion Routines)

SWAC Manual Page Reference	Code Number	Title
E 3.1	00707	Drum-punch
E 3.2	00702	Drum-clear
E 4.1	00380	Memory Punchout
E 6.1	00400	Sorting Code
E 6.2	00401	Row-deleting Code

1. Conversion Routines

Summary of SWAC Conversion Routines

NUMBER	WHAT IS CONVERTED	OUTPUT BOARD	EXTRA WIRES	REMARKS
		DECIMAL TO BINARY		
00206	Numbers (Floating)	36-digit	7 Jackplug	11 Words Output Card No. in 0 Row
00208	Numbers (Triple Prec.)	"	9 "	11 Words Output Card No. in Y Row
00210	Numbers (Integers)	"	9 "	" "
00212	Commands (Relative)	"	7 and 9 Jackplug*	12 Words Output Card No. in Y Row
00214	Commands (SWACPEC) and Numbers (Fractional)	"	9 Jackplug	11 Words Output Card No. in Y Row
00216	Numbers (Double Prec.)	"	9 "	" "
00218	Commands (Floating SWACPEC)	"	9 "	" "
00224	Commands (SWAC) and Numbers (Fractional)	"	9 "	" "
		BINARY TO DECIMAL		
00203	Commands (SWAC)	36-digit	7 Jackplug	11 Words Output
00205	Numbers (Floating)	"	7 "	11 Words Output Card No. in Columns 76, 77, 78
00207	Numbers (Triple Prec.)	72-digit	7 and 9 Jackplug	12 Words Output 2 Numbers per Card
00209	Numbers (Fractional)	"	7 Jackplug	11 Words Output 5 Numbers per Card
00211	Numbers (Integers)	"	7 "	" "
00213	Numbers (Double Prec.)	"	7 "	11 Words Output 4 Numbers per Card
		BINARY TO BINARY		
00200	Numbers (Floating) to Numbers (Fractional)	36-digit	none	10 Words Output
00201	Numbers (Fractional) to Numbers (Floating)	"	"	" "

*00212 converts relative commands to decimal and also to binary. A wire must be added to 36-digit board from "Punch Magnets" column 1, to "Comparing Magnets from Punch Brushes" column 76.

1. Conversion Routines

1. Summary of SWAC Conversion Routines

a. Decimal to Binary Conversion Routines

	ROUTINE NUMBER	DESCRIPTION	For Negat, Punch X in col.:	Decard Format Number	DECIMAL IN		BINARY OUT	
					No. of decimal words per decard	No. of decards between punch-outs	No. of binary words per swacard	No. of swacards punched between inputs
	00200	Commands	**	2	1	10	10	1
	00202	Numbers(abs.val.< 1).	**	4	1	10	10	1
	00204	Commands: X col. 47. Numbers(abs.val.< 1): No X col. 47.	78	2,4	1	10	10	1
	00206	Floating Numbers	See format	13	3	4,3,3	10	1
	00206.1	Floating Numbers	"	14	1	10	10	1
	00206.2	Floating Numbers	"	13	3,3,2	3	8	1
	00208	Numbers(abs.val.< 1).	**	4	1	10	10	1
*	00212	Relative Commands	53	15 16,17	1	10	10	1
*	00214	SWACPEC Commands: No X col.47. Numbers(abs.val.< 1): X col. 47.	34	18,19	1	10	10	1
	00216	Integers	**	8	1	10	10	1
*	00218	Floating SWACPEC Commands	**	20	1	10	10	1
	00220	Commands	48	2	1	100	10	10
	00220.8	Commands	48	2	1	96	8	12
	00224	Commands: No X col.47. Numbers(abs.val.< 1): X col. 47.	48	2,4	1	10	10	1
*	00230	Commands: No X col.33. Floating Numbers: X col. 33.	34	1,12	1	$n\left[\frac{100}{n}\right]$	n ($n \leq 12$)	$\left[\frac{100}{n}\right]$
*	00232	Commands: No X col.33. Integers: X col. 33.	34	1,9	1	$n\left[\frac{100}{n}\right]$	n ($n \leq 12$)	$\left[\frac{100}{n}\right]$
*	00234	Commands: No X col.33. Numbers(abs.val.< 1): X col. 33.	34	1,3	1	$n\left[\frac{100}{n}\right]$	n ($n \leq 12$)	$\left[\frac{100}{n}\right]$
*	00236	Triple Precision Numbers (abs.val.< 1).	**	24	1	3	3	1
	00240	Integers.	48	10	1	100	10	10

*Indicates most efficient or only routine of its kind.

**Any SWAC field position.

1. Conversion Routines

1. Summary of SWAC Conversion Routines
b. Binary to Decimal Conversion Routines

	ROUTINE NUMBER	DESCRIPTION	For Negat, Punch X in col.:	Decard Format Number	BINARY IN		DECIMAL OUT	
					No. of word pulses between collator halts	No. of collator halts between punch-outs	No. of decimal words per decard	No. of decards punched between inputs
	00201	Numbers**(scaled 2^s)	29	4	10	1	1	10
	00203	Commands	48	2	10	1	1	10
	00203.1	Commands	48	2	32	1	1	32
*	00205	Floating Numbers	See Format	13	10	1	3	3,4,4[a]
*	00205.1	Floating Numbers	"	13	10[b]	1	3	3
*	00205.2	Floating Numbers	"	13	8	1	3	3[c]
	00207	Numbers($\|N\|<1$)	"	5	10[d]	1	9	1
	00209	Numbers**(scaled 2^s)	"	6	10	1	5	2
*	00209.1	Numbers**(scaled 2^s)	"	7	10	1	10	1
*	00209.2	Numbers**(scaled 2^s)	"	6	8	1	5[e]	2
	00211	Integers	"	11	10	1	5	2
*	00213	Commands	34	1	10	1	1	10
*	00213.1	Commands	34	1	32	1	1	32
*	00215	Triple Precision Numbers($\|N\|<1$).	See Format	25	See description of routine			
*	00219	Numbers**(scaled 2^s)	"	6	10	60	5	120
*	00219.1	Numbers(scaled 2^s)[f]	"	6	10	60	5	120
*	00221	Numbers**(scaled 2^s)	"	7	10	120	10	120
*	00223	Commands	-	-	10	Direct Output to Tabulator		
*	00229	Numbers**(scaled 2^s)	-	-	10	Direct Output to Tabulator		
*	00231	Integers	-	-	10	Direct Output to Tabulator		

*Indicates most efficient or only routine of its kind.

[a]Eleventh decimal word is sum of previous ten.

[b]Tenth binary word is ignored.

[c]Ninth decimal word is zero.

[d]First binary word is ignored.

[e]Fifth decimal word is zero.

[f]Same as 00219 but can use 10 different scale factors.

**The integer part of each number consists of s binary digits ($0 \leq s \leq 27$).

1. Conversion Routines

1. Summary of SWAC Conversion Routines

c. Miscellaneous Conversion Routines

ROUTINE NUMBER	DESCRIPTION	For Negat, Punch X in col.:	Format Number	IN	OUT
00217	Fixed binary to floating binary	-	-	10 binary words	1 swacard with 10 binary words
00222	Floating binary to fixed binary	-	-	r binary words ($r \leq 10$)	1 swacard with r binary words
00225	Binary to base 4	49	23	n binary words ($n \leq 10$)	n cards with one base 4 word/card
00226	Hex to binary	34	21	10 cards with 1 hex word/card	1 swacard with 10 binary words
00227	Binary to hex	34	26	10 binary words	10 cards with 1 hex word/card
00228	Octal to binary	48	22	100 cards with 1 octal word/card	10 swacards with 10 binary words/card

1. Conversion Routines

2. Decimal Card Formats for Conversion Routines

GENERAL REMARKS

1. "Decard" is a coined word used to describe any decimally punched card used as SWAC input, or punched by the output punch. The term usually refers to cards punched in "SWAC Fields." Any decard to be fed to SWAC must be punched in column 5 (word-pulse column) rows 9 through 1, and X.

2. "Swacard" is a coined word used to describe a binary punched card prepared for SWAC input, or punched by the output punch, and utilizing the "SWAC Fields." Only "SWAC Fields" are read in by the SWAC collator standard board. These include the following columns:

4-14, 27-34, 47-54, 67-79.

The decimal formats only specify the use of certain of these columns as required by the routines; apart from column 5, all SWAC fields not mentioned by the format or the routine must be left blank (or zero). Although the use of some of these columns may sometimes be harmless, it may create such havoc as to justify this blanket rule.

3. Certain columns, although never read in by the SWAC collator, are generally used as follows:

columns 1-3: card number
columns 16-20: problem number.

4. The standard SWAC collator board reads in up to 10 rows per card, rows X and 1-9. Although 11 rows input is possible by the

1. Conversion Routines

addition of an extra jack plug to the collator board, all standard conversion routines are designed for 10 rows input unless otherwise specifically stated. Thus the 0- and Y-rows are ignored and may be punched at will.

5. The X-row is used to denote a negative number or command, and to differentiate between commands and numbers as prescribed by a particular conversion routine or by a particular format.

6. In the following list, the parenthesized numbers designate pertinent conversion routines.

FORMAT 1: COMMANDS. (Input for 00230, 00232, 00234; output from 00213, 00213.1)

COLUMNS	DESCRIPTION
7-9	α
10-12	β
13-14, 27	γ
28-30	δ
31-32	F
33	Blank or zero.
34	X punch denotes a negative command.
35-46, 49-79	Remarks if desired.

FORMAT 2: COMMANDS. (Input for 00200, 00204, 00220, 00220.8, 00224; output from 00203, 00203.1)

COLUMNS	DESCRIPTION
7-9	α
10-12	β
13-14, 27	γ
28-30	δ
31	Blank or zero.
32-33	F
47	X punch for routine 00204 only.
48	X punch denotes a negative command for routines 00203, 00203.1, 00220, 00220.8, 00224.
78	X punch denotes a negative command for routine 00204 only.
Any SWAC field	X punch denotes a negative command for routine 00200 only.
35-46, 49-79	Remarks if desired (except for routine 00200, and except for col. 78 for routine 00204).

1. Conversion Routines

FORMAT 3: NUMBERS (abs. val. < 1). (Input for 00234)

COLUMNS	DESCRIPTION
7-14, 27-29	The 11 decimal digits of the number.
33	X punch denotes a number (other rows may be punched in addition for alphabetic listing).
34	X punch denotes a negative number.
35-46, 49-79	Remarks if desired.

The decimal point is assumed to be just to the left of column 7.

FORMAT 4: NUMBERS (abs. val. < 1). (Input for 00202, 00204, 00208, 00224; output from 00201, scaled 10^{σ}).

COLUMNS	DESCRIPTION
7-14, 27-29	The 11 decimal digits of the number.
29	X punch denotes a negative number
31	The exponent of the decimal scale factor, σ { for routine 00201 only
47	X punch for routine 00204 only.
48	X punch denotes a negative number for routine 00224 only.
78	X punch denotes a negative number for routine 00204 only.
Any SWAC field	X punch denotes a negative number for routines 00202 and 00208 only.
35-46, 49-79	Remarks, if desired, for routine 00224, and except for col. 78 with routine 00204.

The decimal point is assumed to be just to the left of column 7, except for routine 00201 where the integer part consists of σ digits.

FORMAT 5: NUMBERS (abs. val. < 1), 9 numbers per card, 8 digits per number. (Output from 00207)

COLUMNS	DESCRIPTION
6-13	The 8 digits of 1st number; X in col. 6 denotes minus.
14-21	" 2nd " 14 "
22-29	" 3rd " 22 "
30-37	" 4th " 30 "
38-41, 45-48	" 5th " 38 "
49-56	" 6th " 49 "
57-64	" 7th " 57 "
65-72	" 8th " 65 "
73-80	" 9th " 73 "

The decimal point of each number is assumed to be just to the left of its first column.

1. Conversion Routines

FORMAT 6: NUMBERS (scaled 10^{σ}), numbers per card, 11 digits per number. (Output from 00209, 00209.2, 00219, 00219.1)

COLUMNS	DESCRIPTION
6-16	The 11 decimal digits of 1st number; X in col 16 denotes minus.
17-27	" 2nd 27
28-38	" 3rd 38
45-55	" 4th 55
56-66	" 5th 66
80	The exponent of the decimal scale factor, σ.

The integer part of each number occupies σ columns, except for 00219.1 (see description of routine).

FORMAT 7: NUMBERS (scaled 10^{σ}), 10 numbers per card, 7 digits per number. (Output from 00221, 00209.1)

COLUMNS	DESCRIPTION
6-12	The 7 decimal digits of 1st number; X in col 12 denotes minus.
13-19	" 2nd 19
20-26	" 3rd 26
27-33	" 4th 33
34-40	" 5th 40
45-51	" 6th 51
52-58	" 7th 58
59-65	" 8th 65
66-72	" 9th 72
73-79	" 10th 79
80	The exponent of the decimal scale factor, σ.

The integer part of each number occupies σ columns.

FORMAT 8: INTEGERS. (Input for 00216)

These numbers have at most 10 decimal digits which must be punched sequentially in columns 7-14, 27, 28. The decimal point is just to the right of column 28.

An X punch in any of the SWAC fields beyond column 6 denotes a negative integer.

1. Conversion Routines

FORMAT 9: INTEGERS. (Input for 00232)

COLUMNS	DESCRIPTION
7-14, 27-29	The 11 digits of the integer. The decimal point is just to the right of column 29.
33	X punch denotes an integer (other rows may be punched in addition for alphabetic listing).
34	X punch denotes a negative integer.
35-46, 49-79	Remarks if desired.

FORMAT 10: INTEGERS.(Input for 00240)

These numbers have at most 11 digits which must be punched sequentially in columns 67-77. The decimal point is just to the right of column 77.

An X punch in column 48 denotes a negative integer.

FORMAT 11. INTEGERS, 5 numbers per card, 10 digits per number. (Output from 00211)

COLUMNS	DESCRIPTION
6,17,28,45,56	Zero punches.
7-16	The 10 digits of 1st number;X in col 16 denotes minus.
18-27	" 2nd 27
29-38	" 3rd 38
46-55	" 4th 55
57-66	" 5th 66

The respective decimal points are just to the right of columns 16, 27, 38, 55, 66.

FORMAT 12: FLOATING NUMBERS.(Input for 00230)

COLUMNS	DESCRIPTION
7-14, 27	Fractional part of number.
28	X punch denotes a negative exponent.
29-30	Decimal exponent.
33	X punch denotes a number (other rows may be punched in addition for alphabetic listing).
34	X punch denotes a negative number.
35-46, 49-79	Remarks if desired.

1. Conversion Routines

FORMAT 13: FLOATING NUMBERS, 3 numbers per decard. (Input for 00206, 00206.2; output from 00205, 00205.1, 00205.2)

COLUMNS	DESCRIPTION	
7-14, 27	Fractional part of 1st number.	
28-29	Decimal exponent of 1st number.	
30-34, 47-50	Fractional part of 2nd number.	
51-52	Decimal exponent of 2nd number.	
53-54, 67-73	Fractional part of 3rd number.	Blank for cards holding 2 numbers only.
74-75	Decimal exponent of 3rd number.	

Additional X punch in column	7:	Minus	sign	for	fractional	part	of	1st number.
	28	"	"	"	exponent	"	"	" "
	30	"	"	"	fractional	"	"	2nd "
	51	"	"	"	exponent	"	"	" "
	53	"	"	"	fractional	"	"	3rd "
	74	"	"	"	exponent	"	"	" "

FORMAT 14: FLOATING NUMBERS, 1 number per decard. (Input for 00206.1)

COLUMNS	DESCRIPTION
7-14, 27	Fractional part of number.
28-29	Decimal exponent.
Additional X in col. 7	Minus sign for number.
Additional X in col. 28	Minus sign for exponent.

FORMAT 15: DICTIONARY CARD FOR RELATIVE CODING. (Input for 00212)

COLUMNS	DESCRIPTION
7-9	Position of region 12n (n = card no.).*
10-12	" " " 12n + 1
13-14, 27	" " " 12n + 2
28-30	" " " 12n + 3
31-33	" " " 12n + 4
34, 47-48	" " " 12n + 5
49-51	" " " 12n + 6
52-54	" " " 12n + 7
67-69	" " " 12n + 8
70-72	" " " 12n + 9
73-75	" " " 12n + 10
76-78	" " " 12n + 11

*Cards are numbered starting from 000.

1. Conversion Routines

FORMAT 16: RELATIVE COMMANDS. (Input for 00212)

COLUMNS	DESCRIPTION
7-11	α, in relative decimal.
12-14, 27-28	β, " " "
29-33	γ, " " "
34, 47-50	δ, " " "
51-52	F, in decimal.
9,14,31,48	Zero or blank.
53	X punch denotes a negative command.

FORMAT 17: RELATIVE-DECIMAL COMMANDS. (Output from 00212)

COLUMNS	DESCRIPTION
37-39	Card number.
7-53	Format 16.
67-69	α, in decimal.
70-72	β, " "
73-75	γ, " "
76-78	δ, " "

FORMAT 18: SWACPEC COMMANDS. (Input for 00214)

COLUMNS	DESCRIPTION
7-9	A
10-12	B
13-14, 27	C
28-29	S
30-31	D
32-33	F
34	X punch denotes a negative command.

FORMAT 19: SWACPEC NUMBERS (abs. val. <1). (Input for routine 00214)

COLUMNS	DESCRIPTION
7-14, 27, 28	The 10 decimal digits of the number.
34	X punch denotes a negative number.
47	X punch denotes a number.

The decimal point is assumed to be just to the left of column 7.

1. Conversion Routines

FORMAT 20: FLOATING SWACPEC COMMANDS. (Input for 00218)

COLUMNS	DESCRIPTION
7-10	A
11-14	B
27-28	C
29	D
30-31	F
41	First digit of card number (used only if card numbers go beyond 999).
Any SWAC field	X punch denotes a negative command.

FORMAT 21: HEXADECIMAL COMMANDS AND NUMBERS. (Input for 00226)

COLUMNS	DESCRIPTION
7-8	α
9-10	β
11-12	γ
13-14	δ
27	F
34	X punch denotes a negative number or command.

α, β, γ, δ and F are punched in the following hexadecimal form. Each hex digit is indicated by the single-column punches shown directly beneath.

For hex digit:	0	1	2	3	4	5	6	7	8	9	U	V	W	X	Y	Z
Punch in one column:	0	1	2	3	4	5	6	7	8	9	X 4	X 5	X 6	X 7	X 8	X 9

FORMAT 22: OCTAL. (Input for 00228)

These consist of twelve decimal digits which must be punched sequentially in columns 7-14, 27-30. An X punch in col. 48 denotes a minus sign.

FORMAT 23: BASE 4 NUMBERS (abs. val. < 1). (Output from 00225)

These numbers have at most 18 digits which will be punched sequentially in columns 7-14, 27-34, 47, and 48. An X punch in column 49 denotes a negative number.

1. Conversion Routines

FORMAT 24: TRIPLE PRECISION NUMBERS (abs. val < 1). (Input for 00236)

These numbers have at most 32 digits which must be punched sequentially in columns 7-14, 27-34, 47-54, 67-74. An X punch in any SWAC field denotes a negative number. The decimal point is assumed to be just to the left of column 7.

FORMAT 25: PAIRS OF TRIPLE PRECISION NUMBERS (abs. val. < 1). (Output from 00215)

COLUMNS	DESCRIPTION
6-9	A 4 decimal digit integer.
10-41	The 32 decimal digits of 1st number; X in column 10 denotes a negative number.
45-48	A 4 decimal digit integer.
49-80	The 32 decimal digits of 2nd number; X in column 49 denotes a negative number.

FORMAT 26: HEXADECIMAL COMMANDS AND NUMBERS. (Output from 00227)

COLUMNS	DESCRIPTION
67-68	α
69-70	β
71-72	γ
73-74	δ
75	F
34	X punch denotes a negative number or command.

See description of Format 21 for hexadecimal punching.

1. Conversion Routines

3. General Instructions for Operating Conversion Routines

1) Collator input board: Unless otherwise stated the 36-digit collator input board with 10 word pulses (rows 9 through 1, and X) should be used.

2) Number of word pulses on decimal input data: The decards put into the collator for conversion to binary must have 10 word pulses.

3) Number of word pulses on binary input data: The binary cards input into the collator for conversion to decimal may have any number of word pulses (from 1 to 10) commensurate with the input requirements of a given conversion routine. The operator is reminded that when a card has started feeding through the collator, the collator cannot halt until that card has been completely read in. Thus the binary cards must be prepared in such a way that the "number of word pulses input between collator halts," specified by a given conversion routine, is obtained with an integral number of binary cards in each case.

4) Drum: If not mentioned, the drum is not needed.

5) Breakpoint Switch: If not mentioned, the setting of the breakpoint switch is immaterial.

6) Modifications: Possible modifications which are not mentioned here may be found in the code description of particular routines.

1. Conversion Routines

4. Conversion Routine 00225.

1) SWAC Code 00225.

2) Written by G. E. Forsythe, May 1954.

3) Binary to base 4 conversion.

4) This routine will take binary cards with n words per card ($1 \leq n \leq 10$), and convert the words (as fractions) to base 4, punching them one to a card for subsequent listing on a tabulator. It numbers the output cards in binary in Y row. Signs are properly converted, except that -0 is converted to +0. Negative output is an X punch in column 49.

The only parameter is n, the number of words per card.

5) Input cards are in standard form, with n word-pulses and binary words per card.

6) a) Output is 6 words per card, with jack plugs arranged so that cards will be punched in the 3, 2, 1, 0, X, and Y rows.

b) Breakpoint switch off.

c) Drum not used.

d) Ordinary input board. Routine (Card numbers 000 to 007) is followed by a 1 word-pulse card with n punched in the α position. Then follow the binary cards to be converted.

e) In operation, one card feeds in, n cards are punched out, and repeat.

f) Routine takes about half a second per output card - i.e., per number converted (including punchout time).

1. Conversion Routines

g) Any unexpected halts or loops indicate machine failure or wrong number of W P's on input cards. Check cards, and start over at that point.

h) Program halts when input cards are exhausted. Collect the punched output.

i) Output may be listed on IBM tabulator with specially prepared listing board.

7) Each two binary digits is extracted and used to determine one base 4 digit. Sign checked by comparison with zero.

8) The routine is contained within cells 000 to 043 and 060 to 075. Words to be converted are read into cells 240 to 239 + n. Cells 080 to 087, 199 to 203, and 255 are temporaries.

9) Only a fragmentary flow chart has been prepared; it is with the code sheets.

10) The code, with remarks, is in the file with other conversion routines of the 00200 series. It is identical with Forsythe's code 0500.07, in the SWAC code file in the library entitled "Forsythe Semigroups."

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

a. Conversion Routine 00200: Dec. to Bin. Commands.

Input Format: number 2. An X in any SWAC field position indicates a negative command.

Punch: 36-digit board, 11 words (rows 9 through 1, X, Y).

To Operate: 1. Load in routine.

2. SWAC halts in (04): 59,00,00,00,1.

3. Feed in data. SWAC alternately feeds 10 decards and punches 1 ten-word binary card.

Output Identification: Binary cards are serially numbered in F of the Y-row, beginning with zero.

Conversion Time: 10 cards in and 1 card out takes 6.8 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

b. Conversion Routine 00202: Dec. to Bin. Numbers (abs. val. < 1).

Input Format: number 4. An X in any SWAC field position indicates a negative number.

Punch: 36-digit board, 11 words (rows 9 through 1, X, Y).

To Operate: 1. Load in routine.

2. SWAC halts in (08): 9Z,00,00,00,1.

3. Feed in data. SWAC alternately feeds 10 decards and punches 1 ten-word binary card.

Output Identification: Binary cards are serially numbered in F of the Y-row, beginning with zero.

Conversion Time: 10 cards in and 1 card out takes 6.8 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

c. Conversion Routine 00204: Dec. to Bin. Commands and Numbers (abs. val. < 1).

Input Formats: numbers 2 and 4. X in col. 78 indicates a negative command or number. X in col. 47 indicates a command; no X in col. 47 indicates a number.

Punch: 36-digit board, 11 words (rows 9 through 1, X, Y).

To Operate:
1. Read in routine.
2. SWAC halts in (OU): 63,00,00,20,1.
3. Feed in data. SWAC alternately feeds 10 decards and punches 1 ten-word binary card.

Output Identification: Binary cards are serially numbered in F of the Y-row, beginning with zero.

Conversion Time: 10 cards in and 1 card out takes 6.8 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

d. Conversion Routine 00206: Dec. to Bin. Floating Numbers

Input Format: number 13. Three numbers per decard (see description of format 13 for use of X-row).

Punch: 36-digit board, 11 words (rows 9 through 1, 0, X).

To Operate:

1. Read in routine.
2. SWAC halts in (OV): Z3,00,00,00,1.
3. Feed in data.
   a) SWAC feeds 4 decards and punches out 1 ten-word binary card.
   b) SWAC feeds 3 decards and punches out 1 ten-word binary card.
   c) SWAC feeds 3 decards and punches out 1 ten-word binary card.

   Steps (a), (b) and (c) are repeated in that order until all the data have been fed in.
4. If the number of decards is not a multiple of 10: When the collator runs out of cards, OOPB 10 to 30 times to punch out the final binary card.

Output Identification: Binary cards are serially numbered in F of the 0-row, beginning with 1.

Conversion Time: 10 cards in and 3 cards out takes 8 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

e. Conversion Routine 00206.1: Dec. to Bin. Floating Numbers

Input Format: number 14. One number per decard (see description of format 14 for use of X-row).

Punch: 36-digit board, 11 words (rows 9 through 1, 0, X).

To Operate:
1. Read in routine.
2. SWAC halts in (0V): Z3,00,00,00,1.
3. Feed in data. SWAC alternately feeds 10 decards and punches 1 ten-word binary card.

Output Identification: Binary cards are serially numbered in F of the 0-row, beginning with 1.

Conversion Time: 10 cards in and 1 card out takes 6.6 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

f. Conversion Routine 00206.2: Dec. to Bin. Floating Numbers

Input Format: number 13. Three numbers per decard except for every third card, which must have two numbers. (See description of format 13 for use of X-row.)

Punch: 36-digit board, 11 words (rows 9 through 1, 0, Y). Requires an eight-word-pulse master card (rows 9 through 2). Requires word-pulse wire in gang punch position (punch magnet 76 to punch brush 76). This is in order to obtain eight-word-pulse cards for subsequent collator input.

To Operate:
1. Read in routine.
2. SWAC halts in (OV): Z3,00,00,00,1.
3. Feed in data. SWAC alternately feeds 3 decards containing altogether 8 decimal words and punches 1 eight-word binary card.

Output Identification: Binary cards are serially numbered in F of the Y-row, beginning with 1.

Conversion Time: 3 cards in and 1 card out takes 2.6 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

g. Conversion Routine 00208: Dec. to Bin. Numbers (abs. val. < 1)

Input Format: number 4. An X in any SWAC field position indicates a negative number.

Punch: 36-digit board, 10 words (rows 9 through 1, X).

To Operate:
1. Read in routine.
2. SWAC halts in (10): 59,00,00,00,1.
3. Feed in data. SWAC alternately feeds 10 decards and punches 1 ten-word binary card.

Conversion Time: 10 cards in and 1 card out takes 6.5 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

h. Conversion Routine 00212: Dec. to Bin. Relative Commands

Input Formats: numbers 15 and 16. X in col. 53 indicates a negative command.

Punch: 36-digit board, 12 words. Requires wire from "Comp. Magnets from Comp. Brushes": col. 76, to "Punch Magnets": col. 1.

To Operate:

1. Read in Part I of routine, followed by dictionary cards,* followed by Part II of routine.
2. SWAC halts in (02): 64,00,00,10,1.
3. From keyboard, input number of dictionary cards in F; press 00PB.
4. SWAC halts in (02): VV,00,00,10,1.
5. From keyboard, input initial serial number of binary cards in F; press 00PB.
6. SWAC halts in (03): XW,00,00,10,1.
7. From keyboard, input initial serial number of relative-decimal cards in F; press 00PB.
8. SWAC halts in (06): 96,00,00,00,1.
9. Feed in data. SWAC alternately feeds 1 decard and punches 1 relative-decimal card. After 10 relative-decimal cards it punches one ten-word binary card.

Output Identification: The 10 relative-decimal cards (format number 17) of an 11-card group can be distinguished

*To be prepared by the coder as explained in description of routine 00212.

1. Conversion Routines

from the binary card by a "1" punch in col. 80 in the binary card. Relative-decimal cards are serially numbered in decimal in columns 37 through 39, binary cards in Y-row.

Conversion Time: 10 cards in and 11 cards out takes 15 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

i. Conversion Routine 00214: Dec. to Bin. SWACPEC Commands and Numbers (abs. val. < 1)

Input Formats: numbers 18 and 19. X in col. 34 indicates a negative command or number. X in col. 47 indicates a number; no X in col. 47 indicates a command.

Punch: 36-digit board, 11 words (rows 9 through 1, X, Y).

To Operate:

1. Read in routine.
2. SWAC halts in (02): ZU,00,00,10,1.
3. From keyboard, input initial serial number of binary cards in F; press 00PB.
4. SWAC halts in (05): X9,00,00,00,1.
5. Feed in data. SWAC alternately feeds 10 decards and punches 1 ten-word binary card.

Output Identification: Binary cards are serially numbered in F of the Y-row.

Conversion Time: 10 cards in and 1 card out takes 6.5 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

j. Conversion Routine 00216: Dec. to Bin. Integers

Input Format: number 8. An X in any SWAC field position indicates a negative integer.

Punch: 36-digit board, 11 words (rows 9 through 1, X, Y).

To Operate: 1. Read in routine.

2. SWAC halts in (OU): 4Z,00,00,20,1.

3. Feed in data. SWAC alternately feeds 10 decards and punches 1 ten-word binary card.

Output Identification: Binary cards are serially numbered in F of the Y-row, beginning with zero.

Conversion Time: 10 cards in and 1 card out takes 5 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

k. Conversion Routine 00218: Dec. to Bin. Floating SWACPEC Commands

Input Format: number 20. An X in any SWAC field position indicates a negative command.

Punch: 36-digit board, 11 words (rows 9 through 1, X, Y).

To Operate:
1. Read in routine.
2. SWAC halts in (02): 63,00,00,10,1.
3. From keyboard, input initial serial number of binary cards in F; press 00PB.
4. SWAC halts in (0X): 58,00,00,00,1.
5. Feed in data. SWAC alternately feeds 10 decards and punches 1 ten-word binary card.

Output Identification: Binary cards are serially numbered in F of the Y-row.

Conversion Time: 10 cards in and 1 card out takes 6.5 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

ℓ. Conversion Routine 00220: Dec. to Bin. Commands

Input Format: number 2. X in col. 48 indicates a negative command.

Punch: 36-digit board, 11 words (rows 9 through 1, X, Y).

To Operate:

1. Read in routine.
2. SWAC halts in (09): 90,00,00,10,1.
3. From keyboard, input initial serial number of binary cards in F; press 00PB.
4. SWAC halts in (10): 91,00,00,00,1.
5. Feed in data. SWAC alternately feeds 100 decards and punches 10 ten-word binary cards.
6. If number of decards is not a multiple of 100: When collator runs out of cards, press Clear Epsilon button and 00PB to punch the final group of binary cards (fewer than 10 in number).

Output Identification: Binary cards are serially numbered in F of the Y-row.

Halts:

1. A halt in (20) or (0V): 91,00,00,10,1 indicates a decard has an insufficient number of word pulses. Correct faulty card and start over.
2. A halt in (ZZ): ZZ,00,00,10,1 indicates either
   (a) a machine error, or
   (b) a decard has too large a command, or
   (c) a decard has an X punch in a wrong column.
   Correct faulty card and start over.

Conversion Time: 100 cards in and 10 cards out takes 35 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

m. Conversion Routine 00220.8: Dec. to Bin. Commands

Input Format: number 2. X in col. 48 indicates a negative command.

Punch: 36-digit board, 9 words (rows 9 thru 2, Y).

To Operate:

1. Read in routine.
2. SWAC halts in (09): 90,00,00,10,1.
3. From keyboard, input initial serial number of binary cards in F; press 00PB.
4. SWAC halts in (10): 91,00,00,00,1.
5. Feed in data. SWAC alternately feeds 96 decards and punches 12 eight-word binary cards.
6. If number of decards is not a multiple of 96: When collator runs out of cards, press Clear Epsilon button and 00PB to punch the final group of binary cards (fewer than 12 in number).

Output Identification: Binary cards are serially numbered in F of the Y-row.

Halts:

1. A halt in (20) or (0V): 91,00,00,10,1 indicates a decard has an insufficient number of word pulses. Correct faulty card and start over.
2. A halt in (ZZ): ZZ,00,00,10,1 indicates either
   (a) a machine error, or
   (b) a decard has too large a command, or

1. Conversion Routines

(c) a decard has an X punch in a wrong column. Correct faulty card and start over.

Conversion Time: 96 cards in and 12 cards out takes 35 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

n. Conversion Routine 00224: Dec. to Bin. Commands and Numbers (abs. val. <1)

Input Formats: numbers 2 and 4. X in col. 48 indicates a negative command or number. X in col. 47 indicates a number; no X in col. 47 indicates a command. Other punches are ignored.

Punch: 36-digit board, 11 words (rows 9 thru 1,X,Y).

Breakpoint Switch: ON.

To Operate:
1. Read in routine.
2. SWAC halts in (02): XW,00,00,10,1.
3. From keyboard, input initial serial number of binary cards in F; press 00PB.
4. SWAC halts in (07): W7,00,00,00,1.
5. Feed in data. SWAC alternately feeds 10 decards and punches 1 ten-word binary card.

Output Identification: Binary cards are serially numbered in F of the Y-row.

Halt: Routine checks the conversion of commands by reconverting their binary form to decimal; discrepancies with original values lead to breakpoint halt in (17): 10,X2,00,60,2. Discrepancy may be ignored by pressing 00PB to continue.

Conversion Time: 10 cards in and 1 card out takes 6.5 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

o. Conversion Routine 00230: Dec. to Bin. Commands and Floating Numbers*

Input Formats: numbers 1 and 12. X in col. 34 indicates a negative command (see description of format 12 for use of X-row for floating numbers). X in col. 33 indicates a floating number; no X in col. 33 indicates a command.

Punch: 36-digit board, n words per card ($n \leq 11$).

To Operate:

1. Read in routine.
2. SWAC halts in (8W): 90,9V,00,10,1, showing U in F of M register.
3. From keyboard, input initial serial number of binary cards in beta, and n in F. (NOTE: Zero in F is treated by the routine as having n equal to ten.) Press 00PB.
4. SWAC halts in (8Y): 90,79,00,00,1.
5. Feed in data. SWAC alternately feeds $n\left[\frac{100}{n}\right]$ decards and punches $\left[\frac{100}{n}\right]$ binary cards containing n words each.
6. If number of decards is not a multiple of $n\left[\frac{100}{n}\right]$: When collator runs out of cards, press Clear Epsilon button and 00PB to punch the final group of binary cards.

---

*Accepts both normalized and unnormalized floating numbers. Numbers which have a leading decimal are converted normalized. Numbers which have a leading zero are converted to floating binary numbers which have almost the same significance as the decimal numbers.

1. Conversion Routines

7. After last punchout, SWAC halts in (8W): 90,9V,00,10,1 displaying in the M register the last serial number + 1 in alpha and beta, and n in F.

8. To convert a new deck, input from keyboard new initial serial number of binary cards in beta; press 00PB. Follow instructions at Step 4. (NOTE: One cannot input n into F again at this point.)

Output Identification: Binary cards are serially numbered in the row containing the last binary word in columns 37 through 44, 57 through 64.

Halts:

1. A dynamic halt at 4V, with routine obeying 00,00,00,4V,7 repeatedly, indicates the floating decimal number is too large to be represented in floating binary form. Correct error and start over.

2. A halt in (40): 94,79,00,10,1 or in (15): 90,04,00,10,1 indicates either
(a) a decard has too few word pulses, or
(b) routine is being run with too much delay.
Correct error and start over.

Conversion Time: 100 cards in and 10 cards out takes 35 seconds.

NOTE: It is possible to get a forced punchout in the middle of the input deck by use of a $\delta \rightarrow \varepsilon$ one-word-pulse card, which may also have new desired initial serial number in binary.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

p. Conversion Routine 00232: Dec. to Bin. Commands and Integers

Input Formats: numbers 1 and 9. X in col. 34 indicates a negative command or integer. X in col. 33 indicates an integer; no X in col. 33 indicates a command.

Punch: 36-digit board, n words per card ($n \leq 11$).

To Operate:

1. Read in routine.
2. SWAC halts in (8W): 90,9V,00,10,1, showing U in F of M register.
3. From keyboard, input initial serial number of binary cards in beta, and n in F. (NOTE: Zero in F is treated by the routine as having n equal to ten.) Press 00PB.
4. SWAC halts in (8Y): 90,79,00,00,1.
5. Feed in data. SWAC alternately feeds $n\left[\frac{100}{n}\right]$ decards and punches $\left[\frac{100}{n}\right]$ binary cards containing n words each.
6. If number of decards is not a multiple of $n\left[\frac{100}{n}\right]$: When collator runs out of cards, press Clear Epsilon button and 00PB to punch the final group of binary cards.
7. After last punchout, SWAC halts at 8W, showing n in F of M register and last card serial number + 1 in alpha and beta of M register.

1. Conversion Routines

8. To convert a new deck, input from keyboard new initial serial number of binary cards in beta, and new value of n in F. (NOTE: Zero in F will not be treated as having n equal to ten.) Follow instructions at Step 4 above.

Output Identification: Binary cards are serially numbered in the row containing the last binary word in columns 37 through 44, 57 through 64.

Halts: 1. A halt in (40): 94,04,00,10,1 or (15): 90,04,00,4V,1 indicates either
(a) a decard has too few word pulses, or
(b) routine is being run with too much delay.
Correct error and start over.

2. A dynamic halt at 4U, with routine obeying 00,00,00,4V,7 repeatedly, indicates an integer greater than $2^{36} - 1$ (= 68,719,476,735). Correct error and start over.

Conversion Time: 100 cards in and 10 cards out takes 35 seconds.

NOTE: It is possible to get a forced punchout in the middle of the input deck by use of a $\delta \rightarrow \varepsilon$ one-word-pulse card, which may also have new desired initial serial number in binary.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

q. Conversion Routine 00234: Dec. to Bin. Commands and Numbers (abs. val. < 1)

Input Formats: numbers 1 and 3. X in col. 34 indicates a negative command or number. X in col. 33 indicates a number; no X in col. 33 indicates a command.

Punch: 36-digit board, n words per card ($n \leq 11$).

To Operate:

1. Read in routine.
2. SWAC halts in (8W): 90,9V,00,10,1, showing U in F of M register.
3. From keyboard, input initial serial number of binary cards in beta, and n in F. (NOTE: Zero in F is treated by the routine as having n equal to ten.) Press 00PB.
4. SWAC halts in (8Y): 90,79,00,00,1.
5. Feed in data. SWAC alternately feeds $n\left[\frac{100}{n}\right]$ decards and punches $\left[\frac{100}{n}\right]$ binary cards containing n words each.
6. If number of decards is not a multiple of $n\left[\frac{100}{n}\right]$: When collator runs out of cards, press Clear Epsilon button and 00PB to punch the final group of binary cards.
7. After last punchout, SWAC halts at 8W, showing n in F of M register and last card serial number + 1 in alpha and beta of M register.

1. Conversion Routines

8. To convert a new deck, input from keyboard new initial serial number of binary cards in beta, and new value of n in F. (NOTE: Zero in F will not be treated as having n equal to ten.) Follow instructions at Step 4 above.

Output Identification: Binary cards are serially numbered in the row containing the last binary word in columns 37 through 44, 57 through 64.

Halts:

1. A halt in (40): 94,04,00,10,1 or (15): 90,04,00,10,1 indicates either
   (a) a decard has too few word pulses, or
   (b) routine is being run with too much delay.
   Correct error and start over.
2. A dynamic halt at 4V, with routine obeying 00,00,00,4V,7 repeatedly, indicates unacceptable data: a 12 (or more) digit number of absolute value greater than .999,999,999,992. Correct error and start over.

Conversion Time: 100 cards in and 10 cards out takes 35 seconds.

NOTE: It is possible to get a forced punchout in the middle of the input deck by use of a $\delta \rightarrow \varepsilon$ one-word-pulse card, which may also have new desired initial serial number in binary.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

r. Conversion Routine 00236: Dec. to Bin. Triple Precision Numbers (abs. val. < 1)

Input Format: number 24.

Output Format: The ten binary words punched out on each binary card are as follows:

1st word: most significant part 2nd word: second most significant part 3rd word: least significant part	first triple precision binary number.
4th word: most significant part 5th word: second most significant part 6th word: least significant part	second triple precision binary number.
7th word: most significant part 8th word: second most significant part 9th word: least significant part	third triple precision binary number.
10th word: zero.	

Punch: 36-digit board, 11 words (rows 9 through 1, X, Y).

To Operate:

1. Read in routine.
2. SWAC halts in (5Y): VY,00,00,10,1.
3. From keyboard, input initial serial number of binary cards in F; press 00PB.
4. SWAC halts in (62): X1,00,00,00,1.
5. Feed in data. SWAC alternately feeds 3 decards and punches 1 ten-word binary card.

1. Conversion Routines

Output Identification: Binary cards are serially numbered in F of the Y-row.

Conversion Time: 3 cards in and 1 card out takes 7 seconds.

1. Conversion Routines

4. Operating Instructions for Decimal to Binary Conversion Routines

s. Conversion Routine 00240: Dec. to Bin. Integers.

Input Format: number 10. X in col. 48 indicates a negative integer.

Punch: 36-digit board, 11 words (rows 9 thru 1, X, Y).

To Operate:

1. Read in routine.
2. SWAC halts in (09): 90,00,00,10,1.
3. From keyboard, input initial serial number of binary cards in F; press 00PB.
4. SWAC halts in (10): 91,00,00,00,1.
5. Feed in data. SWAC alternately feeds 100 decards and punches 10 ten-word binary cards.
6. If number of decards is not a multiple of 100: When collator runs out of cards, press Clear Epsilon button and 00PB to punch the final group of binary cards (fewer than 10 in number).

Output Identification: Binary cards are serially numbered in F of the Y-row.

Halts:

1. A halt in (20) or (0U): 91,00,00,10,1 indicates a decard has an insufficient number of word pulses. Correct faulty card and start over.
2. A halt in (ZZ): 00,00,00,10,1 indicates either
   (a) a machine error, or
   (b) an integer on a decard is larger than 2^{36}, or

1. Conversion Routines

(c) a data card has an X punch in the wrong column. Correct error and start over.

Conversion Time: 100 cards in and 10 cards out takes 35 seconds.

1. Conversion Routines

5. Conversion routine 00219

1) Code no. 00219

2) Written by J. Dowd, October 1954.

3) Binary to decimal conversion routine for numbers whose absolute value < 1.

4) This routine uses the drum for temporary storage of answers in order to gain speed of output. It takes binary cards with 10 words per card and punches the decimal equivalents (11 digits each), 5 per card, together with the appropriate decimal scale factor (from 0 to 9).

5) The routine receives the input data in multiples of 600 words (i.e., multiples of 60 binary cards, 10 words per card). It punches after 600 words have been converted.

6) Operating procedure:

a. Output. Use standard 72-digit reproducer board with the 7 hubs jackplugged (11 rows/card).

b. Breakpoint on, delay off.

c. Drum is used only for storing answers.

d. Standard SWAC collator input board.

e. Machine operation: Read routine into SWAC on continuous. It halts in(002) waiting for scale factor type-in. Type exponent of binary scale factor into the delta position of R register and press 00PB. SWAC calls for data to convert;

1. Conversion Routines

5. Conversion routine 00219

load data in hopper and start collator. Routine will alternately take in 60 cards (binary) and punch out 120 cards (600 decimal answers). If the data is exhausted before a multiple of 60 cards is reached, the routine will halt at (019) waiting for more input. To proceed to the punch out sub-routine: on CONT., press the "$\delta \rightarrow \varepsilon$" button on the keyboard; then press 00PB. After the punch-out the routine will again be ready for more input data. If a new scale factor is desired at this time, do the following: on CONT., press the clear ε button; then press 00PB twice. SWAC will halt at (002) waiting for scale factor type-in. Proceed as above.

f. The conversion of 600 words takes $3\frac{1}{2}$ minutes.

g. A closed loop after reading from drum means that a drum sum check is in error. If this persists, discontinue the routine.

h. Final program halt when there are no more data to convert.

i. Use standard 5 word decimal list board for tabulating.

7) Mathematical method used is similar to code no. 00209.

8) Routine is stored in cells 000-136. Other cells used are listed with the code.

9) Flow chart is with code.

10) Code is in subroutine file.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

a. Conversion Routine 00201: Bin. to Dec. Numbers (scaled 2^s).

Number of word pulses input between collator halts: 10.

Output Format: number 4, with the exponent of the decimal scale factor punched in col. 31. X in col. 29 indicates a negative number.

Punch: 36-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:

1. Read in routine.
2. SWAC halts in (02): 50,00,00,10,1.
3. From keyboard, input s in delta ($0 \leq s \leq 27$), where s binary digits are allotted to the integer part of each number; press 00PB.
4. SWAC halts in (0W): Y0,00,00,00,1.
5. Feed in data. SWAC alternately feeds 10 binary words and punches 10 decards.

Conversion Time: 1 card in and 10 cards out takes 7 seconds.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

b. Conversion Routine 00203: Bin. to Dec. Commands.

Number of word pulses input between collator halts: 10.

Output Format: number 2. X in col. 48 indicates a negative command.

Punch: 36-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:
1. Read in routine.
2. SWAC halts in (04): Y0,00,00,04,1.
3. Feed in data. SWAC alternately feeds 10 binary words and punches 10 decards.

Conversion Time: 1 card in and 10 cards out takes 7 seconds.

Modifications:
1. To vary n ($n \leq 10$), the number of words appearing on each binary card input, let (07) = XZ + n, 00,00,00,1.
2. To vary m, the number of words converted per binary card input, let
(3Y) = XZ + m, 01,ZZ,15,8 .
m and n may be varied independently ($m \leq n$).

NOTE: Column 4 punches on binary cards will not harm routine.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

c. Conversion Routine 00203.1: Bin. to Dec. Commands.

Number of word pulses input between collator halts: 32.

Output Format: number 2. X in col. 48 indicates a negative command.

Punch: 36-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:
1. Read in routine.
2. SWAC halts in (04): X0,00,00,04,1.
3. Feed in data. SWAC alternately feeds 32 binary words and punches 32 decards.

Conversion Time: 4 cards in and 32 cards out takes 20 seconds.

NOTE: Column 4 punches on binary cards will not harm routine.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

d. Conversion Routine 00205: Bin. to Dec. Floating Numbers.

Number of word pulses input between collator halts: 10.

Output format: number 13, three numbers per decard. Every 11th word is the sum of the previous ten (for checking the conversion). (See description of format 13 for the use of X-row.)

Punch: 36-digit board, 11 words (rows 9 through 1, 0, X).

To Operate:
1. Read in routine.
2. SWAC halts at (06): YY,00,00,00,1.
3. Feed in data. a) SWAC feeds 10 binary words and punches out 4 decards; b) SWAC feeds 10 binary words and punches out 4 decards; c) SWAC feeds 10 binary words and punches out 3 decards. Steps a), b), and c) are repeated in that order until all the data has been fed in.

Output Identification: Decards are serially numbered in decimal in col. 76 through 78 beginning with 001.

Halt: A closed loop with epsilon reading 75,76,77 will appear if the word to be converted has zero everywhere except in alpha (zeroes everywhere is acceptable). Remove faulty binary card and start over.

Conversion Time: 3 cards in and 11 cards out takes 10 seconds.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

e. Conversion Routine 00205.1: Bin. to Dec. Floating Numbers.

Number of word pulses input between collator halts: 10.

Output Format: number 13, three numbers per decard. (See description of format 13 for use of the X-row.)

Punch: 36-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:
1. Read in routine.
2. SWAC halts in (06): YY,00,00,00,1.
3. Feed in data. SWAC alternately feeds 10 binary words and punches 3 decards. (Every tenth binary word is ignored.)

Output Identification: Decards are serially numbered in decimal in col. 76 thru 78 beginning with 001.

Halt: A closed loop with epsilon reading 75,76,77 will appear if the word to be converted has zero everywhere except in alpha (zeros everywhere is acceptable). Remove faulty binary card and start over.

Conversion Time: 1 card in and 3 cards out takes 4.5 seconds.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

f. Conversion Routine 00205.2: Bin. to Dec. Floating Numbers.

Number of word pulses input between collator halts: 8.

Output Format: number 13, three numbers per decard. (See description of format 13 for use of the X-row.)

Punch: 36-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:
1. Read in routine.
2. SWAC halts in (06): YY,00,00,00,1.
3. Feed in data. SWAC alternately feeds 8 binary words and punches 3 decards. (Every ninth decimal number is left blank.)

Output Identification: Decards are serially numbered in decimal in col. 76 thru 78 beginning with 001.

Halt: A closed loop with epsilon reading 75,76,77 will appear if the word to be converted has zero everywhere except in alpha (zeros everywhere is acceptable). Remove faulty binary card and start over.

Conversion Time: 1 card in and 3 cards out takes 4.5 seconds.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

g. Conversion Routine 00207: Bin. to Dec. Numbers ($|N| < 1$)

Number of word pulses input between collator halts: 10.

Output Format: number 5.

Punch: 72-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:

1. Read in routine.
2. SWAC halts in (0W): 46,00,00,00,1.
3. Feed in data. SWAC alternately feeds 10 binary words and punches one decard. (The first of every ten binary words is ignored.)

Conversion Time: 1 card in and 1 card out takes 2 seconds.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

h. Conversion Routine 00209: Bin. to Dec. Numbers (scaled 2^s)

Number of word pulses input between collator halts: 10.

Output Format: number 6, with the exponent of the decimal scale factor punched in col. 80.

Punch: 72-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:

1. Read in routine.
2. SWAC halts in (02): Z0,00,00,10,1.
3. From keyboard, input s in delta ($0 \leq s \leq 27$)[1,2] where s binary digits are allotted to the integer part of each number; press 00PB.
4. SWAC halts in (13): Y0,00,00,00,1.
5. Feed in data. SWAC alternately feeds 10 binary words and punches 2 decards, each containing 5 decimal numbers.

Conversion Time: 1 card in and 2 cards out takes 3 seconds.

Modifications: The routine is designed to input k binary words per card, convert n words per binary card, and output t decimal words per decard (punching out i words from the R register, and j words from the M register), each word consisting of m decimal digits.

Normally: $k = 10$, $n = 10$, $t = 5$, $m = 11$, $i = 3$, and $j = 2$.

1. To vary k, let (17) = XZ + k,00,00,00,0 ($k \leq 10$).

1. Conversion Routines

2. To vary n, let (46) = XZ + n,01,ZU,2U,4 (n ≤ 10).

3. To vary m, let (34) = 00,00,00,00,m and (25) = 2^{-m}, (m ≤ 11).

4. To vary t, let (58) = 00,00,00,00,t (t ≤ 10) and (1U) = 00,00,00,00,i and (3X) = 00,00,00,00,j where i + j = t, im ≤ 36, jm ≤ 35 and tm ≤ 70.

---

1 s may actually be as high as 60 but the decimal scale factor will not be punched on the decards if s > 27.

2 During conversion the decimal numbers lose approximately 0.03s significant decimal digits.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

i. Conversion Routine 00209.1: Bin. to Dec. Numbers (scaled 2^s)

Number of word pulses input between collator halts: 10.

Output Format: number 7, with the exponent of the decimal scale factor punched in col. 80.

Punch: 72-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:

1. Read in routine.
2. SWAC halts in (02): Z0,00,00,10,1.
3. From keyboard, input s in delta ($0 \leq s \leq 27$),[1,2] where s binary digits are allotted to the integer part of each number; press 00PB.
4. SWAC halts in (13): Y0,00,00,00,1.
5. Feed in data. SWAC alternately feeds 10 binary words and punches one decard.

Conversion Time: 1 card in and 1 card out takes 2 seconds.

NOTE: This routine is a modified version of 00209 with

$k = 10,\ n = 10,\ t = 10,\ m = 7,\ i = j = 5.$

---

1 s may actually be as high as 60 but the decimal scale factor will not be punched on the decards if $s > 27$.

2 During conversion the decimal numbers lose approximately 0.03s significant decimal digits.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

j. Conversion Routine 00209.2: Bin. to Dec. Numbers (scaled 2^s)

Number of word pulses input between collator halts: 8.

Output Format: number 6, with the exponent of the decimal scale factor punched in col. 80.

Punch: 72-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:

1. Read in routine.
2. SWAC halts in (02): Z0,00,00,10,1.
3. From keyboard, input s in delta ($0 \leq s \leq 27$);[1,2] where s binary digits are allotted to the integer part of each number; press 00PB.
4. SWAC halts in (13): Y0,00,00,00,1.
5. Feed in data. SWAC alternately feeds 8 binary words and punches 2 decards, each containing 5 decimal numbers. (The last two decimal numbers are zero.)

Conversion Time: 1 card in and 2 cards out takes 3 seconds.

NOTE: This routine is a modified version of 00209 with

$k = 8,\ n = 8,\ t = 4,\ m = 11,\ i = 3,\ j = 1.$

---

[1] s may actually be as high as 60 but the decimal scale factor will not be punched on the decards if $s > 27$.

[2] During conversion the decimal numbers lose approximately 0.03s significant decimal digits.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

k. Conversion Routine 00211: Bin. to Dec. Integers

Number of word pulses input between collator halts: 10.

Output Format: number 11.

Punch: 72-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:
1. Read in routine.
2. SWAC halts in (1Z): V0,00,00,00,1.
3. Feed in data. SWAC alternately feeds 10 binary words and punches 2 decards, each containing 5 decimal numbers.

Conversion Time: 1 card in and 2 cards out takes 3 seconds.

Modifications: The routine is designed to input k binary words per card, convert n words per binary card, and output t decimal words per decard (punching out i words from the R register and j words from the M register), each word consisting of m decimal digits.

Normally: $k = 10$, $n = 10$, $t = 5$, $m = 10$, $i = 3$, $j = 2$.

1. To vary k, let (03) = UZ + k,00,00,00,1 ($k \leq 10$).
2. To vary n, let (12) = UZ + n,01,6V,2X,5 ($n \leq 10$).
3. To vary m, let (50) = $10^m.2^{-36}$ and (0Z) = $2^{-(m+1)}$, ($m \leq 10$).

1. Conversion Routines

4. To vary t, let (06) = 00,00,00,00,i

and (13) = 00,00,00,00,j

i+j = t, im ≤ 36 and jm ≤ 36.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

ℓ. Conversion Routine 00213: Bin. to Dec. Commands

Number of word pulses input between collator halts: 10.

Output Format: number 1. X in col. 34 indicates a negative command.

Punch: 36-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:
1. Read in routine.
2. SWAC halts in (04): Y0,00,00,04,1.
3. Feed in data. SWAC alternately feeds 10 binary words and punches 10 decards.

Conversion Time: 1 card in and 10 cards out takes 7 seconds.

NOTE: Column 4 punches on binary cards will not harm routine.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

m. Conversion Routine 00213.1: Bin. to Dec. Commands

Number of word pulses input between collator halts: 32.

Output Format: number 1. X in col. 34 indicates a negative command.

Punch: 36-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:
1. Read in routine.
2. SWAC halts in (04): X0,00,00,04,1.
3. Feed in data. SWAC alternately feeds 32 binary words and punches 32 decards.

Conversion Time: 4 cards in and 32 cards out takes 20 seconds.

NOTE: Column 4 punches on binary cards will not harm routine.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

n. Conversion Routine 00215: Bin. to Dec. Triple Precision Numbers ($|N|>1$)

Number of word pulses input between collator halts: 10, as follows:

1st word:	a binary integer < 10,000. *	
2nd word:	most significant part	first triple precision binary number.
3rd word:	second most significant part	
4th word:	least significant part	
5th word:	blank.	
6th word:	a binary integer < 10,000. *	
7th word:	most significant part	second triple precision binary number.
8th word:	second most significant part	
9th word:	least significant part	
10th word:	blank.	

Output Format: number 25. (See description of format 25 for the use of X-row.)

Punch: 72-digit board, 12 words.

To Operate:

1. Read in routine.
2. SWAC halts in (01): 7Z,00,00,10,1.
3. From keyboard, input initial serial number of binary cards in F; press 00PB.
4. SWAC halts in (04): 64,00,00,00,1.
5. Feed in data. SWAC alternately feeds one binary card and punches one decard.

1. Conversion Routines

Output Identification: Decards are serially numbered in binary in the Y-row, with the least significant digit in column 80.

Conversion Time: 1 card in and 1 card out takes 2 seconds.

---

* May be used for identification or for any other purpose.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

o. Conversion Routine 00219: Bin. to Dec. Numbers (scaled 2^s)

Number of word pulses input between collator halts: 10.

Output Format: number 6, with the exponent of the decimal scale factor punched in col. 80.

Drum: ON.

Punch: 72-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:

1. Read in routine.
2. SWAC halts in (02): Z0,00,00,10,1.
3. From keyboard, input s in delta $(0 \leq s \leq 27)$,[1,2] where s binary digits are allotted to the integer part of each number; press OOPB.
4. SWAC halts in (13): Y0,00,00,00,1.
5. Feed in data. SWAC alternately feeds 600 binary words and punches 120 decards, each containing five decimal numbers.
6. If number of binary words is not a multiple of 600, when collator runs out of cards press $\delta \rightarrow \varepsilon$ button and OOPB to punch the final group of decards.
7. After last punchout, SWAC halts in (13); see step 4. (If a new scale factor is desired at this time, while on CONTINUOUS press the Clear Epsilon button; then press OOPB twice. SWAC will halt at (02); see step 2.)

Halt: A closed loop with epsilon reading 62, 63, 78, 7U, 7V, 7W, 7Y, and 7Z indicates failure of

1. Conversion Routines

drum sum check. See command in cell 62 for channel number.

Conversion Time: 60 cards in and 120 cards out takes 88 seconds.

Modifications: The routine is designed to input k binary words per card, convert n words per binary card, and output t decimal words per decard (punching out i words from the R register, and j words from the M register), each word consisting of m decimal digits.

Normally: $k = 10$, $n = 10$, $t = 5$, $m = 11$, $i = 3$, and $j = 2$.

1. To vary k, let (17) = XZ + k,00,00,00,0 ($k \leq 10$).
2. To vary n, let (46) = XZ + n,01,ZU,2U,4 ($n \leq 10$).
3. To vary m, let (34) = 00,00,00,00,m and (25) = 2^{-m}, ($m \leq 11$).
4. To vary t, let (58) = 00,00,00,00,t ($t \leq 10$) and (1U) = 00,00,00,00,i and (3X) = 00,00,00,00,j where $i + j = t$, $im \leq 36$, $jm \leq 35$ and $tm \leq 70$..

NOTE: This routine is a modified version of routine 00209 wherein the input and output have been speeded up by making use of the drum.

---

1 s may actually be as high as 60 but the decimal scale factor will not be punched on the decards if $s > 27$.

2 During conversion the decimal numbers lose approximately 0.03s significant decimal digits.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

p. Conversion Routine 00219.1: Bin. to Dec. Numbers (scaled 2^s)

Number of word pulses input between collator halts: 10.

Output Format: number 6, with the exponent of the decimal scale factor punched in col. 80.

Drum: ON.

Punch: 72-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:

1. Read in Part I of routine, Scale Factor card,* Part II of routine.
2. SWAC halts in (02): Z0,00,00,10,1.
3. From keyboard, input s in delta ($0 \leq s \leq 27$),[1,2] where s binary digits are allotted to the integer part of each number whose special scale factor is unity; press 00PB.
4. SWAC halts in (13): Y0,00,00,00,1.
5. Feed in data. SWAC alternately feeds 600 binary words and punches 120 decards, each containing five decimal numbers.
6. If number of binary words is not a multiple of 600, when collator runs out of cards press $\delta \rightarrow \varepsilon$ button and 00PB to punch the final group of decards.
7. After last punchout, SWAC halts in (13); see step 4. (If a new scale factor is desired at this time, while on CONTINUOUS press the Clear Epsilon button; then press 00PB twice. SWAC will halt in (02); see step 2.)

1. Conversion Routines

Halt: A closed loop with epsilon reading 62, 63, 78, 7U, 7V, 7W, 7Y, and 7Z indicates failure of drum sum check. See command in cell 62 for channel number.

Conversion Time: 60 cards in and 120 cards out takes 88 seconds.

Modifications: The routine is designed to input k binary words per card, convert n words per binary card, and output t decimal words per decard (punching out i words from the R register, and j words from the M register), each word consisting of m decimal digits.

Normally: $k = 10$, $n = 10$, $t = 5$, $m = 11$, $i = 3$, and $j = 2$.

1. To vary k, let (17) = XZ+k,00,00,00,0, ($k \leq 10$).
2. To vary n, let (46) = XZ+n,01,ZU,2U,4, ($n \leq 10$).
3. To vary m, let (34) = 00,00,00,00,m and (25) = 2^{-m}, ($m \leq 11$).
4. To vary t, let (58) = 00,00,00,00,t ($t \leq 10$) and (1U) = 00,00,00,00,i and (3X) = 00,00,00,00,j where $i+j = 5$, $im \leq 36$, $jm \leq 35$ and $tm \leq 70$.

NOTE: This routine is a modified version of routine 00219, allowing for ten separate scale factors.

---

* To be prepared by the coder as explained in description of routine 00219.1.

1 s may actually be as high as 60 but the decimal scale factor will not be punched on the decards if $s > 27$.

2 The numbers resulting from multiplication by their special scale factors will further lose approximately 0.03s significant decimal digits during conversion.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

q. Conversion Routine 00221: Bin. to Dec. Numbers (scaled 2^s)

Number of word pulses input between collator halts: 10.

Output Format: number 7, with the exponent of the decimal scale factor punched in col. 80.

Drum: ON.

Punch: 72-digit board, 11 words (rows 9 thru 1, 0, X).

To Operate:

1. Read in routine.
2. SWAC halts in (02): Z0,00,00,10,1.
3. From keyboard, input s in delta ($0 \leq s \leq 27$),[1,2] where s binary digits are allotted to the integer part of each number press 00PB.
4. SWAC halts in (13): Y0,00,00,00,1.
5. Feed in data. SWAC alternately feeds 1200 binary words and punches 120 decards, each containing ten decimal numbers.
6. If number of binary words is not a multiple of 1200, when collator runs out of cards press $\delta \rightarrow \varepsilon$ button and 00PB to punch the final group of decards.
7. After last punchout, SWAC halts in (13); see step 4. (If a new scale factor is desired at this time, while on CONTINUOUS press the Clear Epsilon button; then press 00PB twice. SWAC will halt at (02); see step 2.)

Halt: A closed loop with epsilon reading 62, 63, 78, 7U, 7V, 7W, 7Y, and 7Z indicates failure of

1. Conversion Routines

drum sum check. See command in cell 62 for channel number.

Conversion Time: 120 cards in and 120 cards out takes 103 seconds.

NOTE: This routine is a modified version of routine 00209.1 wherein the input and output have been speeded up by making use of the drum.

---

[1] s may actually be as high as 60 but the decimal scale factor will not be punched on the decards if s > 27.

[2] During conversion the decimal numbers lose approximately 0.03s significant decimal digits.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

r. Conversion Routine 00223: Bin. to Dec. Commands

THIS ROUTINE IS USED FOR DIRECT TABULATOR OUTPUT*

Number of word pulses input between collator halts: 10.

Tabulator: 1. Tabulator Board number 1.

2. Carriage Tape labelled "Decimal listing, 2 groups of 10; double spaced".

3. Expected output: one decimal command per line identified by a cell number (generated by the tabulator), 20 lines per page. See sample output in Conversion Routines Handbook (SWAC room).

Breakpoint Switch: OFF.

To Operate: 1. Start feeding arbitrary deck of cards into the tabulator feed, until first card enters stacker.

2. Reset carriage tape, and set listing paper one line above desired starting list position.

3. Feed routine into SWAC collator.

4. SWAC halts in (04): Y0,00,00,04,1.

5. a) Start feeding data into SWAC collator; SWAC takes in 10 binary words.

b) Tabulator lists one spurious decimal zero word, and the first 9 decimal converted words.

6. a) SWAC feeds 10 binary words.

b) Tabulator lists the tenth decimal word converted from the 10 binary words

1. Conversion Routines

previously taken in.

c) Paper skips a few spaces.

d) Tabulator lists 9 decimal converted words.

e) Return to 6a.

7. a) When SWAC collator runs out of cards, press OOPB ten times.

b) Tabulator lists the last decimal converted word.

c) Paper skips a few spaces.

d) Tabulator lists 9 spurious zeroes.

e) After last tabulator output, SWAC halts in (04): Y0,00,00,04,1.

8. To start over at step 1, and also start the cell-numbering at zero:

a) At the SWAC, press the Clear Epsilon button, and press OOPB twice.

b) At the tabulator, lift the deck of unused cards from the hopper and run out all cards to the stacker by holding the Start button.

c) Return to steps 1,2 (skip 3 and 4), 5a, etc.

Halt: A halt with C register exhibiting --,--,--,10,2 indicates either:

a) Tabulator has run out of cards. At the tabulator, press the STOP button; add cards to the hopper; press the FINAL TOTAL

1. Conversion Routines

button.

or

b) Tabulator may be out of paper. Press the tabulator STOP button; replenish paper supply; press the FINAL TOTAL button.

Conversion Time: 2 cards in and 20 lines out takes 18 seconds.

NOTES:
1. Column 4 punches on binary cards will not harm routine.
2. This routine is a version of routine 00203 with the input and output altered to make use of the tabulator output.

---

* After using tabulator for direct output from SWAC, operator must either turn off tabulator or remove tabulator board; unless this is done, SWAC will not behave properly on subsequent output.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

s. Conversion Routine 00229: Bin. to Dec. Numbers (scaled 2^s)

THIS ROUTINE IS TO BE USED FOR DIRECT TABULATOR OUTPUT*

Number of word pulses input between collator halts: 10.

Tabulator: 1. Tabulator Board number 2.

2. No Carriage Tape required; tape brushes may be raised.

3. Expected output: five decimal words per line, 33 lines per page. See sample output in Conversion Routines Handbook (SWAC room).

To Operate: 1. Start feeding arbitrary deck of cards into the tabulator feed, until the first card enters stacker.

2. Set listing paper two lines above desired starting list position.

3. Feed routine into SWAC collator.

4. SWAC halts in (02): Z0,00,00,10,1.

5. From keyboard, input s in delta ($0 \leq s \leq 27$);[1,2] where s binary digits are allotted to the integer part of each number press 00PB.

6. SWAC halts in (13): 64,00,00,00,1.

7. a) Start feeding data into SWAC collator; SWAC takes in 10 binary words.

b) Tabulator lists two lines of 5 spurious zero words each.

8. a) SWAC feeds 10 binary words.

b) Tabulator lists two lines of 5 decimal words each, converted from the 10 binary words

1. Conversion Routines

previously fed in.

c) Return to 8a.

9. a) When SWAC collator runs out of cards, press 00PB ten times.

b) Tabulator lists two lines of 5 decimal words each, converted from the last 10 binary data words.

c) After last tabulator output, SWAC halts at (13): 64,00,00,00,1.

Halt: A halt with C register exhibiting --,--,--,10,2 indicates either:

a) Tabulator has run out of cards. At the tabulator, press the STOP button; add cards to the hopper; press the FINAL TOTAL button.

or

b) Tabulator may be out of paper. Press the tabulator STOP button; replenish paper supply; press the FINAL TOTAL button.

Conversion Time: 10 cards in and 20 lines out takes 24 seconds.

Modifications: The routine is designed to input k binary words per card, convert n words per binary card, and output t decimal words per decard (printing out i words from the R register, and j words from the M register), each word consisting of

1. Conversion Routines

m decimal digits.

Normally: k = 10, n = 10, t = 5, m = 11,

i = 3, and j = 2.

1. To vary k, let (17) = 63+k,00,00,00,0 (k ≤ 10).
2. To vary n, let (46) = XZ+n,01,ZU,2U,4 (n ≤ 10).
3. To vary m, let (34) = 00,00,00,00,m and
   (25) = 2^{-m}, (m ≤ 11).
4. To vary t, let (58) = 00,00,00,00,t, (t ≤ 10)
   and (1U) = 00,00,00,00,i
   and (3X) = 00,00,00,00,j
   where i+j = t, im ≤ 36, jm ≤ 35 and tm ≤ 70.

NOTE: This routine is a version of routine 00209 with the input and output altered to make use of the tabulator output.

---

1 s may actually be as high as 60 but the decimal scale factor will not be printed if s > 27.

2 During conversion the decimal numbers lose approximately 0.03s significant decimal digits.

* After using tabulator for direct output from SWAC, operator must either turn off tabulator or remove tabulator board; unless this is done, SWAC will not behave properly on subsequent output.

1. Conversion Routines

5. Operating Instructions for Binary to Decimal Conversion Routines

t. Conversion Routine 00231: Bin. to Dec. Integers

THIS ROUTINE IS TO BE USED FOR DIRECT TABULATOR OUTPUT*

Number of word pulses input between collator halts: 10.

Tabulator: 1. Tabulator Board number 2.

2. No Carriage Tape required; tape brushes may be raised.

3. Expected output: five decimal words per line, 33 lines per page. See sample output in Conversion Routines Handbook (SWAC room).

To Operate: 1. Start feeding arbitrary deck of cards into the tabulator feed, until the first card enters stacker.

2. Set listing paper two lines above desired starting list position.

3. Feed routine into SWAC collator.

4. SWAC halts in (1Z): VU,00,00,00,1.

5. a) Start feeding data into SWAC collator; SWAC takes in 10 binary words.

b) Tabulator lists two lines of 5 spurious zero words each.

6. a) SWAC feeds 10 binary words.

b) Tabulator lists two lines of 5 decimal words each, converted from the 10 binary words previously fed in.

c) Return to 6a.

1. Conversion Routines

7. a) When SWAC collator runs out of cards, press 00PB ten times.

b) Tabulator lists two lines of 5 decimal words each, converted from the last 10 binary data words.

c) After last tabulator output, SWAC halts at (1Z): VU,00,00,00,1.

Halt: A halt with C register exhibiting --,--,--,10,2 indicates either:

a) Tabulator has run out of cards. At the tabulator, press the STOP button; add cards to the hopper; press the FINAL TOTAL button.

or

b) Tabulator may be out of paper. Press the tabulator STOP button; replenish paper supply; press the FINAL TOTAL button.

Conversion Time: 10 cards in and 20 lines out takes 24 seconds.

Modifications: The routine is designed to input k binary words per card, convert n words per binary card, and output t decimal words per decard (printing out i words from the R register and j words from the M register), each word consisting of m decimal digits.

1. Conversion Routines

Normally: $k = 10$, $n = 10$, $t = 5$,

$m = 10$, $i = 3$, $j = 2$.

1. To vary k, let (03) = V9+k,00,00,00,1, ($k \leq 10$).

2. To vary n, let (12) = UZ+n,01,6V,2X,5, ($n \leq 10$).

3. To vary m, let (50) = $10^m.2^{-36}$ and (0Z) = $2^{-(m+1)}$, ($m \leq 10$).

4. To vary t, let (06) = 00,00,00,00,i and

(13) = 00,00,00,00,j

$i+j = t$, $im \leq 36$ and $jm \leq 36$.

NOTE: This routine is a version of routine 00211 with the input and output altered to make use of the tabulator output.

---

* After using tabulator for direct output from SWAC, operator must either turn off tabulator or remove tabulator board; unless this is done, SWAC will not behave properly on subsequent output.

1. Conversion Routines

6. Conversion routine 00221

1) Code no. 00221

2) Written by J. Dowd, October 1954.

3) Binary to decimal conversion routine for numbers whose absolute value < 1.

4) This routine uses the drum for temporary storage of answers in order to gain speed of output. It takes binary cards with 10 words per card and punches the decimal equivalents, (7 digits each) 10 per card, together with the appropriate decimal scale factor (from 0 to 9).

5) The routine receives the input data in multiples of 1200 words (i.e., multiples of 120 binary cards, 10 words per card). It punches after 1200 words have been converted.

6) Operating procedure:

a. Output. Use standard 72-digit reproducer board with the 7 hubs jackplugged (11 rows/card).

b. Breakpoint on, delay off.

c. Drum is used only for storing answers.

d. Standard SWAC collator input board.

e. Machine operation: Read routine into SWAC on continuous. It halts in (002) waiting for scale factor type-in. Type exponent of binary scale factor into the delta position of R register and press 00PB. SWAC calls for data to convert;

1. Conversion Routines

6. Conversion routine 00221

load data in hopper and start collator. Routine will alternately take in 120 cards (binary) and punch out 120 cards (1200 decimal answers). If the data is exhausted before a multiple of 120 cards is reached, the routine will halt at (019) waiting for more input. To proceed to the punch out sub-routine; on CONT., press the "$\delta \rightarrow \varepsilon$" button on the keyboard; then press 00PB. After the punch-out the routine will again be ready for more input data. If a new scale factor is desired at this time, do the following; on CONT., press the clear ε button; then press 00PB twice. SWAC will halt at (002) waiting for scale factor type-in. Proceed as above.

f. The conversion of 1200 words takes $4\frac{1}{2}$ minutes.

g. A closed loop after reading from drum means that a drum sum check is in error. If this persists, discontinue the routine.

h. Final program halt when there are no more data to convert.

i. Use standard 10 word decimal list board for tabulating.

7) Mathematical method used is similar to code no. 00209.

8) Routine is stored in cells 000-136. Other cells used are listed with the code.

9) Flow chart is with code.

10) Code is in subroutine file.

1. Conversion Routines

6. Operating Instructions for Miscellaneous Routines

a. Conversion Routine 00217: Fixed Binary Point to Floating Binary Point Numbers.

Number of word pulses input between collator halts: 10.

Punch: 36-digit board, 10 words.

To Operate: 1. Read in routine.

2. SWAC halts in (02): ZZ,00,00,10,1.

3. From keyboard, input into α the exponent k ($|k| < 92$) of the binary scale factor by which the data must be multiplied before conversion; press 00PB.

4. SWAC halts in (04): 46,00,00,00,1.

5. Feed in data. SWAC alternately feeds 10 binary words and punches 1 ten-word binary card.

Conversion Time: 1 card in and 1 card out takes 2 seconds.

1. Conversion Routines

6. Operating Instructions for Miscellaneous Conversion Routines

b. Conversion Routine 00222: Floating Binary Point to Fixed Binary Point Numbers.

Number of word pulses input between collator halts: r (see step 3 below).

Punch: 36-digit board, r words ($r < 11$).

To Operate

1. Read in routine.
2. SWAC halts in (02): ZX,00,00,10,1.
3. From keyboard, input r ($r < 11$) into delta (r is the number of binary words per input card). Press 00PB.
4. SWAC halts in (03): ZW,00,00,10,1.
5. From keyboard, input into delta the "maximum alpha", i.e. the maximum binary number which appears in alpha on any of the data cards (= 128 + floating binary exponent). Press 00PB.
6. SWAC halts in (04): ZV,00,00,10,1.
7. From keyboard, input into F the initial serial number of binary cards. Press 00PB.
8. SWAC halts in (07): 64,00,00,00,1.
9. Feed in data. SWAC alternately feeds r binary words and punches 1 binary card containing r binary words.

Output Identification:

a) Output binary cards are serially numbered in columns 57 to 64 of the last row punched.

1. Conversion Routines

b) The "maximum alpha" input at step 5 above appears in binary notation in the last row punched, columns 37 through 44.

Halt: A halt in (16): 01,00,00,10,1 indicates an input word exceeds the erroneous "maximum alpha" input at step 5 above. Press the Clear Epsilon button; press 00PB twice. Routine halts in (02); see step 2 above. (NOTE: The data card which caused the error halt will not be converted unless the operator can cause it to be the first card fed through the collator at step 9.)

Conversion Time: 1 card in and 1 card out takes 1+ seconds.

1. Conversion Routines

6. Operating Instructions for Miscellaneous Routines

c. Conversion Routine 00225: Binary to Base 4

Number of word pulses input between collator halts: n (see step 3 below).

Output Format: number 23.

Punch: 36-digit board, 6 words (rows 3, 2, 1, 0, X, Y).

To Operate:

1. Read in routine.
2. SWAC halts in (04): 50,00,00,00,1.
3. Either
   a) From keyboard, input n (n < 11) into alpha and press 00PB*, or
   b) Feed an "n in alpha" card into the collator. n is the number of binary words per binary input card.
4. SWAC halts in (08): Z0,00,00,00,1.
5. Feed in data. SWAC alternately feeds n binary words and punches n cards.

Conversion Time: 1 card in and n cards out takes (1 + 0.6n) seconds.

*Warning: If collator is idling, and data cards are in the hopper, improper operation will result.

1. Conversion Routines

6. Operating Instructions for Miscellaneous Routines

d. Conversion Routine 00226: Hexadecimal to Binary

Input Format: number 21.

Collator: eleven words per card. (Add 0-row jackplug.)

Punch: 36-digit board, 10 words.

To Operate:

1. Read in routine.
2. SWAC halts in (02): 4V,00,00,10,1.
3. From keyboard, input into beta the initial serial number of binary cards, and into alpha any desired identification. Press 00PB.
4. SWAC halts in (06): 5U,00,00,00,1.
5. Feed in data. SWAC alternately feeds 10 hexadecimal data cards and punches one binary card.

Output Identification: Binary cards are serially numbered in the row containing the last binary word, in columns 57 through 64.

Identification input into alpha at step 3 appears in the row containing the last binary word, in columns 37 through 44.

Conversion Time: 10 cards in and 1 card out takes 6.8 seconds.

1. Conversion Routines

6. Operating Instructions for Miscellaneous Routines

e. Conversion Routine 00227: Binary to Hexadecimal

Number of word pulses input between collator halts: 10.

Output Format: number 26.

Punch: 36-digit board, 11 words (rows 9 through 1, 0, X).

To Operate:
1. Read in routine.
2. SWAC halts in (04): 5U,00,00,04,1.
3. Feed in data. SWAC alternately feeds 10 binary words and punches 10 hexadecimal cards.

Conversion Time: 1 card in and 10 cards out takes 7 seconds.

NOTE: Column 4 punches on binary cards will not harm routine.

1. Conversion Routines

6. Operating Instructions for Miscellaneous Routines

f. Conversion Routine 00228: Octal to Binary

Input Format: number 22.

Punch: 36-digit board, 11 words (rows 9 thru 1, X, Y).

To Operate:

1. Read in routine.
2. SWAC halts in (09): 90,00,00,10,1.
3. From keyboard, input into F the initial serial number of binary cards; press 00PB.
4. SWAC halts in (10): 91,00,00,00,1.
5. Feed in data. SWAC alternately feeds 100 octal cards and punches 10 binary cards.
6. If number of octal cards is not a multiple of 100, when collator runs out of cards press Clear Epsilon button and 00PB to punch the final group of binary cards.

Output Identification: Binary cards are serially numbered in F of the Y-row.

Conversion Time: 100 cards in and 10 cards out takes 32 seconds.

2. Interpretation and Subroutines

1. Interpretation Routine (File No. 00300)

The SWAC does not have incorporated into its arithmetic system a logical divide, square root, logarithm, etc., therefore routines must be made to do these operations using the fundamental operations of the SWAC. These routines are called subroutines. The subroutines are coded with the operands located in fixed cells and the answers going to fixed cells. In most problems the operands and answers are numerous and therefore cannot be stored in fixed cells used by the subroutine; thus the "Coded Command" and the Interpretation routine are required.

A "Coded Command" is a command specifying a transformation which, in general, is more complicated than the fundamental operations of the SWAC. A coded command may specify some special arithmetic operation such as divide, square root, logarithm, etc. Therefore, in conjunction with an Interpretation routine and subroutines, it is possible to indicate a special operation by one command - the "Coded Command." In general, alpha of the coded command is the address of the first operand; beta, the address of the second operand; gamma, the address of the answer; delta, the exit from the Interpretation routine; and F, the entry into the desired subroutine. To indicate that a command is a "Coded Command" its

2. Interpretation and Subroutines

sign is negative; otherwise the command will be obeyed explicitly in cell (022).

The Interpretation routine interprets a "Coded Command" by sending each operand from any memory cell to the cell used by the subroutine and the answer from the subroutine to any cell. The coded command is supplied by the main routine which is interpreted by the Interpretation routine.

Number of cells: 37, located in cells 000 thru 036.

Cells subject to modification: 012, 013, 022.

Temporaries: 003 thru 009.

Constants: 028 thru 036. (028) = 2^{-1}, (029) = zero, (030) = F extractor, (031) = gamma-delta extractor, (032) = beta extractor, (033) = alpha extractor, (034) = 2^{-36}, (035) = 2^{-24}, (036) = 2^{-8}.

Remarks: Subroutine operands are sent to (003) and (004) by the Interpretation routine. Subroutine answers go to (008). Subroutines with two word answers send results to (008) and (007). Subroutines halt in (002) in event of error, i.e., exceed capacity, denominator zero, etc.

To use Interpretation: It is generally advisable to code the main routine in a block (or sequence) and the coded words in another block to minimize the range of k, the cell number of the word used in the Interpretation routine. Then the Interpretation routine will use only the words in the coded word block.

The main routine must prestore the word in (013) of the Interpretation routine before the initial entry into the Interpretation routine. The word in (013) should be k_o 029 022 022 08; where k_o is the cell number of the first word to be used by the Interpretation routine. In general, k is increased by 1 by the Interpretation routine, therefore for any deviation of sequence of words to be used by the routine,

2. Interpretation and Subroutines

k of (013) must be adjusted by the main routine. The S of word (024) is the cell of the first subroutine entry. This must be put into (024) before using the Interpretation routine.

Entry-exit sequence: (Coded Command)

Main routines enter Interpretation routine at (013). The Interpretation routine enters the subroutine at S + entry of the coded command. Subroutines exit to (010) of the Interpretation routine. The Interpretation exits via (012), which is a special command containing the exit of the coded command in delta. This exit in (012) may be to return to the main routine or to re-enter the Interpretation routine at (013), which will interpret the next command.

Breakpoint: The exit from the subroutine to (010) causes the routine to halt at the breakpoint of (010) displaying the words in cells alpha and beta of (010) in the R and M Registers, respectively. The next command (011) is also a breakpoint. Alpha and beta of (010) and (011) may be changed at the option of the coder. These breakpoints are used in checking out a new routine. Once the routine is checked out, the subroutine exits are changed to (012), thus leaving (010) and (011) available at the option of the coder. If left as breakpoints, they are useful in determining the reason for a halt in (002).

The routine: (All words are written in decimal command form.)

Cell No.	α	β	γ	δ	F	Remarks
000	001	000	000	016	01	Typewriter input to 001, main routine halt.
001	000	000	000	000	00	Temporary
002	001	000	000	016	01	Subroutine halt.
003	000	000	000	000	00	Temporary

2. Interpretation and Subroutines

Cell No.	α	β	γ	δ	F	Remarks
004	000	000	000	000	00	Temporary
005	000	000	000	000	00	"
006	000	000	000	000	00	"
007	000	000	000	000	00	"
008	000	000	000	000	00	"
009	000	000	000	000	00	"
010	000	022	000	096	02	Breakpoint α in R, β in M.
011	000	008	000	096	02	Breakpoint α in R, β in M.
012	008	029	[addr. of result	exit]	05	Answer → address of result.
013	[k]	029	022	022	08	Is (k) ≥ 0? (k) → (022).
014	033	022	006	064	14	(k) < 0; (k) is coded; 2^{-8} α of (k)→(006).
015	006	025	003	003	05	First operand → (003).
016	032	022	007	008	14	2^{-8} β of (k) → (007).
017	007	026	004	004	05	Second operand → (004).
018	031	022	005	064	14	2^{-24} γ and 2^{-32} δ of (k)→(005).
019	005	027	012	000	04	[008, 029, address of result, exit, 05]→ (012).
020	030	022	008	004	14	2^{-32} entry → (008).
021	008	024	008	008	05	[036 013 013, entry+S, 05]→(008).
022		Command or Coded Command				(k) > 0; (k) not coded; obey (022) = (k).
023	036	013	013	013	05	k + 1 → k.
024	036	013	013	S	05	Dummy for (008), k + 1 → k and enter subroutine.
025	000	029	003	016	05	Dummy for (003), (α) → (003).
026	000	029	004	018	05	Dummy for (004), (β) → (004).

2. Interpretation and Subroutines

Cell No.	α	β	γ	δ	F	Remarks
027	008	029	000	000	05	Dummy for (012), (008) → address of result.
028	128	000	000	000	00	2^{-1}
029	000	000	000	000	00	Zero storage.
030	-255	255	255	255	00	F extractor.
031	-255	255	000	000	15	$\gamma - \delta$ extractor.
032	-255	000	255	255	15	β extractor.
033	-000	255	255	255	15	α extractor.
034	000	000	000	000	01	2^{-36}
035	000	000	001	000	00	2^{-24}
036	001	000	000	000	00	2^{-8}

Coded Command — [Address of 1st Operand | Address of 2nd Operand | Address of Answer* | Exit | Entry]

* Subroutines having two answers must transfer the second answer from (007) to the desired cell. The answer in (008) is transferred by command (012) of the Interpretation routine.

2. Interpretation and Subroutines

3. SWACPEC

This method of coding adapts the SWAC to operations analogous to those performed by a CPEC* with a general purpose board. A sequence of arithmetic and logical instructions is programmed, and each operation punched on an IBM card. These instructions appear finally (after suitable processing) as binary words (or numbers) punched 10 to a card. A basic routine in the memory of the SWAC will interpret these commands as they are fed through the input reader of the SWAC (actually a modified IBM collator) at a speed approximately ten times CPEC speed (about 1200 instructions/min.).

The coder programs in the decimal system, and the decimal instructions and numbers (data) are punched on cards just as in the CPEC system. The conversion of these decimal cards to binary is performed by the SWAC, as a preliminary step in the preparation of a suitable control deck.

The coding sheets have been blocked off in groups of ten, so that the coder may see which groups of decimal cards will appear as one binary card.

---

*IBM Card Programmed Electronic Calculator.

2. Interpretation and Subroutines

I. Command Form

1. A command is an instruction which tells the machine where to get numbers, what to do with them, and where to put the result. It is written in the following form ABCSDF±, where

A and B are the addresses of the operands (3 decimal digits)

C is the address of the result (3 digits)

S is the shift (2 digits)

D is a special instruction (2 digits)

F is the functional operation (2 digits)

± is the algebraic sign of the command

Also coded:

N: An x indicates a number; zero or blank a command.

This is for use in the binary conversion routine.

SRI punch:

For the spread read in data. This is a "9" punch in column four on a card containing 10 SRI numbers. (Punched by hand by coder on the binary card, after it is prepared by SWAC.)

2. A, B, and C will in general contain the numbers 096-143, 192-201, 001, 147-150 or 255.

Numbers are stored in the addresses 096-143, inclusive. Addresses of the spread read in (SRI) numbers are 192-201. Storages 192-201 may be used as auxiliary storage but they must be cleared before using a SRI code.

2. Interpretation and Subroutines

255 code is the A or B instruction to read in from the card. The numbers to be read in as A must be in the line following the command.

The number to be read in as B must be on the second line following the command. Both of the numbers must be on the same card as the command, and clearing must be specified. 001 and 147-150 are the addresses of constants.

3. S specifies the shift. "03" is a normal shift. "02", "01", and "00" will shift the number to the right, and give one, two, or three less decimals, respectively. "04", "05", and "06" are left shifts and will give one, two, or three more decimals, respectively.

If an overflow is expected in addition, the numbers must be shifted before they are added. If an overflow occurs in addition when a right shift is specified, the most significant digit will be lost. The SWAC will halt, first typing out the command which failed and the answer, less the most significant digit.

Rounding occurs for all right and normal shifts. If A has i decimals, and B has j decimals, for a "03" shift, we get:

Operation		Decimals in result
A + B	i=j	i
A x B		i + j - 10
A / B		i - j + 10
$\sqrt{A}$	i=2n	(i+10)/2
sin B		10

2. Interpretation and Subroutines

4. D is a special instruction, from 00 to 15, which specifies the following:

No Clearing	Clear A	Clear B	Clear A and B	
00	01	02	03	non-accumulate, no type-out.
04	05	06	07	accumulate*, no type-out
08	09	10	11	non-accumulate, type out C in decimal and command in hexadecimal
12	13	14	15	accumulate*, type out C in decimal and command in hexadecimal

*symbolically, (AB) + C → C. The result of operating on A and B will be added to C before the result is stored in C. Shifting occurs before the result is added to C.

5. Functional operation (F):

	F
A + B	01
A - B	12
A x B	03
A / B	04
$\sqrt{A}$	05
| A |	06
Balance Test	07
Read from drum	09
sin B	10
cos B	11
output	02

2. Interpretation and Subroutines

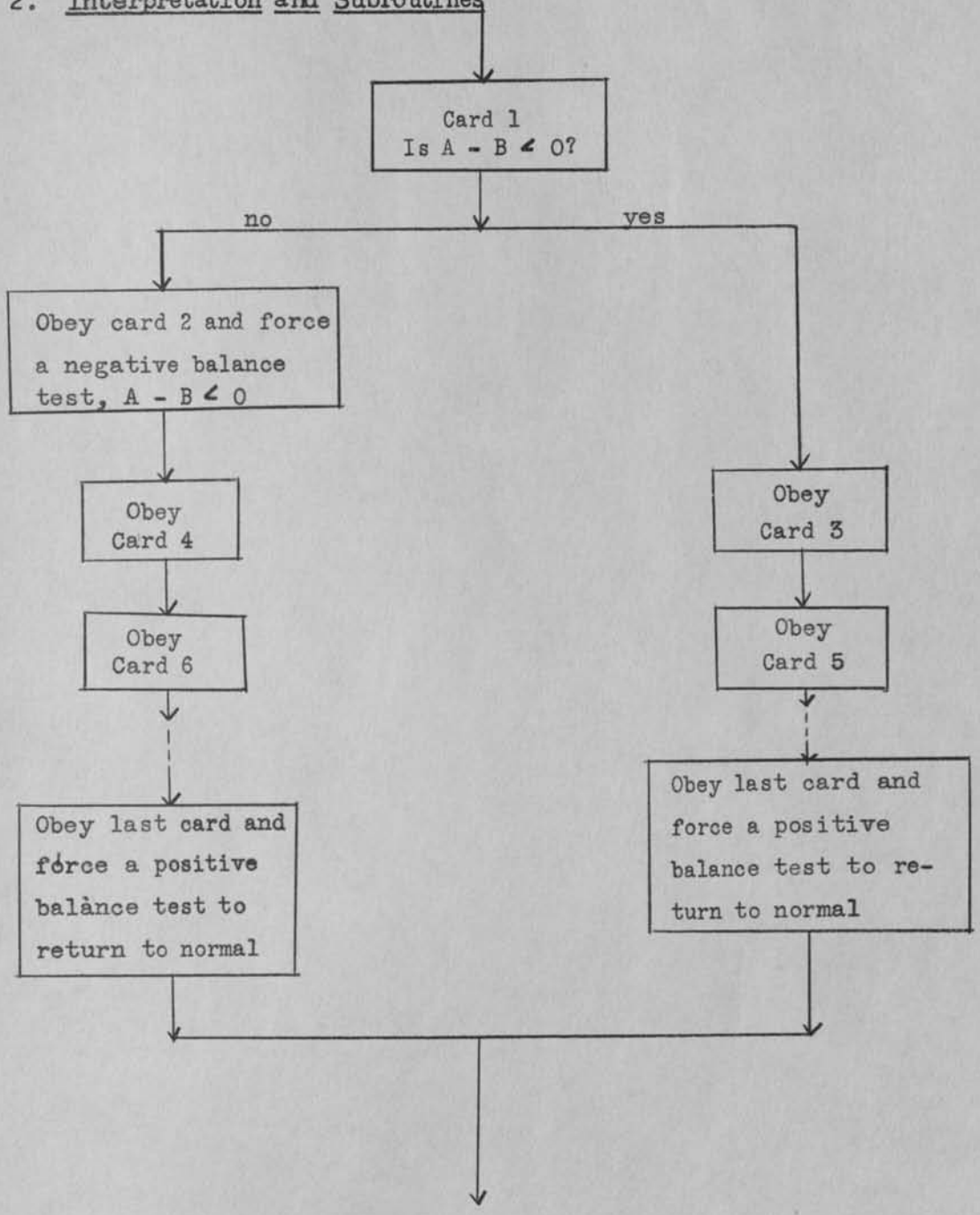
II. Actual Forms of Commands

If there is a number specified here, then no substitutions are available to the coder.

Operation	± Sign					F	Remarks
$\pm(A + B) \rightarrow C$	±	A	B	C	S	D	01	
$\pm(A - B) \rightarrow C$	±	A	B	C	S	D	12	
$\pm(A \times B) \rightarrow C$	±	A	B	C	S	D	03	
$\pm(A / B) \rightarrow C$	±	A	B	C	S	D	04	
$\pm \sqrt{A} \rightarrow C$	±	A	066	C	S	D	05	
$\pm \lvert A \rvert \rightarrow C$	±	A	001	C	S	D	06	
Is $A - B < 0$?		A	B	218	00	09	07	See note 1
$\pm \sin B \rightarrow C$	±	P	B	C	S	D	10	See note 2
$\pm \cos B \rightarrow C$	±	P	B	C	S	D	11	See note 2
to force BT +		001	001	218	00	09	07	
to force BT -		001	000	218	00	09	07	

Note 1. If $A - B < 0$, then the following card, containing ten commands, and every second card thereafter will be skipped until the balance test is positive, i.e., $A - B \geq 0$. In order to make a branch, therefore, we must do the following:

2. Interpretation and Subroutines



2. Interpretation and Subroutines

Note 2. P is the address (147-150) of the power of ten which indicates the decimal point of B (see list of constants). B is normally a ten decimal number (with a "03" shift) with eight good decimals.

III. Built-in Constants.

Constant	Address
zero	001
10^0	147
10^1	148
10^2	149
10^3	150

IV. Spread Read In.

This term is borrowed from CPEC coding. It means that all ten numbers on one card are to be loaded into storage, and this card is known as a SRI card. If a binary card contains a SRI punch (a 9 in column 4) then the ten numbers will be read into storages 192-201 in order: the first number going into 192, the second into 193, etc.

The SRI punch must be put on after the numbers are converted into binary, and storages 192 to 201 must be clear before using the SRI code. If less than ten numbers are required, the coder must specify zeros to fill out the set of ten.

2. Interpretation and Subroutines

V. Output

Operation	Command
punch numbers in 101-110 on one card, in binary	101 000 000 05 00 02
type numbers in 101-110, in decimal	101 074 000 00 00 02
punch numbers in 101-110 on tape, in decimal	101 074 000 01 00 02
display two answers, in hexadecimal, and machine halt (breakpoint)	A B 000 06 00 02
write on drum	111 B C 07 00 02 or 143

a) Typeout and halt when capacity is exceeded:

If there is overflow in addition, or on a left shift, or if a divisor is zero, the SWACPEC will type out the command on which the error occurred, and the decimal equivalent of the answer, less the overflowed digit. No further computation will occur, and it will be necessary to start the routine over at some preceding point, first eliminating the cause of the trouble.

b) Typeout for spot check:

By specifying a suitable value of D, the coder may inspect the answer by any arithmetic computation. The SWACPEC types out the command in hexadecimal, then the answer in decimal, and proceeds with the calculation. The breakpoint output, in which two numbers are

2. Interpretation and Subroutines

displayed, will cause the SWAC to halt. This device will not be too useful since the numbers are displayed in binary form, but it may have some use in checking codes.

c) Typeout answers:

Ten numbers in decimal may be typed out as one operation. These numbers must be located in cells 101-110.

d) Punch tape:

A teletype tape punch is available and may be used for punching of check answers. These may be punched onto IBM cards on the conversion unit in the IBM room. Ten decimal numbers may be punched as one operation.

e) Punch cards:

An IBM type 513 Summary Punch is used by the SWAC to punch out answers, 10 at a time. These answers must be located in addresses 101-110. A single command will cause the punch-out. The numbers are in binary form, each row of the card containing one number. Conversion to decimal is done by a separate routine on the SWAC. The numbers in their final form appear 5 on a card, and may be listed on the IBM tabulator, using a standard plugboard.

2. Interpretation and Subroutines

VI. The Magnetic Drum

The magnetic drum of the SWAC contains auxiliary storage of 4,096 words (numbers) which is available to the SWACPEC coding via the drum read or write commands. It will be possible to store 8, 16, or 32 words of data on the drum with one SWACPEC instruction, and the same flexibility is available for the reading of information from the drum. The C of a command will specify the channel of the drum used, from 016 to 143. The A of a command will take on one of two values only, namely 111, or 143, depending on which one of the two sections of the SWACPEC storage is being used.

The following table gives the combinations for reading or writing with any drum channel.

A	B	Cells used in SWACPEC Storage	Storage positions on any drum channel
111	000	096 - 127	000 - 031
111	004	096 - 111	000 - 015
111	006	112 - 127	016 - 031
111	012	096 - 103	000 - 007
111	013	104 - 111	008 - 015
111	014	112 - 119	016 - 023
111	015	120 - 127	024 - 031
143	004	128 - 143	000 - 015
143	012	128 - 135	000 - 007
143	013	136 - 143	008 - 015

2. Interpretation and Subroutines

No other combinations should be attempted. This is very important, especially on read commands where there is a possibility of erasing part of the SWACPEC routine itself, by incorrect coding.

A one to one correspondence exists between the cell numbers 096 - 127 and the 32 storage positions in any given channel. A similar relation exists between the cells 128 - 143, and the first 16 storage positions of any channel. Thus the same channel may be coded in a write command several times, and only partially filled each time. As long as the same section of the channel is not addressed on successive "writes", the previously stored information will not be disturbed. Likewise, several partial readings of the same channel are possible. Due to this one-to-one correspondence, it is not necessary to specify the storage positions of the drum channels. It suffices to specify the cells of the SWACPEC storage by means of the A and B of the read or write command, and the channel of the drum by the C.

The actual forms taken by the commands are as follows:

Read from DRUM: + A B C 07 00 09

Write on DRUM: + A B C 07 00 02

2. Interpretation and Subroutines

VII. Card Forms

Decimal Cards

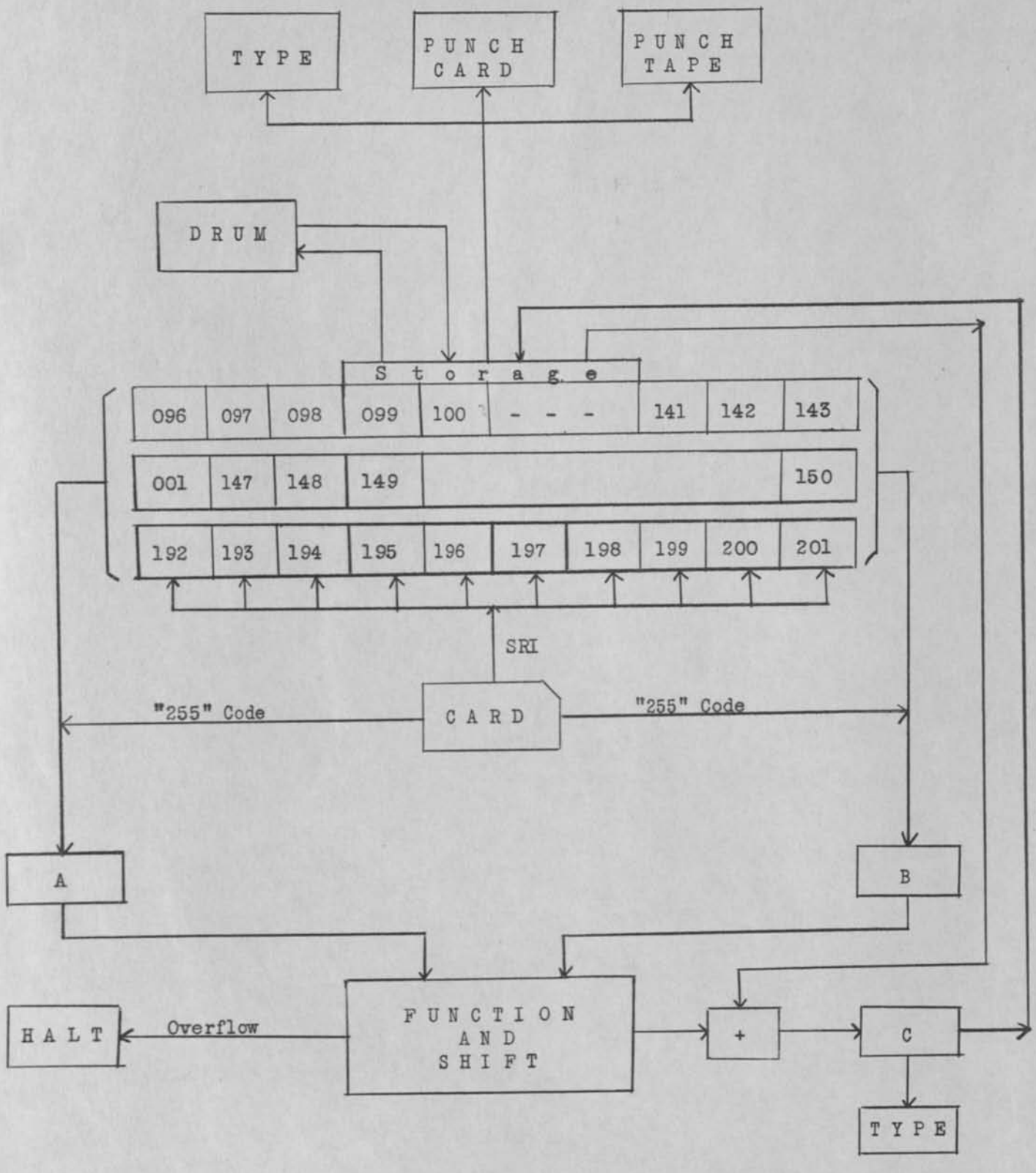
Card Cols.	
1-3	card number
5	word pulse (digits x thru 9)
7-9	A
10-12	B
13-14	C (first two digits)
16-20	Problem number
27	C (last digit)
28-29	S
30-31	D
32-33	F
34	± sign
35	x for number, 0 or blank for command

Note:

Column 4 must never contain a punch on the decimal cards.

Binary Cards

Card Cols.	
1-3	Card number
4	SRI (a "9" only on SRI)
5	word pulse (digits x - 9)
6	± sign
7-14	A
16-20	problem number
27-34	B
47-54	C
67-70	S
71-74	D
75-78	F



SWAC PEC SCHEMATIC DIAGRAM

2. Interpretation and Subroutines

3. SWACPEC

SWACPEC CODING SHEET

Page

Coded by: Problem:

Card No.	A	B	C	S	D	F	+ -	N	S R I	REMARKS

3. Loading, unloading and clearing Routines

1. Drum Punch-out Code.

1) Code No. 00707.

2) Written by Ruth Horgan, August 1954.

3) Drum Punch-out, whole or partial channels.

4) Punches contents of any specified number of channels, beginning with a specified channel. Punches whole, half or quarter channels, and punches either eight or ten words per card.

5) One instruction word is to be typed into SWAC, or fed in from collator. It should be prepared in hexadecimal as follows: Sign, positive; Alpha, zero; Beta, portion of channel (standard notation); Gamma, first channel to be punched; Delta, number of channels; F, either 8 or 10 (u in base 16) — the number of words per card to be punched.

6) Operating procedure:

a. Output. Use standard 36-digit reproducer board. The number of jack-plugs must be the same as the F of the above instruction word, eight or ten. (When ten words per card are punched, an appropriate number of zero words is punched at end of each channel.)

b. Setting of typeout and breakpoint switches is immaterial.

c. Routine does not use drum. Drum may be used entirely for information.

e. Operation. Read routine into SWAC on continuous. It halts in 002, calling for input of one word to cell 254. Either type

3. Loading, unloading and clearing Routines

in the above instruction word and hit 00PB, or input the word from the collator. SWAC punches desired channels and, when completed, halts in 002 awaiting another instruction word.

f. Identification of output. Columns 37 through 44 give (in binary) the serial number (000 to 031) of each word in its channel, and columns 57 through 64 give (in binary) the channel number. Contents of channels are punched in binary.

7) Routine is stored in cells 000 to 064. Other cells used are listed with code.

8) Flow chart is with code.

9) Code is in subroutine file.

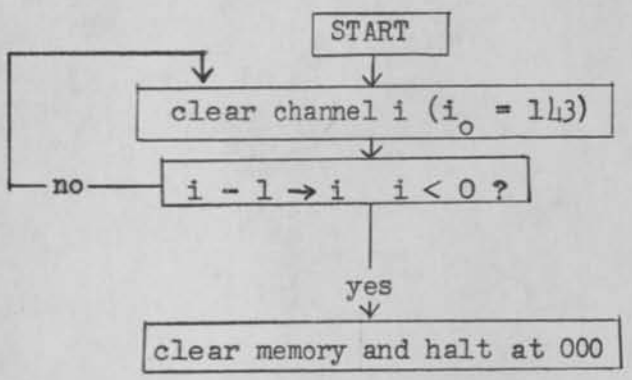
3. Loading, unloading and clearing routines

2. Drum-clear code

1) Code No. 00702
2) Coded by J. Selfridge, 1953.

This one-card routine clears channels 000 to 143* in about 3 sec. and halts in cell 000 with the memory clear. Thus, this card may be used in front of any routine deck which requires a clear drum.

FLOW CHART



THE CODE

card row		to cell	α	β	γ	δ	F	Remarks
9		000						blank
8		001	---	---	001	252	08	clear 001, go to 252
7	$\delta \rightarrow \varepsilon$	000	---	---	---	253	08	clear 000, go to 253

Now cell 001 is cleared and we go to 252 for initial input.

6	252	---	---	001	---	--	used for i - 1 → i
5	253	---	---	[143]	112	02	clear channel 143(1)000
4	254	253	252	253	253	08	i - 1 → i, go to 253 if ≥ 0
3	255	224	---	---	112	01	clear 224 through 255

Now cell 000 is cleared and we go to 253 to start drum clearing.

---

*To clear channels 0 through n, the gamma portion of cell 253 may be changed to n.

4. Code Checking Routines

1. Memory Punchout

1) Code No. 00380, written by John Selfridge, 1954.

2) Description written by Lenore Selfridge, 1956.

3) Title: Memory Punchout Routine

4) Summary:
Routine types the first three words of the high-speed memory, then punches out the entire memory in binary notation.

5) Operating procedure:
a. Output: Use standard 036-digit reproducer board, any number of words per card.
b. Switches: Breakpoint, Off; Typeout, normal.
c. Routine does not use the drum.
d. Input: Use standard collator board, ten words per card.
e. Operation:
Remove any unread cards from inside the collator. Place routine in the collator. At the console, set SWAC at period 7 and in continuous mode of operation.

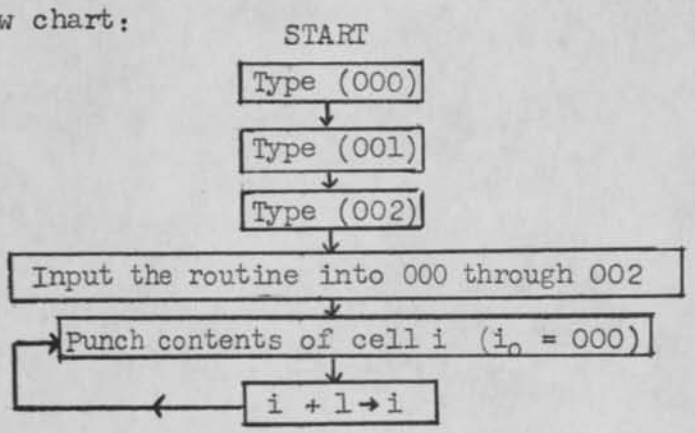
i) Optional for typeout of (000): Type a 2 in F into the R register, hold the R→C switch and press 00PB. Continue to hold the R→C switch throughout the typeout of (000).
ii) Press the Shift key on the console until the R register is clear. Holding the R→C switch, press 00PB once and start the collator. Continue to hold the R→C switch throughout the typeout of (001).
iii) Hold the R→C switch and press 00PB; release the R→C switch during the typeout of (002). (If you held the R→C switch until the typeout of (002) was finished, press 00PB without holding R→C. Cell (002) will type out again.)
iv) Collator halts with the R register showing ZZ 00 00 50 2; press 00PB and the routine starts to punch (000),(001), ...
v) When α of the M register advances beyond ZZ to 00, stop the punching by switching off the toggle (summary punch) switch on the punch unit, and then pressing the stop key of the punch after two blanks have fed.

f. Punching takes about twenty seconds for complete memory (ten words/card).
g. Error halt: If the routine does not behave as described above, chances are that only the first 3 cells of the memory were damaged. Remove all cards from collator without clearing memory and start over.
h. Final halt: Routine must be halted manually as explained in step e.v) above.
i. Output may be converted to decimal using routines 00213 or 00223, or read directly in binary (with cell identification appearing in columns 37 through 44).

4. Code Checking Routines

6) Routine uses cells 000, 001, and 002. Punching is done from cell 000.

7) Flow chart:



8) The code:

Card No.	Card Line	Cell No.	δ↘ε	α	β	γ	δ	F	Remarks
000	9	000	√	001	---	---	---	02	Type out (001)
	8								W.P. punch only
				Now (001) is typed out.					
001	9	000	√	---	---	001	001	05	Dummy command
	8								W.P. punch only
	7		√						W.P. and δ→ε punches
	6	001		001	---	001	001	06	Fix (001)
	5								W.P. punch only
	4								W.P. punch only
	3	000	√	002	---	---	---	02	Type out (002)
	2								W.P. punch only
				Now (002) is typed out and R → C is released.					
002	9	001	√	002	---	---	---	01	Input into (002)
	8	002		---	---	---	001	08	Clear (000); go to (001)
	7	002		---	---	001	---	08	Clear (001); go to (000)
	6	000		002	---	---	---	01	Input into (002)
	5	001		---	---	---	---	--	Zero
	4	002		001	---	---	---	-	Modifier
	3	001		002	---	---	---	05	Modify punch command
(No W.P.)	2	000	√	[255]	---	---	080	02	Punch command
				Now routine starts at cell (001).					

9) Cards are filed in the SWAC room.

6. Other Routines

1. Sorting Code.

1) Code no. 00400.

2) Written by G. E. Forsythe, May 1954.

3) Sorting code.

4) Takes any number of nonnegative 36-digit binary fractions (including not more than 180 distinct fractions), puts the distinct ones in numerical order, rejects duplicates, and punches out the distinct fractions in ascending order of magnitude. Whether or not .000 ··· 0 (binary) was input, it is punched out as the first word. Whether or not .111 ··· 1 (binary) was input, it is punched out as the word following the ordered table.

The one parameter is S_o, where $S_o + 1$ is the number of binary cards to be read in.

5) Numbers to be ordered must be on standard cards, 10 to a card (or ten words on an integral number of cards).

6) a) Uses 10-word punch out. Zero and Y rows left blank.

b) Type-out switch on "normal." Break-point switch on.

c) Drum not used.

d) Standard collator board. Routine is on cards 000 to 007, to be followed by a card with one word pulse and the number (S_o) of last input data card in β. Then follow the input cards 0, 1, ··· , S_o (numbers need not be on the cards).

6. Other Routines

e) Machine takes in all the cards, and puts the distinct binary fractions in cells 076, 077, ··· , n, in descending order of size. After all cards are read in, the number n is typed out in the α-position of the one word typed. The contents of the cells of the memory are then automatically punched out, 10 words to a card: cells n, n-1, n-2, n-3, ··· , 000. After cell 000 is punched out the routine goes into a loop.

f) The routine runs at approximately one input card per second.

g) If more than 180 distinct fractions to be ordered are read in, there is an error halt in cell 057. Fix the input cards, and read the routine in again.

h) Program halts by going into a loop on punch-out. Stop SWAC and collect the punch-out.

i) Output contains the fractions in order from the smallest number (0) in cell n to the largest in cell 076, followed by .111 ··· 1, always in cell 075, and the rest of the memory down to cell 0. Locate the word .111 ··· 1 by inspection and disregard the rest of the punch-out.

7) The sorting is done by a comparison of each new fraction with the ordered list in the memory, starting with cell 076. As soon as a fraction's proper order is found, the other fractions are moved one cell to give it room.

6. Other Routines

8) The routine is in cells 000 to 075.

9) A flow chart is with the code.

10) The code is in manuscript form, filed in the subroutine file, under its code number. It is identical with Forsythe's code 0500.05, in the SWAC code file in the library entitled "Forsythe Semigroups."

6. Other Routines

2. Row-deleting Code No. 00401

1) Suggested by Dr. K. Trueblood; written by Ruth Horgan, August 1955.

2) Description prepared by Ruth Horgan, April 1956.

3) Title: Row-deleting Routine.

4) Summary:
Routine reads in k binary words per card and punches out n of them per card. Sequence of words is not disturbed. Operator designates k and n, $1 \leq k \leq 010^*$, $1 \leq n \leq k$. Routine processes an indefinite number of cards.

If n = k, this routine could be used to "reproduce" binary data cards, moving binary words from one row of a card to another row (without any deletion).

5) Preparation of input data:
Data must be on binary cards, k words per card. Prepare an instruction card with k word-pulse punches, and column 7 punches in the desired n of these k rows. Whichever rows of the instruction card have column 7 punches determine which n of the k rows of each data card will be punched out.

Example: If the instruction card has a 5 and a 9 punch in column 7, two words per data card will be punched out per card -- the words on the 5 and 9 rows of the data card.

6) Operating procedure:
a. Output board: Arrange n jackplugs for desired rows on the 036-digit reproducer board.
b. Setting of typeout and breakpoint switches is immaterial.
c. Drum is not used.
d. Input board: Use standard 036-digit collator board, ten* words per card.
e. Operation: Read the routine cards, the instruction card, and the data cards (in that order) into SWAC on continuous operation. SWAC will halt at 2Z (047), calling for input to 61 (097). Type k in hexadecimal into the alpha part of the R register and push 00PB. SWAC will alternately read one k-word binary card and punch one n-word binary card.

When all the data cards are used, SWAC will halt at 0Y (014), calling for input to cell 60 (096). Press the Clear Epsilon

---

* k may equal eleven; input collator board must be adjusted by operator to accept eleven words. See SWAC memorandum of April, 1955, in NAR library.

6. Other Routines

button and then the OOPB; SWAC will halt at 2Z (047), calling for input to 61 (097) as above. Type in a new value for k, if desired, and push OOPB. SWAC will halt at 03, calling for input to 60 (096). The routine is now ready for a new instruction card and more data cards, if desired.

f. Running time is dependent on card input and output speeds.

g. An error halt at epsilon of 23 (035) indicates an impossible overflow, possibly machine malfunction.

h. There is no final halt. See second paragraph of 6e.

i. Output cards: The resulting cards will have n binary words per card (and the corresponding n word-pulse punches) on the rows corresponding to the n jackplugs placed on the output board.

7. Routine is stored in high-speed memory cells 00 through 3V (000 through 059); data is stored in cells 60 through 6U (096 through 106).

8. Flow chart and code are in NAR library.

9. Cards are filed in SWAC room.

0. List, Mathematical Codes

SWAC Manual Page Reference	Code Number	Title
F 1.1a	00510.0	Matrix Symmetry Check
F 1.1b	00510.1	Matrix Multiplication (row-by-column), Floating Point
F 1.1c	00510.2	To Obtain Transpose of a Matrix
F 1.1d	00510.3	Fixed Point to Floating Row (of a matrix) Conversion
F 1.1e	00510.4	To Re-scale Rows of a Matrix, Floating Point
F 1.1f	00510.5	Drum-load Routine for Code No. 00545.1
F 1.2a	00520.0	Matrix Inversion, Elimination, Floating Point
F 1.2b	00520.1	Matrix Inverse Improvement, Floating Point
F 1.2c	00520.2	Matrix Inversion and Improvement, Floating Point
F 1.2d	00520.3	Matrix Inverse Improvement, Floating Point Input and Fixed Point Output
F 1.2e	00520.4	Computation of Binary Residual Matrix, Floating Point Input
F 1.3a	00530.2	Transformation of Integral Matrices to Triangular (Hermite) Form
F 1.4a	00540.0	Eigenvalues and Eigenvectors of a Real Symmetric Matrix of Order $n \leq 010$, Jacobi Method
F 1.4b	00545.0	To Obtain the Characteristic Polynomial of a Matrix by Frame's Algorithm
F 1.4c	00545.1	Power Method to Find the Dominant Eigenvector of a Matrix in Floating Row Notation
F 3.2a	00660	Overrelaxed Liebmann Process for a Dirichlet Problem in a Square Net
F 3.2b	00661	Overrelaxed Liebmann Process for Laplace's Equation in a Square Net, Mixed Boundary Conditions
F 3.3a	00670	Overrelaxed Liebmann Process for a Square Net, Fundamental Eigenvalue of Laplace's Operator
F 3.4a	00680	Doubling Values of u(x,y) over a Square Net
F 3.4b	00681	Computation of Rayleigh Quotient for Function u(x,y)
F 3.4c	00682	Halving Values of u(x,y) over a Square Net
F 3.4d	00683	Drum-punch Routine Used With Routines 00660,1,2 and 00680,1,2
F 3.4e	00684	Drum-load Routine Used With Routines 00660,1,2 and 00680,1,2

(Section F1 edited by G. E. Forsythe and Ruth Horgan)

0. General

The matrix codes have been assigned numbers from 00500.0 to 00599.9. When filed in the SWAC room, the input decks of IBM cards are filed under these numbers. The fourth digit of the code number is the subsection number of part F1 of this manual. Thus, code 00520.0 will be described in section F 1.2.

Data handling and conversion	Codes 00510.0 to 00519.9
Inverting matrices	Codes 00520.0 to 00529.9
Solving linear systems	Codes 00530.0 to 00539.9
Computing eigenvalues	
Symmetric matrices	Codes 00540.0 to 00544.9
Nonsymmetric matrices	Codes 00545.0 to 00549.9
Solving linear inequalities	Codes 00550.0 to 00559.9
Solving linear games	Codes 00560.0 to 00569.9
Solving nonlinear algebraic systems	Codes 00570.0 to 00579.9

1. Matrices

1. Data handling and conversion

a. Code No. 00510.0

1) Written by Louise Straus, October 1954.

2) Description written by Ruth Horgan, 1955.

3) Title Matrix Symmetry Check

4) Summary

Let matrix A have n rows and n columns, $n \leq 064$. Routine inspects matrix for symmetry, and informs operator if it discovers any discrepancy greater than a given tolerance. After checking one matrix, the routine is ready to inspect another, using the same or different values for n and the tolerance. Matrices may be in any notation; routine merely checks for discrepancies by subtraction in fixed binary point arithmetic.

5) Preparation of input data

Assemble cards for only n rows. For $n \leq 010$, prepare one binary card per row---n elements per card accompanied by 010-n additional words. These additional words need not be zero; the routine ignores them. For $011 \leq n \leq 020$, prepare two binary cards--the first card containing the first ten elements, and the second card containing the remaining n-010 elements followed by 020-n additional words. If $021 \leq n \leq 030$ prepare three cards, if $031 \leq n \leq 040$ prepare four cards, etc., and if $061 \leq n \leq 064$ prepare seven cards.

6) Operating procedure

a) Output: There will be no typeouts and no card punchouts.

b) Breakpoint switch must be on.

c) Drum must be on.

d) Input: Use standard 036-digit collator board, ten words per card.

1. Matrices

e) Operation: Read the nine routine cards into SWAC on continuous operation. Routine halts at 02, calling for input to cell 7X (125). Type n in hexadecimal into the alpha part of 7X and push 00PB. Routine halts in 06, calling for input to cell 7Y (126). Type in the tolerance and push 00PB. Routine next halts in 0V (011), calling for input to cell 80 (128). Feed the matrix cards into the collator, and the routine will begin.

f) Data on running time unavailable.

g) Error halts: At any time that the routine discovers that $|a_{ij} - a_{ji}| >$ the tolerance, there is a display breakpoint halt at 52 (082), showing the two coordinates of the non-symmetric elements in the alpha part of the M and R registers. Push 00PB, and routine halts at epsilon of 53 (083). The M register shows the difference between the two elements, and the R register displays the tolerance. Push 00PB, and SWAC halts at 2W (044), calling for input to cell 7Y (126). Type in a new choice for the tolerance and push 00PB; the routine then continues to check symmetry from the point in the matrix where it halted. It does not return to the first row and column. If it is desired to begin the symmetry check of the given matrix again, perhaps with the tolerance just input to cell 7Y (126), use the R → C switch to obey the command 00,00,01,13,7 and allow the routine to continue.

A halt at the origin, with C register reading 01,00,00,10,1 and the overflow memory tube (alpha 5) registering a digit in 7Z, indicates that there was overflow in the execution of $a_{ij} - a_{ji}$ at 29 (041). From the keyboard, input into the R register the command 00,00,01,43,7 and push 00PB. Routine halts at 52 (082); see paragraph above. (NOTE: In this case, at the 53 (083) halt the M register shows the remainder of the difference between a_{ij} and a_{ji} after overflow.)

h) Final halt: When routine has verified the symmetry of the matrix for the given tolerance, it halts at 02 calling for input of the order n of a new matrix to be inspected. See section e.

1. Matrices

7) Mathematical method used Routine compares the difference between two corresponding elements (a_{ij} and a_{ji}) with a given tolerance.

8) Routine uses high-speed memory cells:

00 through 58 (000 through 088) and 7V through 7Z (123 through 127) for routine, and 80 through ZZ (128 through 255) for data.

The matrix is stored on the drum in channels 10 through 0Z + n (016 through 015 + n). No routine is stored on the drum.

9) Flow chart is with the code.

10) Code is in NAR library. Cards are filed in SWAC room.

1. Matrices

1. Data handling and conversion

b. Code No. 00510.1

1) Written by Louise Straus, 1954.

2) Description prepared by Ruth Horgan, 1955.

3) Title Row-by-column Multiplication of Two Matrices, Floating Binary Point.

4) Summary

Let matrices A and B each have n rows and n columns, $n \leq 032$, all elements in floating binary point notation. Routine forms the row-by-column product AB, and punches it in floating binary point notation.

5) Preparation of input data

Matrix A cards must precede matrix B cards; should B cards precede A cards, the routine forms the product BA. Prepare the cards for each matrix in one group, as follows:

Assemble cards for only n rows. For $n \leq 010$, prepare one binary card per row—n elements per card (in floating binary point notation) accompanied by 010-n additional words. These additional words need not be zero; the routine ignores them. For $011 \leq n \leq 020$, prepare two floating binary cards—the first card containing the first ten elements, and the second card containing the remaining n-010 elements followed by 020-n additional words. Similarly, if $021 \leq n \leq 030$ prepare three cards; if n = 031 or 032 prepare four cards.

Example: if n = 014, the input data would consist of 056 cards—028 cards for A followed by 028 cards for B.

6) Operating procedure

a) Output: Use standard 036-digit reproducer board, ten words per card.

b) Breakpoint switch must be on; typeout switch may be at either setting.

c) Drum must be on.

d) Input: Use standard 036-digit collator board, ten words per card.

1. Matrices

e) Operation: Read routine cards 000 through 018 into SWAC on continuous operation. Routine halts in 69 (105), calling for input to cell ZZ (255). Either type n in hexadecimal into the alpha part of ZZ and push 00PB, or read an "n in alpha" card from the collator. SWAC next halts in 73 (115) calling for input to cell 40 (064). The cards for A followed by the cards for B, and then routine cards 019 through 028 should now be fed into the collator.

Routine halts at epsilon of X4 (212) calling for input to 51 (081), and R register reading 07,09,08,0W,5; push 00PB, and computing will begin. The routine computes the product AB and punches it in floating binary point notation on the same number of cards as was required for matrix A.

f) It requires about 8 minutes to compute the product of two 032-by-032 matrices, and about $2\frac{1}{2}$ minutes for two 018-by-018 matrices.

g) Error halts: A halt at 02 with C register reading 01,00,00,10,1 indicates that there was overflow within the floating binary point subroutine.

A dynamic halt at 3Y (062), with the SWAC obeying the command 21,07,03,3Y,8 repeatedly, indicates "underflow"--a number has exceeded the minimum capacity of the floating binary point range. Use the R → C switch to obey the command 00,00,08,0W,7, and allow routine to continue; this substitutes zero for the answer which was too small.

h) Final halt: After punchout, routine halts at Y8 (232).

i) Output: The answer cards are in floating binary point notation. The elements are arranged in the same order, and on the same number of cards, as were the input cards for matrix A.

7) The operations $\Sigma_j a_{ij}a_{jk}$ are performed with standard floating binary point subroutines.

8) Routine is stored in cells:

00 through 1Z (000 through 031),
60 through 9Z (096 through 159), and
W8 through ZZ (200 through 255).

1. Matrices

All other cells of the high-speed memory are used for data. The matrix A is stored on drum channels 50 through 4Z + n (080 through 079 + n), and B on channels 10 through 0Z + n (016 through 015 + n). Product AB is stored on channels 30 through 2Z + n (048 through 047 + n).

9) Flow chart is in NAR library code file. Cards are filed in SWAC room.

10) Code exists only in detailed flow chart form.

1. Matrices

1. Data handling and conversion

c. Code No. 00510.2

1) Written by Louise Straus, October 1954.

2) Description prepared by Ruth Horgan, 1955.

3) Title Transposing a Matrix

4) Summary

Let matrix G have n rows and n columns, n ≤ 032; routine punches the transpose of G.

5) Preparation of input data

Elements of G may be in any notation; the routine merely transfers them within the SWAC storage without inspecting them. Prepare cards for only n rows.

For n ≤ 010, prepare one binary card per row--n elements per card accompanied by 010-n additional words. For 011 ≤ n ≤ 020, prepare two binary cards--the first card containing the first ten elements, and the second card containing the remaining n-010 elements followed by 020-n additional words. Similarly, if 021 ≤ n ≤ 030 prepare three cards; if n = 031 or 032 prepare four cards.

6) Operating procedure

a) Output: Use standard 036-digit reproducer board, ten words per card.

b) Setting of typeout and breakpoint switches is immaterial.

c) Drum must be on.

d) Input: Use standard 036-digit collator board, ten words per card.

e) Operation: Read routine cards 000 through 005 into SWAC. It halts at 02 calling for input to cell 2V (043). Either type n in hexadecimal into the alpha part of 2V and push 00PB, or read an "n in alpha" card from the collator. SWAC next halts in 0X (013) calling for input to cell 40 (064). The cards for matrix G should be fed into the collator. Routine forms the transpose of G and punches it in binary, with no change in the kind of notation, on the same number of cards as was G.

1. Matrices

f) It takes 1½ minutes to obtain the transpose of a 028-by-028 matrix.

g) There are no significant error halts.

h) Final halt: After punchout, routine halts in 29 (041).

i) Output: The punched cards will be in the same binary notation, with the same arrangement of rows and columns, as were the input data cards.

7) Mathematical method used Routine merely transfers elements within the SWAC storage.

8) Routine uses high-speed memory cells:

00 through 3V (000 through 059),
40 through 3Z+n (064 through 063+n), and
80 through 7Z+n (128 through 127+n).

Matrix is stored on drum channels 10 through 0Z + n (016 through 015 + n), and the transpose on channels 30 through 2Z + n (048 through 047 + n).

9) Flow chart is with the code.

10) Code is in NAR library. Cards are filed in SWAC room.

1. Matrices

1. Data handling and conversion

d. Code No. 00510.3

1) Written by Louise Straus, 1954.

2) Description prepared by Ruth Horgan, 1955.

3) Title Fixed Binary Point to Floating Row Conversion

4) Summary

Prepares data for routines 00709 and 00545.1. Let each row of a matrix have n elements, $n \leq 030$; each element is stored times 2^{-k}. Routine inspects each row for its largest element and shifts it as far left as possible, keeping a tally of minus one in delta for each left shift (combined properly with the given k). Routine then shifts all other elements in the row the same number of positions to the left.

Punches n + 1 elements per row--the exponent p (tally of shifts combined with k) and the n elements. (Thus 2^p becomes a common scale factor for all elements of the row.) Routine proceeds to the next row, and scales it independently. Continues until the supply of data cards in the collator is exhausted--regardless of the value of n.

5) Preparation of the input data

Prepare the matrix cards in fixed binary point notation. Assemble cards for only n rows. For $n \leq 010$, prepare one binary card per row--n elements per card (in fixed binary point notation) accompanied by 010-n additional words. For $011 \leq n \leq 020$, prepare two binary cards--the first card containing the first ten elements, and the second card containing the remaining n-010 elements followed by 020-n additional words. Similarly, if $021 \leq n \leq 030$ prepare three cards.

The additional words per row, used to assemble a multiple of ten, need not be zero; see last paragraph, section 6i.

6) Operating procedure

a) Output: Use standard 036-digit reproducer board, ten words per card.

b) Setting of typeout and breakpoint switches is immaterial.

c) Routine does not refer to drum.

1. Matrices

d) Input: Use standard 036-digit collator board, ten words per card.

e) Operation: Read the eight routine cards into SWAC on continuous operation. It halts in 02, calling for input to cell 94 (148). Either type n in hexadecimal into the alpha part of 94 and push 00PB, or input an "n in alpha" card from the collator. SWAC halts in 03 calling for input to cell ZZ (255); type the exponent k of the given scale factor (in hexadecimal) into the delta part of ZZ, and push 00PB. Routine next halts in 19 (025), calling for input to cell 96 (150). Feed the data cards into the collator. SWAC reads in the data one matrix row at a time, and punches the scaled data one matrix row at a time.

Note that if n is a multiple of ten, one more card per row will punch out than had been read in. See 6i below.

If all the elements of a row are zero, the exponent p is given the arbitrary low value -2^{16}.

f) Computing time is negligible; running time can be measured by card input and output speeds.

g) There are no significant error halts.

h) Final halt: Routine halts at 19 (025) when supply of input data cards is used.

i) Output: The first word punched out per row contains the exponent p, an integer in binary, in the delta part of the word. The n scaled elements of the row, in binary, then follow in order. Hence, there are n + 1 words punched per row. If $n < 010$, SWAC punches one card per row. If $010 \leq n < 020$, SWAC will punch two cards per row. Similarly, three or four cards will punch per row for $020 \leq n < 030$ or $n = 030$.

The additional words punched per row (to complete a multiple of ten words per row) will be zero. During the read-in of the original matrix, more than n elements per row might have been read into SWAC. These additional words, however, will have been cleared during the routine, and will punch as zeros.

7) Mathematical method used: See section 4).

1. Matrices

8) Routine is stored in cells 000 through 076, 148, 149, 200; data in cells 150 through 149 + n. (Cells 00 through 4W, 94, 95, W8; data in cells 96 through 95 + n.)

9) Flow chart is with code.

10) Code is in NAR library. Cards are filed in SWAC room.

1. Matrices

1. Data handling and conversion

e. Code No. 00510.4

1) Written by Louise Straus, 1954.

2) Description prepared by Ruth Horgan, 1955.

3) Title: To Re-scale Rows of Floating Binary Point Matrix

4) Summary

Let matrix A have n rows and n columns, n ≤ 032; all elements are in floating binary point notation. Given n exponents between -255 and +255, the routine multiplies each row of the matrix by the respective power of two and punches out the resulting matrix.

5) Preparation of input data

Prepare matrix cards for only n rows. For n ≤ 010, prepare one binary card per row--n elements per card (in floating binary point notation) accompanied by 010-n additional words. These additional words need not be zero; the routine ignores them. For 011 ≤ n ≤ 020, prepare two floating binary cards--the first card containing the first ten elements, and the second card containing the remaining n-010 elements followed by 020-n additional words. Similarly, if 021 ≤ n ≤ 030 prepare three cards; if n = 031 or 032 prepare four cards.

Arrange the exponents of the desired powers of two as if they represented another row of the matrix, beginning with the exponent for the first row, then that for the second row, ... , ending with the exponent for the last row. These exponents are to be stored in binary as integers in alpha.

6) Operating procedure

a) Output: Use standard 036-digit reproducer board, ten words per card.

b) Setting of typeout and breakpoint switches is immaterial.

c) Routine does not refer to drum.

d) Input: Use standard 036-digit collator board, ten words per card.

e) Operation: Read the seven routine cards into SWAC on continuous operation. It halts in 02, calling for input to cell ZW (252). Either type the order n in hexadecimal

1. Matrices

into the alpha part of ZW and push 00PB, or feed an "n in alpha" card from the collator. Routine next halts in 06 calling for input to cell W8 (200). Feed into the collator the cards containing the exponents described in 5). There will be n words, stored in binary on 1, 2, 3, or 4 cards.

Routine then halts in 0V (011), calling for input to cell 96 (150). Feed the matrix cards through the collator. Routine will read in one row of the matrix (1, 2, 3, or 4 cards; multiple of ten words), multiply it by the appropriate power of two, and punch it on the same number of cards. It will process row after row of the matrix until the supply of cards in the collator is exhausted.

It is possible to re-scale any m rows of a matrix of order n, where $m < n$. The operator need only feed to the collator the cards for m rows, n elements each. He must still prepare n exponents, but only the first m of these will be used by the routine for re-scaling. Routine will halt, having used up all data cards, before it refers to the $(m + 1)$-th exponent.

f) Data on running time unavailable.

g) Error halts:

A halt in 23 (035) indicates that one of the newly scaled elements has exceeded the floating notation range. Should the operator wish to suspend re-scaling of the row being processed during the overflow and advance to the next row, he should re-enter the routine at epsilon of 09. Note that there will be no punchout at all for the skipped row.

A halt at 34 (052) indicates "underflow" —a number has exceeded the minimum capacity of the floating binary point range. Push 00PB; this will substitute zero for the answer which was too small (currently in cell 47 (071)) and routine will continue.

h) Final halt: Routine halts when supply of data cards is used. Should there be more than n rows represented by the cards in the collator, the routine will continue to read them in and attempt to multiply them by powers of two. These will be meaningless. Having used up its n given powers of two (cells W8 through W7 + n), the routine would continue through consecutive cells, using whatever it found for powers of two and punching supposedly re-scaled rows.

1. Matrices

i) Output: The punched matrix will be in floating binary point notation. Elements will be arranged in the same order as they were in the initial matrix, with additional zero words where required for a multiple of ten words.

7) Probably the main use of this routine is to "precondition" a matrix A before inverting it, say by routine No. 00520.0.

8) Routine is stored in high-speed memory cells 00 through 40 (000 through 064), given powers of two in cells W8 through W7 + n (200 through 199 + n), and a row of the matrix in cells 96 through 95 + n (150 through 149 + n); other temporaries are listed with the code.

9) Flow chart is in NAR library.

10) Code exists only in detailed flow chart form. Cards are filed in SWAC room.

1. Matrices

1. Data handling and conversion.

f. Code No. 00510.5

1) Written by Louise Straus, 1954.

2) Description written by Ruth Horgan, 1955.

3) Title Drum Load for Power Method Eigenvector Routine, No. 00545.1.

4) Summary

Let matrix A have n rows and columns, $n \leq 030$. The elements of each row are in floating row notation, possibly as punched out in conversion routine code no. 00510.3, n + 1 words per row. The first word per row is the floating row exponent p as described in memorandum on code no. 00510.3. This drum load routine writes successive rows of matrix A on successive channels of the drum, beginning with the first row on channel 10 (016). It sums the first n + 2 words of each channel and stores the sum in the last word of that channel.

5) Preparation of input data

Each row of matrix A requires n + 1 binary words of input; the first word is the exponent in delta, and the other words are the n elements in floating row notation. For $n < 010$, prepare one binary card per row--the exponent, the n elements, and 9-n additional words of zero. For $010 \leq n < 020$ prepare two binary cards per row--the first card containing the row exponent and first 9 elements, the second card containing the remaining n-9 elements followed by 019-n additional words. Similarly, if $020 \leq n < 030$ prepare three cards, and if n = 030--four cards.

6) Operating procedure

a. Output: There are no typeouts or punchouts.

b. Setting of typeout and breakpoint switches is immaterial.

c. Drum must be on.

d. Input: Use standard 036-digit collator board, ten words per card.

e. Operation: Read the four routine cards into SWAC on continuous operation. It halts in 02, calling for input to cell 5Z (095). Either type the order n in hexadecimal into the alpha part of 5Z and push 00PB, or feed an "n in alpha" card from the collator. Routine next halts in CW (012), calling for input to cell 60 (096). Feed the matrix cards through the collator. Routine reads in one row of the matrix at a time, writes it on the drum (including the sum as described), and proceeds to the next row.

1. Matrices

f. Running time dependent on card input speed.

g. There are no significant error halts.

h. Final halt: Routine halts when supply of input data cards is exhausted. If operator has represented more than n rows in the cards he placed in the collator, the routine will continue to read them in and write on the drum--though still expecting only n + 1 words per row. There is no halt coded in the routine to inform the operator that more than n channels of the drum have been loaded; the routine will continue to go through the motions of drum-writes on channels not used by code no. 00515.1.

i. Output: No typeouts nor punchouts.

7) Routine is stored in high-speed memory cells 00 through 21 (000 through 033), and data in cells 5Z through 60 + n (095 through 096 + n). The matrix is stored on drum channels 10 through 0Z + n (016 through 015 + n) if there are only enough cards for n rows. If there are more than n rows in the data, the data will be stored on successive channels beginning with 10 (016)--continuing until the data cards are used.

8) Flow chart is with the code.

9) Code is in the NAR library. Cards are filed in SWAC room.

1. Matrices

2. Inverting matrices

a. Code No. 00520.0

1) Written by Dr. J. T. Pettit, Institute of Geophysics, 1953.

2) Description prepared by Ruth Horgan and G. E. Forsythe, 1955.

3) Title: Matrix Inversion by Row Elimination, Floating Binary Point

4) Summary:

Let matrix A have n rows and n columns, $n \leq 032$, all elements in floating binary point notation. Routine obtains an approximation B to the inverse A^{-1} and punches it in floating binary point on cards. Routine is then ready to compute the inverse of a new matrix, and continues indefinitely. (For accuracy, see paragraph 7 below.)

5) Preparation of input data

Prepare cards for only n rows. For $n \leq 010$, prepare one binary card per row--n elements per card (in floating binary point notation) accompanied by 010-n additional words. These additional words need not be zero; the routine ignores them. For $011 \leq n \leq 020$, prepare two floating binary cards--the first card containing the first ten elements, and the second card containing the remaining n-010 elements followed by 020-n additonal words. Similarly, if $021 \leq n \leq 030$ prepare three cards; if n = 031 or 032 prepare four cards.

6) Operating procedure*

a) Output: Use standard 036-digit reproducer board, ten words per card.

b) Breakpoint switch must be on; typeout switch may be at either setting.

c) Drum is used.

d) Input: Use standard 036-digit collator board, ten words per card.

e) Operation: Read cards 000 through 090 of the routine into SWAC on continuous operation. Routine halts at 3Z (063) calling for input to cell 1X (029). Either type n in

*See code file in NAR library for chronological summary of input, computation, and output.

1. Matrices

hexadecimal into the gamma part of cell 1X and push 00PB, or read an "n in gamma" card from the collator. Routine next halts in 02 calling for input to cell UO (160); start the matrix cards reading through the collator.

The routine computes B, and punches it in floating binary point notation on the same number of cards as was the input--and in the same order. It halts again at 3Z (063), awaiting the next word of "n in gamma" and the next matrix to be inverted.

f) Computing the inverse of a 032-by-032 takes about twenty minutes. An 018-by-018 can take between two and three minutes.

g) Error halts:
A halt at 54 (084) indicates that the matrix A is singular.

A dynamic halt at 3Y (062), with the SWAC obeying the command 21,07,03,3Y,8 repeatedly, indicates "underflow"-- a number has exceeded the minimum capacity of the floating binary point range. Use the R --> C switch to obey the command 00,00,08,0W,7 and allow the routine to continue; this substitutes zero for the answer which was too small.

A halt at 02 with C register reading 01,00,00,10,1 indicates that there was overflow within the floating binary point subroutine.

In the event of drum-load failure during initial loading, the routine halts in a display breakpoint command at epsilon of 0X (013). The R register displays the number of the incorrect channel (in gamma) and the M register displays the amount of discrepancy. Push 00PB and SWAC halts in another breakpoint command at 1V (027). The C register indicates which of the 032 words had caused the halt. The R register displays the word read from the drum, and the M register displays the word from cards which should have been written on the drum. Push 00PB, and the routine will again attempt to write the 032 words on the drum and verify them by comparison. If there is disagreement again, there will be another halt at 0X (013). This will continue until there is agreement or until the operator tires.

h) Final halt: After punching B, the routine halts at 3Z (063), calling for the order n of the next matrix. Routine processes an indefinite number of matrices.

1. Matrices

i) Output: The approximate inverse B is punched in floating binary point notation; the arrangement of the elements is identical to the arrangement in the matrix A.

7) Mathematical method used

The routine diagonalizes A (one column at a time) by a complete row elimination, and simultaneously develops B by carrying out the same row operations on I. There is no search for a largest pivot. Each prospective pivot is checked to be sure it is not zero, followed by some appropriate row operation if the pivot is zero.

All operations are carried out in floating binary, with 028 significant binary digits.

One can use this (or any other) matrix inversion routine to find the solution bA^{-1} of a system of linear equations $xA = b$ (where x, b are row vectors) as follows:

Invert the bordered matrix

$$C = \begin{bmatrix} 1 & -b \\ 0 & A \end{bmatrix}_{1 \quad n}$$

of order n + 1. Since

$$C^{-1} = \begin{bmatrix} 1 & bA^{-1} \\ 0 & A^{-1} \end{bmatrix}$$

the first row of C^{-1} contains the solution as its last n components.

A word of caution to the unwary: No routine can invert all non-singular matrices A. Errors in this routine arise from two sources: (1) In all operations, A is represented by the "digital" matrix A*, where the star denotes the rounding off of A to 028 significant binary digits. (2) In inverting A* all operations are rounded to 028 significant digits, and the output B is therefore not $(A^*)^{-1}$, and certainly it is not the true inverse A^{-1}. The relative importance of these two effects is little understood, but a small amount of evidence suggests the following:

1. Matrices

Suppose A, A^{-1} have the orders of magnitude 2^p, 2^q, respectively. Then B, $(A^*)^{-1}$, and A^{-1} all agree to something like 028-p-q binary digits. Whether or not the digits have any significance, code 00520.0 ordinarily punches out answers, with error halts only as in paragraph 6e.

8) All cells of the high-speed memory are used for routine and/or data. Many of the drum channels are used; a list of unused channels is with the code.

9) Flow chart is not available.

10) Code is in NAR library. Cards are filed in SWAC room.

1. Matrices

2. Inverting matrices

b. Code No. 00520.1

1) Written by Louise Straus, 1954.

2) Description prepared by Ruth Horgan and G. E. Forsythe, 1955.

3) Title Matrix Inverse Improvement, Floating Binary Point.

4) Summary:

Let matrix A and the first approximation B_o to its inverse each have n rows and n columns, $n \leq 032$. All the elements are in floating binary point notation. Routine attempts to improve the inverse, informs the operator of the maximum (in absolute value) element of $I - B_i A$, and according to the operator's wishes either executes another improvement step or punches out the last B_i. (For accuracy, see paragraph 7 below.)

5) Preparation of input data

Prepare all the matrix cards for A in one group, and those for B_o in another; the cards for A will precede those for B_o through the collator. Assemble the cards for each matrix as follows:

Prepare cards for only n rows. For $n \leq 010$, prepare one binary card per row--n elements per card (in floating binary point notation) accompanied by 010-n additional words. These additional words need not be zero; the routine ignores them. For $011 \leq n \leq 020$, prepare two floating binary cards--the first card containing the first ten elements, and the second card containing the remaining n-010 elements followed by 020-n additional words. Similarly, if $021 \leq n \leq 030$ prepare three cards; if n = 031 or 032 prepare four cards.

Example: if n = 017, the input data will consist of 068 cards--034 cards for A followed by 034 cards for B_o.

6) Operating procedure *

a) Output: Use standard 036-digit reproducer board, ten words per card.

b) Breakpoint switch must be on; typeout switch may be at either setting.

c) Drum is used.

---

*See code file in NAR library for chronological summary of input, computation, and output.

1. Matrices

d) Input: Use standard 036-digit collator board, ten words per card.

e) Operation: Read cards 000 through 091 of the routine into SWAC on continuous operation. Routine halts at 02 calling for input to cell ZY (254). Either type n in hexadecimal into the gamma part of the word and push 00PB, or read an "n in gamma" card from collator. Routine then halts at 0X (013), calling for input to cell 60 (096); read in A followed by B_0, followed by routine cards 092 through 154.

Routine halts in ZX (253) and then in ZY (254) calling for input to cells 05 and 9V (155). Type into these cells in hexadecimal the tolerances to be used in determining the symmetry of A and B_i, respectively. If A is non-symmetric, type in all Z's (equivalent of all 9's in decimal notation) at both input commands.

Routine then enters symmetry check, and if symmetry is acceptable will read the remainder of the routine cards from the collator. SWAC will halt at 00; push 00PB, and improvement takes place. At the end of each iteration of the improvement, routine halts at U8 (168) in a display breakpoint command. In the M register is displayed in floating binary notation the maximum error from the previous iteration; the R register displays the maximum error from the current iteration. See section 7) for description of maximum error. To return for another iteration, push 00PB; there will be another halt at U8 (168) after the next iteration.

To punch out the improved inverse after the halt at U8 (168), set SWAC on COMMAND mode of operation. Push 00PB (to get to period 7), clear epsilon, return to CONTINUOUS operation and push 00PB. The last improved inverse will punch out, on the same number of cards and in the same order as was the matrix initially read to SWAC from cards.

Before punching is begun, the routine does a symmetry check on the inverse to be punched. If the symmetry check is satisfied, the punchout takes place immediately. If it is not satisfied, see paragraph g) for symmetry check halts. Routine halts in 23 (035) when punchout is complete.

f) It takes ten minutes per improvement cycle for a matrix of order 029, and twenty seconds for a matrix of order eight.

g) Halts:

Symmetry check halts:

A halt at epsilon of 28 (040) with C register displaying 00,00,00,10,1 indicates that the matrix A failed to satisfy a symmetry check. The same halt with a C register of

1. Matrices

01,00,00,10,1 indicates that B_i failed to satisfy symmetry check. Handle both of these halts in the same manner; push 00PB and the routine next halts at a display breakpoint command, epsilon of 29 (041). The M register displays the difference between the two non-symmetric elements; the R register displays the contents of cell 45 (069), the tolerance just used for comparison.

Push 00PB again and routine halts at 2U (042), calling for input to cell 45 (069). Type in a new choice for the tolerance and push 00PB. The routine will check the remaining elements of the matrix under consideration (A or B_i) with the new value in 45 (069). If there is satisfaction, the routine will continue computing; if the symmetry check had just followed a "request" for punchout, punchout will now take place.

If the symmetry check fails with the new value in 45 (069), routine will again halt at 28 (040). Follow instructions given above for altering the value in 45 (069).

The next time in the computing that the routine is to execute a symmetry check on A or B_i, it will not use the last value typed into 45 (069) by the operator. It will use as tolerance the value initially typed into cell 05 or 9V (155) and transferred to 45 (069) for use; see first paragraph in part e.

Error halts:

A halt at 02 with C register of 01,00,00,10,1 indicates that there was overflow within the floating binary point subroutine.

A dynamic halt at 3Y (062), with the SWAC obeying the command 21,07,03,3Y,8 repeatedly, indicates "underflow"— a number has exceeded the minimum capacity of the floating binary point range. Use the R → C switch to obey the command 00,00,08,0W,7 and allow routine to continue; this substitutes zero for the answer which was too small.

In the event of drum-load failure during initial loading, the routine halts in a display breakpoint command at epsilon of 0X (013). The R register displays the number of the incorrect channel (in gamma) and the M register displays the amount of discrepancy. Push 00PB and SWAC halts in another breakpoint command at 1V (027). The C register indicates which of the 032 words had caused the halt. The R register displays the word read from the drum, and the M register displays the word from cards which

1. Matrices

should have been written on the drum. Push 00PB, and the routine will again attempt to write the 032 words on the drum and verify them by comparison. If there is disagreement again, there will be another halt at 0X (013). This will continue until there is agreement or until the operator tires.

h) Final halt: Routine halts at 23 (035) when punchout of improved inverse is complete.

i) Output: The improved inverse B_i is punched out in floating binary point notation on the same number of cards, and with the same arrangement of elements, as described for preparation of input matrices. See 5).

All of the punchout cards will have the same punching in columns 37 through 44 and 57 through 64. This will represent the first sixteen binary digits in floating binary notation of the last maximum error accepted by the operator at the halt in U8 (168). Column 37 is the most significant, and column 64 the least significant, of the sixteen digits.

7) Mathematical method used, apart from symmetry checks:

The routine uses an improvement method due to Schulz [Z.A.M.M. 13 (1933), 57-59], also known as Newton's method. Corresponding to each approximate inverse B_i, the routine computes

$$B_{i+1} = (2I - B_iA)B_i \quad .$$

All computations are performed with SWAC's floating binary point subroutines. If there were perfect precision, one would have

$$I - B_{i+1}A = (I - B_iA)^2 \quad .$$

Hence, if $\|I - B_iA\| < 1$ in some norm, one would have quadratic convergence of B_i to A^{-1}, as $i \to \infty$. In practice, round-off soon causes the process to terminate with a roughly fixed error.

The maximum error which is displayed in each cycle is the element of $I - B_iA$ with maximum absolute value. Since the critical computation of the first "residual" matrix $I - B_oA$ is done with the SWAC floating binary point scheme, each element is limited in accuracy to about 2^{-28} times the largest subtotal occurring in the accumulating sum for that element. If the elements of A, A^{-1} have orders of magnitude 2^p and 2^q, respectively, then the accumulating sums for $I - B_oA$ grow to about 2^{p+q} before decreasing to near zero.

1. Matrices

Thus the matrix $I - B_0A$ has about 028-p-q significant binary digits. Hence B_1 may be expected to have something like 029-p-q significant binary digits, even if B_0 were much more accurate. Limited experience suggests that, after a few iterations, B_i agrees with the A^{-1} to something like 030-p-q digits. (See also the discussion of errors in paragraph 7 under code 00520.0.)

If the maximum element of $I - B_0A$ is too large (i.e., $>> 1$) this routine may diverge entirely, with $B_i \rightarrow \infty$. In this case B_0, though a poor approximation to A^{-1}, is probably better than B_i ($i \geq 1$).

8) All cells of the high-speed memory and many channels of the drum are used for routine and/or data. A list of unused channels is with the code.

9) Flow chart is with code 00520.2; see code 00520.1 for exceptions.

10) Code is in NAR library. Cards are filed in SWAC room.

1. Matrices

2. Inverting matrices

c. Code No. 00520.2

1) Written by J. Pettit and L. Straus, 1954.

2) Description prepared by Ruth Horgan and G. E. Forsythe, 1955.

3) Title Combined Matrix Inversion and Improvement, Floating Binary Point.

4) Summary

Routine 00520.2 is a combination of routines 00520.0 and 00520.1. Given a matrix A with n rows and n columns, $n \leq 032$, this routine first obtains an approximate inverse B_0 by the method of routine 00520.0. It then applies the method of routine 00520.1 to improve the inverse for as many iterations as the operator desires. It informs the operator of the maximum (in absolute value) element of $I - B_iA$, and either executes another iteration of improvement or punches out the last B_i. All computations, input, and output are in floating binary point notation.

5) Preparation of input data

Prepare cards for only n rows. For $n \leq 010$, prepare one binary card per row--n elements per card (in floating binary point notation) accompanied by 010-n additional words. These additional words need not be zero; the routine ignores them. For $011 \leq n \leq 020$, prepare two floating binary cards--the first card containing the first ten elements, and the second card containing the remaining n-010 elements followed by 020-n additional words. Similarly, if $021 \leq n \leq 030$ prepare three cards; if n = 031 or 032 prepare four cards.

6) Operating procedure *

a) Output: Use standard 036-digit reproducer board, ten words per card.

b) Breakpoint switch must be on; typeout switch may be at either setting.

c) Drum is used.

d) Input: Use standard 036-digit collator board, with ten word input.

---

*See code file in NAR library for chronological summary of input, computation, and output.

1. Matrices

e) Operation: Read cards 000 through 136 of the routine into SWAC on continuous operation. Routine halts at 3Z (063) calling for input to cell 1X (029). Either type n in hexadecimal into the gamma part of 1X and push 00PB, or read an "n in gamma" card from collator. Routine next halts at 02 calling for input to cell U0 (160); start the matrix cards through the collator, followed by routine cards 137 through 199. Routine halts in ZX (253) and then in ZY (254) calling for input to cells 05 and 9V (155). Type in hexadecimal into these cells the tolerances to be used in determining the symmetry of A and B_i, respectively. If A is a non-symmetric matrix, type in all Z's (equivalent of all 9's in decimal notation) at both input commands.

Routine then enters symmetry check, and if symmetry is acceptable will enter the inverse routine. Routine checks symmetry of matrix and of inverse; if symmetry is acceptable, the remainder of routine cards feed through the collator. SWAC will halt at 00; push 00PB.

At the end of each iteration of the improvement, routine halts at U8 (168) in a display breakpoint command. In the M register is displayed in floating binary point notation the maximum error from the previous iteration; the R register displays the maximum error from the current iteration. See section 7) for description of maximum error. To return for another iteration, push the 00PB; there will be another halt at U8 (168) after the next iteration.

To punch out the improved inverse after the halt at U8 (168), set SWAC on COMMAND mode of operation. Push 00PB (to get to period 7), clear epsilon, return to CONTINUOUS operation and push 00PB. The last improved inverse will punch out, on the same number of cards, and in the same order, as was the matrix initially read to SWAC from cards. Before the punching is begun, the routine does a symmetry check on the inverse to be punched. If the symmetry check is satisfied, the punchout takes place immediately. If not, see paragraph g) for symmetry check halts. Routine halts in 23 (035) when punchout is complete.

f) For n of ten, the inverse part of routine takes about 30 seconds and the improvement about 25 seconds per cycle. For n = 8, improvement takes 21 seconds per cycle. For n = 032, inverse takes 20 minutes, and improvement about the same time.

1. Matrices

g) Halts:

Symmetry check halts:

A halt at epsilon of 28 (040) with C register displaying 00,00,00,10,1 indicates that the matrix A failed to satisfy a symmetry check. The same halt with a C register of 01,00,00,10,1 indicates that B_i failed to satisfy symmetry check. Handle both of these halts in the same manner; push 00PB and the routine next halts at a display breakpoint command, epsilon of 29 (041). The M register displays the difference between the two non-symmetric elements; the R register displays the contents of cell 45 (069), the tolerance just used for comparison.

Push 00PB again and routine halts at 2U (042), calling for input to cell 45 (069). Type in a new choice for the tolerance and push 00PB. The routine will check the remaining elements of the matrix under consideration (A or B_i) with the new value in 45 (069).

If the symmetry check is satisfied, the routine will continue computing; if the symmetry check had just followed a "request" for punchout, punchout will now take place. If the symmetry check fails with the new value in 45 (069), routine will again halt at 28 (040). Follow instructions given above for altering the value in 45 (069).

The next time in the computing that the routine is to execute a symmetry check on A or B_i, it will not use the last value typed into 45 (069) by the operator. It will use as tolerance the value initially typed into cell 05 or 9V (155) and transferred to 45 (069) for use; see first paragraph in part e.

Error halts:

A halt (during the inverse part of routine) at epsilon of 54 (084) indicates that matrix A is singular.

A halt at 02 with C register of 01,00,00,10,1 indicates that there was overflow within the floating binary point subroutine.

A dynamic halt at 3Y (062), with the SWAC obeying the command 21,07,03,3Y,8 repeatedly, indicates "underflow"—a number has exceeded the minimum capacity of the floating binary point range. Use the R → C switch to obey the command 00,00,08,0W,7 and allow routine to continue; this substitutes zero for the answer which was too small.

1. Matrices

In the event of drum-load failure during initial loading, the routine halts in a display breakpoint command at epsilon of OX (013). The R register displays the number of the incorrect channel (in gamma) and the M register displays the amount of discrepancy. Push OOPB and SWAC halts in another breakpoint command at 1V (027). The C register indicates which of the 032 words had caused the halt. The R register displays the word read from the drum, and the M register displays the word from cards which should have been written on the drum. Push OOPB, and the routine will again attempt to write the 032 words on the drum and verify them by comparison. If there is disagreement again, there will be another halt at OX (013). This will continue until there is agreement or until the operator tires.

h) Final halt: Routine halts at 23 (035) when punchout of improved inverse is complete.

i) Output: The improved inverse B_1 is punched out in floating binary point notation on the same number of cards, and with the same arrangement of elements, as described for preparation of input matrix. See paragraph 5).

All of the punchout cards will have the same punching in columns 37 through 44 and 57 through 64. This will represent the first sixteen binary digits (in floating binary notation) of the last maximum error accepted by the operator at the halt in U8 (168). Column 37 is the most significant, and column 64 the least significant, of the sixteen digits.

7) Mathematical method used: See descriptions of Codes No. 00520.0 and 00520.1.

8) All cells of the high-speed memory and many channels of the drum are used for routine and/or data. A list of unused channels is with the code.

9) Flow chart is with the code.

10) Code is in NAR library. Cards are filed in SWAC room.

1. Matrices

2. Inverting matrices

d. Code No. 00520.3

1) Written by Louise Straus, 1954.

2) Description written by G. E. Forsythe, 1955.

3) Title Matrix Inverse Improvement, Floating Binary Input and Binary Output

4) Summary

Let matrix A and the first approximation B_0 to its inverse each have n rows and n columns, $n \leq 032$. All elements of A and B_0 are in floating binary point notation. Routine converts A and B_0 to binary notation. It then attempts to improve the inverse, informs the operator of the maximum (in absolute value) element of $I - B_i A$, and, according to the operator's wishes, either executes another improvement step or punches out the last B_i in binary notation. (In a certain sense B_i is very accurate; for details see paragraph 7 below.)

5) Preparation of input data

Prepare all the matrix cards for B_0 in one group, and those for A in another; the cards for B_0 will precede those for A through the collator (reversed order from code no. 00520.1). Each row of each matrix must have forty binary words, the last 040-n of which are zero words.

Also, prepare one card with three words in this order:

9 row) n in the alpha position
8 row) the coded value of the largest exponent of any element of B_0, placed in alpha position. (For example, 8U would say that the largest element of B_0 had 8U in its alpha position.)
7 row) same for A.

Thus the input data will consist of 1 + 8n cards.

6) Operating procedure

a) Output: Use standard 036-digit reproducer board, ten words per card.

b) Breakpoint switch on. Typewriter switch: either setting.

c) Drum must be on.

1. Matrices

d) Input: Use standard 036-digit collator board, ten words per card.

e) Operation: The input deck consists of:

cards 000 through 015, followed by the card with 3 binary words (see paragraph 5), the 4n cards for B_o, the 4n cards for A, and then cards 016 through 093 of the routine.

Read this deck into SWAC on continuous operation. SWAC stops at 10 (016) for input to cell 9Z (159). Push 00PB, and the improvement routine begins. If all is well, it stops in a display breakpoint command at cell U8 (168). In the M register is displayed (in unscaled binary notation) the maximum error (defined in paragraph 7) from the previous iteration (zero the first time); in the R register is the maximum error from the current iteration. For a new iteration, push 00PB. There will be a halt at U8 (168) after each iteration.

To punch out the improved inverse after one has attained the desired accuracy, wait for a halt in U8 (168), and then set SWAC on the COMMAND mode of operation. Push 00PB to get to period 7, clear epsilon, return to CONTINUOUS operation, and push 00PB. The last improved inverse will punch out in binary notation on just as many cards as necessary (010, 020, 030, or 040 WP per row of the matrix), ten WP per card.

f) It takes 17 seconds per improvement cycle for a matrix of order 7. Other times are not known.

g) Error halts:

The main error halt is at cell V5 (181). This may indicate that the maximum exponents (see paragraph 5) have been incorrectly prepared, and should be fixed. (However, the absence of a halt at cell V5 (181) does not guarantee the correctnessof the exponents.)

A halt at V5 (181) or at 0Z (015) may occur if any element of the residual matrix $I - B_oA$ exceeds one in absolute value. For the latter difficulty the only cure is to seek a better approximate inverse B_o elsewhere—perhaps by using code no. 00520.1 first. Routine 00520.3 will not run when the elements of B_oA are too large in absolute value.

h) Final halt: The routine halts at cell 13 (019) when the punchout is complete.

1. Matrices

i) Output: The output is in pure binary, not floating binary. All elements are binary fractions, and to give the improved inverse should be multiplied by 2 to the exponent prepared for B_o under paragraph 5 above.

7) Mathematical method used The basic improvement method used here, as in code no. 00520.1, is the Schulz algorithm in the form

$$B_{i+1} = (2I - B_i A)B_i \ .$$

In the present code, however, the elements of the matrices B_o and A are first converted to 36-digit binary numbers, with a scale for each matrix. The scaled products $b_{ih}a_{hj}$ of the matrix multiplications are kept with 72-digit precision, and accumulated with a third cell for overflow. The final product $\Sigma_{h=1}^{n} b_{ih}a_{hj}$ is unscaled and rounded to a pure binary fraction. As a result, the residual matrix $I - B_o A$ is known to 36-digit precision. Because of the care used, if $\|I - B_o A\| < 1$ in some norm, the B_i will eventually agree to practically 36 significant binary digits with the inverse of the floating binary matrix A.

The maximum error displayed at each breakpoint is the element of $I - B_i A$ of greatest absolute value.

The precision obtainable with the code seems to be rarely needed. The floating point representation of matrix A ordinarily differs from the true A so much that the B_i obtained from this code is far too precise a representation for A^{-1}. See paragraph 7 under code no. 00520.0.

Perhaps the main use of code no. 00520.3 is to get a bound without round-off error on the absolute values of the elements of $I - B_o A$. If the matrix $I - B_o A$ is wanted, use code no. 00520.4.

It is believed that it should not be difficult to change this routine so that it would accept B_o and A in pure binary form. This would be a valuable improvement.

8) All the high-speed memory and much of the drum is used.

9) and 10) Cards are filed in SWAC room. Flow charts and code are unavailable. (It was considered that code no. 00520.3 would be entirely superseded by codes no. 00520.1 and 00520.2.) It is known that subsections of the routine begin at cards 000, 016, 021, 025, 053, 081, and 089. Originally these were separate codes. A decimal listing of the code, with a very few remarks, is filed in the NAR library.

1. Matrices

2. Inverting matrices

e. Code No. 00520.4

1) Written by Louise Straus, 1954.

2) Description written by G. E. Forsythe, 1955.

3) Title: Computation of Binary Residual Matrix, Floating Binary Input.

4) Summary

The floating binary matrices A and an approximate inverse B_o are as in code no. 00520.3. The present code punches out $I - B_oA$ in binary, provided its largest element does not exceed one in absolute value. The routine is very accurate, and the round-off error in $I - B_oA$ does not exceed 2^{-37}. With some card handling, one can proceed to compute the B_i defined in code no. 00520.3, punching out $I - B_iA$ ($i = 1, 3, \cdots$), and punching out the last B_i, all in binary. Unless one wants to have some $I - B_iA$ punched out, code no. 00520.3 is preferable, because it gives the last B_i without card handling.

5) Preparation of input data: Prepare the input data just as for code no. 00520.3.

6) Operating procedure

a) Output: Use standard 036-digit reproducer board, ten words per card.

b) Breakpoint switch: on. Typewriter switch: either setting.

c) Drum must be on.

d) Input: Use standard 036-digit collator board, ten words per card.

e) Operation: The input deck is as in code no. 00520.3. Read the deck through the collator. SWAC punches the n rows of $I - B_oA$, the first punch command being in cell 55 (085), with alpha of C register reading 80 (128).

After the punchout, SWAC halts at cell U8(168) with a display breakpoint as in code no. 00520.3 (maximum errors from current and previous iterations displayed in M and R registers). Press 00PB, and SWAC halts at cell 09, calling for collator input to cell 50 (080).

1. Matrices

Here one must take the output of the punchout and run them through the collator. Then SWAC punches out the n rows of $I - B_1A$, etc.

If one desires to get B_i punched out, proceed as in code no. 00520.3 at U8 halt (last paragraph of section 5e).

f) Data on running time unavailable.

g) and h) See code no. 00520.3 for description of halts.

i) Output: The elements of $I - B_iA$ are unscaled binary fractions. The elements of B_i are as in code no. 00520.3.

7) Mathematical method used See code no. 00520.3.

8) All the high-speed memory and much of the drum are used.

9) and 10) Cards are filed in SWAC room. Flow charts and code are unavailable. This code was originally made of independent codes, subsequently tied together. Subsections of the code begin on cards 000, 016, 021, 025, 053, 081, and 089. A decimal listing of the code, with a very few remarks, is filed in the NAR library.

1. Matrices

3. Solving linear systems

a. Code No. 00530.2

1) Written by F. B. Meek, March 1954.

2) Description written by F. B. Meek.

3) Title Transformation of Integral Matrices to Triangular (Hermite) Form.

4) Summary

The purpose of the code is to obtain a triangular matrix B from a given matrix A by a series of row operations, equivalent to premultiplication by a unimodular matrix U, i.e., to obtain B = UA. B is to have positive integers b_{jj} on the main diagonal, zeros below the main diagonal, and integral elements b_{ij} $(i < j)$ above the main diagonal such that $0 \leq b_{ij} < b_{jj}$.

The code will attempt to handle any $n \times n$ matrix* A whose elements are integers $< 2^{26}$ in absolute value, $n \leq 133$. The code obtains B by a series of elementary row operations on A, namely, interchanging two rows, multiplying a row by minus one, and multiplying a row by an integer, K_i, and adding it to another row, row i. The only serious limitation to the code is that the integers K_i must be stored on the drum using only 026 binary digits. If the integers K_i become $\geq 2^{26}$ in absolute value, the code halts in a breakpoint (see list of failure halts, 6g below) and no further computation can be done.

This code also computes the determinant of A. The absolute value of the determinant of A is clearly the product of the diagonal elements of B. The code forms this product and it also keeps track of the sign of the determinant of A.

5) Preparation of input data

a) The "n card" must be prepared and inserted in the routine deck after card no. 161 (the cards are numbered 000 through 162 in columns 23, 24, 25). The "n card" has ten WP; the command 01,00,00,10,1 along with a δ to $\mathcal{E}$ punch in the X row; 10 • C punched in β of the 9 row; and n punched in β of the 8 row. Here n is the order of the matrix, C is

*A slight modification would allow the code to handle rectangular $(m \times n)$ matrices, $m \leq 133$, no limit on n. Thus a non-homogeneous system of linear equations can be reduced to triangular form and easily solved by desk calculation.

1. Matrices

the number of cards per matrix column, 10 • C is the number of words the SWAC will read in when it reads in a matrix column. It will later be seen that C = (n + 2)/010 rounded up to the nearest integer.

b) The binary matrix deck can be prepared using conversion routine 00216. The decimal matrix deck is made up of the cards for column 1 followed by the cards for column 2 and so on to column n. Let a_{ij} be the element of the given matrix A in the i-th row and j-th column. Let $a_{n+1,j} = \Sigma_{i=1}^{n} a_{ij}$. The decimal cards for column j contain the following numbers, one per card, and in this order: $j \cdot 2^{20} = j \cdot 1{,}048{,}576$; $a_{n+1,j}$; a_{1j}; a_{2j}; a_{3j}; ··· ; a_{nj}; and as many word-pulse cards as is necessary to make the total number of cards for column j a multiple of ten.

6) Operating procedure

a) Normal typewriter output. Standard 036-digit card-punch output, ten words per card.

b) Breakpoint switch on.

c) Drum on.

d) Standard collator input, ten words per card. The cards to be input are the routine deck (including the "n card," see 5a) followed by the binary matrix deck.

e) Operation:

Clear the memory and epsilon, press 00PB, and start the collator. SWAC will read in the drum-clearing routine, clear the drum, clear the memory, read in the drum-loading routine, load drum channels 10 through 2X with routines, clear the memory, read in the main routine and the "n card," and halt in a breakpoint with $\varepsilon = 28$. Push 00PB, and SWAC will read in the initial re-start card, read in the cards for column 1, compute a little bit, type out one word or punch a card, read in the cards for column 2, compute a little bit, type out one word or punch a card, and so on.

In general, the routine reads in column j, applies to it the elementary row transformations which have been applied to the previous columns, and makes some further transformations in order to make the elements a_{ij}, $i > j$, equal to zero and to make the elements a_{ij}, $i < j$, less

1. Matrices

than a_{jj}. If the resulting column is a 1 in the j-th position and zeros elsewhere it then types a word containing j in β to indicate this, or, if not, it punches as many cards as necessary to punch all a_{ij}, $i \neq j$. The first card punched will contain (and in this order) j, i = 1, a_{ij}, a_{2j}, a_{3j}, j, i = 4, a_{4j}, a_{5j}, a_{6j}.

SWAC will continue in this way until it has finished with column n, in which case it will type the determinant of A (mod 2^{36}) in binary and halt in a breakpoint with $\mathcal{E}$ = 48, α of C = 57, β of C = 50, β of M = j = n, R = the determinant of A, or until it has filled the 097 drum channels available for recording the row transformations. In the latter case, it will punch out the column j^* it is currently working on, read in column $j^* + 1$ and apply to it the transformations which have been applied to the previous columns and punch it out, and so on to column n. Finally it will punch a re-start card containing j^* and the current value of the determinant of A, and halt in a breakpoint with $\mathcal{E}$ = 28, α of C = 00, β of C = 55, β of M = j^*. At this point, run the last two cards out of the punch (use the switch which disconnects SWAC from the punch!), set aside the punchouts of columns j, $j < j^*$, put the last card, i.e., the re-start card, in front of the cards for column j^* and put the re-start card, the cards for columns j^*, $j^* + 1$, $\cdots$, n in the collator feed, push 00PB, and start the collator. One may, if one desires, get off SWAC at this point and continue at some later time by replacing the initial re-start card (no. 162 of the routine deck) with the newly punched re-start card and replacing the original matrix deck with the cards for columns j^*, $j^* + 1$, $\cdots$, n. The SWAC will continue as outlined above until it finishes with column n or until it again fills the drum channels with the records of the transformations. In the latter case, continue as above. In the former case, the transformation is complete and the output cards[+] can be converted with 0021l and listed with the "72 digit—5 word Decimal" tab board.

f) The time in minutes (not including the time necessary to convert the output from binary to decimal with 0021l) necessary to complete the computation is about .01 n^2, where n is the order of the matrix.

[+] Not the cards which were read back into SWAC.

1. Matrices

g) Error halts:

All coded failure halts are breakpoints of the output type. They are the following (all in hex):

ε	α of C	β of C	R	M	Relative code reference	Explanation
48	50	59	$\bar{j}$	j	11008	$\bar{j} \neq j$
28	04	5u	$\Sigma_{i=1}^{n} a_{ij}$	$a_{n+1,j}$	12008	$\Sigma a_{ij} \neq a_{n+1,j}$
*37	0u	50	0	j	16023	A is singular
2y	0u	07	0	$\Delta\Sigma$	17014	drum channel check sum not right
**4v	02	05	D	q	19011	division routine error
*4v	52	55	i	K_i	20011	$\lvert K_i \rvert$ is too large
*44	52	54	$\bar{i}$	$K_{\bar{i}}$	24004	$\lvert K_{\bar{i}} \rvert$ is too large
*49	51	55	i*	K_{i*}	26009	$\lvert K_{i*} \rvert$ is too large
*49	52	55	i	K_i	29009	$\lvert K_i \rvert$ is too large
4w	5u	50	$a_{n+1,j}$	j	30012	$\Sigma_{i=1}^{n} a_{ij} \neq 1$
**4u	07	52	$\Delta\Sigma$	drum channel in γ	31010	drum error
3w	04	03	a_{ij}	$a_{ij} + a_{xj}K_i$	32028	overflow
4z	03	04	i_1	i_2	33015	$i_1 = i_2$
4z	02	50	i	j	35015	i = x
4u	52	06	i	r	38010	$i > j$, $r \neq 0$

*After these breakpoints, push 00PB and it will punch out the current column j and halt in another breakpoint with ε = 09.

**Push 00PB and let the machine try again.

1. Matrices

h) The final program halt is a breakpoint with $\mathcal{E} = 48$, α of C = 57, β of C = 50. See the third paragraph of 6)e).

i) The output cards (not the cards for j^*, $j^* + 1$, •••) should be converted from binary to decimal with 00211. The decimal cards can be listed with the "72 digit--5 word Decimal" tab board.

7) The method used in the code is essentially that method which is indicated by Theorems 103.4 and 103.5, pages 228-230 of Introduction to Abstract Algebra by C. C. MacDuffee.

8) Most of the high-speed memory and much of the drum used.

9) and 10) This code is identical with Meek's code 54-01.7. The flow chart, code, and other information pertinent to the code are in the SWAC code file in the NAR library entitled 00530.2. Cards are filed in the SWAC room.

1. Matrices

4. Computing eigenvalues

a. Code No. 00540.0

1) Written by Leo G. Komai, Physics Department; March 1956.

2) Description written by Leo Komai.

3) Title. Eigenvalues and Eigenvectors of a Real Symmetric Matrix of Order n $\leq$ 010, Jacobi Method.

4) Summary:

The routine is loaded on the drum. The n^2 elements of the matrix are fed in on cards. The routine then proceeds to compute and punch out the n eigenvalues. If desired, the routine will also compute and punch out the n eigenvectors.

5) Preparation of input data

The input data must be in binary form and consists of the following:

a) A binary card which includes two words.
   i) n in alpha, where n ($\leq$ 010) is the order of the matrix.
   ii) $\pm$ Identification which will be punched out with the eigenvalues. A positive word of identification will result in computation and punching out of the eigenvalues as well as the eigenvectors (see 6.g. iii). A negative or zero word of identification will result in computation and punching out of the eigenvalues only.

b) n 010-word binary cards, each of which contains, in the proper order, one row of the matrix (i.e., n words followed by 010 - n zeros). The n^2 matrix elements, a_{ij}, must be punched as numbers of absolute value less than 1, with the binary point just to the left of alpha 1, and must also be such that $\Sigma_{i,j=1}^{n} a_{ij}^2 < 1$.

6) Operating procedure

a) Use standard 36-digit output board with ten words out.
b) Breakpoint on.
c) Drum is used for storing routine and answers.
d) Standard SWAC collator input board.
e) Machine operation:
   i) Read routine into SWAC on continuous.
   ii) SWAC halts in (01): 9u, 00, 00, 00, 1.
   iii) Read in input data (see 5 above).
   iv) SWAC starts computing.

1. Matrices

v) SWAC punches out two binary cards. The first binary card contains the n eigenvalues (the binary point just to the left of alpha 1) followed by 010 - n zeros. The second binary card contains $\pm$ identification (see 5a), eight zeros, and N, the number of rotations performed by the routine, in alpha (see 7).

If identification was negative:

vi) SWAC halts in (01): 9u, 00, 00, 00, 1 for new input data, see step ii) above.

If identification was positive:

vii) The routine computes the eigenvectors and punches out 010 binary cards as follows:
The first n binary cards contain the eigenvectors (one eigenvector per card: n words followed by 010 - n zeros). The next 010 - n binary cards are zeros. SWAC halts, awaiting new input data. (See step ii above.)

f) The routine requires roughly 60 to 120 seconds for a tenth-order matrix, and less for smaller matrices.

g) Failure halts:

i) Halt in:
(0x): 09,1y,00,60,2
indicates failure to load routine properly on drum;
M register: shows 40,00, ξ ,70,1
R register: shows erroneous digits
where ξ is the number of the failing channel.

Press 00PB; machine halts in:
(1v): 40 + λ, 60 + λ, 00,60,2.
The failure is here pinpointed by displaying on the R register the contents of (40 + λ), the drum input data, and on the M register the contents of (60 + λ), the drum output data, which have failed to compare. Press 00PB; routine will try to load the same drum channel again until drum input and output data agree.

ii) Halt in:
(98): 00,00,00,60,2
indicates a machine failure which produces an impossible overflow, or an overflow due to excessively large matrix elements (see 5b).

1. Matrices

iii) Halt in:
(43): 00,00,00,60,2 or (91): 00,00,00,60,2
indicates that the required number of rotations is greater than 944 and thus exceeds the storage space allotted to them. Press 00PB to continue computation of eigenvalues. The routine will not compute the eigenvectors.

h) Final program halt:

After completing computation and punch of eigenvalues and (if desired) eigenvectors, SWAC always halts in:
(01): 9u,00,00,00,1
awaiting new input data (see 6.e.ii.).

i) Binary output may be converted to decimal using any binary to decimal conversion routine for numbers of absolute value less than one.

7) The original Jacobi method is used for the determination of the eigenvalues. An n^{th} order matrix is an operator in an n-dimensional vector space. The routine selects the largest off-diagonal element of the matrix and performs successive rotations of the basis vectors in the plane defined by this largest off-diagonal element. The search for the largest off-diagonal element occurs initially and after each rotation. The exact angle required for the complete annihilation of the off-diagonal element a_{ij} is given by $\tan 2\theta = 2a_{ij}(a_{ii} - a_{jj})^{-1}$. Instead of forming the appropriate trigonometric functions of this angle, the routine stores the sines and cosines of the 025 angles $\theta_p = 2^{011-p}$ minutes $(0 \leq p \leq 024)$ on the drum, and selects the functions of the angle $\pm \theta_p$, where p is chosen so that $2^p|a_{ij}| \geq |a_{ii} - a_{jj}| > 2^{p-1}|a_{ij}|$. Thus, if $a_{ii} = a_{jj}$, there is a rotation of $\pm 2^{11}$ minutes $\approx 34^o$.

1. Matrices

8) Cells and Drum Channels being used:

Cells 156-255 are used in the eigenvalue routine to store the matrix to be diagonalized and, in the eigenvector routine, to store the complete rotation matrix whose rows are the eigenvectors. The remaining cells are used for storing the routine proper.

Drum channels 016-143 are used in the following manner:

1st quarter	2nd quarter	3rd quarter	4th quarter
016-040: cosines and sines and constants	Coded rotations stored here, 016-133 incl.	016-019: subroutines	
041: subroutine	143: subroutine		

The eigenvalue routine is stored in channels 134-138.

The eigenvector routine is stored in channels 139-142.

9) Flow chart and code are in the NAR library.

10) Cards are filed in the SWAC room.

1. Matrices

4. Computing eigenvalues

b. Code No. 00545.0

1) Written by Louise Straus, 1954; G. E. Forsythe and Ruth Horgan know a little about it.

2) Description written by G. E. Forsythe.

3) Title To Obtain the Characteristic Polynomial of a Matrix by Frame's Algorithm.

4) Summary

This routine will take a matrix A of order n ≤ 014 in floating row form, and type out the negative coefficients of its characteristic polynomial in decimal fractions and exponents of two.

5) Preparation of input data

The input deck contains 069 + 4n cards, arranged in the following order:

Drum-load routine, cards 000 through 008.

A marker card (with no column 5 punch), card 009.

A one-word-pulse card with n in delta, in binary.

Next 2n cards: Rows 1 to n of matrix A, two cards per row. Each row has 020 binary words. The first word contains in delta an integer p, where 2^p is a common factor for all elements of the row. Then follow the n elements, and 019-n words of zero.

Next 2n cards: Rows 1 to n of the identity matrix in the same representation as A. In addition, the sixteenth word of each row of the identity matrix (but not A) must be the negative sum (ignoring overflows) of the binary words in the first fifteen words of the row. Thus the first row of the identity matrix should have these twenty words:

1) 00 00 00 01 0 (hex)
2) 80 00 00 00 0
3) to 15)
00 00 00 00 0
16) -80 00 00 01 0
17) to 20)
00 00 00 00 0

1. Matrices

A set of properly prepared identity matrices is filed with code no. 00545.0 cards. It is necessary to put the first 2n of these into the input deck.

Remainder of routine, cards 010 through 067.

6) Operating procedure

a) Output: Output is entirely by typewriter, arranged to type eleven digits per word.

b) Typewriter switch: CONVERTED setting. Breakpoint switch: OFF.

c) Drum must be on.

d) Input: Use standard 036-digit collator board, ten words per card.

e) Operation: Read the input deck into SWAC on continuous operation. SWAC will halt at epsilon of 3V (059); push 00PB. Routine then alternately computes and types out two words, without ever stopping.

f) Data on running time unavailable.

g) Error halt: A halt in epsilon of 16 (022) indicates failure of memory sum on drum-read. On COMMAND mode of operation, find the bad channel and half channel. When corrected, start over.

h) Final halt: There is no final program halt.

i) Output: Consider the characteristic polynomial in the form

$$A^n - c_1 A^{n-1} - c_2 A^{n-2} - \cdots - c_n I \quad .$$

A typical typeout is

00000 00002 0, 81249 99999 8, -00000 00017 9,
-12345 67899 9, etc.

This is to be interpreted in decimal as follows:

$$c_1 = .81249\ 99999\ 8 \times 2^2$$

$$c_2 = .12345\ 67899\ 9 \times 2^{-18}$$

etc.

1. Matrices

The numbers c_{n+1}, c_{n+2}, ... are zero, except for round-off error. The exponents are really decimal integers, but there is some conversion round-off.

7) Mathematical method used

The routine follows a method of evaluating the coefficients of the characteristic polynomial discovered by Souriau, Frame, and Faddeev-Sominskiĭ. See P.S. Dwyer, "Linear Computations," page 225. The inner products of the matrix multiplications are accumulated in triple precision, and finally rounded to single precision floating vectors.

The round-off errors in the c_i grow alarmingly fast, so the routine has a limited applicability. The early coefficients c_1, c_2, c_3 are ordinarily quite accurate enough, but c_n has a relative error something like $2^{-40}p^{n/2}$, where $P = |\lambda_n/\lambda_1|$ is the ratio of the largest (in modulus) to the least (in modulus) eigenvalue. For a discussion of the round-off error, see G.E. Forsythe and Louise Straus, J. Math. Physics, 34 (1955), 152-156.

8) Routine uses all the high-speed memory, and much of the drum.

9) and 10) Flow chart and code are not available. Cards are filed in SWAC room.

1. Matrices

4. Computing eigenvalues

c. Code No. 00545.1

1) Written by Louise Straus, 1954.

2) Description prepared by Ruth Horgan and G. E. Forsythe, 1955.

3) Title Power Method to Find Dominant Eigenvector of a Matrix in Floating Row Notation.

4) Summary

Let matrix A have n rows and columns, $n \leq 030$, and let vector x_1 have n elements. The routine forms $Ax_0 = x_1$, $Ax_1 = x_2$, $Ax_2 = x_3$, ••• . After a specified number, N, of iterations, the routine punches vectors x_i at each succeeding iteration until halted by operator. All operations are carried out in floating row notation; for description, see "Fixed Binary Point to Floating Row Conversion" routine, code no. 00510.3.

5) Preparation of input data

Routine expects to find matrix A on the drum, one row per channel, on consecutive channels beginning with channel 10 (016). Of the 032 words of a given channel, the first word should be the exponent (in delta) associated with the row; the next n words should be the n elements of the row, in floating row notation.

The last word should be the sum of the first n + 1 words, with overflow ignored. The routine assumes that only the first n + 1 words of each channel have been included in the given channel sum; if additional non-zero words had been included in the given channel sum by a drum-load process, this routine (00545.1) would incorrectly indicate drum-read failure. Drum-load routine number 00510.5 will load A on the drum in the prescribed manner.

Prepare the initial vector x_0 in floating row notation, beginning with the exponent p (in delta) followed by the n elements, as if it were a row of A. It is never stored on the drum in this routine. It is to be placed before card 010 of the routine, and loaded to cells 60 (096) through 60 + n (096 + n) during initial routine input. If desired, the operator may prepare more than n + 1 words for x_0 and they will read in without mishap, if to cells less than WO (192). Any words read in beyond 7Y (126), however, will be destroyed during computation. All others are left undisturbed; if they come within the range of the r cells punched per vector, they will also be punched (see i).

1. Matrices

Prepare a binary instruction card of three words as follows:

1st word: n, the order of the matrix, in hexadecimal in delta part of word.

2nd word: N, the number of iterations to be executed before punchout, in hexadecimal in delta part of word.

3rd word: r, in hexadecimal in delta part of word. (r = 010, 020, or 030. Parameter r is the smallest multiple of ten which is greater than or equal to n. It is the number of words punched out per vector.)

This three-word card follows the last routine card at time of initial loading.

6) Operating procedure

a) Output: Use standard 036-digit reproducer board, ten words per card.

b) Set typeout switch at "normal". Breakpoint switch setting immaterial.

c) Drum must be on.

d) Input: Use standard 036-digit collator board, ten words per card.

e) Operation: Read routine into SWAC on continuous operation, with the initial vector cards inserted between routine cards 009 and 010. Following routine card 017 should be the instruction card described above, containing three words. Routine will immediately begin, and will start punching x_i only after N iterations. The punchout of each vector x_i is preceded by the typeout of its exponent, in hexadecimal, in the delta part of the word.

The typeout and punchouts will then occur at each iteration until halted by operator. To halt the routine press the STOP button on the console during a typeout; do not release the button until the typeout is completed, or SWAC will not halt. Remove the vectors x_i from the reproducer, and inspect for convergence. If more iterations are desired, place SWAC on continuous operation, press 00PB, and routine will continue.

1. Matrices

Before N iterations have been completed, the operator may wish to reduce N and have punchout take place sooner. To change N, press the STOP button on the console; use the R → C switch to obey the command ZV,00,00,00,1. Type the new value for N in hexadecimal into the delta part of cell ZV (251) and allow SWAC to continue.

Once the punchout of vectors has begun, the operator may wish to compute for M more iterations without punchout. To do this, stop the routine during a typeout as described above; epsilon is now at 05. Use the R → C switch to obey the command ZV,00,00,00,1; type the value for M in hexadecimal into the delta part of ZV (251). Let the routine pick up again at 06; it will punch the vector whose exponent was typed out at 05, and continue computing. The next typeout and punchouts will occur after the next M iterations.

f) Data on running time unavailable.

g) Error halt: If the sum-check of a drum channel fails, the routine informs operator of the drum-read failure by going into a loop. It obeys the following sequence of commands repeatedly:
16; 17; 11; 12 through 14, n + 1 times; 15.

h) Final halt: Routine continues indefinitely if not halted by operator; see section e.

i) Output: Preceding each vector punchout is a one-word typeout—the exponent p corresponding to that vector. The punchout is r words per vector, ten binary words per card; the exponent is not punched out. Punchout is from cells 61 (097) through 60 + r (096 + r). If more than n + 1 words were read into SWAC in vector x_0, any of those additional words stored in cells less than 61 + r (097 + r) will be punched out in x_i. Otherwise there will be r-n words of zero punched to complete the ten word multiple per vector.

If it is desired to read some vector x_i back into SWAC as x_0 with a subsequent initial loading of the routine, it should be preceded by a binary card with its exponent in delta. Reading r instead of n + 1 words into x_0 is permissible; see second paragraph of 5).

7) Mathematical method used

The products involved in computing $x_{i+1} = Ax_i$ are formed and accumulated with double precision, with a third cell reserved for overflow. The largest in magnitude of the components of x_{i+1} is then normalized and compressed to 036 significant digits of a single word; the exponent p of x_{i+1} is correspondingly corrected. All other components of x_{i+1} are altered to fit the new exponent.

1. Matrices

The power method for getting eigenvectors is described, for example, by Aitken [Roy. Soc. Edinburgh, Proc., A 57 (1936-37), 269-304]. The vector $x_i = A^i x_o$ is asymptotically (as $i \to \infty$) in the invariant space spanned by the eigenvectors belonging to the eigenvalues of maximum modulus. When there is only one such eigenvalue, x_i asymptotically approaches one line in n-space. This case may be recognized by the convergence of the ratios of corresponding components of x_{i+1} and x_i, as $i \to \infty$.

When two eigenvalues dominate (for example, a pair of complex conjugates), it is more difficult to recognize convergence. For details on this case, see J. B. Rosser et. al. [J. Research, Nat. Bur. Stand. 47 (1951), 291-297].

8) Routine and data use almost all the high-speed memory. The matrix is stored on drum channels 10 through 0Z + n (016 through 015 + n).

9) Flow chart is with the code.

10) Code is in NAR library. Cards are filed in the SWAC room.

1. Matrices

4. Computing Eigenvalues (Nonsymmetric Matrices)

d. Code No. 00545.2

1) Written by G. E. Forsythe, June 1956.

2) Computation of the eigenvalues of a real tridiagonal matrix by the quotient-difference algorithm

3) Summary (written by G. E. Forsythe)

Consider the real tridiagonal matrix:

$$J = \begin{bmatrix} \alpha_1 & \beta_1 & & & & \\ 1 & \alpha_2 & \beta_2 & & & \\ & 1 & \alpha_3 & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \beta_{n-1} & \\ & & & & 1 & \alpha_n \end{bmatrix} \qquad (1 \leq n \leq 23) \quad .$$

Let the eigenvalues of J be $\lambda_1, \cdots, \lambda_n$, ordered so that:

(0) $$|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n| \quad .$$

For p = 1, 2, ••• , code 00545.2 computes the following quantities of H. Rutishauser's quotient-difference (QD) algorithm: [see references 1-3 below]:

(1) $$e_0^{(p)}, q_1^{(p)}, e_1^{(p)}, q_2^{(p)}, e_2^{(p)}, \cdots, q_n^{(p)}, e_n^{(p)} \quad .$$

Here $e_0^{(p)} = e_n^{(p)} = 0$ (all p). One gets the first row $q_i^{(1)}$ and $e_i^{(1)}$ from the relations:

1. Matrices

(2) $q_i^{(1)} = \alpha_i - e_{i-1}^{(1)} \quad (i = 1, \cdots, n)$;

(3) $e_i^{(1)} = \beta_i / q_i^{(1)} \quad (i = 1, \cdots, n-1)$.

One gets subsequent rows by the following relations ($p = 1, 2, \cdots$):

(4) $q_i^{(p+1)} = q_i^{(p)} + e_i^{(p)} - e_{i-1}^{(p+1)} \quad (i = 1, \cdots, n)$;

(5) $e_i^{(p+1)} = e_i^{(p)} q_{i+1}^{(p)} / q_i^{(p+1)} \quad (i = 1, \cdots, n-1)$.

The use of these quantities to compute the λ_i will be discussed below.

There are two parameters: n (the order of J) and k (the type-out frequency).

In the event any number exceeds capacity, there is automatic downscaling of all the $q_i^{(p)}$ and $e_i^{(p)}$ to one-tenth their previous values.

After any type-out (say the q-th) the operator has the option of changing once and for all to the accelerated form of the QD algorithm described on p. 507 of Ref. 2. In this one computes a quantity $t^{(p)}$ along with each row of $q_i^{(p)}$, $e_i^{(p)}$, as follows:

(6) $$\begin{cases} t^{(q)} = 0 \\ t^{(p+1)} = t^{(p)} + q_n^{(p)} \quad (p \geq q) \end{cases}$$.

Also, formula (4) is altered in the acceleration to read:

(4') $q_i^{(p+1)} = q_i^{(p)} + e_i^{(p)} - e_{i-1}^{(p+1)} - q_n^{(p)}$.

Note: $t^{(p)} = 0$ for the unaccelerated QD algorithm.

It is not possible to return from the accelerated

1. Matrices

form (4',5,6) of the QD algorithm to the normal form of the algorithm.

For p = rk (r = 1, 2, ···), the code types out $p.10^{-6}$, $t^{(p)}$, and then the quantities of (1), all as decimal fractions. Here k is an integer parameter which can be altered easily by the operator. At the operator's option one can punch out the same quantities in binary form.

4) Preparation of data

For a matrix of order n ($1 \leq n \leq 23$), one must prepare 2n + 5 words of data to be input in this order:

(1) n (in alpha) [order of matrix]

(2) zero (the initial value of $p.10^{-6}$) [row number]

(3) zero (the initial value of t) [Rutishauser's acceleration parameter, called t' on the coding sheets]

(4) zero = β_0

(5) α_1

(6) β_1

(7) α_2

(8) β_2

* * *

(2n+2) β_{n-1}

(2n+3) α_n

(2n+4) zero = β_n

(2n+5) k (in gamma) [type-out frequency].

1. Matrices

Cover the word-pulse of k, to run the cards out of the collator on loading.

5) Operating procedure

a) Output: Typewriter output converted, with option of punching if desired. Arrangement of jack plugs on reproducer immaterial!

b) Switches: Breakpoint switch on. Converted output switch on.

c) Drum: Not used at all.

d) Input: Read in routine, cards 000 to 025, numbered in columns 1-3. Follow with the 2n+5 words of data.

e) Machine operation: If the word-pulse of k was covered, SWAC halts in cell 0Y. Press 00PB. Then SWAC computes for a while, types 2n+3 words, and halts in cell 095 = 5Z with a display breakpoint. Reposition the typewriter carriage, and press 00PB. SWAC runs and then types again. Repeats.

Occasionally it may type out 100000000 followed by 0000[12]000 and halt. Reset typewriter carriage, press 00PB; SWAC will continue as before. This means that a downscaling of all q_i and e_i by 0.1 occurred for p = [12].

To enter the acceleration routine, press " $\delta \rightarrow \varepsilon$ " at a breakpoint halt in cell 095 = 5Z.

In the accelerated version of the QD algorithm, after $q_n^{(p)}$ has become zero, one can effectively change n to n-1 by transferring epsilon to 11h = 72 by use of the "R → C" switch. This should be done in cell 095 = 5Z on a breakpoint halt.

1. Matrices

If in the acceleration one wants to change n to n - 2, one has only to transfer to an epsilon of 255 = ZZ by use of the "R → C" switch.

If in the acceleration one wants to change n to any value m $\leq$ n - 2 it is only necessary to type m into the α - position of cell 146 = 92, and then transfer control to an epsilon of 111 = 6Z.

To punch out the current row of e_i and q_i, press "Cl ε" at a breakpoint halt in cell 095 = 5Z. The 2n + 3 words will be punched out and SWAC will go into a loop in cell 102 = 66. Halt SWAC with the STOP button. To return to the QD algorithm, change epsilon to 095 = 5Z with the "R → C" input switch.

To alter k, at a breakpoint halt in cell 095 = 5Z type the new value into the gamma position of cell 016 = 10 with the "R → C" input switch.

f) Duration: SWAC requires about .015n seconds per line of QD algorithm, or .015nk seconds between type-outs. Type-outs take almost 2 seconds per word.

g) Failure halts: Overflows in the QD algorithm automatically result in the downscaling of the whole line to 0.1 times its former value. Other overflows, which should be SWAC errors, go to cell 000, and should halt SWAC, since (000) is zero.

If a zero divisor should occur in the QD algorithm, it would halt SWAC in cell 051 = 33 with the command 00, 00, 00, 00, 1. If this occurs, one must start over with all values of

1. Matrices

the α_i translated to a new value $\alpha_i + \gamma$.

h) There is no final program halt.

i) The 2n + 3 words of output are the following:

(1) Current value of $p.10^{-6}$

(2) Current value of t

(3) $e_o^{(p)} = 0$

(4) $q_1^{(p)}$

(5) $e_1^{(p)}$

* * *

(2n + 1) $e_{n-1}^{(p)}$

(2n + 2) $q_n^{(p)}$

(2n + 3) $e_n^{(p)} = 0$.

If typed, these are converted to decimal.

6) The basic material on the mathematical method may be found in the references cited.

As Rutishauser shows, for the unaccelerated QD algorithm, if

$$|\lambda_{r-1}| > |\lambda_r| > |\lambda_{r+1}|,$$

(7) $\lim_{p\to\infty} q_r^{(p)} = \lambda_r$; while $\lim_{p\to\infty} e_{r-1}^{(p)} = \lim_{p\to\infty} e_r^{(p)} = 0$,

so that λ_r is immediately read from the type-outs. If

(8) $$|\lambda_{r-1}| > |\lambda_r| = |\lambda_{r+1}| > |\lambda_{r+1}|,$$

then $\lim_{p\to\infty} e_{r-1}^{(p)} = \lim_{p\to\infty} e_{r+1}^{(p)} = 0$, but $q_r^{(p)}$, $e_r^{(p)}$, and $q_{r+1}^{(p)}$ do not approach limits as $p \to \infty$. However, the following limits

1. Matrices

do exist:

$$a_2 = \lim_{p\to\infty} a_2^{(p)}, \text{ and } b_2 = \lim_{p\to\infty} b_2^{(p)} \quad .$$

Here

(9) $$a_2^{(p)} = q_r^{(p)} + e_r^{(p)} + q_{r+1}^{(p)} \quad ;$$

(10) $$b_2^{(p)} = q_r^{(p)}\, q_{r+1}^{(p)} \quad .$$

Moreover, λ_r and λ_{r+1} are the zeros of $\lambda^2 - a_2\,\lambda + b_2$.

(10') If $|\lambda_{r-1}| > |\lambda_r| = |\lambda_{r+1}| = |\lambda_{r+2}| > |\lambda_{r+3}|$, then let

(11) $$a_3 = \lim_{p\to\infty} a_3^{(p)} = \lim_{p\to\infty} \left\{q_r^{(p)} + e_r^{(p)} + q_{r+1}^{(p)} + e_{r+1}^{(p)} + q_{r+2}^{(p)}\right\} \quad ;$$

(12) $$b_3 = \lim_{p\to\infty} b_3^{(p)} = \lim_{p\to\infty} \left\{q_r^{(p)}\, q_{r+1}^{(p)} + q_r^{(p)} e_{r+1}^{(p)} + q_r^{(p)} q_{r+2}^{(p)} + e_r^{(p)} e_{r+1}^{(p)} + e_r^{(p)} q_{r+2}^{(p)} + q_{r+1}^{(p)}\, q_{r+2}^{(p)}\right\} \quad ;$$

(13) $$c_3 = \lim_{p\to\infty} c_3^{(p)} = \lim_{p\to\infty} \left\{q_r^{(p)}\, q_{r+1}^{(p)}\, q_{r+2}^{(p)}\right\} \quad .$$

Then λ_r, λ_{r+1}, λ_{r+2} are the zeros of $\lambda^3 - a_3\lambda^2 + b_3\lambda - c_3$. Similar formulas hold for groups of four more eigenvalues λ_i of equal modulus.

Thus a knowledge of the quantities (1) permits all the λ_i to be computed readily by the QD algorithm.

The expressions (9), (10), ··· , (13) differ from Rutishauser's in using only coefficients from a single line of the algorithms.

For the accelerated QD - algorithm the interpretation of the answers can be more complicated. The key point is that the effect of the acceleration in theory is to produce a p-th line

which is just what the unaccelerated algorithm would have produced as a p-th line, starting from $J - t^{(p)}I$. Hence the eigenvalues corresponding to the p-th line are $\lambda_i - t^{(p)}$. The confusion arises because $t^{(p)}$ may or may not converge, as $p \to \infty$. If $|\lambda_{n-1}| > |\lambda_n|$, then $t^{(p)} \to \lambda_n$. But if $|\lambda_{n-1}| = |\lambda_n|$, $t^{(p)}$ will normally not converge.

Whether or not $t^{(p)}$ converges, if $|\lambda_{r-1}| > |\lambda_r| > |\lambda_{r+1}|$, one seems empirically to find that

$$e_{r-1}^{(p)} \to 0 \text{ and } e_r^{(p)} \to 0, \text{ as } p \to \infty \quad .$$

In that case,

$$\lim_{p\to\infty} \left\{ q_r^{(p)} + t^{(p)} \right\} = \lambda_r \quad . \tag{14}$$

In particular, if $|\lambda_{n-1}| > |\lambda_n|$, one has $\lim_{p\to\infty} q_n^{(p)} = 0$ and

$$\lim_{p\to\infty} t^{(p)} = \lambda_n \quad .$$

The effect of the acceleration on a quadratic factor is as follows. Suppose (8) holds. If $\mu_r^{(p)}$, $\mu_{r+1}^{(p)}$ are the zeros of $\mu^2 - a_2^{(p)}\mu + b_2^{(p)}$, then

$$\lim_{p\to\infty} \left\{ \mu_r^{(p)} + t^{(p)} \right\} = \lambda_r \quad ;$$

$$\lim_{p\to\infty} \left\{ \mu_{r+1}^{(p)} + t^{(p)} \right\} = \lambda_{r+1} \quad .$$

In other words, if the zeros of $\mu^2 - a_2^{(p)}\mu + b_2^{(p)}$ are increased by $t^{(p)}$, the corresponding quadratic factor converges to $\lambda^2 - a_2\lambda + b_2$. To be explicit,

1. Matrices

$$\lim_{p\to\infty}\left\{a_2^{(p)} + 2t^{(p)}\right\} = a_2 \quad ;$$

$$\lim_{p\to\infty}\left\{b_2^{(p)} + a_2{}^{(p)}t^{(p)} + (t^{(p)})^2\right\} = b_2 \quad .$$

In a similar way, if (10') holds, one gets cubic factors which may be interpreted by augmenting the roots by $t^{(p)}$. Thus

$$\lim_{p\to\infty}\left\{a_3{}^{(p)} + 3t^{(p)}\right\} + a_3 \quad ;$$

$$\lim_{p\to\infty}\left\{b_3^{(p)} + 2a_3^{(p)}t^{(p)} + 3(t^{(p)})^3\right\} = b_3 \quad ;$$

$$\lim_{p\to\infty}\left\{c_3^{(p)} + b_3^{(p)}t^{(p)} + a_3^{(p)}(t^{(p)})^2 + (t^{(p)})^3\right\} = c_3 \quad .$$

The acceleration results in quadratic convergence of (14) when $|\lambda_{n-1}| > |\lambda_n|$. Its effect is certainly much less when

$$|\lambda_{n-2}| > |\lambda_{n-1}| = |\lambda_n| \quad .$$

However, it seems to be worthwhile.

Once an eigenvalue or quadratic or cubic factor has been isolated, one should decrease n by 1 or 2 or 3, and continue the algorithm without changing the q_i and e_i already eliminated. Directions for changing n are given under 5e above. This presupposes that $e_m^{(p)} = 0$.

This code operates in fixed-point single precision. The operation (5),

$$e_i^{(p+1)} = q_{i+1}^{(p)}\, e_i^{(p)} / q_i^{(p+1)} = AB/C \ ,$$

is the major source of loss of significance. In this experimental code, no step has been taken to save digits. It just forms AB, and then divides by C. An overflow leads to replacing A, B, C by A/10, B/10, C/10, with resultant loss of digits later. As a

1. Matrices

result, the code is not very accurate when there are complex eigenvalues.

The difficulty with complex eigenvalues of the form $re^{\pm 2\pi is}$ is this: If s is irrational, the corresponding values of $q^{(p)}$ (in theory) have as limit points (for $p \to \infty$) all real numbers. As a result, there must eventually be downscaling out of all significance. One hopes that the quadratic factor will have been separated before such disastrous downscaling has occurred.

For n = 23, all α_i = .2, all β_i = .01, the code (p = 100) computed λ_{23} = .001711029, where the true value is .0017110277+ = $4 \sin^2(\pi/48)$. For p = 500, the eigenvalues λ_1, λ_2, λ_3 had not been separated. A hand calculation from that point yields λ_1 = .398288991, where the true value is λ_1 = .3982889722-. Such accuracy seems possible only for matrices without complex eigenvalues.

The code types p = 1 for the top line, in contrast to Rutishauser's use of p = 0 here.

7) Cell Allocation

Cells 00 - 9y and zz are used for the code, constants and temporaries. Cells 9z - wy hold the old row of data. Cells wz - zy hold the new row of data.

8) Flow chart N.A.R. Library has a flow chart containing only cell numbers.

9) Code N.A.R. Library has the coding sheets.

1. Matrices

Code 00545.2 is not elegant or adaptable, but it runs. It is hoped later to rewrite it and combine it with a code for getting the tridiagonal form J of an arbitrary matrix A.

References:

1. H. Rutishauser: Z. angew. Math. Physik vol. 5 (1954), pp. 496-508.
2. H. Rutishauser: Z. angew. Math. Physik vol. 6 (1955), pp. 387-401.
3. P. Henrici, multilithed typescript on QD algorithm to appear in NBS Applied Math Series.

3. Partial Differential Equations

2. Boundary value problems

a. Code no. 00660.

1) Written by G. E. Forsythe, February 1955.

2) Description written by G. E. Forsythe.

3) Title

Overrelaxed Liebmann Process for Dirichlet Problem in a Square Net

4) Summary

Consider the set of lattice points P (x,y), where $016 \leq x \leq 143$, $000 \leq y \leq 031$, and x and y are integers. Consider the Laplace difference equation in two dimensions:

$$(1)\quad r(x,y) = u(x+1,y) + u(x-1,y) + u(x,y+1) + u(x,y-1) - 4u(x,y) = 0 \ .$$

Let R be a subset of P at which the values of u are to be determined. Let C be the subset of P at which the values of u are prescribed. For each point (x,y) in R, it is necessary that all four points $(x \pm 1, y \pm 1)$ be in $R \cup C$. There is no other restriction on R and C.

Code 00660 solves the difference equation (1) for all (x,y) in R, subject to u(x,y) being prescribed on C.

There are three parameters in the Code: β, the overrelaxation factor (see below); n, the maximum value of x in R ($n \leq 142$); f, the maximum value of y in R ($f \leq 030$).

The method used is successive overrelaxation of the values of u(x,y) in lexicographic order (see below).

5) Preparation of data

The set P is to be mapped on the drum, with x being the channel number and y, the cell number on each channel. Select a drum input routine, and prepare the drum data in a form suitable for it. Code 00684 is a suitable one; it uses 33 words per channel on 11-word input.

The set $R \cup C$ should be inclosed in as small a rectangle as convenient. Let this be $016 \leq x \leq n+1$, $000 \leq y \leq f+1$. Prepare data for drum channels 016, ..., n + 1, and for cells 000 to f + 1 in these channels. It is irrelevant what is in other channels and cells.

The points of C are identified by a 1 in the most significant binary digit; (this is removed before the arithmetic operations begin). At each point (x,y) of C put the prescribed value of u(x,y), scaled down far enough (perhaps less than 2^{-8} is a safe rule). At each point (x,y) of R one may put any desired trial values of the field u(x,y). (Zero is a very convenient start.)

6) Operating procedure

a. Output: There is no output from this code. It is ordinarily followed by a drum-punch routine—for example, 00683.

3. Partial Differential Equations

b. Switches: Breakpoint switch on.

c. Drum is used continually.

d. Input: First clear the drum, and then load the drum with the data for R and C. (I recommend Code 00684 for this.) Then clear the memory (automatic with Code 00684), and read in the program deck of Code 00660, consisting of cards 000 to 013, numbered in binary in columns 75-78 of the Y row.

e. Machine operation:

SWAC will first halt at an epsilon of 002, calling for an input to cell 69 = 105. If n is to be 8y = 142, just press 00PB. Otherwise, type correct n into gamma and press 00PB. SWAC halts at an epsilon of 003, calling for an input to cell 6u = 106. If f is to be 1y = 030, just press 00PB. Otherwise, type correct f into alpha, and then press 00PB. Next, SWAC halts at an epsilon of 009, calling for input into cell 69 = 105. If beta is to be left at its built-in value of unity, just press 00PB. Otherwise, type $2^{-3}\,\Delta\beta$ (2^{-3} times the desired correction to beta), and press 00PB.

SWAC now proceeds to relax away on the field R from channels 017 to n + 1, in order. After each sweep of the field, SWAC halts in cell 77 = 119 with a display breakpoint. In the R register is displayed the sum Σ of the absolute values of the residuals. In the M register is displayed the ratio of the current value of Σ to the value of Σ from the previous sweep. (If this ratio exceeds unity, the M register displays $-\Sigma$.)

After the display breakpoint the operator has three choices:

(1) he may press 00PB, and cause SWAC to make another relaxing sweep.

(2) he may press "Clear epsilon" and then 00PB. This causes SWAC to halt in cell 80 = 128 with a display breakpoint. In the R register is displayed $2^{-3}\beta$, while in the M register is displayed 2^{-3} times the last correction made to beta. Press 00PB again and SWAC halts in cell 009, where the operator may type in a new scaled correction to beta, as above. Another 00PB starts the relaxation sweep again.

(3) he may press "Clear memory" and "Clear epsilon," and read in a drum-punch routine to get the answers off the drum (for example, routine 00683).

The relaxation cycles may be speeded up by turning the breakpoint switch off.

f. Duration:

The only extended times occur during the relaxation. The exact time depends on the values of n and f. When n is 068, and f is 030, the time is about 8.5 seconds per sweep.

g. Failure halts: (Occur only with breakpoint switch on.)

Halt in 4z = 079: Machine error in division routine. Read code in again.

3. Partial Differential Equations

Halt in 54 = 084: Overflow in correcting beta. Beta is too large. Read code in again.

Halt in 82 = 130: Overflow in computing residual from (1). The values of u(x,y) are too large. Rescale them downward on the data cards, re-load the drum, and start over.

h. Final program halt: There is none. The operator stops when he is satisfied with the smallness of Σ.

i. Dealing with output: If one wants a decimal listing of the answers, they should be taken out on a drum-punch routine (e.g., 00707) in a form suitable for conversion with code 00219 or 00229, for example.

7) Mathematical method used
The new value of u(x,y) is always computed from the formula:

$$\text{new } u(x,y) = u(x,y) + \tfrac{1}{4}\beta\, r(x,y) \qquad (1 \leq \beta < 2).$$

The new value is immediately put back in the field, and is used for the next computation of r(x,y). The field is scanned lexicographically, with x as a major, and y as a minor index. The ideas are sketched in an article by G. E. Forsythe entitled, "Difference methods on a digital computer for laplacian boundary value and eigenvalue problems," 1955*. There are references to original papers by David Young and by S. P. Frankel.

8) The drum holds the whole field RUC. The routine is entirely within the high-speed memory. Cells u0 = 160 to zz = 255 are used to hold data transferred from the drum, while the routine is within cells 000 to 82 = 130.

9) Flow chart and code are in the NAR library.

10) Cards are filed in the SWAC room.

* To appear in Commun. Pure Appl. Math., 1956.

3. Partial Differential Equations

2. Boundary value problems

b. Code no. 00661

1) Written by G. E. Forsythe, February 1955.

2) Description written by G. E. Forsythe.

3) Title

Overrelaxed Liebmann Process for Laplace's Equation in a Square Net, Mixed Boundary Conditions.

4) Summary

This code is a modification of Code no. 00660, in which it is possible to deal with more general boundary conditions. Not only can u(x,y) be prescribed on the boundary, but it is possible to prescribe also that the normal derivative of u be zero at any point of the boundary. This is achieved by the following method.

Consider the set of lattice points P: (x,y), where $016 \leq x \leq 124$, $000 \leq y \leq 031$, and x and y are integers. Consider the Laplace difference equation in two dimensions:

$$(1)\quad r(x,y) = u(x+1, y) + u(x-1,y) + u(x,y+1) + u(x,y-1) - 4u(x,y) = 0 \quad .$$

Let R be a subset of P at which the values of x are to be determined. Let $C_{0,0}$ be the subset of P at which the values of u are prescribed. Let $C_{p,q}$ be the subset of points (x,y) of P at which the values of u(x,y) are to be made equal to u(x+p, y+q); here $-004 \leq p \leq 011$, $-004 \leq q \leq 011$. For each point (x,y) in R, it is necessary that all four points $(x \pm 1, y \pm 1)$ be in R or in one of the sets $C_{p,q}$. For each point (x,y) in $C_{p,q}$ it is necessary that the point (x+p, y+q) be in R or in one of the sets $C_{p,q}$.

Code 00661 solves the difference equation (1) for all (x,y) in R, subject to u(x,y) = u(x+p, y+q) for (x,y) in $C_{p,q}$ (for each p,q). For p = q = 0, the boundary condition is that u(x,y) be prescribed.

There are three parameters in the code: β, the overrelaxation factor (see below); n, the maximum value of x in R or in some $C_{p,q}$ other than $C_{0,0}$ ($n \leq 124$); f, the maximum value of y in R or in some $C_{p,q}$ other than $C_{0,0}$ ($f \leq 031$). For x = 016 and for x = n + 1, all points (x,y) must be in $C_{0,0}$ for certain technical reasons.

The method used is successive overrelaxation of the values of u(x,y) in lexicographic order (see below).

5) Preparation of data

The set P is to be mapped on the drum, with x being the channel number, and y, the cell number on each channel. Select a drum input routine, and prepare the drum data in a form suitable for it. Code 00684 is a suitable one; it uses 33 words per channel on 11-word input.

3. Partial Differential Equations

The sets R and $\{C_{p,q}\}$ should be inclosed in as small a rectangle as convenient. Let this be $016 \leq x \leq n + 1$, $000 \leq y \leq f + 1$. Prepare data for drum channels 016, ... , $n + 1$, and for cells 000 to $f + 1$ in these channels. It is irrelevant what is in other channels and cells.

It is essential that the drum channels number 016 and $n + 1$ consist entirely of points of $C_{0,0}$. If this is not automatic, one must increase n by one and add a channel of points in $C_{0,0}$.

The points of C belonging to any of the $C_{p,q}$ are identified by a 1 in the most significant digit. In addition, the values of p,q are tagged in each point of $C_{p,q}$ as follows: the digits alpha-six, alpha-seven, alpha-eight, and alpha-nine contain $p + 4$; the digits beta-one, beta-two, beta-three, and beta-four contain $q + 4$. All these identification digits are removed before the arithmetic begins.

At each point of $C_{0,0}$ put the prescribed value of u(x,y), scaled down far enough (it must be less than 2^{-12} to avoid conflicts with the identification digits). At each point (x,y) of R or of any $C_{p,q}$ (for $|p| + |q| > 0$), put any desired trial values of the field u(x,y). Zero is ordinarily a convenient trial value.

Although the set $C_{0,0}$ of this code has the same function as the set C of Code 00660, the identification tag is different. Hence code 00661 cannot accept the data cards of Code 00660. This unfortunate situation could be remedied.

6) Operating procedure

a. Output: There is no output from this code. It is ordinarily followed by a drum-punch routine — for example, 00683.

b. Switches: Breakpoint switch on.

c. Drum is used continually.

d. Input: First clear the drum, and then load the drum with the data. (I recommend Code 00684.) Then read in the program deck of Code 00661, consisting of cards 000 to 016, numbered in binary in columns 75-78 of the Y row.

e. Machine operation:

The code runs very much like 00660. SWAC will first halt at an epsilon of 002, calling for input to cell 52 = 082. If n is to be 54 = 084, just press 00PB. Otherwise type correct n into gamma, and press 00PB. SWAC halts at an epsilon of 003, calling for an input to cell 53 = 083. If f is to be 1z = 031, just press 00PB. Otherwise, type correct f into alpha, and then press 00PB. Next, SWAC halts at an epsilon of 009, calling for input into cell 52 = 082. If beta is to be left at its built-in value of unity, just press 00PB. Otherwise, type $2^{-3}\,\Delta\beta$ ($\Delta\beta$ is a desired additive correction to beta), and press 00PB.

3. Partial Differential Equations

SWAC now proceeds to relax away on the field from channels 017 to n + 1, in order. (The relaxation of channel n + 1 is necessary only in order to get the new values of u for channel n back to the drum.) After each sweep of the field, SWAC halts in cell 60 = 096 with a display breakpoint. In the R register is displayed the sum Σ of the absolute values of the residuals r(x,y) at the points of R. In the M register is displayed the ratio of the current value of Σ to the value of Σ from the previous sweep. A quotient >1 is indicated by $-\Sigma$.

After the display breakpoint the operator has three choices:

(1) he may press 00PB, and cause SWAC to make another relaxing sweep. (If the breakpoint switch is turned off, SWAC automatically goes through option (1), of course.)

(2) he may press "Clear epsilon" and then 00PB. This causes SWAC to halt in cell 9v = 155 with a display breakpoint. In the R register is displayed $2^{-3}\beta$, while in the M register is displayed $2^{-3}\Delta\beta$, i.e., 2^{-3} times the last correction made to beta. Press 00PB again, and SWAC halts in cell 009, where the operator may type in a new scaled correction to beta, as above. Another 00PB starts the relaxation sweep.

(3) he may press "Clear memory" and then "Clear epsilon", and read in a drum-punch routine to get the answers off the drum (for example, routine 00683).

f. Duration:

The only extended times occur during the relaxation. The exact time depends on the values of n and f, and on the number of points in the $C_{p,q}$ other than $C_{0,0}$. I have no data on the routine, except my memory that it is perhaps 50 per cent slower than 00660 in one case.

g. Failure halts: (Occur only with breakpoint switch on.)

Halt in 3x = 061: Overflow in adding $2^{-3}\Delta\beta$ to $2^{-3}\beta$.

Halt in 46 = 070: Overflow in getting new u(x,y).

Halt in 4z = 079: Overflow in computing r(x,y).

Rescale the u(x,y) downward on the data cards, and reload the drum.

Halt in 93 = 147: Error in division subroutine. Read code in again.

h. Final program halt: There is none. The operator stops when he is satisfied that Σ is small enough.

i. Dealing with output: If one wants a decimal listing of the answers, they should be taken out on a drum-punch routine (e.g., 00707) in a form suitable for conversion with code 00219 or 00229, for example.

7) Mathematical method used

For points of R, the new value of u(x,y) is always

3. Partial Differential Equations

computed from the formula

$$\text{new } u(x,y) = u(x,y) + \tfrac{1}{4}\beta r(x,y) \quad (1 \leq \beta < 2).$$

For points of any $C_{p,q}$, the code simply follows the rule

$$\text{new } u(x,y) = u(x + p, y + q) \quad .$$

In either case the new value is immediately put back into the field, and is used for the next computation employing that point. The field is scanned lexicographically, with x as a major, and y as a minor index. The ideas are sketched in an article by G. E. Forsythe entitled, "Difference methods on a digital computer for laplacian boundary value and eigenvalue problems," 1955.* There are references to original papers by David Young and by S. P. Frankel.

The use of the $C_{p,q}$ to approximate normal derivatives is described in the above paper. I should expect that one would always use p = -2, -1, 0, 1, or 2, with q the same.

8) Cell allocation

The drum holds the whole field of R and the $C_{p,q}$. The routine is entirely within the high-speed memory. It sends data out to channel 7y = 126 at one point in the calculation, however, so that the field must use only channels up to 7x = 125. This use of a channel for temporary storage is necessary, but the number could easily be changed. See the code.

Cells u0 = 160 to zz = 255 are used to hold data in the high-speed memory, while the routine is within cells 000 to 9z = 159.

9) Flow chart and code are in the NAR library.

10) Cards are filed in the SWAC room.

* Commun. Pure Appl. Math., 1956.

3. Partial Differential Equations

3. Eigenvalue problems

a. Code no. 00670

1) Written by G. E. Forsythe, February 1955.

2) Description written by G. E. Forsythe.

3) Title

Overrelaxed Liebmann Process for a Square Net, Fundamental Eigenvalue of Laplace's Operator.

4) Summary

Consider the set of lattice points P: (x,y), where $016 \leq x \leq 143$, $000 \leq y \leq 031$, and x and y are integers. Consider the difference equation in two dimensions:

$$(1)\quad r(x,y) \equiv u(x+1,y) + u(x-1,y) + u(x,y+1) + u(x,y-1) - \mu\, u(x,y) = 0 .$$

Let R be a subset of P at which the values of u are to be determined. Let C be the subset of P at which the values of u are prescribed. For each point (x,y) in R, it is necessary that all four points $(x \pm 1, y \pm 1)$ be in $R \cup C$. There is no other restriction on R and C.

Code 00670 solves the difference equation (1) for all (x,y) in R, subject to u(x,y) being prescribed on C. In addition, μ is a parameter which can be adjusted by the operator in the range $0 < \mu < 4$.

For the fundamental eigenvalue problem, one would prescribe u(x,y) to be zero on C, and adjust μ until the operator from u to r is semidefinite. Code 00681 will compute a Rayleigh quotient to assist in locating μ. Codes 00680 and 00682 help with scaling the u(x,y).

There are four parameters in the code: μ; β, the overrelaxation factor (see Code 00660); n, the maximum value of x in R ($n \leq 142$); f, the maximum value of y in R ($f \leq 030$).

The method used is successive overrelaxation of the values of u(x,y) in lexicographic order; see Code 00660.

5) Preparation of data

The set P is to be mapped on the drum, with x being the channel number, and y, the cell number on each channel. Select a drum input routine, and prepare the drum data in a form suitable for it. Code 00684 is a suitable one; it uses 33 words per channel on 11-word input.

The set $R \cup C$ should be inclosed in as small a rectangle as convenient. Let this be $016 \leq x \leq n+1$, $000 \leq y \leq f+1$. Prepare data for drum channels 016, ..., n + 1, and for cells 000 to f + 1 in these channels. It is irrelevant what is in other channels and cells.

The points of C are identified by a 1 in the most significant binary digit; this is removed before the arithmetic operations begin. At each point (x,y) of C put the

3. Partial Differential Equations

prescribed value of u(x,y), scaled down far enough (perhaps less than 2^{-8} is a safe rule). For the eigenvalue application one would normally have u ≡ 0 on C. At each point of R one may put any desired trial values of the field u(x,y). For the eigenvalue application with u ≡ 0 on C it is reasonable to make u ≡ 2^{-9} on R.

6) Operating procedure

a. Output: There is no output from this code. It is ordinarily followed by a drum-punch routine—for example, 00683.

b. Switches: Breakpoint switch on.

c. Drum is used continually.

d. Input: First clear the drum (automatic with Code 00684) and then load the drum with the data for R and C. Then clear the memory (automatic with Code 00684), and read in the program deck of Code 00670, consisting of cards 000 to 013, numbered in binary in columns 75-78 of the Y row.

e. Machine operation: SWAC will first halt at an epsilon of 002, calling for an input to cell 69 = 105. If n is to be 8y = 142, just press 00PB. Otherwise type correct n into gamma, and then press 00PB. SWAC halts at an epsilon of 003, calling for an input to cell 6u = 106. If f is to be 1y = 030, just press 00PB. Otherwise type correct f into alpha, and then press 00PB. Next, SWAC halts with a display breakpoint at an epsilon of 84 = 132. In the R register is displayed the "built-in" value of $\frac{1}{4}\mu$, namely 15/16, i.e., z0 in alpha. The M register is blank. Press 00PB. If $\frac{1}{4}\mu$ is to be left at its value of 15/16, just press 00PB again. Otherwise, type in an appropriate correction $\frac{1}{4}\Delta\mu$ to $\frac{1}{4}\mu$, and press 00PB again.

SWAC now proceeds to relax away on the field R from channels 017 to n + 1, in order. After each sweep of the field, SWAC halts in cell 77 = 119 with a display breakpoint. In the R register is displayed the sum Σ of the absolute values of the residuals r(x,y). In the M register is displayed the ratio of the current value of Σ to the value of Σ from the previous sweep. (If this ratio would exceed unity, the M register displays -Σ.)

After the display breakpoint the operator has four choices:

(1) he may press 00PB, and cause SWAC to make another relaxing sweep;

(2) he may press "Clear epsilon" and then 00PB. This causes SWAC to halt in cell 80 = 128 with a display breakpoint. In the R register is $2^{-3}\beta$, while in the M register is displayed 2^{-3} times the last correction

3. Partial Differential Equations

made to beta. Press OOPB again and SWAC halts in cell 009, where the operator may type in a new scaled correction $2^{-3}\ \Delta\beta$ to beta. Another OOPB starts the relaxation sweep again.
(3) he may press "delta to epsilon" and then OOPB. This causes SWAC to halt in cell 84 = 132 with a display of the current value of $\frac{1}{4}\mu$ in the R register, and the last value of $\frac{1}{4}\Delta\ \mu$ in the M register. Press OOPB again and SWAC halts in cell 85 = 133, where a new value of $\frac{1}{4}\Delta\ \mu$ may be typed in. Press OOPB again and the relaxation sweep begins again.
(4) he may clear the machine and read in another routine - e.g., 00680, 00681, 00682, or 00683 to deal with the data on the drum.

The relaxation cycles may be speeded up by turning the breakpoint switch off. Then alternative (1) is followed at each sweep.

f. Duration: The only extended times occur during the relaxation. The exact time depends on the values of n and f. When n is 068, and f is 030, the time is about 8.5 seconds per sweep.

g. Failure halts: (Occur only with breakpoint switch on.)
Halt in 4z = 079: Machine error in division routine, Read code in again.
Halt in 54 = 084: Overflow in correcting beta. Beta is too large. Read code in again.
Halt in 82 = 130: Overflow in computing residual from (1). The values of u(x,y) are too large. Rescale them downward with Code 00682 (assuming that u ≡ 0 on C).

h. Final program halt: There is none. The operator stops when he is satisfied with the smallness of Σ.

i. Dealing with output: If one wants a decimal listing of the answers, they should be taken out on a drum-punch routine (e.g., 00707) in a form suitable for conversion with code 00219 or 00229, for example.

7) Mathematical method used
The new value of u(x,y) is always computed from the formula

$$\text{new } u(x,y) = u(x,y) + \tfrac{1}{4}\beta\, r(x,y) \quad .$$

The new value is immediately put back in the field, and is used for the next computation of r(x,y). The field is scanned lexicographically, with x as a major, and y as a minor index. The ideas are sketched in an article by G. E. Forsythe entitled, "Difference methods on a digital computer for laplacian boundary value and eigenvalue problems," 1955. There are references to original papers by David Young and by S. P. Frankel.

The least eigenvalue λ_1 of the Laplace difference operator is defined as the least λ for which

3. Partial Differential Equations

$$u(x+1,y) + u(x-1,y) + u(x,y+1) + u(x,y-1) - 4u(x,y) + \lambda u(x,y) = 0 \ .$$

Let μ_1 be the largest μ for which (1) holds; then $\mu_1 = 4 - \lambda_1$. When $u \equiv 0$ on C, λ_1 is positive and near 0, so that μ_1 is slightly less than 4. Thus $\frac{1}{4}\mu_1$ is slightly less than 1, and positive. Hence the "built-in" value of 15/16 for $\frac{1}{4}\mu_1$ is reasonable.

The following remarks are based on the boundary condition $u = 0$ on C. If $\mu = \mu_1$, the u(x,y) will converge to the desired eigenfunction. For any value of $\mu > \mu_1$, the u(x,y) will converge to zero. For any value of $\mu < \mu_1$, the u(x,y) will diverge to ∞ until an overflow error halt occurs. But in going to 0 or ∞, the u(x,y) will come fairly close to the shape of the desired eigenfunction, provided μ is reasonably close to μ_1. At that point, one should use Code 00681 to get a better approximation λ to λ_1. With this better approximation, and $\frac{1}{4}\mu = 1 - \frac{1}{4}\lambda$, a return to Code 00670 will improve the eigenfunction approximation u(x,y). Codes 00680 and 00682 may be used to rescale the values of u(x,y) as needed.

The process described here will work only for the smallest eigenvalue λ_1. To attempt to get higher λ_i would be to try to use successive overrelaxation to solve an indefinite matrix problem with positive diagonal terms--a known impossibility. To get the higher λ_i, one would have to use a modified relaxation process valid for indefinite systems--for example, the Kaczmarz projection method.

8) The drum holds the whole field R$\cup$C. The routine is entirely within the high-speed memory. Cells u0 = 160 to zz = 255 are used to hold data transferred from the drum, while the routine is within cells 000 to 8v = 139.

9) Flow chart

There is no flow chart, as the code differs so little from 00660.

10) Code

The code is identical with 00660, except for the cells on a listing filed in the NAR library. Cards are filed in the SWAC room.

3. Partial Differential Equations

4. Miscellaneous

a. Code no. 00680

1) Written by G. E. Forsythe, February 1955.

2) Description written by G. E. Forsythe.

3) Title

Doubling Values of u(x,y) Over a Square Net.

4) Summary

Consider the set of lattice points P: (x,y), where 016 $\leq$ x $\leq$ 143, 000 $\leq$ y $\leq$ 031, and x and y are integers. Let R be a subset of P at which the values of u are to be doubled. Code 00680 simply doubles them.

There are two parameters in the code: n, the maximum value of x in R (n $\leq$ 142); f, the maximum value of y in R (f $\leq$ 030).

5) Preparation of data

The data are ordinarily on the drum as a result of Code 00670. The set R should be inclosed in some rectangle: 016 $\leq$ x $\leq$ n + 1, 000 $\leq$ y $\leq$ f + 1. The points within this rectangle, but not in R, are identified by a 1 in the most significant binary digit. The points of R have no such marking.

6) Operating procedure

a. Output: There is no output from this code. It is ordinarily followed by routine 00670 or 00681.
b. Switches: Breakpoint switch on.
c. Drum is used continually.
d. Input: Read in the program deck, consisting of cards 000 to 013, numbered in binary in columns 75-78 of the Y row. Either ten or eleven word input will work.
e. Machine operation:

SWAC halts at cells 002 and 003. Proceed as in Code 00660. Next, SWAC halts at cell 009. Press 00PB . SWAC proceeds to double the values and halt with a display breakpoint in cell 77 = 119. Now you are done. To double again, push 00PB.

f. Duration: A few seconds.
g. Failure halts: None.
h. Final program halt: First display breakpoint.
i. Dealing with output: Leave it on the drum.

7) Mathematical method used

None.

8) The drum holds the field. The routine is within the high-speed memory, and is a minor modification of Code 00660.

9) Flow chart

There is none.

3. Partial Differential Equations

10) Code

The code is the same as 00660, except for the following cells:

Cell No.	α	β	γ	δ	F	Remarks
020:	080	080	080	132	07	
132:	000	116	110	065	14	$u_o \rightarrow (110)$
133:	000	000	000	033	08	$033 \rightarrow \varepsilon$
036:	104	116	109	065	14	$u_o \rightarrow (109)$
037:	109	106	106	038	04	ignore overflow $\Sigma + \frac{1}{2} u_o \rightarrow \Sigma$

In fact, commands in 022 to 028 and 121 to 127 are not used, while the division is irrelevant. The display in R register at 119 is now Σu_i, but overflow in Σ is ignored. One can double repeatedly.

11) Cards are filed in the SWAC room.

3. Partial Differential Equations

4. Miscellaneous

b. Code no. 00681

1) Written by G. E. Forsythe, February 1955.

2) Description written by G. E. Forsythe.

3) Title

Computation of Rayleigh Quotient for Function u(x,y) Stored on the Drum in the Form of Code 00670.

4) Summary

Consider the set of lattice points P: (x,y), where 016 $\leq$ x $\leq$ 143, 000 $\leq$ y $\leq$ 031, and x and y are integers. Consider the operator r(x,y) = u(x + 1,y) + u(x - 1,y) + u(x,y + 1) + u(x,y - 1) - 4u(x,y). Let R be a subset of P on which the values of u are unknown for the eigenvalue problem of Code 00670. Code 00681 computes the Rayleigh quotient

$$Q = \frac{-\Sigma u(x,y)\ r(x,y)}{\Sigma u(x,y)^2}$$

where the summations are over the points of R.

There are two parameters in the code: n, the maximum value of x in R(n $\leq$ 142); f, the maximum value of y in R (f $\leq$ 030).

5) Preparation of data

Ordinarily Code 00681 will be used in conjunction with Code 00670, so that the data will already be on the drum in the correct manner. Should it be desired to use Code 00681 independently, consult Code 00670 for the format.

6) Operating procedure

a. Output: The only output is a display in the M register.

b. Switches: Breakpoint switch on.

c. Drum is used continually.

d. Input: Read in the program deck, consisting of cards 000 to 013, numbered in binary in columns 75-78 of the Y row. Either ten or eleven word input will work.

e. Operation: SWAC halts at epsilons of 002 and 003, just like Code 00660. Proceed as in that code. SWAC then halts at an epsilon of 009, calling for input into cell 69 = 105. Press 00PB.

SWAC now computes Q and halts in cell 77 = 119 with a display breakpoint. At this point the routine is complete.

f. Duration: Not over 12 seconds.

g. Failure halts: The halts in 4z = 079, 54 = 084, 82 = 130 are as in Code 00660 and not at all likely to occur. There are two likely halts:

Halt in 89 = 137: An overflow halt in computing the numerator of Q.

3, Partial Differential Equations

Halt in 8u = 138: An overflow halt in computing the denominator of Q.

In case of either of the last two halts, clear the machine, and read in Code 00682. Run it once to halve all values of u(x,y). Then re-run Code 00681. Repeat the cure if necessary.

h, Final program halt: The display breakpoint mentioned above.

i, Dealing with output: Copy the M register; call it λ_1. For dealing with Code 00670, compute $\frac{1}{4}\mu_1 = 1 - \frac{1}{4}\lambda_1$. Actually, this is easily done right from the M register in one's head. [A command or two should be added to compute $\frac{1}{4}\mu_1$ for Code 00670.]

The R register holds N (see below). If $N < 2^{-2r}(r=1,2,--)$ one should run code 00680 r times to multiply the values of u(x,y) by 2^r, and thus increase the significant digits of Q. If one makes u(x,y) too large by mistake, use Code 00682 to decrease them.

7) Mathematical method used

One accumulates the numerator in the form

$$N = -4 \Sigma\, u(x,y)\, r(x,y).$$

One accumulates the denominator in the form

$$D = 4 \Sigma\, u(x,y)^2.$$

One divides N by D,

8) The drum holds the whole field R. The routine is entirely within the high-speed memory. Cells uo = 160 to zz = 255 hold the data transferred from the drum, while the routine is in cells 000 to 8v = 139, Most of the commands of Code 00660 have been left in the memory, but are either not performed or are irrelevant to the answer.

9) Flow chart

There is no flow chart, as the code is such a minor modification of Code 00660.

10) Code

The code is the same as Code 00660 (q. v.), except for the following cells:

Bkpt.	Cell No.	α	β	γ	δ	F	Remarks
	013:	080	080	106	132	07	clear (106) = N
	132:	000	000	113	015	05	clear (113) = D
	036:	109	116	159	133	11	$4u_o r_o \rightarrow$ (159)
	133:	116	116	158	000	10	$4u_o^2 \rightarrow$ (158)
	134:	106	159	106	137	06	$N - 4u_o r_o \rightarrow N$
	135:	158	113	113	138	04	$4u_o^2 + D \rightarrow D$
	136:	000	000	000	038	07	$038 \rightarrow \varepsilon$

3. Partial Differential Equations

Bkpt.	Cell No.	α	β	γ	δ	F	Remarks
x	137:					04	overflow halt } bkpt on
x	138:					05	overflow halt } bkpt on
	040:	000	000	000	[041	07]	don't change drum (filler)
	086:	000	000	000	000	04	dummy for 040 (filler)
	051:	080	086	040	029	05	preset 040
	093:	000	192	192	093	04	dummy for 035
	035:	000	[193	193]	093	04	don't change u_o

037 is unused.

119 displays N and Q.

D is to be found in cell 058 at that time.

11) Cards are filed in the SWAC room.

3. Partial Differential Equations

4. Miscellaneous

c. Code no. 00682

1) Written by G. E. Forsythe, February 1955.

2) Description written by G. E. Forsythe.

3) Title
Halving Values of u(x,y) Over a Square Net.

4) Summary
Consider the set of lattice points P: (x,y), where $016 \leq x \leq 143$, $000 \leq y \leq 031$, and x and y are integers. Let R be a subset of P at which the values of u are to be halved. Code 00682 simply halves them.
There are two parameters in the code: n, the maximum value of x in R ($n \leq 142$); f, the maximum value of y in R ($f \leq 030$).

5) Preparation of data
The data are ordinarily on the drum as a result of Code 00670. The set R should be inclosed in some rectangle: $016 \leq x \leq n + 1$, $000 \leq y \leq f + 1$. The points within this rectangle, but not in R, are identified by a 1 in the most significant binary digit. The points of R have no such marking.

6) Operating procedure
a. Output: There is no output from this code. It is ordinarily followed by another routine.
b. Switches: Breakpoint switch on.
c. Drum is used continually.
d. Input: Read in the program deck, consisting of cards 000 to 013, numbered in binary in columns 75-78 of the Y row. Either ten or eleven word input will work.
e. Machine operation:
SWAC halts at cells 002 and 003. Proceed as in Code 00660. Next, SWAC halts at cell 009. Press 00PB. SWAC proceeds to halve the values and halt with a display breakpoint in cell 77 = 119. Now you are done.
f. Duration: A few seconds.
g. Failure halts: None.
h. Final program halt: First display breakpoint.
i. Dealing with output: Leave it on the drum.

7) Mathematical method used
A right shift, so the halving is rounded.

8) The drum holds the field. The routine is within the high-speed memory, and is a minor modification of Code 00660.

9) Flow chart
There is none.

3. Partial Differential Equations

10) Code

The code is the same as 00680, except for the following changes:

Cell No.	α	β	γ	δ	F	Remarks
033:	134	118	118	072	14	y → Y (118)
035:	110	000	[193]	093	04	$\frac{1}{2}u_o \rightarrow u_o$ (set in 034)
093:	110	000	192	093	04	(dummy for 035)
132:	000	116	110	066	14	$\frac{1}{2}u_o \rightarrow$ (110)
134:	255	000	000	000	00	(extractor for 033)

One can halve repeatedly.

Display in R register at 119 is Σu_i, but overflow in Σ is ignored.

11) Cards are filed in the SWAC room.

3. Partial Differential Equations

4. Miscellaneous

d. Code No. 00683.

1) Written by G. E. Forsythe, February, 1955.

2) Description written by G. E. Forsythe.

3) Title

Drum Punch Routine, 11 Words per Card, for Routines 00660, 1, 2 and 00680, 1, 2.

4) Summary

There is one parameter n, the number of the last channel to be punched. The routine punches all channels from 016 to n, three cards per channel, 11 words per card. The first word gives in δ the channel number only. The identifier punches give cell number and channel number. The output is in a form ready to be read back onto the drum by routine 00684.

5) No data preparation needed.

6) Operating procedure

a. Use 36-digit punch output, with jackplugs arranged to punch rows 9, 8, ... , 1, 0, X.
b. Switch settings immaterial.
c. Drum is used, of course.
d. Standard collator input with either 10 or 11 word input. Start with clear memory. Routine deck has three cards.
e. Machine halts in cell 025 for input into cell 029. Type into δ of R register the number n of last channel to be punched. Press 00PB. The routine then proceeds to punch three cards per channel from channels 016 to n.
f. The routine takes the time to punch three cards per channel, or about 2 seconds per channel.
g. No failure halts have been programmed.
h. After channel n has been punched, the routine halts in cell 015 with an 01 command. One could clear epsilon, press 00PB and start over.
i. The output can be input to the drum with code 00684. The numbering of the channels is in the β identifier punch, while the α identifier punch gives the cell numbers (from 000 to 031) for each channel.

7) No mathematical method used.

8) Cells 000 to 029 are used for the routine, with cells 127 to 159 to hold the channel number and channel. Cells 161, 162, 163 and 255 are temporaries.

9) There is no flow chart.

3. Partial Differential Equations

10) A code listing is in the NAR library.

11) Cards are filed in the SWAC room.

Note: Should one want to punch channels from m ($\neq$016) to n, one must change γ(017) and β(018) to m immediately after the input of n. At present these cells contain 016 in these positions and nothing else.

3. Partial Differential Equations

4. Miscellaneous

e. Code No. 00684

1) Written by G. E. Forsythe, February 1955.

2) Description written by G. E. Forsythe.

3) Title

Drum Load Code for Routines 00660, 1, 2 and 00680, 1, 2.

4) Summary

There is one parameter k ($1 \leq k \leq 32$), the number of words to be loaded into each channel. This must be the same for each channel loaded. The routine first clears the drum. Then it takes in k + 1 words per channel from the collator, of which the first contains the channel number, and loads the next k words into the first k cells of the channel. After all channels are loaded, the routine clears the memory and epsilon, and can be immediately followed by a fresh routine.

5) Preparation of data

Prepare k + 1 words per channel. The first word has the channel number n in γ. The second to the (k + 1)-th words are the k words to be read into channel n. There may be any number of word pulses per card up to eleven. For full channels (k = 32) it is convenient to take three cards with 11 words per card.

All the drum data cards are inserted ahead of the blank card in the routine deck.

6) Operating procedure

a. No output.

b. Switches immaterial.

c. Drum is being loaded, and is also used to clear the memory.

d. Use collator board with either 10 or 11 word input. Input deck has 5 cards: No. 0 is a copy of 00702, the drum-clear routine; numbers 1 - 3 are the routine. No. 4 is a blank card marking the place to insert input data. No. 5 has a $\delta \rightarrow \epsilon$ punch and a word pulse, to initiate the clearing of the memory.

e. Machine halts on input for one to type into 015. If k = 32, one may just press 00PB. Otherwise ($1 \leq k \leq 31$), type k into γ of R register and press 00PB. The machine takes in the channels, performs one output-to-drum command per channel, and finally clears the memory and epsilon in a series of eight fictitious drum input commands.

f. The routine essentially takes only the card-read time and the drum-transfer time. If each channel is on three cards, the time is about 0.77 seconds per channel.

g. There are no programmed failure halts. If trouble develops, start over.

3. Partial Differential Equations

h. Final halt is a clear machine with epsilon = zero.
i. No output.

7) No mathematics used.

8) The routine uses cells 000 to 016, 100, 127 to 127 + k, and 248 to 255.

9) There is no flow chart, as the code is too simple.

10) The code is filed in the NAR library.

11) Cards are filed in the SWAC room.

All date entries (*) refer to special memorandums concerning the operation of and coding for SWAC. These memorandums are filed in the N.A.R. library in a folder entitled "SWAC Announcements, Notes, and Coding Guide."

PLEASE NOTE

Minor corrections are being made on the following pages from Section F:

F 1.4e-1, 2, 3, 4
F 4.1a-1, 2, 3, 4, 5
F 7.4a-1, 2, 3, 4, 5, 6, 7
F 7.4b-1, 2, 3, 4
F 7.4c-1

This material will be distributed in a few weeks.

November 14, 1958

C. B. Tompkins
Director, Numerical Analysis Research
University of California
405 Hilgard Avenue
Los Angeles 24, California

Dear Sir:

This office maintains lists of names of those to whom the "Manual of the SWAC Computing System" has been issued. In this connection your name appears on our list for distribution of new pages and correction to the SWAC manual.

If you wish us to continue this service please return this sheet making any necessary changes in address. Until we receive a reply we shall assume you are no longer interested, and your name will be removed from the list.

Attached to this memorandum is a list of pages and dates of issue by which you may check your copy of the SWAC manual for correct compilation to date. If you lack any of the pages please indicate by number which one you need on the bottom of the sheet to be returned to our office. If you have no plans to use your copy of the MANUAL, we would like to have it back for others to use.

Sincerely yours,

CB Tompkins.

C. B. Tompkins, Director

C. B. Tompkins
Director, Numerical Analysis Research
University of California
405 Hilgard Ave.
Los Angeles 24, California

Copy No. 240

Dear Sir:

Please continue the "Manual of the SWAC Computing System" service I have been receiving on the below address. I have printed in any necessary address changes.

Reserve Book Room

Main Library

Campus

Pages I lack in my copy are:

SEQUENCE OF PAGES IN SWAC MANUAL

(rev. 11-13-58)

PAGE	DATE
Title Page	
ii	3-15-54
iii	11-2-55
iv	3-15-54
v	3-8-56
vi	3-15-54
vii	2-21-56
viii	"
ix	3-8-56
A 1.2-1	3-15-54
2	"
3	"
A 1.3-1	"
2	"
A 1.4-1	"
A 1.5-1	"
2	"
3	"
4	"
A 1.6-1	"
2	"
3	"
4	"
A 1.7-1	"
2	"
3	"
A 2.1-0	10-4-55
1	3-15-54
2	"
A 2.2-1	10-4-55
2	"
3	"
A 2.3a-1	"
2	"
3	"
4	"
5	"
6	"
A 2.3b-1	"
2	"
3	"
4	"
5	"
6	"
7	"
8	"
9	"
10	"
A 2.3b-11	10-4-55
12	"
13	"
14	"
15	"
16	"
A 2.3c- 1	"
2	"
3	"
4	11-2-55
5	"
6	"
7	"
A 2.3d- 1	10-4-55
2	"
3	"
A 2.3e- 1	11-2-55
2	"
3	"
4	"
5	"
6	"
7	"
8	"
A 2.3f- 1	10-4-55
A 2.3g- 1	"
B 1.1 - 1	3-15-54
B 1.2 - 1	"
2	"
B 1.3 - 1	"
B 1.4 - 1	"
B 2.1 - 1	"
2	"
3	"
4	"
5	"
6	"
7	"
B 2.2a- 1	3-15-54
2	"
3	"
B 2.2b- 1	"
2	"
3	"
4	"
5	"
B 2.2c- 1	"
2	"
B 2.2d- 1	"
B 3.1 - 1	3-15-54
2	"
3	2-4-54
B 3.2 - 1	3-15-54
2	"
3	"
B 4.1 - 1	"
2	"
3	"
4	"
B 4.2 - 1	"
2	"
3	"
4	"
B 6.1 - 1	"
2	"
3	"
4	"
6	9-15-56
7	"
C 1 - 1	3-15-54
2	"
C 1.1 - 1	
2	"
3	"
4	"
5	"
6	"
7	"
8	"
C 1.2 - 1	9-27-57
C 1.3 - 1	7-18-58
C 1.4 - 1	7-17-58
C 2.1a- 1	3-15-54
2	"
3	"
4	"
5	"
6	"
7	"
C 2.1b- 1	"
2	"
3	"
C 2.1c- 1	"
2	"
3	"
4	"
C 2.1d- 1	"
2	"

PAGE	DATE
C 2.1d- 3	3-15-54
C 2.2 - 1	"
2	"
3	"
4	"
5	"
6	"
C 2.3a- 1	"
2	"
3	"
4	"
5	"
C 2.3b- 1	3-8-56
2	"
3	"
C 2.5a- 1	3-15-54
2	"
3	"
4	"
C 2.5b- 1	"
2	"
3	"
4	"
5	"
6	3-11-58
C 2.5c- 1	3-15-54
2	"
C 2.5d- 1	"
C 2.5e- 1	10-22-58
2	"
C 2.9 - 1	7-19-54
2	"
C 2.10- 1	3-8-56
2	"
C 3.1 - 1	3-15-54
D 1.1a- 1	3-29-56
2	"
3	"
4	"
D 1.1b- 1	"
2	"
3	"
4	"
D 1.2a- 1	3-15-54
2	"
D 1.2b- 1	"
2	"

PAGE	DATE
D 1.2c- 1	3-15-54
2	"
D 1.2d- 1	"
2	"
D 1.2e- 1	"
2	"
D 1.2f- 1	"
2	"
D 1.2g- 1	"
D 1.2h- 1	"
D 1.3a- 1	"
2	"
3	"
D 1.3b- 1	"
D 1.4 - 1	"
2	"
D 1.5 - 1	"
D 1.6 - 1	"
2	"
D 1.7 - 1	"
2	"
3	"
D 1.8 - 1	"
2	"
D 2.1a- 1	"
2	"
D 2.1b- 1	"
2	"
D 2.1c- 1	"
D 2.1d- 1	"
2	"
D 2.1e- 1	"
D 2.2a- 1	"
2	"
3	"
D 2.2b- 1	"
2	"
3	"
D 2.3 - 1	"
2	"
3	"
4	"
5	"
6	"
D 2.4 - 1	"
2	"
3	"

PAGE	DATE
D 2.5 - 1	3-15-54
2	"
3	"
4	"
5	"
6	"
7	"
8	"
D 2.6 - 1	"
2	"
D 3.1 - 1	"
D 3.2 - 1	"
2	"
3	"
D 3.4 - 1	"
E 0 - 1	4-20-56
E 1.1a- 1	2-21-56
E 1.1b- 1	"
E 1.1c- 1	"
E 1.2 - 1	"
2	"
3	"
4	"
5	"
6	"
7	"
8	"
9	"
10	6-17-58
11	"
E 1.3 - 1	2-21-56
E 1.4a- 1	"
E 1.4b- 1	"
E 1.4c- 1	"
E 1.4d- 1	"
E 1.4e- 1	"
E 1.4f- 1	"
E 1.4g- 1	"
E 1.4h- 1	"
2	"
E 1.4i- 1	"
E 1.4j- 1	"
E 1.4k- 1	"
E 1.4l- 1	"
E 1.4m- 1	"
2	"

PAGE	DATE
E 1.4n- 1	2-21-56
E 1.4o- 1	"
2	"
E 1.4p- 1	"
2	"
E 1.4q- 1	"
2	"
E 1.4r- 1,2	"
3,4	11-17-58
E 1.4s- 1	2-21-56
2	"
E 1.5a- 1	2-1-56
E 1.5b- 1	"
E 1.5c- 1	"
E 1.5d- 1	"
E 1.5e- 1	"
E 1.5f- 1	"
E 1.5g- 1	"
E 1.5h- 1	"
2	"
E 1.5i- 1	"
E 1.5j- 1	"
E 1.5k- 1	"
2	"
E 1.5l- 1	"
E 1.5m- 1	"
E 1.5n- 1	"
2	"
E 1.5o- 1	"
2	"
E 1.5p- 1	"
2	"
E 1.5q- 1	"
2	"
E 1.5r- 1	"
2	"
3	"
E 1.5s- 1	"
2	"
3	"
E 1.5t- 1	"
2	"
3	"
E 1.5u- 1	6-17-58
2	"
3	"
E 1.5v- 1	"
2	"
E 1.5w- 1	"
2	"

PAGE	DATE
E 1.6a- 1	2-1-56
E 1.6b- 1	"
2	"
E 1.6c- 1	"
E 1.6d- 1	"
E 1.6e- 1	"
E 1.6f- 1	"
E 2.1 - 1	3-15-54
2	"
3	"
4	"
5	"
E 2.3 - 1	"
2	"
3	"
4	"
5	"
6	"
7	"
8	"
9	"
10	"
11	"
12	"
13	"
14	"
E 3.1 - 1	9-3-54
2	"
E 3.2 - 1	4-16-56
E 4.1 - 1	4-19-56
2	"
E 6.1 - 1	8-2-54
2	"
3	"
E 6.2 - 1	4-16-56
2	"
F 0 - 1	4-20-56
F 1.0 - 1	3-8-56
F 1.1a- 1	"
2	"
3	"
F 1.1b- 1	"
2	"
3	"
F 1.1c- 1	"
2	"
F 1.1d- 1	"
2	"
3	"

PAGE	DATE
F 1.1e- 1	3-8-56
2	"
3	"
F 1.1f- 1	"
2	"
F 1.2a- 1	"
2	"
3	"
4	"
F 1.2b- 1	"
2	"
3	"
4	"
5	"
F 1.2c- 1	"
2	"
3	"
4	"
F 1.2d- 1	"
2	"
3	"
F 1.2e- 1	"
2	"
F 1.3a- 1	"
2	"
3	"
4	"
5	"
F 1.4a- 1	4-16-56
2	"
3	"
4	"
F 1.4b- 1	3-8-56
2	"
3	"
F 1.4c- 1	"
2	"
3	"
4	"
F 1.4d- 1	7-4-56
2	"
3	"
4	"
5	"
6	"
7	"
8	"
9	"

PAGE	DATE	PAGE	DATE	PAGE	DATE
F 1.4d- 10	7-4-56	H 1	6-7-56		
11	"	2	"		
F 1.4e- 1	5-29-57	3	"		
2	"	4	"		
3	"	5	"		
4	"	6	"		
F 3.2a- 1	4-20-56	7	"		
2	"	8	"		
3	"	9	"		
F 3.2b- 1	"				
2	"				
3	"				
4	"				
F 3.3a- 1	"				
2	"				
3	"				
4	"				
F 3.4a- 1	"				
2	"				
F 3.4b- 1	"				
2	"				
3	"				
F 3.4c- 1	"				
2	"				
F 3.4d- 1	"				
2	"				
F 3.4e- 1	"				
2	"				
F 4.1a- 1	3-18-57				
2	"				
3	"				
4	"				
5	"				
F 7.4a- 1	10-3-56				
2	"				
3	"				
4	"				
5	"				
6	"				
7	"				
F 7.4b- 1	6-12-57				
2	"				
3	"				
4	"				
F 7.4c- 1	7-5-57				

SEQUENCE OF PAGES IN SWAC MANUAL

(rev. 11-13-58)

PAGE	DATE
Title Page	
ii	3-15-54
iii	11-2-55
iv	3-15-54
v	3-8-56
vi	3-15-54
vii	2-21-56
viii	"
ix	3-8-56
A 1.2-1	3-15-54
2	"
3	"
A 1.3-1	"
2	"
A 1.4-1	"
A 1.5-1	"
2	"
3	"
4	"
A 1.6-1	"
2	"
3	"
4	"
A 1.7-1	"
2	"
3	"
A 2.1-0	10-4-55
1	3-15-54
2	"
A 2.2-1	10-4-55
2	"
3	"
A 2.3a-1	"
2	"
3	"
4	"
5	"
6	"
A 2.3b-1	"
2	"
3	"
4	"
5	"
6	"
7	"
8	"
9	"
10	"
A 2.3b-11	10-4-55
12	"
13	"
14	"
15	"
16	"
A 2.3c- 1	"
2	"
3	"
4	11-2-55
5	"
6	"
7	"
A 2.3d- 1	10-4-55
2	"
3	"
A 2.3e- 1	11-2-55
2	"
3	"
4	"
5	"
6	"
7	"
8	"
A 2.3f- 1	10-4-55
A 2.3g- 1	"
B 1.1 - 1	3-15-54
B 1.2 - 1	"
2	"
B 1.3 - 1	"
B 1.4 - 1	"
B 2.1 - 1	"
2	"
3	"
4	"
5	"
6	"
7	"
B 2.2a- 1	3-15-54
2	"
3	"
B 2.2b- 1	"
2	"
3	"
4	"
5	"
B 2.2c- 1	"
2	"
B 2.2d- 1	"
B 3.1 - 1	3-15-54
2	"
3	2-4-54
B 3.2 - 1	3-15-54
2	"
3	"
B 4.1 - 1	"
2	"
3	"
4	"
B 4.2 - 1	"
2	"
3	"
4	"
B 6.1 - 1	"
2	"
3	"
4	"
6	9-15-56
7	"
C 1 - 1	3-15-54
2	"
C 1.1 - 1	
2	"
3	"
4	"
5	"
6	"
7	"
8	"
C 1.2 - 1	9-27-57
C 1.3 - 1	7-18-58
C 1.4 - 1	7-17-58
C 2.1a- 1	3-15-54
2	"
3	"
4	"
5	"
6	"
7	"
C 2.1b- 1	"
2	"
3	"
C 2.1c- 1	"
2	"
3	"
4	"
C 2.1d- 1	"
2	"

PAGE	DATE
C 2.1d- 3	3-15-54
C 2.2 - 1	"
2	"
3	"
4	"
5	"
6	"
C 2.3a- 1	"
2	"
3	"
4	"
5	"
C 2.3b- 1	3-8-56
2	"
3	"
C 2.5a- 1	3-15-54
2	"
3	"
4	"
C 2.5b- 1	"
2	"
3	"
4	"
5	"
6	3-11-58
C 2.5c- 1	3-15-54
2	"
C 2.5d- 1	"
C 2.5e- 1	10-22-58
2	"
C 2.9 - 1	7-19-54
2	"
C 2.10- 1	3-8-56
2	"
C 3.1 - 1	3-15-54
D 1.1a- 1	3-29-56
2	"
3	"
4	"
D 1.1b- 1	"
2	"
3	"
4	"
D 1.2a- 1	3-15-54
2	"
D 1.2b- 1	"
2	"

PAGE	DATE
D 1.2c- 1	3-15-54
2	"
D 1.2d- 1	"
2	"
D 1.2e- 1	"
2	"
D 1.2f- 1	"
2	"
D 1.2g- 1	"
D 1.2h- 1	"
D 1.3a- 1	"
2	"
3	"
D 1.3b- 1	"
D 1.4 - 1	"
2	"
D 1.5 - 1	"
D 1.6 - 1	"
2	"
D 1.7 - 1	"
2	"
3	"
D 1.8 - 1	"
2	"
D 2.1a- 1	"
2	"
D 2.1b- 1	"
2	"
D 2.1c- 1	"
D 2.1d- 1	"
2	"
D 2.1e- 1	"
D 2.2a- 1	"
2	"
3	"
D 2.2b- 1	"
2	"
3	"
D 2.3 - 1	"
2	"
3	"
4	"
5	"
6	"
D 2.4 - 1	"
2	"
3	"

PAGE	DATE
D 2.5 - 1	3-15-54
2	"
3	"
4	"
5	"
6	"
7	"
8	"
D 2.6 - 1	"
2	"
D 3.1 - 1	"
D 3.2 - 1	"
2	"
3	"
D 3.4 - 1	"
E 0 - 1	4-20-56
E 1.1a- 1	2-21-56
E 1.1b- 1	"
E 1.1c- 1	"
E 1.2 - 1	"
2	"
3	"
4	"
5	"
6	"
7	"
8	"
9	"
10	6-17-58
11	"
E 1.3 - 1	2-21-56
E 1.4a- 1	"
E 1.4b- 1	"
E 1.4c- 1	"
E 1.4d- 1	"
E 1.4e- 1	"
E 1.4f- 1	"
E 1.4g- 1	"
E 1.4h- 1	"
2	"
E 1.4i- 1	"
E 1.4j- 1	"
E 1.4k- 1	"
E 1.4l- 1	"
E 1.4m- 1	"
2	"

OK OK OK

PAGE	DATE
E 1.4n- 1	2-21-56
E 1.4o- 1	"
2	"
E 1.4p- 1	"
2	"
E 1.4q- 1	"
2	"
E 1.4r- 1,2	"
3,4	11-17-58
E 1.4s- 1	2-21-56
2	"
E 1.5a- 1	2-1-56
E 1.5b- 1	"
E 1.5c- 1	"
E 1.5d- 1	"
E 1.5e- 1	"
E 1.5f- 1	"
E 1.5g- 1	"
E 1.5h- 1	"
2	"
E 1.5i- 1	"
E 1.5j- 1	"
E 1.5k- 1	"
2	"
E 1.5l- 1	"
E 1.5m- 1	"
E 1.5n- 1	"
2	"
E 1.5o- 1	"
2	"
E 1.5p- 1	"
2	"
E 1.5q- 1	"
2	"
E 1.5r- 1	"
2	"
3	"
E 1.5s- 1	"
2	"
3	"
E 1.5t- 1	"
2	"
3	"
E 1.5u- 1	6-17-58
2	"
3	"
E 1.5v- 1	"
2	"
E 1.5w- 1	"
2	"

PAGE	DATE
E 1.6a- 1	2-1-56
E 1.6b- 1	"
2	"
E 1.6c- 1	"
E 1.6d- 1	"
E 1.6e- 1	"
E 1.6f- 1	"
E 2.1 - 1	3-15-54
2	"
3	"
4	"
5	"
E 2.3 - 1	"
2	"
3	"
4	"
5	"
6	"
7	"
8	"
9	"
10	"
11	"
12	"
13	"
14	"
E 3.1 - 1	9-3-54
2	"
E 3.2 - 1	4-16-56
E 4.1 - 1	4-19-56
2	"
E 6.1 - 1	8-2-54
2	"
3	"
E 6.2 - 1	4-16-56
2	"
F 0 - 1	4-20-56
F 1.0 - 1	3-8-56
F 1.1a- 1	"
2	"
3	"
F 1.1b- 1	"
2	"
3	"
F 1.1c- 1	"
2	"
F 1.1d- 1	"
2	"
3	"

PAGE	DATE
F 1.1e- 1	3-8-56
2	"
3	"
F 1.1f- 1	"
2	"
F 1.2a- 1	"
2	"
3	"
4	"
F 1.2b- 1	"
2	"
3	"
4	"
5	"
F 1.2c- 1	"
2	"
3	"
4	"
F 1.2d- 1	"
2	"
3	"
F 1.2e- 1	"
2	"
F 1.3a- 1	"
2	"
3	"
4	"
5	"
F 1.4a- 1	4-16-56
2	"
3	"
4	"
F 1.4b- 1	3-8-56
2	"
3	"
F 1.4c- 1	"
2	"
3	"
4	"
F 1.4d- 1	7-4-56
2	"
3	"
4	"
5	"
6	"
7	"
8	"
9	"

OK

OK

PAGE	DATE	PAGE	DATE	PAGE	DATE
F 1.4d- 10	7-4-56	H 1	6-7-56		
11	"	2	"		
F 1.4e- 1	5-29-57	3	"		
2	"	4	"		
3	"	5	"		
4	"	6	"		
F 3.2a- 1	4-20-56	7	"		
2	"	8	"		
3	"	9	"		
F 3.2b- 1	"				
2	"				
3	"				
4	"				
F 3.3a- 1	"				
2	"				
3	"				
4	"				
F 3.4a- 1	"				
2	"				
F 3.4b- 1	"				
2	"				
3	"				
F 3.4c- 1	"				
2	"				
F 3.4d- 1	"				
2	"				
F 3.4e- 1	"				
2	"				
F 4.1a- 1	3-18-57				
2	"				
3	"				
4	"				
5	"				
F 7.4a- 1	10-3-56				
2	"				
3	"				
4	"				
5	"				
6	"				
7	"				
F 7.4b- 1	6-12-57				
2	"				
3	"				
4	"				
F 7.4c- 1	7-5-57				