IBM

speedcoding system

for the

type 701

ELECTRONIC

DATA PROCESSING

MACHINES

INTERNATIONAL BUSINESS MACHINES CORPORATION

This edition replaces a restricted, tentative version issued early in 1953. Information in this edition is as of September 10, 1953.

Printed in U.S.A.

Form 24-6059-0

foreword

THE IBM Type 701 Speedcoding System was designed to minimize the amount of time spent in problem preparation. It is applicable to small computing problems and to many large computing problems. Because programming for the IBM Electronic Data Processing Machines, Type 701 and Associated Equipment, is developing rapidly and because it is advantageous to make changes in the system easily and informally, this manual will be obsoleted fairly often, and new revisions published.

It is hardly possible to assign credit for the IBM Speedcoding System because the group in IBM which developed the system has benefited from the suggestions of so many. Historically, the ingenious development of general purpose systems for the IBM Card Programmed Calculator certainly influenced the basic thinking. Once the IBM 701 was announced, scientists concerned with preparing for these machines actively considered the problem of reducing problem preparation. Many useful and provocative ideas in this area were discussed in Poughkeepsie, New York, during the week of August 25-28, 1952, when representatives from the following organizations met to consider programming ideas: Boeing Airplane Company, Douglas Aircraft Company (El Segundo and Santa Monica), General Electric Company, International Business Machines Corporation, Lockheed Aircraft Corporation, Los Alamos Scientific Laboratory, National Bureau of Standards, U. S. Naval Ordnance Laboratory, U. S. Naval Ordnance Test Station (Inyokern), North American Aviation, Inc., United Aircraft Corporation, Bell Telephone Laboratories, RAND Corporation, U. S. Atomic Energy Commission. At that meeting there were specific discussions of systems analogous to Speedcoding.

The group at IBM which developed the Speedcoding System undertook to do so after discussing such systems with Dr. Willard Bouricius, head of the Type 701 Planning Group at Los Alamos Scientific Laboratory. This group, under the direction of Mr. Bengt Carlson, had completed a program with objectives similar to those of Speedcoding. Many discussions were held with Mr. Stuart Crossman's group at United Aircraft, particularly with Mr. Walter Ramshaw, whose assistance was extremely helpful in organizing and collating information and text.

Finally, appreciation is expressed to those at IBM who have been most closely associated with the Speedcoding project since its beginning in January, 1953. These are: Mr. John Backus, who supervised the project, Mr. Harlan L. Herrick, Mr. Donald A. Quarles, Jr., Mr. Sherwood Skillman, Mr. John Pulos, and Miss Lucy A. Siegel. The project was carried out under the general direction of Mr. John Sheldon.

contents

TYPE 701 SPEEDCODING SYSTEM

GENERAL INFORMATION

SPEEDCO I is the name of a system—an integrated combination of a large-scale digital computer and a method by which that computer may readily be programmed to solve scientific and engineering problems.

The computer used is the IBM Type 701. Its internal high-speed memory on cathode-ray tubes will be referred to as the *electrostatic memory*. When the amount of storage available in the electrostatic memory is not large enough, magnetic drums are used to store and supply large blocks of information for ready access at frequent intervals. The *drum memory* is also capable of retaining its contents while the power is turned off, so that intermediate results remain available overnight when the machine is shut down. Any part of the information on the drums may be selectively altered by SpeedCo I at any time.

If a larger secondary memory is needed, or if information is to be filed away for future reference, magnetic tapes may be used instead of magnetic drums. Magnetic tape is a storage and input-output medium that provides compactness, allows rapid reading and writing, and can be re-used many times.

To achieve greater computing efficiency, the machine works internally in the binary number system. The user of SpeedCo I will find, however, that this fact does not in any way affect the programming. During SpeedCo I operation all numbers introduced into the machine and all results printed by the machine are expressed in the decimal number system. SpeedCo I automatically performs all necessary conversions between the decimal and binary number systems.

The programs may be written and introduced into the computer in various ways. Usually the instructions are key-punched on cards and read into the machine. If the program is to be preserved for future use, it can then be recorded on tape and filed away in compact form. To prepare the machine for calculation, the appropriate magnetic tapes are inserted in the tape units, and the cards containing the instructions and data of the problem are placed in the hopper of the card reader. Pushing a button then causes the machine to store the program and data of the problem and start calculating. From then on, operation of the computer is fully automatic, with all its components under complete control of the program, although it is possible for the operator to interrupt the calculation manually at any time.

The primary unit of information handled by SpeedCo I consists of 72 binary digits and is called a *word*.

STORAGE

INFORMATION may be stored in electrostatic storage, on magnetic drums, and on magnetic tape. The following description covers, in general terms, the nature and extent of each of these storage media.

Electrostatic Storage

The heart of the machine is the electrostatic storage unit, through which all information to and from all other components of the machine must pass. Electrostatic storage consists of a bank of cathode-ray tubes. Information is stored on the screens of the tubes through the presence or absence of charged spots at certain locations on the screens. In this way a certain number of binary digits (or *bits*) may be stored on each tube. SpeedCo I provides for 714 words of this sort of storage.

The principal advantages of electrostatic storage over other types are the very small time necessary to extract information from any given location and send it to the computing unit, and the fact that the programmer has random access to any electrostatic storage location. Information is lost when the power is turned off.

Magnetic Drums

Additional storage capacity is provided by two magnetic drums. These drums are rotating cylinders surfaced with a material that can be magnetized locally. Binary digits are stored on a drum through the presence or absence of small magnetized areas at certain locations on the surface of the drum. Each drum has a storage capacity of 1024 words. Information is

transmitted to and from drum storage only through electrostatic storage. When such a transfer of information occurs, the machine is said to write on or read from the drum.

Any part of the information on a drum can be selectively altered by SpeedCo I at any time. Because access to individual words on a drum is slow compared to electrostatic storage access, it is more efficient to use the drums for storing large blocks of information. After the first word of such a block has been located, the remaining words are read at the rate of 400 per second. Magnetic drums will retain stored information even when the power is turned off.

Magnetic Tapes

There is also a tape-storage section which includes four magnetic tape units. Each tape, which may be up to 1400 feet long, is wound on a reel. The tape itself is a non-metallic, oxide-coated band one-half inch wide. Binary information is recorded on tape by means of magnetized spots. A block of words recorded consecutively on a tape is called a *record* or a *unit record*. The amount of information contained on each tape depends on the lengths of the individual records, because there is a certain amount of space between successive records to allow for starting and stopping the tape. It is possible to store approximately 140,000 words on each tape. The machine can read or write on a tape only through the medium of electrostatic storage. On the average, about 10 milliseconds are needed for the tape to accelerate to its reading or writing speed, after which the reading or writing of a unit record takes place at the rate of 625 words per second. Because the tapes are removable, a library of standard programming and mathematical tables may be kept on tapes.

COMPUTING

THE NUMBERS handled by SpeedCo I are expressed in scientific, or floating-point, notation. Every number read, stored, computed or printed is of the form

$$F \times R^E$$

where F is the fractional part, R is the radix of the number system used, and E is the exponent.

All numbers that are of direct concern to the programmer (that is, all numbers read into or printed by the machine) are expressed in the decimal system. For these numbers the following relationships apply:

1. $R = 10$
2. $0 \leqq |E| \leqq 236$
3. $\begin{cases} \frac{1}{10} \leqq |F| < 1 \text{ or } F = 0 \\ \text{For less restricted input, see Appendix B} \end{cases}$

Note particularly that the absolute value of the decimal exponent of any quantity read into or printed by the machine may not be greater than 236. Any attempt to read or print an exponent that does not fall in this range will result in an automatic machine stop.

The fractional part F is a ten-decimal digit fraction. Hence the maximum precision attainable by SpeedCo I is one part in 10^{10}.

Numbers that do not come to the direct attention of the programmer (that is, numbers handled inside the machine) are expressed in the binary system. For these numbers the corresponding relationships are:

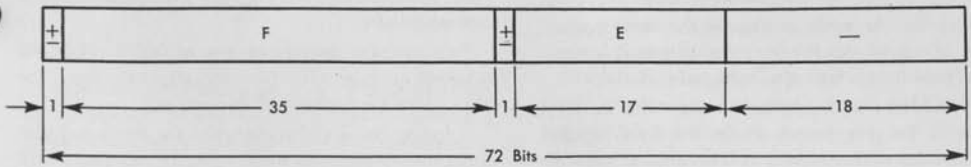
1. $R = 2$
2. $0 \leqq |E| \leqq 131,071$
3. $\frac{1}{2} \leqq |F| < 1 \text{ or } F = 0$

As has been pointed out above, the necessary conversions between the decimal information read into and printed by the machine and the binary information used inside the machine are performed automatically by SpeedCo I. Note, however, that SpeedCo I permits the binary quantities inside the machine a much greater magnitude range than is permitted those quantities to be converted to decimal and printed. In this connection the programmer should not attempt to print any quantity whose decimal exponent is larger, in absolute value, than 236.

It is important to note that because of the intrinsic error involved in conversion from the decimal floating-point form to the binary floating-point form, the programmer should not expect an *exact* correspondence between his decimal input and the resulting binary numbers stored in the machine. For example, the floating-decimal number with fractional part +.1 and exponent part +1 (which is a floating-decimal representation of the integer 1) happens to result in the stored binary number $1 + 2^{-34}$.

Numbers are stored in the various memory organs of the machine as shown in Figure 1.

Because $3\frac{1}{3}$ bits are about equivalent in information content to one decimal digit, the 35 bits allotted to the fractional part are more than adequate to preserve the precision of the ten decimal-digit-input quantities.



Bit Positions	Item
1	Sign of the quantity
2 - 36	Fractional part
37	Sign of the exponent
38 - 54	Exponent
55 - 72	These bit positions are not used

FIGURE 1

Each portion of the electrostatic memory capable of storing a word is called a *storage location* or a *memory cell*. These locations are numbered. The number associated with any particular location is called its *address*. In SpeedCo I the 714 locations available to the programmer are numbered from 300 to 1013, inclusive.

In describing a program, it is frequently necessary to refer to a particular location in memory. These references are often conveniently abbreviated by referring merely to the address itself. For instance, the phrase "x is stored at a" is customarily used in place of the more precise phrase "the quantity x is located in that memory cell whose address is the integer a." The same phrase is often further abbreviated to:

$$L\ (x) = a$$

Similarly, it is often necessary to refer to the word stored in a particular memory cell. This word is usually referred to as "the contents of a" or "the quantity at a" where in both cases the precise phrase abbreviated is "the quantity located in that memory cell whose address is the integer a." The symbolic abbreviation used here is:

$$Q\ (a) = x$$

To cause SpeedCo I to carry out a numerical calculation, the programmer must supply four pieces of information. The first of these, referred to as OP_1, tells the machine which numerical operation it is to perform. The second and third, designated A and B, are the addresses in electrostatic storage, of the two operands. The fourth, C, is the address of the electrostatic memory cell in which the result is to be stored.

This information having been supplied, SpeedCo I then selects the two numbers located at memory cells A and B, performs upon them the operation OP_1, and sends the result to electrostatic memory cell C where it is stored in place of the previous contents of C.

The following paragraphs describe the elementary arithmetic operations in general terms. The actual methods and instructions necessary for performing these operations are explained later.

Addition and Subtraction

The addition operation is performed as follows.

The exponent at A is compared with the exponent at B. If these exponents are not equal, the fractional part associated with the smaller exponent is shifted to the right as many binary places as are necessary to make the two exponents equal.

The fractional parts are then added algebraically. Next the machine checks the fractional part of the result. If the absolute value of this fractional part is in the range equal to or greater than one-half and less than one, the fractional part and the exponent are stored at C. If this absolute value falls outside this range, however, the fractional part itself is shifted to the left or right as necessary to restore it to this range, and the exponent is modified accordingly. The new fractional part and exponent are then stored at C. In the case where the two operands are exactly equal and of opposite sign, plus zero is stored as the fractional

part of the result, and the decimal equivalent of the stored exponent of the result becomes minus 235.

Subtraction proceeds in exactly the same way—except, of course, that the fractional part at B is algebraically subtracted from the fractional part at A.

Both addition and subtraction may also be performed, if the programmer so desires, by using the absolute value of the quantity at B in conjunction with either the algebraic or the absolute value of the quantity at A.

Multiplication

Two computations are performed. In one, the fractional part at A is multiplied by the fractional part at B. In the other, the exponent at A is added to the exponent at B. The absolute value of the product of the fractional parts is then checked.

If this absolute value is equal to or greater than one-half, the product of the fractional parts is then the fractional part of the product, and the sum of the exponents is the exponent of the product. These numbers are therefore stored at C. If, on the other hand, the absolute value of the product of the fractional parts is less than one-half, the fractional part of the product is obtained by doubling the product of the fractional parts, and the exponent of the product is formed by reducing the sum of the exponents by one. These modifications having been made, the result is then stored at C.

Two multiplication operations are available. The one forms the product with its correct algebraic sign. The other forms the negative of the product.

Division

Here also two computations are performed. In one, one-half of the fractional part at A is divided by the fractional part at B to obtain a 35-bit rounded quotient. In the other, the exponent at B is subtracted from the exponent at A. The absolute value of the quotient of the fractional parts is then checked. If this absolute value is less than one-half, the fractional part of the quotient is taken as twice the quotient of the fractional parts, and the difference of the exponents is the exponent of the quotient. If, on the other hand, the absolute value of the quotient of the fractional parts is greater than or equal to one-half, the fractional part of the quotient is taken as the quotient of the fractional parts, and the exponent of the quotient is formed by increasing the difference of the exponents by one. These modifications having been made, the result is then stored at C.

Two division operations are available. The one forms the quotient with its correct algebraic sign. The other forms the negative of the quotient.

If during any division operation the fractional part of the divisor happens to be zero, the calculator will stop and a signal light called the Divide-Check light will light up on the operator's panel.

Elementary Arithmetic Operations: General

Note that each of the above ten operations takes two numbers already expressed in the standard floating-binary form previously specified, performs upon them the desired elementary arithmetic operations, and then adjusts the result so that it, too, is in the standard form.

Note also that the result returned to memory by any one of these operations is not rounded—except in the case of division, where either a 35-bit or a 34-bit rounded quotient is obtained.

CONTROL

SpeedCo I is a stored program system in which the programmer's instructions to the machine are all stored in the machine's memory before the calculation begins. The procedure leading to this result is:

1. The programmer analyzes his problem and breaks its solution down into the basic steps that SpeedCo I can perform.
2. By means of an alphamerical code, determined by the design of SpeedCo I, he translates these steps into a form that can be interpreted by the machine. Each of these steps, which will hereafter be referred to as an *instruction*, is then stored in the machine's memory.
3. Data necessary for the solution of the problem are also stored in the memory of the machine.
4. Upon completion of this storing process, calculation automatically begins with the execution of the instruction which is at that time in electrostatic memory cell 300. From this point on the machine operates without any further intervention on the part of the programmer, automatically locating and executing all succeeding instructions of the program.

A complete analysis of the SpeedCo I instruction system follows.

Program Counter

The numerical representation of an instruction occupies the space of one word in memory. SpeedCo I instructions may temporarily be stored on drums or tape, but at the time they are to be used they must be in electrostatic storage.

A program contains a set of instructions, usually to be executed in sequence, which will cause the machine to compute a desired result. These instructions are ordinarily introduced into consecutively-numbered storage locations in the order in which they are to be executed. The reasons for this follow.

Each time an operation is to be performed, the machine looks up the instruction in electrostatic memory, executes it, and then goes back to the memory for the next instruction. The order in which instructions are executed is controlled by the *program counter*. This counter contains the address of the instruction currently being executed. After each execution, the number in this counter is automatically increased by one. Consequently, the machine automatically takes its next instruction from that electrostatic storage location whose address is one higher than the address of the cell from which the current instruction was obtained. In this way the machine continues to execute instructions in the sequence in which they were stored in memory.

This normal sequence of instructions can be altered by means of certain transfer operations explained below. By means of these operations, any electrostatic storage location from 300 to 1013 inclusive can be designated as the source of the next instruction. The transfer operation used accomplishes this result by placing into the program counter the address of the designated storage location. Following this, the machine looks up and executes the instruction in the designated memory cell; thereafter, execution of the program proceeds sequentially from this new electrostatic storage location. Under these circumstances, a *transfer of control* is said to have occurred.

An important observation about stored program technique should be noted. Instructions are stored in the machine just like numerical data; the only distinction between the two is the way they are interpreted by the machine. Hence the addresses that are part of SpeedCo I instructions may be modified while they are in the machine through the use of special operations. Thus, one part of a program may modify another instruction of the same program by directing the machine to compute a new address, or new addresses, for the instruction to be modified.

A further consequence of the fact that both data and instructions are stored in the machine's memory in the form of binary words is that if, for any reason, a transfer operation enters the address of a piece of data into the program counter, the data at that location will be interpreted as if it were an instruction. Exactly what will happen when SpeedCo I attempts to execute this instruction is in general quite unpredictable, but certainly such misuse of a transfer operation is bound to result in erroneous calculations.

A further similar difficulty will arise if a transfer operation enters any address from 0 to 299 or 1014 to 1023 into the program counter. During SpeedCo I operation these electrostatic-storage regions are always occupied by a block of control information. If it were not for this control information, the 701 would operate under the guidance of its own built-in control circuits only, and would therefore behave as a fixed-point binary calculator. The combination of the built-in 701 circuitry and the control information (stored in electrostatic memory cells 0 to 299 and 1014 to 1023) enables SpeedCo I to read and print in the decimal system, to calculate on a floating-point basis and to perform those other SpeedCo I operations that are not part of the machine's built-in order code.

Because this control information has been stored in these electrostatic locations, it is necessary that SpeedCo I programs be so written as to use only those memory cells with addresses from 300 to 1013 inclusive. In particular, a transfer operation that enters any number not in this interval (300 to 1013) into the program counter, is always in error, since it will cause the control information at that location to be interpreted as if it were an instruction. Here too it is impossible to predict exactly what will happen when SpeedCo I attempts to execute this instruction, but it is again true that the results will be erroneous.

Instruction Layout

Each operation that SpeedCo I can execute is assigned an alphabetical-code designation and a numerical-code designation. The alphabetical designations have been chosen on a mnemonic basis, and consist of combinations of from two to five letters each. These alphabetical designations are solely a programming convenience, however, because the calculator is not

able to recognize them. If the programmer has prepared his program using these alphabetical designations, his instruction cards must first be processed on standard IBM sorting and gang punching equipment. The effect of this preliminary processing will be to punch in the cards of his instruction deck the numerical code designations that correspond to the various alphabetical code designations used in his program. This having been accomplished, the instruction cards are then ready to be loaded by SpeedCo I, because the machine uses the numerical designations to identify the desired operations.

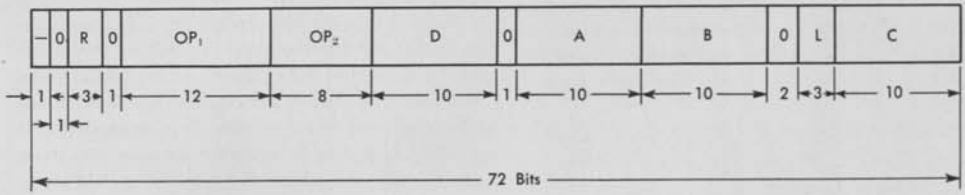
If the programmer is willing to forego the mnemonic advantages of the alphabetical code, there is no reason why he cannot prepare his program entirely in numerical form, specifying each desired operation by giving its numerical designation. This procedure has the advantage of making the preliminary sorting and gang-punching operations referred to above unnecessary.

Each numerical code designation is a three-digit integer. The complete list of all SpeedCo I operations, together with their alphabetical and numerical code designations, appears in the section entitled *Operations.*

In general, each SpeedCo I instruction may specify two distinct operations, referred to as OP_1 and OP_2. A summary of the OP_1 operations is as follows: elementary arithmetic operations, computation of square root, evaluation of elementary trigonometric and exponential functions, transfers of information between tapes or drums and electrostatic storage, and final-result printing.

The addresses A, B and C are used in connection with the OP_1 operations.

Frequently it is desirable to be able to execute the same series of instructions a number of times, each time increasing or decreasing any or all of the addresses A, B and/or C by one. As will be explained in the section entitled *Address Modification,* SpeedCo I provides a convenient means of accomplishing this purpose. In order to designate which addresses, if any, should be so modified, the programmer includes in each instruction a quantity called the R-code. In addi-



Bit Positions	Item
1	Sign of instruction (automatically supplied by SpeedCo I)
2	Always zero
3 - 5	R-code
6	Always zero
7 - 18	OP_1
19 - 26	OP_2
27 - 36	D
37	Always zero
38 - 47	A
48 - 57	B
58 - 59	Always zero
60 - 62	L
63 - 72	C

FIGURE 2

tion to the above information, each instruction (in all but a few exceptional cases to be pointed out later) can also call for an OP_2 operation.

A summary of the OP_2 operations follows: conditional and unconditional transfers of control, address-modification operations, and error-checking operations.

Each instruction also contains an address portion designated D. The D address is used in connection with almost all OP_2 operations.

Finally, every instruction contains a portion designated L. The numbers coded here, together with the positions of certain switches on the operator's panel, control a detailed listing of the entire program—instructions, intermediate results and final results. This feature is extremely helpful in tracking down programming errors when the program is first being tested.

Instructions are stored in the various memory organs of the machine as shown in Figure 2.

OPERATIONS

FOLLOWING is a list of all SpeedCo I operations. For convenient reference, the descriptions are given in abbreviated form. Complete descriptions of all these operations appear in other sections.

OP_1 Operations

Alphabetical Code	Numerical Code	Name	Description
ADD	658	Add	$Q(A) + Q(B) = Q(C)$
SUB	696	Subtract	$Q(A) - Q(B) = Q(C)$
ADDAB	699	Add absolute	$Q(A) + \|Q(B)\| = Q(C)$
ABADD	703	Absolute add	$\|Q(A)\| + \|Q(B)\| = Q(C)$
SUBAB	707	Subtract absolute	$Q(A) - \|Q(B)\| = Q(C)$
ABSUB	711	Absolute subtract	$\|Q(A)\| - \|Q(B)\| = Q(C)$
MPY	715	Multiply	$[Q(A)] \times [Q(B)] = Q(C)$
NGMPY	731	Negative multiply	$-[Q(A)] \times [Q(B)] = Q(C)$
DIV	734	Divide	$[Q(A)] \div [Q(B)] = Q(C)$
NGDIV	748	Negative divide	$-[Q(A)] \div [Q(B)] = Q(C)$
SQRT	782	Square root	$\sqrt{Q(A)} = Q(C)$
SINE	780	Sine	$\sin [Q(A)] = Q(C)$
ARTAN	781	Arc tangent	$\tan^{-1} [Q(A)] = Q(C)$
EXP	783	Exponential	$e^{[Q(A)]} = Q(C)$
LN	784	Logarithm	$\log_e [Q(A)] = Q(C)$
MOVE	690	Move	The block of information stored in electrostatic cells A to B is stored in electrostatic cells C to C + B − A. See Appendix E.
WRTPJ	532	Write tape J	The block of information stored in electrostatic cells A to B is written on the designated tape. The most recent previous instruction affecting the same tape must be either WRITE or REWIND.
WRTPK	533	Write tape K	
WRTPL	534	Write tape L	
WRTPM	535	Write tape M	
RFTPJ	435	Read forward tape J	The first B − A + 1 words of the next block of information stored on the designated tape are read and stored in electrostatic cells A to B. (Notes 1, 3, and 4.)
RFTPK	437	Read forward tape K	
RFTPL	439	Read forward tape L	
RFTPM	441	Read forward tape M	

RBTPJ	416	Read backward tape J	The first B − A + 1 words of the preceding block of information stored on the designated tape are read and stored in electrostatic cells A to B. (Notes 2, 3, and 5.)
RBTPK	417	Read backward tape K	
RBTPL	418	Read backward tape L	
RBTPM	419	Read backward tape M	
SFTPJ	556	Skip forward tape J	The designated tape is advanced (without being read) to the end of the next block of information. (Notes 1 and 4.)
SFTPK	557	Skip forward tape K	
SFTPL	558	Skip forward tape L	
SFTPM	559	Skip forward tape M	
SBTPJ	546	Skip backward tape J	The designated tape is backspaced (without being read) to the beginning of the preceding block of information. (Notes 2 and 5.)
SBTPK	547	Skip backward tape K	
SBTPL	548	Skip backward tape L	
SBTPM	549	Skip backward tape M	
RWTPJ	572	Rewind tape J	The designated tape is rewound to its starting position. The most recent previous instruction affecting the same tape may have been anything except WRITE.
RWTPK	574	Rewind tape K	
RWTPL	576	Rewind tape L	
RWTPM	578	Rewind tape M	
EFTPJ	564	End file tape J	The next 6 feet of the designated tape are erased. This erased length identifies the end of the tape during future tape reading operations. The most recent previous instruction affecting the same tape must be WRITE.
EFTPK	566	End file tape K	
EFTPL	568	End file tape L	
EFTPM	570	End file tape M	
WRDRP	497	Write drum P	The block of information stored in electrostatic cells A to B is written on the designated drum starting at drum address C. The writing process is then automatically checked. If any errors are detected the program skips the next two instructions.
WRDRQ	498	Write drum Q	
RFDRP	526	Read forward drum P	The block of information stored on the designated drum starting at drum address C is read and stored in electrostatic cells A to B.
RFDRQ	528	Read forward drum Q	
PRINT	580	Print	The block of data stored in electrostatic cells A to B is printed. For identification purposes the C address and a line number are printed at the left end of each line of the report.
EJECT	767	Eject	The printer paper is ejected. (See Appendix C.)
NOOP	751	No operation	

Note 1: The most recent previous instruction affecting the same tape must be either READ FORWARD, READ BACKWARD, SKIP FORWARD, SKIP BACKWARD or REWIND.

Note 2: The most recent previous instruction affecting the same tape must be either READ FORWARD, READ BACKWARD, SKIP FORWARD, SKIP BACKWARD or END FILE.

Note 3: If the recomputed check sum fails to agree with the check sum on the tape, the program skips the next two instructions.

Note 4: If the designated tape is already at the end-of-file gap when this instruction is given, the program skips the next instruction.

Note 5: If the designated tape is already at the beginning-of-file gap when this instruction is given, the program skips the next instruction.

OP_2 Operations

Alphabetical Code	Numerical Code	Name	Description
TR	104	Transfer	Control is unconditionally transferred to electrostatic cell D.
TRPL	109	Transfer plus	Control is transferred to electrostatic cell D if Q (C) is positive.
TRMN	115	Transfer minus	Control is transferred to electrostatic cell D if Q (C) is negative.
TRZ	112	Transfer zero	Control is transferred to electrostatic cell D if the fractional part of Q (C) is zero.
SNTRP	117	Sense and transfer P	Control is transferred to electrostatic cell D if operator's panel sense switch P (or Q) is down. (These sense switches, P and Q, are numbered 4 and 5, respectively, on the operator's panel.)
SNTRQ	120	Sense and transfer Q	
TIA	128	Transfer and increase R_A	R_A, R_B, and R_C are quantities stored in electrostatic memory which can be used to modify the addresses A, B, and C, respectively. Execution of one of these instructions increases each of the designated R-quantities by one and transfers control to electrostatic cell D.
TIB	126	Transfer and increase R_B	
TIC	125	Transfer and increase R_C	
TIAB	130	Transfer and increase R_{AB}	
TIBC	127	Transfer and increase R_{BC}	
TIAC	129	Transfer and increase R_{AC}	
TIABC	131	Transfer and increase R_{ABC}	
TDA	135	Transfer and decrease R_A	Same as the corresponding transfer-and-increase operations except that each of the designated R-quantities is decreased by one.
TDB	133	Transfer and decrease R_B	
TDC	132	Transfer and decrease R_C	
TDAB	137	Transfer and decrease R_{AB}	
TDBC	134	Transfer and decrease R_{BC}	
TDAC	136	Transfer and decrease R_{AC}	
TDABC	138	Transfer and decrease R_{ABC}	
SETRA	139	Set R_A	The designated R-quantity is replaced by the D address.
SETRB	250	Set R_B	
SETRC	145	Set R_C	
SKRA	152	Skip R_A	The program skips the next instruction if the designated R-quantity is equal to the D address.
SKRB	159	Skip R_B	
SKRC	162	Skip R_C	
RADDA	199	Reset and add A	The designated address of the instruction located at D is added into the address counter after this counter has been reset to zero.
RADDB	202	Reset and add B	
RADDC	205	Reset and add C	
RADDD	208	Reset and add D	
ADDA	177	Add A	The designated address of the instruction located at D is added to the contents of the address counter.
ADDB	184	Add B	
ADDC	190	Add C	
ADDD	193	Add D	
SUBA	211	Subtract A	The designated address of the instruction located at D is subtracted from the contents of the address counter.
SUBB	216	Subtract B	
SUBC	221	Subtract C	
SUBD	226	Subtract D	

STA	251	Store A	The designated address of the instruction located at D is replaced by the contents of the address counter.
STB	252	Store B	
STC	235	Store C	
STD	244	Store D	
SKIP	165	Skip	The program skips the next instruction if the D address is equal to the contents of the address counter.
PRCH	232	Prepare check	This operation must always be given prior to the first use of the start-check operation.
STCH	253	Start check	This operation causes the instruction of which it is a part to become the first instruction in a *checking loop*, that is, in a sequence of instructions which will be performed twice to check for machine errors.
ECHTR	254	End check and transfer	This operation causes the instruction of which it is a part to become the last instruction in a checking loop. The first time it is encountered it causes a transfer to D, where D is usually the location of the preceding start-check operation. The second time it is encountered it compares the results of the two passes through the loop. If the results are the same the next instruction is skipped. If there is a discrepancy this skip does not occur.
STOP	123	Stop and transfer	Upon encountering this instruction the calculator stops. If it is then restarted by the operator, the first instruction executed is a transfer of control to D. At the time of the stop, the address D is visible (in binary) on the accumulator register lights on the operator's panel.
	000	No operation	

(Note that the alphabetical code for no OP_1 is NOOP, and for no OP_2 is simply a blank.)

ELEMENTARY MATHEMATICAL FUNCTIONS

SPEEDCO I provides for the direct computation of the following elementary mathematical functions: square root, sine, arc tangent, exponential, and natural logarithm.

For each of these operations the programmer must specify the appropriate alphabetical or numerical OP_1 code, the address A of the independent variable, and the address C at which the result is to be stored. These instructions do not require the specification of a B address; hence the number coded there is immaterial.

The present available set of these five elementary functions is designed to give at least seven significant decimal digits of accuracy. For details as to the range of the argument and accuracy of approximation, see Appendix A.

Problems may arise where the accuracy provided is inadequate. In such cases it is possible to modify the SpeedCo I control information to obtain greater accuracy. Such modifications may, however, increase the storage requirements for this control information. Hence the programmer must balance the advantages of greater precision against the disadvantage that the electrostatic storage capacity may decrease from the 714 words now available with the present approximations.

Descriptions of alternative sets of approximations will be added to the appendix as they become available.

INPUT-OUTPUT COMPONENTS

THE CARD READER, printer, magnetic tapes and magnetic drums are all classified as input-output components of the machine, because they all share the com-

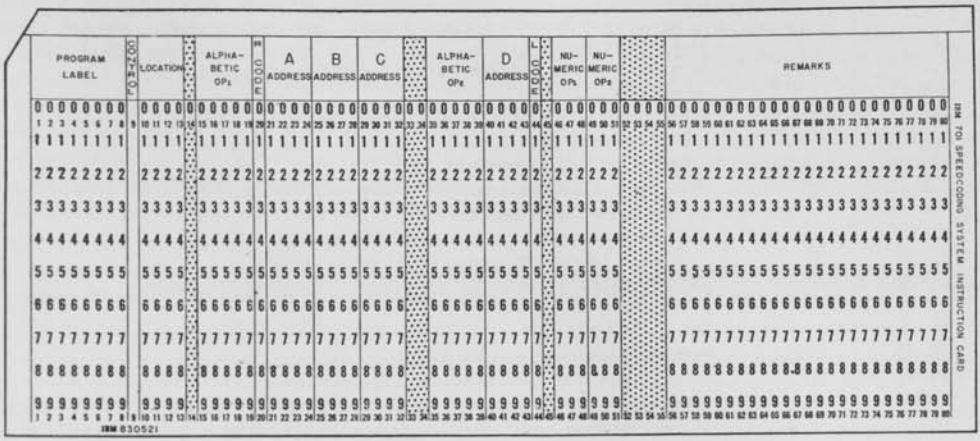


FIGURE 3

mon property of being able to automatically receive information from, or transmit information to, electrostatic storage. In fact, it must be remembered that, whenever information is transmitted from one component of the machine to another, it must pass through electrostatic storage.

Any machine component capable of automatically transmitting information both to and from electrostatic storage (such as tapes and drums) may be regarded as an auxiliary storage (as distinguished from the electrostatic *working storage*). However, the common input-output terminology will be used in this section.

The computer has full automatic control over all input-output components. As will be seen from what follows, this control is exercised by means of the stored program.

Punched Cards

SpeedCo I uses punched cards as its primary input medium because of their great flexibility and because of the availability of apparatus for key-punching, verifying, and duplicating. Errors in key-punching are easily detected and corrected. Input data may readily be prepared on several key-punches simultaneously, and the cards may then be collected for entry into the computer. Cards are particularly desirable for manual access to a file because they can easily be separated, and their contents can be printed on them.

Instruction Card Form (Figure 3)

SpeedCo I instructions are initially entered into the calculator by means of punched cards, one instruction being punched per card. The punching fields are shown in Figure 4.

Note that the instruction card form provides space for both the alphabetical and the numerical OP_1 and OP_2 code designations. If the program has been written using the alphabetical designations, the following procedure applies: (1) the alphabetical designations are key-punched from the programmer's manuscript into card columns 15-19 and 35-39; (2) after the key-punching has been verified, a sort and gang-punch process is carried out which punches the corresponding numerical designations into columns 46-48 and 49-51.

If, on the other hand, the program was written using the numerical designations for OP_1 and OP_2, these numerical designations have only to be key-punched and verified. In this case the alphabetical fields (columns 15-19 and 35-39) are left blank, and no sorting and gang-punching operations are necessary.

Note that the Type 701 is controlled by the numerical code punching only, the alphabetical punching being unintelligible to the machine. For this reason, SpeedCo I instruction cards should never be fed into the Type 701 with only the alphabetical OP_1 and OP_2 code designations punched. All columns, except those

Card Columns	Symbol	Description
1 - 7		Not used by SpeedCo I.
8		Must never have zero and one punch simultaneously.
9		Must always be blank.
10 - 13	Location	Electrostatic memory location at which the instruction is to be stored.
14		Always punched zero.
15 - 19	OP_1	Alphabetical code designation for OP_1.
20	R	The digit coded here controls which of the addresses A, B and C are to be increased by R_A, R_B and R_C, respectively.
21 - 24	A	Address in electrostatic storage of the first operand.
25 - 28	B	Address in electrostatic storage of the second operand.
29 - 32	C	Address in electrostatic memory at which the result is to be stored.
33 - 34		Always punched zero.
35 - 39	OP_2	Alphabetical code designation for OP_2.
40 - 43	D	Address used in connection with the OP_2 operation.
44	L	This digit, taken in conjunction with the positions of the three list switches labeled 1, 2, and 3 on the operator's panel, controls a detailed listing of instructions and results.
45		Not used by SpeedCo I.
46 - 48	OP_1	Numerical code designation for OP_1.
49 - 51	OP_2	Numerical code designation for OP_2.
52 - 55		Always punched zero.
56 - 80		Not used by SpeedCo I.

FIGURE 4

specifically designated to be blank or not used and except the alphabetical operation codes, *must* have a single numerical punch.

Data Card Form

Data are initially entered into the calculator by means of punched cards, five or fewer pieces of data being punched per card. The punching fields are shown in Figure 5.

A sign must be specified for each fractional part and each exponent. In each case the sign is punched over the least significant digit of the fractional part or exponent to which it applies. A 12 punch means plus, and an 11 punch means minus. If the word count is less than 5, the irrelevant signs need not be punched, but the card itself must be filled out with zeros or, at least, some single numerical punch for each irrelevant column.

In no case may the absolute value of an exponent exceed 236.

Card Reader

The process of loading the data and instructions of a program into the calculator is carried out as follows.

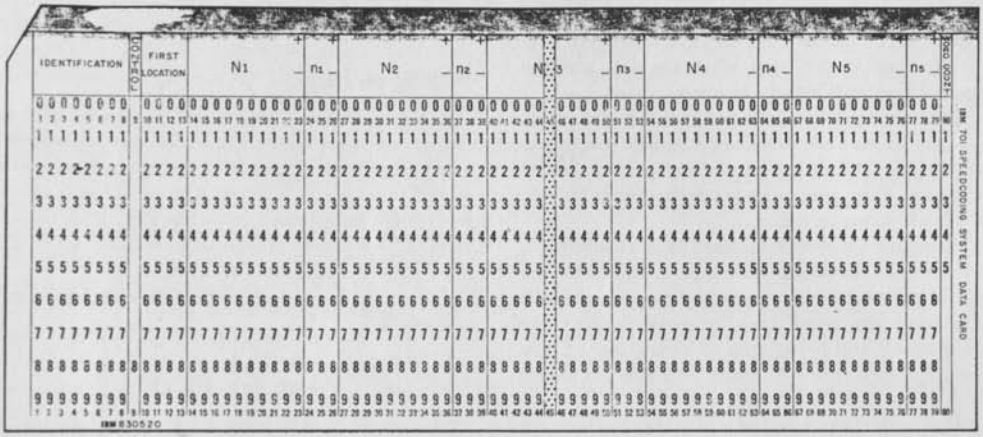
The data and instruction cards are key-punched and verified, and any necessary sort-and-gang-punch operations are performed. The programmer will be supplied with a certain constant deck of cards labelled SPEEDCODING I. (This deck will contain all of the control information necessary to cause the 701 to operate according to the SpeedCo I system. The last card of this deck will be labelled Transfer Control Card.) From this constant deck and his own variable deck he will make up a single deck by inserting his variable deck between the last and the next-to-last card of the constant deck, and adding three blank cards after the last card of the constant deck. The resulting deck is then placed in the hopper of the card reader, and the card-reader start button is depressed to ready the card reader. The calculator is then started by pressing the load button on the operator's panel. No further manual intervention is necessary, subsequent operation being entirely automatic.

The cards in the hopper feed through the card reader, and the information on them is transmitted to electrostatic storage. SpeedCo I then directs and controls the conversion of the decimal information read from the cards.

In the case of instruction cards, the information is

converted to binary and assembled in the standard binary instruction form given in the section entitled *Control*. In the case of data cards, the conversion process involves not only the change from decimal to binary, but also includes the change from the powers-of-ten punched on the cards to the powers-of-two employed within the calculator. This converted data is then assembled in the standard floating-binary-number form given in the section on *Computing*. In both cases the converted information is stored in electrostatic memory at the designated locations. The execution of the problem then automatically begins with execution of the instruction stored at location 300.

During this read-in process the card reader operates at the rate of 150 cards per minute. Hence data is read and converted at the rate of 750 numbers per minute, while the rate for instructions is 150 per minute.



Card Columns	Symbol	Description
1 - 7		Not used by SpeedCo I.
8		Always double-punched zero and one.
9		Always punched eight.
10 - 13	Location	Electrostatic memory location at which the first piece of data on the card is to be stored. The remaining four pieces of data will be stored in the next n consecutively-numbered locations, where n = 0, 1, 2, 3 or 4 depending upon whether the word count is 1, 2, 3, 4 or 5, respectively.
14 - 23	N_1	First fractional part.
24 - 26	n_1	First exponent.
27 - 36	N_2	Second fractional part.
37 - 39	n_2	Second exponent.
40 - 44 46 - 50	N_3	Third fractional part. (In punching N_3 column 45 must be skipped since this card column is not read by SpeedCo I.)
51 - 53	n_3	Third exponent.
54 - 63	N_4	Fourth fractional part.
64 - 66	n_4	Fourth exponent.
67 - 76	N_5	Fifth fractional part.
77 - 79	n_5	Fifth exponent.
80	Word count	Number of pieces of data on the card (= 1, 2, 3, 4 or 5).

FIGURE 5

All calculations made in the process of converting the decimal information on the cards to the required binary form are automatically checked. Any discrepancy causes the machine to stop and give an error indication.

Magnetic Tape

Magnetic tapes may be used either as a high-capacity long-term memory or as input from a previous problem that had stored its results on tape. Input data from cards, including programs, can be transcribed by the computer on tape to conserve storage space or to save time when the data must be repeatedly entered into the computer.

There are four tape units designated by the letters J, K, L and M. Each contains a magnetic tape of any length up to 1400 feet. After the tape has been placed in motion, it can read or write information at the rate of 625 words per second.

Information is recorded on tape in six channels that run parallel to the length of the tape. A bit of information is represented by a magnetized spot in a channel. A set of six bits recorded in a line perpendicular to the six channels will be referred to as a *group* of bits. Twelve groups recorded serially on a tape are needed to store one binary word of 72 bits.

A seventh channel on the tape serves to check the reading and writing in the other six channels by the so-called *redundancy-check* principle. That is, either a 0 or a 1 is recorded in the seventh channel so that across the seven channels there is an odd number of 1's in each set of seven bits. When the tape is read, the number of 1's is automatically checked. If the number is even, the calculator stops, and a signal light on the operator's panel called the *tape-check* light is turned on.

If the number of 1's is odd (as it should be when correct), the machine continues the reading process. It should be emphasized that the operation of this seventh channel is completely automatic and requires no attention whatever on the part of the programmer.

A schematic diagram of how a word of 72 bits is recorded on tape is shown in Figure 6. Each x denotes a binary 1 or 0 recorded in that position on the tape. The 72 bits recorded in the six recording channels represent the full word. The tape moves in the direction of the arrow. The group numbered 1 contains the first six bits of the word. The remaining eleven groups contain the following bits of the word in groups of six. Thus, group 2 contains bits 7 through 12 of the word, etc.

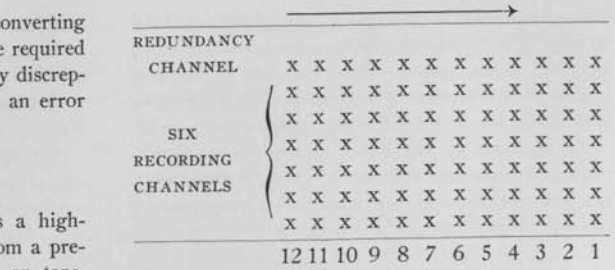


FIGURE 6

Writing on Tape

The general procedure for writing a file of information on tape is as follows. A write-tape instruction is given. This instruction causes the selected tape to be started in motion and causes it to record, as one unit record, the information stored in that block of consecutively-numbered electrostatic memory cells which begins at cell A and ends at cell B.

For example, consider the following write-tape instruction:

OP_1	A	B	C
WRTPK	0301	0387	—

This instruction will cause the block of information stored in electrostatic cells 301 to 387 inclusive to be written, as a unit record, on tape K.

The writing process requires that all previous magnetic marks be erased from that portion of the tape being written on. To accomplish this, an erasing apparatus precedes the recording apparatus by approximately two inches. As the tape moves under the impetus of the write-tape instruction, the erasing apparatus is continually active, while the recording apparatus does not operate until told to do so by SpeedCo I.

If the write-tape instruction is given when the tape unit is in the rewound condition (i.e., in position to write a file of records), the actual writing on tape is delayed eight-tenths of a second. The erase circuits, however, are functioning during this time, and the result is a blank portion of tape called the *beginning-of-file gap*.

As soon as the beginning-of-file gap has been written, the recording process begins. The selected tape unit takes the word located at A and records it on the tape as 12 six-bit groups in the manner shown in the diagram above. Next the word at A+1 is recorded.

This is followed immediately by the recording of the word at A+2. This process continues until the word located at B has been recorded.

During this process two other operations have also been taking place. First, as each group of bits was recorded on tape, the corresponding redundancy bit was automatically computed and recorded. Second, as the successive words of the record being written were selected from electrostatic storage a calculation was carried on that resulted in the formation of a weighted sum of all the binary information comprising the record. This sum (called the *check sum*) is then automatically recorded on the tape immediately behind the word taken from location B.

Note that the length of the unit record is variable, the number of words recorded being given by the expression B — A+1. This number of words and their associated check sum having been written, the tape unit automatically disconnects itself from the calculator and stops. Because of the time necessary for the tape to come to a complete stop, and the two-inch distance between the erase head and writing head, there results a small section of erased tape. The gap caused by this erasure is called an *end-of-record gap*.

Note that the write-tape instruction makes no use of the address C; and so any number coded in the C address field is irrelevant to the execution of the instruction.

The first unit record of the file has now been written. To write a second unit record, a second write-tape instruction is programmed. In this way a series of records may be recorded. Note again that the records may be of variable size if desired.

The complete recording on a tape consists of a number of unit records that make up a file of information.

Tapes can be re-used many times, and a new file can be written over an old file, the old one being erased in the process. Each time a new file is written, it is started at the beginning of the tape, and only one usable file of unit records can be on a tape at one time. Different files, however, will have different lengths, so that there is a possibility that beyond the last record of the most recent record file, bits may be left over from a previous use of the tape. These residual bits of information may not be properly spaced in relation to the record just written. This may result in an error on a later reading of the new record.

To avoid having to erase the entire tape every time, and for certain control purposes to be mentioned later, an instruction called END FILE TAPE has been provided. This instruction, which must be given after writing any file, erases a further section of tape after the last unit record. The section of tape erased in this way is called an *end-of-file gap*.

For example, to end a file of records just written on tape K, the following instruction should be given:

OP_1	A	B	C
EFTPK	—	—	—

In this instruction the addresses A, B and C are irrelevant to the execution of the instruction.

Figure 7 shows schematically how a typical file of information is recorded on tape. The arrow designates the forward direction of tape motion. Writing can be done only when the tape is moving forward. A beginning-of-file gap is followed by a number of unit records with intervening end-of-record gaps. Note that these gaps are of a fixed length regardless of the lengths of the unit records themselves. Finally, an end-of-file gap appears after the last unit record. Note, too, that the machine operates so that the lengths of the two gaps at each end of a file are equal to each other, but are longer than the intervening end-of-record gaps.

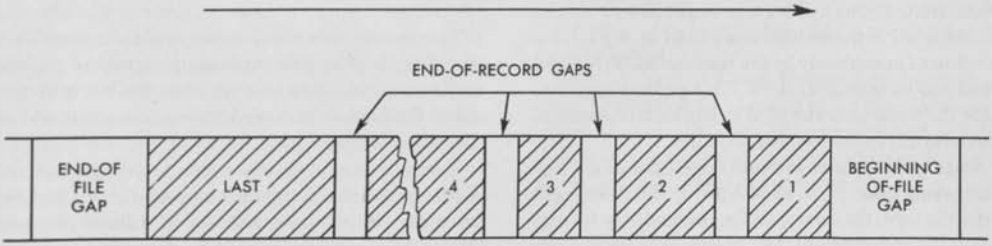


FIGURE 7

To recapitulate, there are three kinds of gaps in the recording of information on tape:

1. The normal spacing between successive groups of six bits within a unit record.
2. The longer gap between unit records. This gap is long enough to allow the tape to start and stop between records.
3. The still longer gaps at the ends of the file.

Reading from Tape

The general procedure for reading a file of information from tape is as follows.

Assume that the tape is in position to read the first unit record of the file. A read-forward-tape instruction is given. This instruction causes the selected tape to be started in motion and causes it to transmit the first B − A + 1 words of the first unit record to electrostatic memory where they are stored in that block of consecutively-numbered memory cells which begins at cell A and ends at cell B.

For example, consider the following read-forward-tape instruction:

OP_1	A	B	C
RFTPK	0851	0885	—

If the first unit record of the file contains 35 words (in establishing the number of words in a unit record on tape, the check sum is not included in the count), this instruction will cause those 35 words to be read from tape K and stored blockwise in electrostatic memory cells 851 to 885 inclusive. If, on the other hand, the first unit record contains more than 35 words, this instruction will cause the first 35 of these words to be read from the tape and stored in the designated memory cells.

In detail, the read-forward process proceeds as follows. The selected tape transmits the first word to electrostatic memory where it is stored at A. Next the second word is transmitted and stored at A+1. This is followed immediately by the transmittal of the third word and its storage at A+2. This process continues until the word to be stored at B has been transmitted and recorded in electrostatic memory.

During this process two other operations have also been taking place. First, as each group of bits was read from the tape, the corresponding redundancy bit was also read and checked. Any discrepancy in this redundancy check will stop the calculator and light the tape-check light.

Second, as the successive words of the record being read were transmitted to electrostatic storage, the check sum was recomputed and compared with the check sum read from the tape. Any discrepancy in this comparison will cause the calculator to ignore the next two instructions of the program and skip to the third instruction following the read-forward-tape instruction.

Note that both of these checking procedures are carried out for the entire unit record being read, even in those cases where only part of the record is being stored in electrostatic memory. In such cases the storing process is discontinued after the (B − A + 1)st word has been read and stored, but the tape unit continues transmitting until all words of the unit record, including the check sum, have been transmitted. During this latter phase, when the words being read from tape are not being stored in electrostatic memory, the redundancy-check process and the check sum calculation still continue. Hence if the calculator does not stop, and if it does not skip the two instructions following the read-forward-tape instruction, the programmer is assured that the tape-writing and reading process has been correctly performed.

Note also that since the reading process is always continued to the end of the record, even when only part of the record is stored in electrostatic memory, the tape always stops at the end-of-record gap; this concludes the record just read.

Because the read-forward-tape instruction makes no use of the address C, any number coded in the C address field is irrelevant to the execution of the instruction.

If for any reason the A and B addresses of a read-forward-tape instruction are so chosen that B − A+1 is larger than the number of words in the record being read, the calculator will stop and a signal light on the operator's panel called the *copy-check* light will be turned on.

The process described above reads the first unit record of the file. The remaining records of the file may be read in the same way. After the last unit record of the file has been read, the tape is positioned at the end-of-file gap.

If, at this time, another read-forward-tape instruction is given, the designated tape unit starts up and attempts to read another unit record. Instead of a unit record, however, it finds the end-of-file gap. This causes the calculator to ignore the next instruction of the program and skip to the second instruction follow-

ing the read-forward-tape instruction. The programmer may then take advantage of this automatic skip to go into a new phase of the program. This end-of-file skip cannot be obtained from a blank tape; at least one unit record must be written on the tape to distinguish the normal space at the start of the file from the end-of-file gap obtained by use of the end-file-tape instruction.

It should be emphasized, however, that this is not the only function of the end-of-file gap (see section entitled *Writing on Tape*). Even if the program is so written as to make no use whatever of the end-of-file skip feature, it is still necessary to give an end-file-tape instruction after writing a file on tape.

Any OP_2 operation may be used with a read-forward-tape instruction. However, it is important to note that SpeedCo I is so constructed that such an OP_2 operation will only be executed if neither the check-sum-discrepancy skip nor the end-of-file skip has occurred.

In certain types of programs it is convenient to be able to read the unit records which make up a file on tape in the reverse order from the order in which they are written. To facilitate such tape reading programs, the read-backward-tape instruction has been provided as part of SpeedCo I.

The read-backward-tape instruction is very similar to the read-forward-tape instruction.

They differ only in the following respects. When READ BACKWARD TAPE is given, the designated tape is first backspaced one unit record. During this backspacing no reading, checking or storing takes place. The unit record is then read in the forward direction and checked, exactly as in the read-forward-tape case. The designated tape then backspaces once more—again without reading, checking or storing. After execution of the read-backward-tape instruction, the tape stops at the end-of-record gap that precedes, on the tape, the unit record just read.

For example, assume that the tape to be read is stopped at the end-of-record gap between the fourth and fifth unit records on the tape. The instruction READ FORWARD TAPE will read and store the fifth record, and the tape will stop at the end-of-record gap between the fifth and sixth unit records. If the instruction given had been a read-backward-tape instruction, however, the fourth record would have been read and stored, and the tape would have stopped at the end-of-record gap between the third and fourth unit records.

Note that the addresses A and B have the same significance in the instruction READ BACKWARD TAPE as they have in the instruction READ FORWARD TAPE. Thus the instruction READ BACKWARD TAPE causes $B - A + 1$ words to be transmitted to electrostatic memory and stored in cells A through B. If the unit record contains more than this number of words, only the first $B - A + 1$ words of the record are stored in electrostatic memory (where *first* is used in the sense *first written*).

The redundancy-check process and the check-sum calculation are carried out for the entire unit record involved, even though only part of the record may be stored.

If any discrepancy is detected in the redundancy check, the calculator will stop, and the tape-check light will be turned on. If the recomputed check sum fails to agree with the check sum read from the tape, the calculator will skip the next two instructions.

If A and B are so chosen that $B - A + 1$ exceeds the number of words in the record being read, the calculator will stop, and the copy-check light will be turned on.

In all three of the above cases the error is detected during the second (or read-forward) phase of the read-backward-tape instruction. For this reason the tape comes to rest, not at the end-of-record gap that precedes the record read, but at the end-of-record gap that follows the record read. The final back-spacing, which is normally part of the read-backward-tape instruction, does not take place when one of these three error conditions is encountered.

The C address of the read-backward-tape instruction is irrelevant to the execution of the instruction.

When reading a tape backward, an end-of-file gap is recognized just as in reading forward. After the unit record at the beginning of the file has been read backward, the tape will be positioned at the beginning-of-file gap. A further read-backward-tape instruction causes the tape unit to treat the beginning-of-file gap as if it were an end-of-file gap, and the calculator skips the next instruction.

As in the case of a read-forward-tape instruction, any OP_2 operation may be used with a read-backward-tape instruction, but the OP_2 operation will only be executed if neither the check-sum-discrepancy skip nor the end-of-file skip has occurred.

Tape Skipping

Cases will frequently arise where the unit records of a file on a tape must be read in a more-or-less ran-

dom order. The instructions READ FORWARD TAPE and READ BACKWARD TAPE do not by themselves conveniently permit such tape reading, because the instruction READ FORWARD TAPE reads only the record which immediately follows the end-of-record gap at which the tape is stopped, while the instruction READ BACKWARD TAPE reads only the record which immediately precedes this same end-of-record gap. In order to facilitate random tape reading, therefore, the instructions SKIP FORWARD TAPE and SKIP BACKWARD TAPE have been provided.

The instruction SKIP FORWARD TAPE advances the designated tape one unit record, the tape stopping at the end-of-record gap following the one at which it was stopped when the instruction was given. No reading or checking is performed.

The instruction SKIP BACKWARD TAPE backspaces the designated tape one unit record, the tape stopping at the end-of-record gap preceding the one at which it was stopped when the instruction was given. Again, no reading or checking is performed.

For example, suppose that the last operation involving tape K was a read-backward-tape-K instruction that read and stored the fifth unit record of the file. Tape K is therefore stopped at the end-of-record gap between the fourth and fifth records. If the next record to be read is the second, the programmer could accomplish this by giving the instruction SKIP BACKWARD TAPE K two times, followed by the instruction READ BACKWARD TAPE K.

The two skip-backward-tape-K instructions would backspace tape K to the end-of-record gap between the second and third records; and the read-backward-tape-K instruction would read and store the second record, the tape finally stopping at the end-of-record gap between the first and second records.

The instructions SKIP FORWARD TAPE and SKIP BACKWARD TAPE both embody an end-of-file skip feature. If SKIP FORWARD TAPE is given when the designated tape is stopped at the end-of-file gap, or if SKIP BACKWARD TAPE is given when the tape is stopped at the beginning-of-file gap, the calculator ignores the next instruction and proceeds immediately to the execution of the second instruction following the tape skip instruction.

Any OP_2 operation may be used with a skip-forward-tape or skip-backward-tape instruction. However, it is important to note that such an OP_2 operation will only be executed providing an end-of-file skip has not occurred.

The A, B and C addresses of the instructions SKIP FORWARD TAPE and SKIP BACKWARD TAPE are irrelevant to the execution of the instructions.

After the instruction SKIP FORWARD TAPE or SKIP BACKWARD TAPE has been given and interpreted and the actual tape motion has begun, the calculator proceeds with the interpretation and execution of the following instructions of the program without waiting for the tape to come to a stop. If a subsequent instruction calls for the same tape while it is still in motion, the program is automatically delayed until the tape stops.

Tape Rewinding

The instruction REWIND TAPE is used to cause the designated tape to return to the starting point of its file of records.

When writing a file of unit records on a tape, the instruction REWIND TAPE should not be given until after the instruction END FILE TAPE has been used to terminate the file.

The A, B and C addresses of the instruction REWIND TAPE are irrelevant to the execution of the instruction.

After the instruction REWIND TAPE has caused the rewinding action to begin, the calculator proceeds to the following instructions of the program without waiting for the tape to stop. If a subsequent instruction calls for the same tape before the actual rewinding is complete, the program is then automatically delayed until the tape comes to a stop.

Tape Status

It is not possible both to read and write on a single passage of a tape in one direction through the tape unit. If information is being written on a tape, the file should first be completed by writing an end-of-file gap before the information is read. The tape can then be read backward immediately, or it can be rewound and then read in the forward direction. When a tape has been used for reading, it must first be rewound before any new information can be written on it.

Circuits associated with each tape unit remember whether the tape is being read or written. When the circuits are set up for reading, the tape unit is said to be in *read status;* when the circuits are set up for writing, the tape unit is said to be in *write status.* If the tape unit is in neither read status nor write status, it is said to be in *neutral status.*

When a tape unit is in read status, it may be used only for reading and skipping; when in write status, it may be used only for writing. If the instruction WRITE TAPE is given for a tape unit that is in read status, the instruction is not executed, the calculator stops, and the copy-check light is turned on. Similarly, if any one of the instructions READ FORWARD TAPE, READ BACKWARD TAPE, SKIP FORWARD TAPE, or SKIP BACKWARD TAPE is given for a tape unit that is in write status, the instruction is not executed, the calculator stops, and the copy-check light is turned on.

If the instruction END FILE TAPE is given for a tape unit that is in read status, the instruction is not executed, and the tape remains stationary.

Any write, read or skip instructions will be executed in the normal manner, if the tape called for is in neutral status. After the execution of a normal write instruction, the tape unit used will be left in write status. After the execution of a normal read or skip instruction, the tape unit used will be left in read status. Irrespective of its previous status, the execution of a rewind instruction will always restore a tape unit to neutral status. An end-file-tape instruction will return the tape to neutral status only if it was originally in write status.

Figure 8 shows the status that will result when each one of the tape instructions is given under all possible conditions.

REWIND TAPE will be executed with the tape unit in any status. However, REWIND TAPE should not be given while the unit is in write status. END FILE TAPE, which resets the tape unit to neutral, should always precede REWIND TAPE in those cases where the tape was originally in write status.

A rewind-tape instruction must always be given before writing on a tape previously in read status, even after reading backward all the way to the beginning-of-file gap.

Once a tape has been rewound, further rewind-tape instructions may be given, but they will be ignored as long as the tape remains rewound.

After an end-of-file gap has been written, the tape unit will be in neutral status. At this point the programmer must not give any of the instruction READ FORWARD TAPE, SKIP FORWARD TAPE, WRITE TAPE, or END FILE TAPE (each of which would move the tape further forward) although the neutral status would permit such meaningless action. Only READ BACKWARD TAPE, SKIP BACKWARD TAPE, or REWIND TAPE can follow END FILE TAPE.

When a tape is being read or skipped in the forward direction and the end-of-file gap has once been sensed, no further read-forward-tape or skip-forward-tape instructions (which would move the tape further forward) should be given. Only READ BACKWARD TAPE, SKIP BACKWARD TAPE, or REWIND TAPE can follow the detection of an end-of-file gap. Similarly, after sensing the beginning-of-file gap on reading or skipping backward, or after giving the instruction REWIND TAPE, the programmer must not give READ BACKWARD TAPE or SKIP BACKWARD TAPE.

Magnetic Drums

The magnetic drum storage is divided into two blocks of 1024 words each, with addresses consisting of the integers from 0 to 1023. Each block will be referred to as a drum. The two drums are designated

Instruction	Resulting status if original status was:		
	Read	Neutral	Write
READ FORWARD TAPE	Read	Read	Write*
READ BACKWARD TAPE	Read	Read	Write*
SKIP FORWARD TAPE	Read	Read	Write*
SKIP BACKWARD TAPE	Read	Read	Write*
WRITE TAPE	Read*	Write	Write
END FILE TAPE	Read**	Neutral	Neutral
REWIND TAPE	Neutral	Neutral	Neutral

*Instruction not executed. Copy-check stop.
**Instruction not executed. No copy-check stop.

FIGURE 8

by the letters P and Q. These drums provide an auxiliary memory that is, for many purposes, more accessible than tape memory. Individual words on a drum can be selectively altered at any time. Drums are used to a large extent for storing tables of data and sections of long programs that may not fit into the electrostatic memory.

Information is usually recorded on the drum as blocks of words called unit records. The words of a unit record are stored in locations with consecutive addresses, although the first word of a record may be placed at any drum address.

The following paragraphs describe the process of writing a unit record on a drum.

A write-drum instruction is given. This causes the designated drum to be connected to the calculator and causes it to record, as one unit record, the information stored in that block of consecutively-numbered electrostatic memory cells which begins at cell A and ends at cell B. The first word of the unit record is written on the drum at drum address C, and the last word is written at drum address C+B − A.

For example, consider the following write-drum instruction:

OP_1	A	B	C
WRDRQ	0947	1013	0032

This instruction will cause the block of information stored in electrostatic cells 947 to 1013, inclusive, to be written on drum Q as a unit record beginning at drum address 32 and ending at drum address 98.

The instruction WRITE DRUM also includes a checking feature that operates as follows. After the block of information from electrostatic storage has been written on the drum, and before the calculator goes on to the next instruction of the program, the information just written on the drum is read back into electrostatic storage and checked, word for word, against the information originally in electrostatic storage. If there is no discrepancy, the calculator proceeds to the interpretation and execution of the next instruction of the program. If, on the other hand, the calculator detects any discrepancy between the original information and the information read back from the drum, the next two instructions of the program are ignored, and the calculator skips to the third instruction following the write-drum instruction. Hence, if the calculator does not skip the two instructions following the write-drum instruction, the programmer is assured that the drum writing process has been correctly performed.

Any OP_2 operation may be used with a write-drum instruction. However, it is important to note that such an OP_2 operation will only be executed if a checksum-discrepancy skip has not occurred.

The drum reading operation is very similar. It is carried out as follows. A read-forward-drum instruction is given. This causes the designated drum to be connected to the calculator and causes it to transmit to electrostatic storage that block of B − A + 1 words which is stored on the drum beginning at drum address C.

Upon its transmittal to electrostatic memory, this information is stored in that block of consecutively-numbered electrostatic memory cells which begins at cell A and ends at cell B.

For example, consider the following read-forward-drum instruction:

OP_1	A	B	C
RFDRQ	0811	0820	0041

This instruction will cause the block of information stored on drum Q, beginning at drum address 41 and ending at drum address 50, to be transmitted to electrostatic storage and stored in electrostatic cells 811 to 820 inclusive.

No checking features are included in the drum-reading operation.

Note that a unit record written on a drum does not preserve its identity as a unit record, there being no end-of-record gaps to mark its beginning and end. For instance, the information read in the above read-forward-drum example would be the 10th through the 19th words of the unit record written in the preceding write-drum example, unless another write-drum instruction involving this region of the drum had intervened.

If only a single word is to be written or read, the electrostatic address of this word must be used as both the A and the B addresses of the instruction concerned.

Drum reading and writing have none of the status limitations that affect tape reading and writing, nor is there anything in drum reading and writing which corresponds to the end-of-file features of tape reading and writing.

The programmer must not attempt to read or write beyond drum address 1023. For example, it is not

possible to write a 7-word record beginning at drum address 1022 and ending at drum address 4.

Loading Procedures for Lengthy Programs

In the section entitled *Card Reader* a procedure is described for loading short programs into the calculator. This procedure does not apply, however, to programs which involve more than 714 words of instructions and data. In such cases, the additional storage capacity of tapes or drums must be utilized. This is done in the following way.

During the writing of the program the programmer subdivides it into a number of blocks, each of which requires no more than 714 words of storage to accommodate all of its input data, its instructions, its output data and any *erasable storage* needed to retain intermediate results temporarily.

The punched cards that contain the instructions and data of the problem are then grouped in blocks in the same way, the cards of each block being separated from those of the next block by means of a control card. This control card contains information that completely specifies a tape-writing or drum-writing operation.

The layout of the loading control card is as follows:

Card Columns	Description
1 - 7	Not used by SpeedCo I.
8	Must not be punched with zero and one simultaneously.
9	Always punched nine.
10	Always punched zero.
11 - 14	Electrostatic storage address of first word of unit record to be written on tape or drum.
15	Always punched zero.
16 - 19	Electrostatic storage address of last word of unit record to be written on tape or drum.
20	Always punched zero.
21 - 24	Drum address of first word of unit record to be written on drum. (In case tape writing is specified, the field is irrelevant but must be filled out with zeros.)
25 - 28	Always punched zero.
29	Tape or drum identification: J, K, L or M for tape; P or Q for drum.
30 - 80	Not used by SpeedCo I.

With one important exception, the loading process is the same as in the simpler case already described. That is, the programmer inserts his variable deck of instruction, data and loading control cards between the last and the next-to-last cards of the constant deck labelled SPEEDCODING I, adds three blanks, puts this resulting deck in the hopper of the card reader, depresses the start button of the card reader and the load button on the operator's panel.

The loading action is now as follows. The first block of data and instruction is read by the card reader and stored in the designated electrostatic memory locations. The first control card is then read, the card reader stops, and the information of the first block is taken from electrostatic storage and written as a unit record on the designated tape or drum. The card reader automatically resumes operation, and the instructions and data of the second block are read, converted and stored in electrostatic storage. The second control card stops the card reader again and causes the information of the second block to be written on the designated tape or drum. The card reader resumes operation. This process continues until all blocks except the last have been dealt with. The last block is read in, converted, and stored in electrostatic memory, following which execution of the program begins with the execution of the instruction which is then stored at location 300.

Because the first instruction executed is one of those in the last block loaded, the deck of cards fed through the card reader must be arranged to take care of this.

The important exception mentioned above in this kind of loading is: if any of the loading control cards have specified tape writing, the specified tapes should first be given an end-file-tape instruction and a rewind-tape instruction if the programmer desires to begin reading these tapes from the beginning-of-file in his problem, or merely an end-file-tape instruction if he desires to begin reading backwards. These preparatory tape instructions *must* be provided by the program itself unless the programmer desires merely to continue writing on the tapes in question after the loading process is completed and his program begins execution.

The actual sequence of tape events in any SpeedCo I loading is as follows. In normal operation, all four tapes are always rewound (for suppression of tape rewinding, see Appendix D) before any instruction cards, data cards, or loading control cards are read by the card reader. If no loading cards pertaining to tapes are contained in the programmer's deck, the tapes stay in this rewound, neutral status until they are first used by the program. If any loading cards pertaining to

tapes *are* contained in this deck, the effect on the tapes is exactly the same as though a write-tape instruction had been executed as many times as there are such loading-control cards, and in the same order as they occur in the deck.

The above process may easily be modified to accommodate the case of a lengthy program which is to be executed using the same instructions, but not the same data, many times in the future. In this event a significant time saving will usually be achieved if the instructions are transferred from cards to tape once for all, and are thereafter loaded from tape. If this is done, the loading process for future runs will be to put the program tape on the correct tape unit and to load the data by means of the card reader. The only other requirement is that a few special instruction cards be loaded along with the data cards. After the data and these special cards have been read, converted and stored in electrostatic memory, the calculation will begin with a tape-reading operation. As a result of this operation the first block of the program will be read from the tape into electrostatic storage. The calculation then begins, in normal fashion, with the execution of the instruction that has been stored in location 300 by the tape-reading operation just completed.

Printer

The final results of SpeedCo I are printed on a line printer which prints at the rate of 150 lines per minute. Five results and a line-identification number print on each line, so that numbers are printed at the rate of 750 per minute.

The general procedure for printing a unit record is as follows. A print instruction is given. This causes the printer to be connected to the calculator and causes it to print, as one unit record, the results stored in that block of consecutively-numbered electrostatic memory cells which begins at cell A and ends at cell B. The results are printed five numbers to the line. An identification number prints opposite each line of results. This number is automatically computed using the formula $100\ C+n$, where C is the number coded as the C address of the print instruction and n is the number of the line in the block.

For example, consider the following print instruction:

OP_1	A	B	C
PRINT	0461	0473	0407

This instruction will cause the block of results stored in electrostatic cells 461 to 473 inclusive to be printed and identified as follows:

Identification	Results
407 01	$Q(461)$ $Q(462)$ $Q(463)$ $Q(464)$ $Q(465)$
407 02	$Q(466)$ $Q(467)$ $Q(468)$ $Q(469)$ $Q(470)$
407 03	$Q(471)$ $Q(472)$ $Q(473)$

The results are printed in floating decimal. The conversion from the floating-binary information within the calculator to the floating-decimal information printed is performed automatically by SpeedCo I. Once the conversion has been performed SpeedCo I checks the process by reconverting the decimal information to binary and comparing the results with the original binary information. The mechanical action of the printer is also automatically checked. After the type mechanism has been physically set up to print a line, it transmits to the calculator a series of signals indicating what characters the type wheels are positioned to print. SpeedCo I then compares this information with the setup signals previously sent by the calculator to the printer. If either of these two comparison checks reveals any discrepancy, the calculator stops and gives an error indication. Hence if the print operation is concluded without this error indication arising, the programmer is assured that the information printed is the true decimal equivalent of the binary unit record specified.

The checking procedure is incomplete in only one respect. The print mechanism does not send back to the calculator any indication of the signs it prints—hence if there is an error in the algebraic sign of a quantity printed, no error indication will be given.

It is very important that the programmer remember that the absolute value of the decimal exponent of any quantity to be printed may never exceed 236.

TRANSFER OPERATIONS

ONE OF THE very important advantages of a stored-program calculator is the ease with which it can be controlled to carry out the instructions stored in its memory in some order other than the order in which they appear there. By use of the various operations available in SpeedCo I, for instance, the programmer may cause the calculator to repeat a designated sequence of instructions a specified number of times, or to repeat the sequence as often as necessary to produce

a specified result, or to choose between two or more alternate procedures depending upon the sign or the size of a computed quantity, or to depart in various other ways from its normal routine of executing instructions in the order in which they are stored.

These transfers of control are effected, as is explained in the section entitled *Control*, by placing into the program counter the address of some instruction other than the one that immediately follows the current instruction. Because the program counter controls which instruction is to be executed next, the program does not continue in the usual sequential manner, but instead transfers control to the new location given.

The following transfer instructions are available. They are all OP_2 operations.

Transfer (TR)

When the program encounters this instruction the number coded in its D address field is placed into the program counter, so that the instruction located at D is the next instruction executed.

For example, consider the following transfer instruction:

LOC	OP_2	D
0487	TR	0811

In this case the program will execute the OP_1-A-B-C portion of the instruction located at 487, and will then enter the number 811 into the program counter. As a result the next instruction executed will be the one stored at 811, following which the program will be executed sequentially from this new location until another transfer operation is encountered.

Transfer Plus (TRPL)

When the program encounters this instruction, the number coded in its D address field is placed into the program counter if, and only if, the quantity computed and stored at C by the OP_1-A-B-C part of the same instruction is positive. (Zero is regarded as positive.)

For example, consider the following instruction:

LOC	OP_1	A	B	C	OP_2	D
0519	ABSUB	0684	0832	0712	TRPL	0498

This instruction will cause the calculator to compute the value of $|Q(684)| - |Q(832)|$. This result will then be stored at location 712. If this result was positive, the next instruction executed will be the instruction located in cell 498. If, on the other hand, this result was negative, the next instruction executed will be the instruction located in cell 520.

Transfer Minus (TRMN)

When the program encounters this instruction, the number coded in its D address field is placed into the program counter if, and only if, the quantity computed and stored by the OP_1-A-B-C part of the same instruction is negative. Hence, if the result at C is negative, control transfers to D whereas, if it is positive, the program continues sequentially. (Zero is regarded as positive.)

Transfer Zero (TRZ)

When the program encounters this instruction, the number coded in its D address field is placed into the program counter if, and only if, the fractional part of the quantity computed and stored by the OP_1-A-B-C part of the same instruction is equal to zero.

For example, consider the following instruction:

LOC	OP_1	A	B	C	OP_2	D
0519	ABSUB	0684	0832	0712	TRZ	0498

This instruction will cause the calculator to compute the value of $|Q(684)| - |Q(832)|$. This result will then be stored at location 712. If the fractional part of this result was equal to zero, the next instruction executed will be the instruction located in cell 498. If on the other hand this fractional part was non-zero, the next instruction executed will be the instruction located in cell 520. The programmer is warned that because of the intrinsic error involved in conversion, exact agreement of numbers resulting from his decimal computations does not necessarily produce exact agreement between the corresponding binary numbers.

Transfer Plus, Minus and Zero

The operations TRANSFER PLUS, TRANSFER MINUS and TRANSFER ZERO are all conditional transfer operations whose effect depends upon the value of the quantity computed and stored at C. For this reason no one of these instructions should be given as part of a SpeedCo I instruction which does not, during its OP_1-A-B-C part, compute and store a number at C.

A situation may arise when the programmer wants to determine both whether a computed result is zero or non-zero, and also whether the same computed re-

sult is positive or negative. Suppose for example that a transfer is to occur if the difference between two numbers (x and y) is less than or equal to zero, but is not to take place if this difference is greater than zero. The following pair of instructions illustrates one of several possible methods of accomplishing this result:

LOC	OP_1	A	B	C	OP_2	D
f	SUB	$L(x)$	$L(y)$	$L(x-y)$	TRZ	g
$f+1$	SUB	$L(x)$	$L(y)$	$L(x-y)$	TRMN	g

If $x - y$ is zero, a transfer to g will occur as a result of instruction f. If $x - y$ is non-zero, this transfer will not occur and instruction $f+1$ will be executed next. Here the instruction is repeated, except that OP_2 is a TRANSFER MINUS. Hence if $x - y$ is negative, a transfer to g will take place. Thus the combination of instructions f and $f+1$ results in a transfer to g if $x - y$ is equal to or less than zero, whereas, if $x - y$ is non-zero and positive, the program continues sequentially with instruction $f+2$.

Sense and Transfer (SNTRP, SNTRQ)

These two operations are also conditional transfer operations. Each of them is associated with a switch on the operator's panel called a sense switch. Sense switches P and Q are labelled 4 and 5, respectively, on the operator's panel. If the designated switch is in the ON position when SENSE AND TRANSFER is given, control is transferred to the location specified by the D address of the sense-and-transfer instruction. If, on the other hand, the switch is in the OFF position, the transfer does not take place. These operations are intended primarily to permit the operator to modify the action of the program manually while it is stored in the machine.

For instance, in a program that includes an iterative process of some sort, it might be desirable to print the results of every iteration in some cases and to omit iterative-result printing entirely in others. This could be accomplished as follows. Let the print instruction which prints the results of each iteration be located in cell f. Then, in the OP_2 operation field of instruction $f - 1$, code the instruction SENSE AND TRANSFER P (or Q), and in the D address field of the same instruction code the address $f+1$. Then, if the designated sense switch is OFF, printing will take place at the end of each iteration, whereas, if the switch is ON, printing will be omitted.

Other Transfer Operations

In addition to the operations described above, SpeedCo I includes two other types of transfer instructions called TRANSFER AND INCREASE and TRANSFER AND DECREASE. These instructions are explained in the section entitled *Address Modification*.

ADDRESS MODIFICATION

TO REALIZE fully the advantages of stored-program calculation, it is necessary that means be provided to modify the addresses of the program while the program is being executed. This requirement arises in the fact that in a great many problems the solution is quite repetitious in form, the same sequence of instructions being executed over and over, each time operating upon different data. To facilitate solution of this typical kind of problem, SpeedCo I provides several means for modifying the address parts of stored instructions as the calculation proceeds.

One of these modification features involves the use of three non-negative quantities designated R_A, R_B, and R_C, which are stored as part of the SpeedCo I control information. These R-quantities serve as increments that may selectively be added to the A, B and/or C addresses of designated instructions of the program.

Whenever such an addition takes place, it occurs while the instruction concerned is being interpreted, and prior to the time of its execution. Thus the effective address is not the address stored as part of the instruction, but is the stored address increased by the corresponding R-quantity.

It is most important that the programmer bear in mind that the address of the instruction, as stored, is not changed by this incrementing process. While it is true that the result of executing such an incremented instruction is precisely the same as if the stored instruction had been changed in storage and then executed, as stated above the actual situation is that the change is made during the process of interpreting the instruction.

Whether none, one, two or all three of the possible additions takes place in the case of any particular instruction, is determined by the value the programmer assigns to the R-code for that instruction, where the R-code is a single digit punched in column 20 of the instruction card. The following table indicates the effect of each possible value of the R-code:

R-code value	Effective addresses		
0	A	B	C
1	A	B	$C + R_C$
2	A	$B + R_B$	C
3	A	$B + R_B$	$C + R_C$
4	$A + R_A$	B	C
5	$A + R_A$	B	$C + R_C$
6	$A + R_A$	$B + R_B$	C
7	$A + R_A$	$B + R_B$	$C + R_C$

Consider, for example, the following instruction:

OP_1	A	B	C
ADD	0410	0563	0741

Assume that the R-quantities stored when this instruction is encountered are:

$R_A = 0007$ $R_B = 0019$ $R_C = 0104$

Then, the following table indicates what action will take place for each possible value of R:

R	Action
0	$Q(410) + Q(563) = Q(741)$
1	$Q(410) + Q(563) = Q(845)$
2	$Q(410) + Q(582) = Q(741)$
3	$Q(410) + Q(582) = Q(845)$
4	$Q(417) + Q(563) = Q(741)$
5	$Q(417) + Q(563) = Q(845)$
6	$Q(417) + Q(582) = Q(741)$
7	$Q(417) + Q(582) = Q(845)$

Transfer and Increase Operations

SpeedCo I provides seven OP_2 operations known as transfer-and-increase operations. Typical of these is the instruction TRANSFER AND INCREASE R_C. When the program encounters this instruction, the number coded in its D address field is placed into the program counter, and the quantity R_C is increased by one.

In normal usage a transfer-and-increase instruction transfers control back to the beginning of a sequence of instructions that has just been executed, meanwhile modifying any or all of the three R-quantities. The result is that during the next execution of this same sequence of instructions, the data operated upon will be located in cells immediately adjacent to the cells that contain the data used during the first execution.

These seven operations are the following:

Name	Abbreviation
TRANSFER AND INCREASE R_C	TIC
TRANSFER AND INCREASE R_B	TIB
TRANSFER AND INCREASE R_{BC}	TIBC
TRANSFER AND INCREASE R_A	TIA
TRANSFER AND INCREASE R_{AC}	TIAC
TRANSFER AND INCREASE R_{AB}	TIAB
TRANSFER AND INCREASE R_{ABC}	TIABC

When any one of the instructions in this group is encountered, the designated R-quantity or quantities are increased by one, and control is transferred to D.

Transfer and Decrease Operations

SpeedCo I provides seven OP_2 operations known as transfer-and-decrease operations. These operations are very similar to the transfer-and-increase operations explained above. The only difference is that, as the name implies, the designated R-quantity or quantities are decreased by one each time one of these instructions is executed.

These seven operations are the following:

Name	Abbreviation
TRANSFER AND DECREASE R_C	TDC
TRANSFER AND DECREASE R_B	TDB
TRANSFER AND DECREASE R_{BC}	TDBC
TRANSFER AND DECREASE R_A	TDA
TRANSFER AND DECREASE R_{AC}	TDAC
TRANSFER AND DECREASE R_{AB}	TDAB
TRANSFER AND DECREASE R_{ABC}	TDABC

When any one of the instructions in this group is encountered, the designated R-quantity or quantities are decreased by one, and control is transferred to D. If an attempt is made to decrease any of the R-quantities below zero, an automatic error stop will occur.

Set R_A, Set R_B, Set R_C

Through the use of the transfer-and-increase and transfer-and-decrease instructions the quantities R_A, R_B and R_C can be increased or decreased one unit at a time. It is often desirable, however, to change these R-quantities by more than unity. For instance, if the program requires more than one independent use of the R-quantities, it will usually be necessary to reset them to zero after the first use and prior to the second.

The OP_2 operations SET R_A, SET R_B and SET R_C have been provided for this purpose. When any one of these instructions is encountered the designated R-quantity is replaced by the number coded in the D address field of the instruction. It is of course impossible to set any of the R-quantities to a negative value, since D is not provided with a sign.

Skip R_A, Skip R_B, Skip R_C

The normal use of a transfer-and-increase instruction forms what is called a *loop* in the program. Each time the transfer-and-increase instruction is encountered, the designated R-quantities are increased by one and control is transferred back to an earlier instruction of the program. Obviously, however, some means must be provided to stop this process after the loop has been repeated the required number of times. It is for this purpose that the OP_2 operations SKIP R_A, SKIP R_B and SKIP R_C have been provided. When any one of these instructions is encountered, the designated R-quantity is compared with the number coded in the D address field of the instruction. If these two numbers are equal, the calculator ignores the next instruction of the program entirely and proceeds directly to the interpretation and execution of the second instruction following the skip instruction. If, however, the numbers compared are unequal, this skip does not take place.

In making use of this instruction, bear in mind that the execution of the OP_1-A-B-C part of a SpeedCo I instruction always precedes the execution of the OP_2-D part of that same instruction.

Examples Involving Address Modifications

An example of the use of the above instructions is given below.

Assume that it is necessary to compute the sum of 34 quantities stored in locations 688 through 721 and to store the sum in cell 945. Assume further that cell 1013 contains the number zero and that instruction 416 is the next instruction to be executed. The sequence of instructions shown below could be used for this purpose:

LOC	OP_1	R	A	B	C	OP_2	D
0416	ADD	0	1013	1013	0945	SETRA	0000
0417	ADD	4	0688	0945	0945	SKRA	0033
0418	NOOP	0	0000	0000	0000	TIA	0417

1. Instruction 416 initializes the operation by first resetting the previous contents of 945 to zero and then resetting R_A to zero.
2. Since the R-code for instruction 417 is a 4, the effective A address for this instruction will be A+R_A. During the first execution of instruction 417, $R_A = 0$. Hence the effective A address is 688. The quantity at 688 is therefore added to zero and the sum is stored in 945. The 33 in the D field is then compared with R_A and, since they are unequal, no skip occurs. Instruction 418 then increases R_A to one and transfers control back to 417.
3. During the second execution of instruction 417, $R_A = 1$ and therefore the effective A address is $688+1 = 689$. Hence the quantity at 689 is added to the contents of 945 and the sum is stored in 945. Since $33 \neq 1$, no skip occurs and instruction 418 increases R_A to 2 and transfers control back to 417.
4. During the third execution of instruction 417, $R_A = 2$ and therefore the effective A address is $688+2 = 690$. Hence the quantity at 690 is added to the contents of 945 and the sum is stored in 945. Since $33 \neq 2$, no skip occurs and instruction 418 increases R_A to 3 and transfers control back to 417.
5. Instructions 417 and 418 thus form a loop. During the nth execution of this loop, $R_A = n - 1$ and, therefore, the effective A address is $688+(n-1)$. Hence the quantity stored in this cell is added to the contents of 945 (which at this point contains the sum of the quantities stored in the first $n - 1$ cells), and the sum is stored in 945. The 33 in the D field of instruction 417 is then compared with R_A. If these numbers are unequal, R_A is increased from $n - 1$ to n and control is transferred back to 417.
6. During the 34th execution of the loop, $R_A = 33$ and therefore the effective A address is $688+33 = 721$. Hence the quantity at 721 is added to the contents of 945 and the final result is stored in 945. The 33 in the D field of instruction 417 is then compared with R_A and since R_A now equals 33 the skip occurs, control transfers to 419 and the next phase of the calculation begins.

A more complex example of the use of these techniques is the following matrix multiplication problem.

Assume that it is necessary to postmultiply the matrix A of order 18×20 by the matrix B of order 20×16. Let the elements of A be stored in electrostatic cells 500 through 859 in the order $a_{1,1}, a_{1,2}, a_{1,3},$

$\ldots, a_{1,20}, a_{2,1}, a_{2,2}, a_{2,3}, \ldots, a_{2,20}, a_{3,1}, a_{3,2}, \ldots, a_{3,20}, \ldots, a_{18,1}, a_{18,2}, a_{18,3}, \ldots, a_{18,20}$ and let the elements of the first column of B be stored in electrostatic cells 900 through 919 in the order $b_{1,1}, b_{2,1}, b_{3,1}, \ldots, b_{20,1}$. The remaining columns of B will be assumed to have been written on tape J, each column comprising one unit record. Let the electrostatic storage cells set aside for the elements of each successive column of the product be those with addresses running from 950 through 967. Two cells of erasable storage are required; let them be 1000 and 1001. Let the contents of 1013 be zero. Assume that tape J is in read (or neutral) status and that it is stopped at the beginning-of-file gap; also that tape K is in write (or neutral) status and stopped at the same place. Finally, assume that $R_A = R_B = R_C = 0$ and that instruction 354 is the next instruction to be executed.

The following sequence of instructions could be used for this purpose:

LOC	OP_1	R	A	B	C	OP_2	D
0354	ADD	0	1013	1013	1000	—	—
0355	MPY	6	0500	0900	1001	SKRB	0020
0356	ADD	0	1000	1001	1000	TIAB	0355
0357	ADD	1	1013	1000	0950	SETRB	0000
0358	ADD	0	1013	1013	1000	SKRC	0017
0359	NOOP	0	0000	0000	0000	TIC	0355
0360	NOOP	0	0000	0000	0000	SETRC	0000
0361	WRTPK	0	0950	0967	0000	SETRA	0000
0362	RFTPJ	0	0900	0919	0000	—	—
0363	NOOP	0	0000	0000	0000	TR	0355
0364	EFTPK	0	0000	0000	0000	TR	0366
0365	SBTPJ	0	0000	0000	0000	STOP	0362

1. Instruction 354 resets 1000, the location at which each successive element of C will be accumulated.

2. Instructions 355 and 356 form a loop which evaluates the successive elements of C. In the case of $c_{1,1}$, for instance, the first execution of this loop places the product $a_{1,1}\, b_{1,1}$ into cell 1000 and changes R_A and R_B from 0 to 1. The second execution of this loop then computes $a_{1,2}\, b_{2,1}$, adds it to the contents of 1000, stores the sum in 1000 and changes R_A and R_B from 1 to 2. The third execution of this loop then computes $a_{1,3}\, b_{3,1}$, adds it to the contents of 1000, stores the sum in 1000 and changes R_A and R_B from 2 to 3. This continues until the loop has been executed 20 times. During the 20th cycle $a_{1,20}\, b_{20,1}$ is computed, added to the contents of 1000 and the sum $(c_{1,1})$ is stored in 1000. In addition R_A and R_B are changed from 19 to 20. Control then transfers back to 355 and an irrelevant multiplication of $a_{2,1}$ by the contents of cell 920 takes place. The skip R_B instruction next transfers control to instruction 357 which transfers $c_{1,1}$ from cell 1000 to cell 950 and resets R_B to zero. Instruction 358 resets cell 1000 to zero. Instruction 359 changes R_C from 0 to 1 and transfers control back to 355.

3. The loop which consists of instructions 355 and 356 is then executed 20 times forming $C_{2,1}$ in cell 1000. Instruction 357 next transfers $c_{2,1}$ from cell 1000 to cell 951 and again resets R_B to zero. Instruction 358 then resets cell 1000 to zero. Instruction 359 changes R_C from 1 to 2 and returns control to 355.

4. Since the matrix A has 18 rows, this larger loop (instructions 355 through 359) must be executed 18 times in order to compute the first column of C. During the 18th execution of this loop $R_C = 17$. Hence instruction 358 this time causes a skip to instruction 360 which resets R_C to zero. Instruction 361 then writes the completed first column of C on tape K and resets R_A to zero. Instruction 362 reads the second column of the matrix B into the same block of electrostatic cells previously occupied by the first column of B. Instruction 363 transfers control back to 355.

5. From here on the process is completely cyclic. It terminates when all 16 columns of B have been processed. At that time instruction 362 attempts to read the 17th column of B from tape J. Since there is nothing there to read, an end-of-file skip occurs which transfers control to 364. Instruction 364 writes an end-of-file gap on tape K and transfers control to 366 where the next phase of the calculation begins.

6. If during any of the executions of the instruction READ FORWARD TAPE J the recomputed check sum fails to agree with the check sum read from the tape, two instructions are skipped, thus transferring control to 365. The OP_1 part of this instruction causes the tape to backspace to the start of the record just read. The OP_2 part of this instruction being a stop-and-transfer instruction, the calculator then stops. If it is manually restarted, control will transfer to 362 and the calculator will

again attempt to read the record whose check sum failed. If the original error was a random tape reading error, the program will continue in normal fashion. If not, the calculator will stop again. In this latter event a tape writing error is indicated, and the calculations which initially wrote the successive columns of *B* on tape J should be repeated.

Address Counter

The preceding sections deal with address modifications accomplished by means of the R-quantities. As was emphasized in the above, the use of this method of address modification means that the addresses of the stored instructions are not changed, since the incrementing process takes place after the machine has begun to interpret the instruction and before it is executed.

The programmer is not limited, however, to this method of address modification. SpeedCo I also provides a counter called the *address* counter. Fixed point addition and subtraction may be carried out in this counter, the operands being addresses and address increments. It should be noted that when this method of address modification is used, the addresses of the stored instructions are changed in memory.

The address counter is capable of adding to or subtracting from its contents a number taken from the A, B, C or D address field of any SpeedCo I instruction which happens to be in electrostatic storage at the time. This counter is also capable, as a single programmed operation, of resetting itself to zero and then adding. Finally, the contents of the address counter can be stored in the A, B, C or D address field of any SpeedCo I instruction which is at that time in electrostatic storage.

The address counter may contain any integer in the range from −131,071 to +131,071. When the program controls this counter to store its contents in the A, B, C or D address field of a designated instruction, however, what is actually stored is the absolute value of the contents modulo 1024. Hence in order to determine what will be stored from the address counter at any given time, the absolute value of the contents of this counter must be reduced by the largest integral multiple of 1024 which leaves a positive remainder less than 1024. It is this remainder which will be stored. The following tabulation gives a few typical cases:

Address Counter Reads	Quantity Stored Is
+0308	+0308
−1003	+1003
+1348	+0324
−1888	+0864
+2348	+0300

The following 16 OP_2 operations control the functioning of the address counter.

Four operations cause the address counter to reset itself to zero and to add the A, B, C or D address of a designated instruction. They are:

RESET AND ADD A (RADDA)
RESET AND ADD B (RADDB)
RESET AND ADD C (RADDC)
RESET AND ADD D (RADDD)

The instruction RESET AND ADD A, for example, will cause the contents of the address counter to be replaced by the A address of that instruction whose location is given in the D field of the reset-and-add-A instruction.

Four operations cause the address counter to add to its contents the A, B, C or D address of a designated instruction. They are:

ADD A (ADDA)
ADD B (ADDB)
ADD C (ADDC)
ADD D (ADDD)

The instruction ADD B, for example, will cause the address counter to add to its contents the B address of that instruction whose location is given in the D field of the add-B instruction.

Four operations cause the address counter to subtract from its contents the A, B, C or D address of a designated instruction. They are:

SUBTRACT A (SUBA)
SUBTRACT B (SUBB)
SUBTRACT C (SUBC)
SUBTRACT D (SUBD)

The instruction SUBTRACT C, for example, will cause the address counter to subtract from its contents the C address of that instruction whose location is given in the D field of the subtract-C instruction.

Finally, four operations cause the absolute value (modulo 1024) of the contents of the address counter to be stored in the A, B, C or D address field of a designated instruction. They are:

STORE A (STA)
STORE B (STB)
STORE C (STC)
STORE D (STD)

The instruction STORE B, for example, will cause the absolute value (modulo 1024) of the contents of the address counter to be stored as the B address of that instruction whose location is given in the D field of the store-B instruction.

As an example of the use of the address counter, suppose that, starting with the execution of instruction 428, it is necessary to modify the matrix multiplication coding given above in order to permit the postmultiplication of a 12×17 matrix by a 17×13 matrix. This requires that the D address of instruction 355 be changed from 20 to 17 and that the D address of instruction 358 be changed from 17 to $12 - 1 = 11$. The following sequence of instructions would accomplish this:

LOC	OP_1	R	A	B	C	OP_2	D
0428	NOOP	0	0017	0006	0000	RADDA	0428
0429	XXXXX	X	XXXX	XXXX	XXXX	STD	0355
0430	XXXXX	X	XXXX	XXXX	XXXX	SUBB	0428
0431	XXXXX	X	XXXX	XXXX	XXXX	STD	0358

Instruction 428 replaces the previous contents of the address counter by 17, the A address of instruction 428. Instruction 429 stores this value (17) as the D address of instruction 355. Instruction 430 subtracts 6 from the contents of the address counter; the 6 being obtained from the B field of instruction 428. Instruction 431 stores the difference (11) as the D address of instruction 358.

In the above example the necessary constants are stored as the A and B addresses of the NOOP operation located at 428. However, it is usually not necessary to code special NOOP operations for this purpose. In most programs a considerable number of the instructions will have blank OP_2-D parts. Constants (such as 17 and 6 in the above example) can be coded as D addresses of such instructions, the OP_2 field in those cases being left blank. Any tape instructions (which always involve at least one irrelevant field) can also be used for this purpose.

Address modification by means of the address counter and its associated instructions may readily be used to create program loops. Here again means have been provided to terminate the repetition of such a loop after it has been executed the required number of times. The instruction which controls this termination is the OP_2 operation SKIP. When the SKIP instruction is encountered, the calculator compares the D address of the SKIP instruction with the absolute value of the contents of the address counter. If these two numbers are equal, the calculator ignores the next instruction of the program and proceeds directly to the interpretation and execution of the second instruction following the SKIP instruction. If on the other hand the two numbers are unequal, the program continues in the usual sequential fashion.

Address Modification: General

In employing address modification techniques (either the R-quantity method or the address counter method) the programmer must guard carefully against the possibility of computing a modified address that is less than 300 or greater than 1013. If for any reason the program should compute an address in the range from 0 to 299 inclusive, or from 1014 to 1023 inclusive, and should then execute the instruction of which this address is a part, the results are certain to be incorrect, since these regions of electrostatic memory are occupied by SpeedCo I control information, and not by data or instruction of the program.

Combined Use of Address Counter and R-quantities

In many cases it will be found very desirable to be able to reset and add to, add to, or subtract from the address counter one of the quantities R_A, R_B, or R_C, or, conversely, to be able to replace R_A, R_B, or R_C by the contents of the address counter. SpeedCo I does not directly provide for performing these manipulations by special logical operations, but they can be effected by judicious use of the previously described operations and the location 0025, which contains key control information associated with the R-quantities and may (only in the cases given below) be regarded as an instruction location.

LOC	OP_2	D
f	RADDX	0025

The above instruction at *f* replaces the contents of the address counter by the quantity R_x (where x = A, B or C). This operation is of particular value in case it is desired to save the number R_x for later use.

LOC	OP_2	D
g	ADDX	0025

The above instruction at g adds the quantity R_x to the contents of the address counter.

LOC	OP_2	D
h	SUBX	0025

The above instruction at h subtracts the quantity R_x from the contents of the address counter.

To perform the reverse operation of replacing one of the R-quantities by the absolute value (modulo 1024) of the contents of the address counter, one may use the following idea:

LOC	OP_2	D
f	STD	$f+1$
$f+1$	SETRX	(N)

Suppose the absolute value (modulo 1024) of the contents of the address counter is N. Thus the execution of the above instruction at f results in storing N, as indicated, in the D-field of the instruction at $f+1$, which is then executed and results in setting R_x to the value N, as required.

An example of the use of these instructions would be in a case where it is desired to replace R_A by a completely new value for a different logical use, but at the same time to save the present value of R_A. Suppose that the present value of R_A is 800, and consider the following set of instructions:

LOC	OP_2	D
0500	RADDA	0025
0501	STD	0600
0502	SETRA	0000
.	.	.
.	.	.
.	.	.
.	.	.
0600	SETRA	(0800)

The effect of the above instruction at 500 is to store the value 800 of R_A in the address counter. The instruction at 501 then stores this value 800 as the D-address in a SETRA instruction, to be used much later in the program. Instruction 502 then changes the value of R_A from 800 to 0. The program then continues, making use of this new value of R_A for perhaps a completely new logical purpose, and when instruction 600 is executed, the old value 800 of R_A is restored. Note that the address counter is also free for completely new logical functions after the execution of the instruction at 501.

CHECKING

IN ADDITION to those checking features that are an integral part of certain of the input-output instructions, SpeedCo I provides means for obtaining a general over-all check of any part of the program. This check is based upon the principle of performing the computations twice and comparing the results. The determination as to what part, if any, of the program is to be checked in this way is entirely up to the programmer. He may use this checking feature to check the entire program, or he may use it only for certain parts of the program, or he may elect not to use it at all.

This checking feature involves the use of the following OP_2 operations:

PREPARE CHECK	(PRCH)
START CHECK	(STCH)
END CHECK AND TRANSFER	(ECHTR)

At some time prior to the first use of this checking feature, or at the beginning of the program if the entire program is to be checked, the instruction PREPARE CHECK must be given. This instruction has the effect of initializing certain portions of the SpeedCo I control information. This initializing has to take place before any checking can be done. Note that the instruction PREPARE CHECK ordinarily need not be given more than once in any one program, even in those cases where the checking feature is used only intermittently, with checked portions of the program alternating with unchecked portions. For the exception to this rule, see the last paragraph of this section. Note also that the instruction PREPARE CHECK makes no use of the D address. Hence the number coded there is irrelevant.

The instructions START CHECK and END CHECK AND TRANSFER occur in pairs. A start-check instruction marks the beginning of a checking loop and the following end-check-and-transfer instruction marks the end of the same loop. The calculations included in this loop are performed twice in order to check for errors.

In detail, the procedure is as follows. The instruction START CHECK is coded as the OP_2 part of a SpeedCo I instruction. (Note that the instruction

START CHECK makes no use of the D address. Hence any number coded there is irrelevant.) When the calculator encounters this SpeedCo I instruction, the presence of the start-check operation causes the calculator not only to store the computed result for that instruction in location C, but also to reset to zero a counter called *check counter 1* and to enter the same result into this counter. From this point on in the calculation every computed result is not only stored in the desired location C in the usual way, but is also added into check counter 1. This addition into check counter 1 is not floating point addition, however. For reasons of speed and convenience this addition is carried out in fixed point fashion, each floating point number being treated as if it were two fixed point numbers, the exponents being added in along with the fractional parts. Thus while the quantity accumulated has no mathematical or physical significance, it is a function of all of the computed results which enter into its composition, and hence it may be used as a check sum.

This check summing process continues until the calculator encounters a SpeedCo I instruction which has the operation END CHECK AND TRANSFER as its OP_2 part. The presence of this end-check-and-transfer operation causes the computed result for that instruction to be the last result added into check counter 1. Control then transfers to the location specified in the D field of this instruction.

In many cases this would be the location of the preceding start-check operation. (Cases will arise where certain initializing operations will have to be performed before control can be returned to the location of the start-check instruction; these will be discussed below.) In the present case, however, the calculator transfers control back to the location of the preceding start-check operation and all of the instructions of the checking loop except for the operation PRINT are then executed a second time. At the start of this second execution, check counter 2 is reset to zero and during this second execution a check sum is accumulated in check counter 2. When the end-check-and-transfer instruction is encountered the second time, the calculator compares the contents of check counters 1 and 2. If they are equal the calculator ignores the next instructioin of the program and proceeds directly to the interpretation and execution of the second instruction following the end-check-and-transfer instruction. If there is any disagreement, however, this skip does not take place and the instruction following the end-check-and-transfer instruction is interpreted and executed in the usual way.

Consider, for example, the following short checking loop:

LOC	OP_1	R	A	B	C	OP_2	D
0379	MPY	0	$L(x)$	$L(x)$	0400	STCH	0
0380	MPY	0	$L(y)$	$L(y)$	0401	——	——
0381	ADD	0	0400	0401	0402	——	——
0382	SQRT	0	0402	0000	0403	ECHTR	0379
0383	NOOP	0	0000	0000	0000	STOP	0379

Instruction 379 computes x^2 and starts the check summing procedure in check counter 1. Instructions 380-382 compute y^2, $x^2 + y^2$ and $\sqrt{x^2 + y^2}$ and continue the check summing in check counter 1. The end-check-and-transfer operation in instruction 382 then transfers control back to 379. The same four operations are repeated, but this time the check sum accumulates in check counter 2. The second execution of instruction 382 causes the two check sums to be compared. If they agree, control skips to 384 and the next phase of the program begins. If there is any disagreement, instruction 383 is not skipped and the calculator stops. If it is restarted manually the entire process described above will be repeated.

Note that this process checks only those instructions which compute and store a result at C. Drum reading instructions, for instance, are not directly checked by this process, although if the program is so written as to consist of an unbroken series of checking loops then the drum reading instructions will be indirectly checked, since the data read will enter into calculations which produce the check sums.

In many cases the D address of the end-check-and-transfer instruction cannot be the location of the preceding start-check instruction. For example, suppose that one of the instructions of the checking loop has the OP_1 part READ FORWARD TAPE M. One of the effects of this instruction is to move the tape forward one unit record. Hence the second execution of the checking loop would not produce the same results as the first, since a different unit record would be read. The tape must therefore be backspaced before the second pass through the checking loop begins.

Certain types of address modifications also preclude the direct repetition of a checking loop. Suppose, for instance, that at the start of the first pass through the loop, $R_B = 17$ and that as a result of thirteen transfer-

and-increase-R instructions within the loop the final value of R_B is 30. Then R_B must be reset to 17 before the second pass is begun.

The above adjustments could be programmed as shown below:

LOC	OP_1	A	B	C	OP_2	D
f	XXXXX	XXXX	XXXX	XXXX	STCH	0000
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
0480	XXXXX	XXXX	XXXX	XXXX	ECHTR	0483
0481	NOOP	0000	0000	0000	STOP	0483
0482	NOOP	0000	0000	0000	TR	0486
0483	SBTPM	0000	0000	0000	——	——
0484	NOOP	0000	0000	0000	SETRB	0017
0485	NOOP	0000	0000	0000	TR	*f*

Assume that during the first execution of the loop, tape M is moved forward one unit record and R_B changes from 17 to 30. Instruction 480 then transfers control to 483. Instruction 483 backspaces tape M to its original position. Instruction 484 resets R_B to 17, its original value. Instruction 485 transfers control back to *f*, and the second pass through the loop begins. At the end of the second pass the two check sums are compared. If they agree, control skips to instruction 482 which in turn transfers control to 486 where the next phase of the program begins. If the check sums do not agree, however, the skip does not take place, instruction 481 is executed, and the calculator stops. If it is manually restarted control transfers to 483, conditions are initialized once more and the entire process is repeated.

It is important to note that, should any such initializing sequence require the performance of numerical calculations, the results of these calculations will not become part of either check sum. SpeedCo I recognizes that calculations performed after an end-check-and-transfer instruction and before a start-check instruction do not belong to the checking loop, and therefore does not add such results into either check counter.

Two simple rules which must be remembered in using the three checking operations, PREPARE CHECK, START CHECK, and END CHECK AND TRANSFER, are the following:

1. If an END CHECK AND TRANSFER is being executed, the last executed checking operation must have been a START CHECK.
2. If a START CHECK is being executed, the last executed checking operation must have been either an END CHECK AND TRANSFER or a PREPARE CHECK.

Rule 2 is particularly pertinent in case the programmer has arranged a transfer operation out of a checked loop, that is, between a START CHECK and an END CHECK AND TRANSFER. He might wish to do this, for example, on a tape end-of-file or tape error condition. In such a case it is essential to give a PREPARE CHECK before the execution of the next START CHECK.

LISTING

LISTING is the process of printing, in complete detail, all of the pertinent information concerning one or more instructions of a program while the program is being executed. The SpeedCo I listing operation prints on a single line the following information for each instruction listed, from left to right in the order shown:

1. Location of the instruction
2. Alphabetical OP_1 code
3. R-code
4. A address
5. B address
6. C address
7. Alphabetical OP_2 code
8. D address
9. L-code
10. Absolute value of contents of address counter modulo 1024
11. R_A
12. R_B
13. R_C
14. Contents of effective A
15. Contents of effective B
16. Contents of effective C

All numerical information is printed in the decimal number system.

The actual listing process will not be completely concurrent with the execution of the instruction being listed. Instead the information will gradually be stored up on a drum until a block of information corresponding to ten instructions has been accumulated. At this point the program will be temporarily interrupted and the printing will take place. As only blocks of ten lines

are listed it will sometimes be necessary to artificially continue the program beyond the natural stopping point in order to list all lines desired.

It is very important to remember that the above information listed for any one particular instruction pertains to the status of the program *after* the OP_1 of that instruction has been completely executed, but *before* the execution of the OP_2 of the instruction has been commenced.

If any item is irrelevant (for example, the contents of A, B and C in a no-OP_1 or tape or drum instruction), the field corresponding to this item will be blank.

If OP_1 of the listed instruction is SUB, the negative of the contents of B will be printed instead of the contents of B. The other exceptions are as follows:

$|Q(A)|$ and $|Q(B)|$ are listed for ABADD
$|Q(A)|$ and $-|Q(B)|$ are listed for ABSUB
$|Q(B)|$ is listed for ADDAB
$-|Q(B)|$ is listed for SUBAB
$-Q(A)$ is listed for NGMPY and NGDIV

If the exponent part of $Q(A)$, $Q(B)$ or $Q(C)$ is greater in absolute value than 236, the corresponding listed information for the value will be 0 for the fractional part and plus or minus 999, depending on whether the exponent is plus or minus, respectively.

Listing is not intended as a means of printing results, but is instead an aid in debugging new, unchecked programs.

When the complete program is being listed, the large volume of printed output required slows the calculator down to an effective speed of only about 75 instructions per minute. The listing process is selective, however. It is possible to so code the problem that, depending on sense switch settings, the information for only some of the instructions executed will be printed. This decrease in the volume of printed output will bring about a corresponding increase in the effective calculating speed during listing. No listing will ever take place for instructions being executed the second time around a checked loop (see *Checking*).

Whether the information for any particular instruction will be printed is determined by the number coded in its L field and by the position of the corresponding operator's panel sense switch. The L-code of an instruction may be 0, 3 or 6. These three codes are associated with the three sense switches labelled 1, 2, 3, respectively, on the operator's panel. When the "0" switch is in the ON position, the information for every instruction with a zero in its L field will be printed. When the "0" switch is off the information for such instructions will not be printed. The listing of instructions with a 3 or a 6 in their L fields is similarly controlled by the "3" and the "6" switches.

TIMING

THE AVERAGE and maximum times required for the complete execution of each of the SpeedCo I OP_1 and OP_2 operations, except for the elementary functions SQRT, EXP, LN, SINE and ARTAN, are listed in Figure 9. (For the execution time of the elementary functions, see the Appendix.)

The OP_1 and OP_2 times for each instruction are completely independent, so the programmer may compute the time required for the complete execution of any instruction by simply adding the OP_2 time to the OP_1 time, unless the OP_2 is no operation, in which case the complete instruction time is simply the OP_1 time. Note, however, that unlike $NOOP_2$, the OP_1 operation NOOP requires a non-zero time.

Execution Time for OP_1 (Figure 9)

Condition 1: The most recent previous instruction affecting the same tape has been WRTP( ).

Condition 2: The most recent previous instruction affecting the same tape has been RWTP( ).

Condition 3: The most recent previous instruction affecting the same tape has been RFTP( ) or SFTP( ).

Condition 4: The most recent previous instruction affecting the same tape has been RBTP( ) or SBTP( ).

Condition 5: The most recent previous instruction affecting the same tape has been EFTP( ).

Condition 6: The PRINT instruction is not being executed for the second time around a checking loop.

Condition 7: The PRINT instruction is being executed for the second time around a checking loop.

Note 1: These times must be amended if the next instruction to be executed affecting the same tape occurs sooner than a certain time. The tape unit will be disconnected from the calculator after the times

OP_1	Average Time (milliseconds)	Maximum Time (milliseconds)	Condition
ADD	4.200	19.272	
SUB	4.368	19.440	
ADDAB	4.428	19.500	
ABADD	4.656	19.728	
SUBAB	4.428	19.500	
ABSUB	4.656	19.728	
MPY	3.546	3.552	
NGMPY	3.714	3.720	
DIV	3.666	3.720	
NGDIV	3.834	3.888	
MOVE	$3.180 + (.816)(B - A + 1)$	$3.180 + (.816)(B - A + 1)$	
WRTP( )	$14.000 + (1.600)(B - A + 1)$	$19.000 + (1.600)(B - A + 1)$	1
	$904.000 + (1.600)(B - A + 1)$	$1204.000 + (1.600)(B - A + 1)$	2
RFTP( )	$14.000 + (1.600)(n)$	$19.000 + (1.600)(n)$	3
	$24.000 + (1.600)(n)$	$31.000 + (1.600)(n)$	4
	$904.000 + (1.600)(n)$	$1504.000 + (1.600)(n)$	2
	where n = number of words in the unit record.		
RBTP( )	$57.000 + (4.800)(n)$	$76.000 + (4.800)(n)$	4
	$67.000 + (4.800)(n)$	$88.000 + (4.800)(n)$	3
	$946.000 + (4.800)(n)$	$1260.000 + (4.800)(n)$	5
	where n = number of words in the unit record.		
SFTP( )	14.000 (see Note 1)	19.000 (see Note 1)	3
	24.000 (see Note 1)	31.000 (see Note 1)	4
	904.000 (see Note 1)	1504.000 (see Note 1)	2
SBTP( )	14.000 (see Note 1)	19.000 (see Note 1)	4
	24.000 (see Note 1)	31.000 (see Note 1)	3
	904.000 (see Note 1)	1204.000 (see Note 1)	5
RWTP( )	2.832 (see Note 2)	2.832 (see Note 2)	
EFTP( )	2.832 (see Note 2)	2.832 (see Note 2)	
WRDR( )	$103.072 + (5.120)(B - A + 1)$	$144.382 + (5.120)(B - A + 1)$	
RFDR( )	$53.072 + (2.560)(B - A + 1)$	$73.727 + (2.560)(B - A + 1)$	
PRINT	$1663.000 + (400.000)(I)$	$1683.000 + (400.000)(I)$	6
	where I = smallest integer greater than or equal to $(1/5)(B - A + 1)$		
	2.896	2.896	7
EJECT	3.036	3.036	
NOOP	2.736	2.736	

FIGURE 9

given, and the program will proceed, but an automatic delay will occur if the calculator is required to perform a function involving the same tape before the tape has skipped entirely over the unit record. If this delay occurs, the amount which must be added to the times given will be, in milliseconds, $(1.600)(n) - t$, where n is the number of words in the unit record being skipped and t is the time elapsed between the time the tape unit disconnects and the beginning of the execution of the next instruction affecting the same tape.

Note 2: Here also these times must be amended if the next instruction to be executed affecting the same tape occurs sooner than a certain time. The tape will be disconnected from the calculator after the times given, and the actual physical rewinding or end-of-file writing will take place simultaneously with the execution of the following instructions unless these physical functions are not completed before an instruction affecting the same tape is reached in the program. In this case there will be an automatic delay, the amount of which can be computed by using the following facts:

a) The check sum for each unit record occupies a half word of space.

b) Each word in a unit record occupies $\frac{3}{25}$ inches of tape.

c) The tape length of the unit record gap is one inch.

d) The tape length of the beginning-of-file gap is 72 inches.

e) The tape length of the end-of-file gap is 72 inches.

f) The tapes move at a rate of 75 inches per second of time.

Thus it requires $72/75 = 0.96$ seconds for the calculator to write the end-of-file gap, so if the time consumed after an EFTP( ) operation before reaching the next instruction affecting the same tape is more than 0.96 seconds, there will be no extra loss of time. Or suppose a tape file consists of 100 unit records of ten words each. The tape length for this file in inches would be $2 \cdot 72 + [100 \cdot (10 + 1/2)] \cdot 3/25 + 99 \cdot 1 = 369$; therefore the time required to rewind this file would be $369/75 = 4.920$ seconds. If the next instruction affecting the same tape occurs sooner than this time, there will be a delay until this time interval has been completed.

Execution Time for OP_2 (Figure 10)

The information in Figure 10 should be adequate to give the programmer a means of computing the machine time necessary for the execution of his program, exclusive of listing time and initial loading time.

In listing, the average and maximum times required per instruction listed are 1.325 seconds and 1.370 seconds, respectively.

The machine time necessary for the initial loading of instructions and data from punched cards to electrostatic storage, tapes, and/or drums is almost entirely a function of the number of cards to be loaded.

Only one instruction can be loaded on an instruction card but up to five floating decimal numbers can be loaded on a data card, all at the regular card reading speed of 150 cards per minute, or 0.4 seconds per card, unless a loading control card intervenes, which may stop the card reader to give the calculator time enough to load the specified data and/or instructions on the specified tape or drum. The maximum time the card reader may stop is about two seconds and should average about 0.6 seconds. (For small blocks to be loaded on tapes or drums it may not stop at all.)

The amount of machine time consumed in reading the constant deck labelled SPEEDCODING I, which must always be used with the variable program cards (see pages 18 and 27) is either about 55 seconds or about two seconds, depending upon which of the two possible constant decks is used (one deck will consist of about 130 cards and will be used in case the two drums set aside for storage of the control information which constitutes the SpeedCo I system do not already contain this information; the other deck will consist of two cards and will be used in case the two drums do already contain the required information, which should always be the case if these two drums have not been used since the last SpeedCo problem was run on the calculator).

Hence the programmer can approximate the loading time, in seconds, by the formula

$$(0.4)(n) + (0.6)(c) + e$$

where

n = total number of instruction, data, and loading control cards.

c = total number of loading control cards.

e = 2 or 55 depending upon whether the SpeedCo I system is or is not already on the two special drums, respectively.

OP_2	Time (milliseconds)
TR	0.768
TRPL	0.876 or 0.648 (for plus or minus condition, respectively)
TRMN	0.924 or 0.696 (for minus or plus condition, respectively)
TRZ	0.876 or 0.648 (for zero or non-zero condition, respectively)
SNTRP	0.864 or 0.588 (for on or off condition, respectively)
SNTRQ	0.864 or 0.588 (for on or off condition, respectively)
TIA	2.256
TIB	2.256
TIC	2.256
TIAB	2.256
TIBC	2.256
TIAC	2.256
TIABC	2.256
TDA	2.880
TDB	3.036
TDC	2.880
TDAB	3.036
TDBC	3.036
TDAC	2.880
TDABC	3.036
SETRA	2.160
SETRB	2.328
SETRC	2.280
SKRA	0.984 or 0.756 (for equal or non-equal condition, respectively)
SKRB	1.032 or 0.804 (for equal or non-equal condition, respectively)
SKRC	1.032 or 0.804 (for equal or non-equal condition, respectively)
RADDA	1.560
RADDB	1.656
RADDC	1.656
RADDD	1.320
ADDA	1.392
ADDB	1.488
ADDC	1.488
ADDD	1.152
SUBA	1.548
SUBB	1.644
SUBC	1.644
SUBD	1.152
STA	1.596
STB	1.740
STC	1.656
STD	1.368
SKIP	0.984 or 0.756 (for equal or non-equal condition, respectively)
PRCH	0.660
STCH	1.440 or 1.596 (for first or second time, respectively)
ECHTR	1.368 or 1.260 (for first or second time, respectively)
STOP	0.816
No operation	0.000

FIGURE 10

appendix

APPENDIX A

Elementary Functions — Set 1

THE NAMES and important characteristics of the presently used set (Set 1) of elementary function operations are listed below.

Square Root (SQRT)

This operation uses the usual Newton iteration technique to compute $\sqrt{Q(A)}$ to at least 9 significant decimal digits for any value of $Q(A)$ for which the absolute value of the decimal exponent part does not exceed 39,456. Any larger exponent will give completely meaningless results. If the fractional part of $Q(A)$ is negative, the value obtained from $\sqrt{Q(A)}$ will be the same as that obtained for $Q(A) = 0$, namely 0 for the fractional part and the least integer greater than or equal to one-half the exponent part of $Q(A)$ for the exponent part.

Average Time: 8.2 milliseconds
Maximum Time: 9.016 milliseconds

Sine (SINE)

This operation uses a series approximation to compute sin $Q(A)$. Since it is necessary that the argument in the series lie between $-\pi/2$ and $+\pi/2$, the original argument $Q(A)$ is first reduced to an equivalent argument lying in this interval. This reduction of angle results in a loss of significance in the argument of about n decimal places, where n = absolute value of the decimal exponent part of $Q(A)$. The series approximation itself gives an accuracy of at least 8 decimal places. Hence the final result sin $Q(A)$ is accurate to about 8 decimal places if $n < 3$, and is accurate to about $10 - n$ decimal places if $2 < n < 11$. If $10 < n < 77$, the result obtained for sin $Q(A)$ will be 0 for the fractional part and $+7$ for the binary exponent part (i.e., about $+2$ for the decimal exponent). This result is also obtained for any $Q(A)$ whose fractional part is 0. for $n > 76$, the result will generally be meaningless.

Average Time: 13 milliseconds
Maximum Time: 25.416 milliseconds

Inverse Tangent (ARTAN)

This operation uses a series approximation to compute the principal value of arctan $Q(A)$ to at least 7 decimal places for any value of $Q(A)$ for which the absolute value of the decimal exponent part of $Q(A)$ does not exceed 76. For larger values of the exponent the result will generally be meaningless.

Average Time: 13 milliseconds
Maximum Time: 27.648 milliseconds

Exponential (EXP)

This operation uses a series approximation to compute exp $[Q(A)]$ to at least 8 significant decimal digits. Since the output of the program provides only for decimal exponents up to $\pm$ 39,456, the argument $Q(A)$ should be small enough so that exp $[Q(A)] < 10^{39,456}$, which means roughly that the decimal exponent part of $Q(A)$ should not be larger than 5. The smallest allowable negative value of the input exponent is -76. For smaller negative exponents, the result will generally be meaningless.

Average Time: 13 milliseconds
Maximum Time: 14.040 milliseconds

Natural Logarithm (LN)

This operation uses a binary digit generation method to compute $\log_e Q(A)$. The highest allowable absolute value for the decimal exponent part of $Q(A)$ is 39,456. Any larger value will give completely meaningless results. If the exponent part of the result obtained is non-positive (i.e., if $|\log_e Q(A)| < 1$, which is equivalent to the condition $1/e < Q(A) < e$), the result is accurate to at least nine decimal places. If the exponent part of the result is positive, the result is accurate to at least nine significant decimal digits. If the fractional part of $Q(A)$ is 0, the result obtained is the same as if the binary fractional part of $Q(A)$ had been the value 1/2 [hence the result obtained would be approximately ($\log_e 10$) times the decimal exponent part of $Q(A)$]. If the fractional part of $Q(A)$ is negative, the result obtained is the same as if the fractional part had been positive.

Average Time: 33 milliseconds
Maximum Time: 55.836 milliseconds

APPENDIX B

Less Restricted Type of Input Data

THE FRACTIONAL part N_1 of the numbers punched on the data cards to be loaded need not be restricted to the conditions $1/10 \leqq N_1 < 1$ or $N_1 = 0$, as stated on Page 8. These conditions, which essentially say that all non-zero N_1 must not have any leading zeros, are altered to the following condition:

The total number of leading zeros for the five (or fewer) non-zero N_1 to be loaded for any card should not exceed 20.

Thus one may load a card with $N_1 = .0000\text{x}\ \text{xxxx}$ for $i = 1, 2, 3, 4, 5$, since the number of leading zeros for this card would be $4 \times 5 = 20$.

If it is attempted to load a data card containing more than a total of 20 leading zeros for the N_1 on the card, the result may be a copy check stop when the machine attempts to read the next card.

APPENDIX C

Additional OP_1-Code: Eject

TO THE group of OP_1 operations in the SpeedCo I Control System, there has been added an operation which has the alphabetic code EJECT and the numerical code 767. The execution of this OP_1 effects the ejection of the printer paper, except on the second cycle of a STCH to ECHTR checking loop in which case this OP_1 acts as an NOOP. Any OP_2 may be used in combination with this OP_1 on an instruction. The other fields of the instruction are not specially restricted, and have no effect upon the ejection. When an instruction involving the EJECT operation is listed, the fields containing the contents of the effective A, B, and C (items 14, 15 and 16 of page 38) will, as in the case of an NOOP, consist of zeros.

APPENDIX D

Suppression of Tape Rewinding

IN THE normal operation of the loading system, all four tapes J, K, L, and M are rewound prior to the reading by the card reader of any instruction cards, data cards, or loading control cards. (See Page 27.) Sense switch 6 (heretofore not used by SpeedCo I) on the operator's panel may be used to selectively control this rewinding action. If, prior to the loading of either of the two SpeedCo I constant decks (see Page 41), sense switch 6 is placed in the ON (i.e., DOWN) position, none of the four tapes will be rewound by the loading system, while, if sense switch 6 is placed in the OFF (i.e., NORMAL) position, all four tapes will be rewound as described above. This selective control thus makes it possible to retain the prior status and the prior position of each of the four tapes during loading.

APPENDIX E

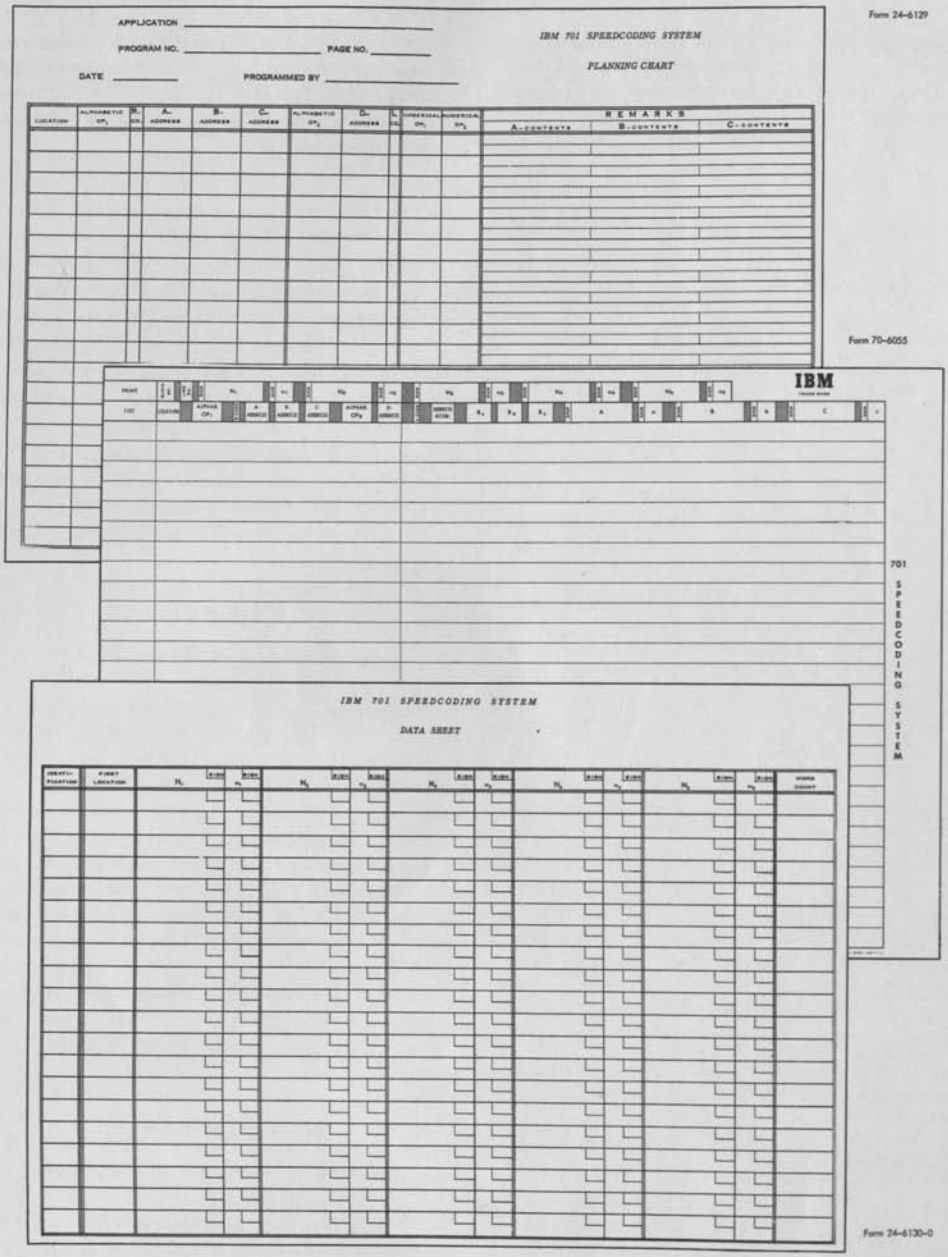
Additional OP_1-Code: Move

AN OP_1 operation has been added to the SpeedCo I Control System which has the alphabetic code MOVE and the numerical code 690. This operation makes it possible to move a block of information (which may consist of either instructions or data or both) from one set of consecutive locations to another. The address A of the instruction on which the OP_1 is MOVE specifies the first location of the block to be moved, B the last, and C the first location of the relocated block. Providing there is no overlap between the locations of the block to be moved and the locations of the relocated block, the execution of this operation does not alter the contents of the former locations. Any OP_2 operation may be used in combination with MOVE, and although the storing of the relocated block takes place prior to the execution of the OP_2, this does not prevent the successful operation of the MOVE and the associated OP_2 even if the instruction on which they occur is located within the region to be occupied by the relocated block. When an instruction involving the MOVE operation is listed, the fields containing the contents of the effective A, B, and C (items 14, 15, and 16 of Page 38) will consist of zeros.

In all cases, the operation MOVE successively moves the contents of A to C, the contents of A + 1 to C + 1 . . . . , the contents of B to C + (B − A), with the requirement that $A \leqq B$. From this information, it may be deduced that there is one special case in which the block to be moved would not be relocated in the manner previously described. This special case arises when $A < C \leqq B$. The result of executing a MOVE operation in this special case may be deduced in general from the following example. Let A = 400, B = 405, C = 402, and assume the R-code value is zero. Then, letting primes indicate the new contents

as distinguished from the old, the result of the MOVE operation is: $Q'(402) = Q(400)$, $Q'(403) = Q(401)$, $Q'(404) = Q'(402) = Q(400)$, $Q'(405) = Q'(403) = Q(401)$, $Q'(406) = Q'(404) = Q(400)$, and $Q'(407) = Q'(405) = Q(401)$.

If an instruction (or an instructional constant) is moved by the MOVE operation, the programmer should be mindful of the fact that the address parts of these instructions (or instructional constants), or of others, may need to be changed.



PLANNING CHARTS AND PRINTED FORMS FOR THE IBM SPEEDCODING SYSTEM

Form 24-6059-0 (5-54:2M-W)

Form 24-6059-0 (5-54:2M-W)