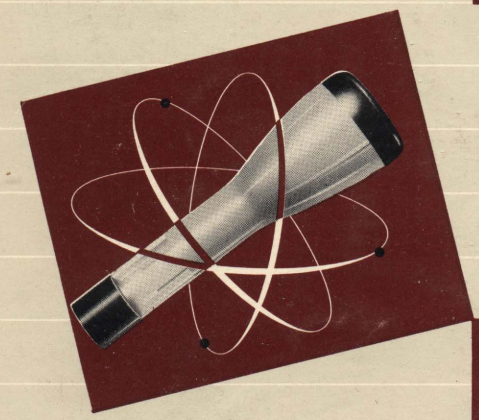
KIM COPY

PRINCIPLES OF OPERATION

TYPE 701

AND ASSOCIATED EQUIPMENT



IBM ELECTRONIC

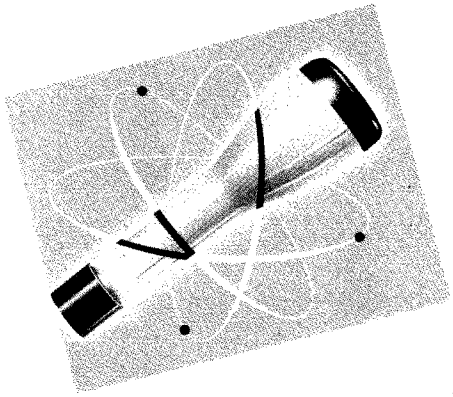
DATA PROCESSING

MACHINES

PRINCIPLES OF OPERATION

TYPE 701

AND ASSOCIATED EQUIPMENT



INTERNATIONAL BUSINESS MACHINES CORPORATION

MINOR REVISION

This edition, Form 24-6042-1, is a minor revision of the preceding edition but does not obsolete Form 22-6042-0. Principal change in this edition is:

PAGE	SUBJECT
31	An additional Logical Operation, EXTRACT, has been inserted. This moves material back so that an additional page (32a) has been provided for "break-over" of the *Sense Operations* section.

International Business Machines Corporation

590 Madison Avenue, New York 22, N. Y.

Printed in U.S.A.

Form 24-6042-1

This manual describes the operation of an installation of IBM Electronic Data Processing Machines consisting of the following units:

1 Type 701 Electronic Analytical Control Unit

2 Type 706 Electrostatic Storage Units

1 Type 711 Punched Card Reader

1 Type 716 Alphabetical Printer

1 Type 721 Punched Card Recorder

2 Type 726 Magnetic Tape Readers and Recorders

1 Type 731 Magnetic Drum Reader and Recorder

Other units, which are not listed, have to do with power supply and distribution.

foreword

Of IBM's many important contributions to the field of electronic business and scientific equipment, none has shown greater promise than the new class of equipment known as Electronic Data Processing Machines. This equipment, expanding electronics into previously untouched areas, has been made possible by drawing on IBM's tremendous reservoir of experience in electronics.

These new machines are being designed for the higher speeds and larger capacities demanded by problems of increasing complexity which confront business, industry, government and science. These problems include procurement and supply, logistics, econometrics, production control, engineering development, and scientific research.

IBM Electronic Data Processing Machines will incorporate the newest devices for input, output, and storage, including magnetic tapes, magnetic drums, and cathode-ray tubes. Individual machines will be of portable size and specialized function. Some units will serve for control, arithmetic, and logical operations; others will provide for the input, output, or storing of data.

This manual describes a representative installation of present equipment—one that is intended primarily for engineering and scientific calculations. For simplicity, the complete name of this installation, IBM Electronic Data Processing Machines Type 701 and Associated Equipment, will be abbreviated to 701.

Among the outstanding features of the 701 are its large capacity high-speed electrostatic storage, intermediate magnetic drum storage, magnetic tape units, a versatile and fast input-output system, and computing speed characterized by a multiplication time of 456 microseconds.

In order to achieve maximum versatility, every function of the machine is under control of the stored program. This versatility allows the machine to execute instructions at the rate of about 14,000 per second on typical problems. Also, functions such as input-output operation, which are determined by fixed circuitry on some computers, are under complete control of the program and, hence, under complete control of the operator. The great advantage of this system lies in the fact that a customer may build up a library of programs which will accomplish his special applications at peak machine efficiency. No compromise in efficiency is necessary in the design of the machine to accommodate an *average* application. Furthermore, a customer may efficiently calculate on *any* 701 installation simply by using his own library of programs.

contents

PART I

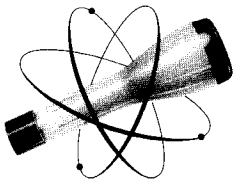
Description of Characteristics

PART II

Programming and Examples

PART I

description of characteristics



GENERAL

THE 701 is a large-scale electronic digital computer controlled by a stored program of the one-address type, and utilizing various types of internal storage.

The internal high-speed memory is on cathode-ray tubes and 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 the machine 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 a greater computing efficiency, the machine works internally in the binary-number system. The input and output, however, may be accomplished on standard IBM cards in the familiar decimal-number system by programming that does not interfere with maximum reading, punching, and printing speeds.

Results of a computation are printed on a modified Type 407 accounting machine operating at a speed of over 10,000 characters per minute. Control of the automatic tape-controlled carriage may be accomplished either manually, by control panel wiring, or by the stored program itself. Output can also take the form of cards punched in either binary or decimal; this again depends on the programming.

The programs may be written and introduced into the computer in various ways. Usually the instructions are key-punched on cards in their original form 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 a compact form. To prepare the machine for calculation the appropriate magnetic tapes are inserted in the tape units, cards are placed in the punch hopper, if necessary, and the cards containing the instructions and data of the problem are placed in the hopper of the card reader. By pushing one button the machine may be made to store the program and data of the problem and start calculating. From then on, operation of the computer is fully automatic, with all of the components being under the 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 is defined as a full word which consists of 35 bits (binary digits) and a sign, or 36 bits in all. However, any of these full words can be split into two "half words," each having 17 bits and a sign, or 18 bits in all. Since $3\frac{1}{3}$ bits are about equivalent in information content to one decimal digit, the full word has a precision of about ten decimal digits, and the half-word corresponds to about five decimal digits.

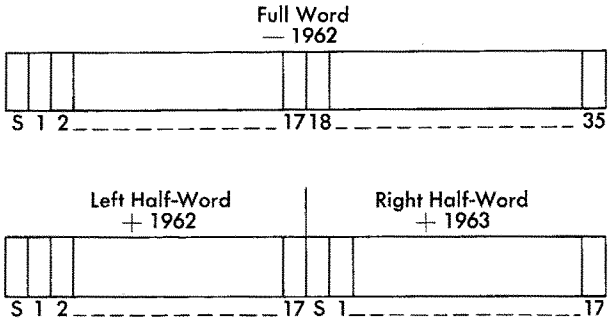
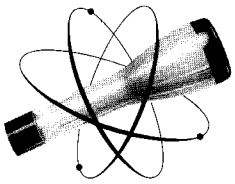


FIGURE 1

Figure 1 shows schematically a full word and the two half-words contained in the full word. The position of each of the 35 binary bits within the full word is numbered 1 through 35 from left to right. The sign bit of the number is represented on the extreme left and is labeled S. If this full word is divided between the 17th and 18th positions, the positions in the left half-word are designated exactly as in the full word. However, the 18th position of the full word now becomes the sign position of the *right* half-word, while the remaining positions are numbered 1 through 17. A word is considered negative if the binary digit 1 occupies the sign position; it is considered positive if the sign position contains the digit 0.



STORAGE

INFORMATION may be stored in electrostatic storage, on magnetic drums, on magnetic tape, and on punched cards.

The purpose of this section is simply to point out, in general terms, the types and extent of storage available on the 701. Punched cards are a well-known form of permanent storage and will receive extensive discussion in the input-output section of this manual. Details of the instructions necessary for manipulation of information contained on drums and tapes will also be found in the same section.

ELECTROSTATIC

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 screen of each tube through the presence or absence of charged spots at certain locations on the screen. In this way, a certain number of binary digits (or "bits") may be stored on each tube. One electrostatic storage unit can accommodate ~~1024~~ 2048 full words or ~~2048~~ 4096 half words. However, two such units may be used to provide a maximum storage of ~~2048~~ 4096 full words or ~~4096~~ 8192 half words. It is assumed in what follows that maximum electrostatic storage has been provided for this installation.

Principal advantages of electrostatic storage over other types is 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.

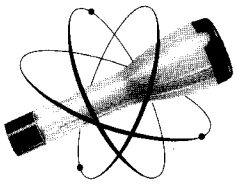
MAGNETIC DRUMS

ADDITIONAL storage capacity is provided by four 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 2048 full 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 the machine at any time. Because access to individual words on a drum is slow in relation 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 800 per second. Magnetic drums will retain stored information after the power is 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 *unit record*. The amount of information contained on each tape depends on the lengths of the individual records, since there is a certain amount of space between each record to allow for starting and stopping the tape. It is possible to store upwards of 200,000 words on each tape. The machine can read or write information on a tape only through the medium of electrostatic storage. It takes, on the average, about 10 milliseconds for the tape to accelerate to its reading or writing speed after which the reading or writing of a unit record takes places at the rate of 1250 words per second. Since the tapes are removable, a library of standard programming and mathematical tables may be kept on tapes.



ADDRESS SYSTEM

MEMORY LOCATIONS

FULL AND HALF-WORD locations in electrostatic storage, together with the tapes, drums, printer, card reader, and punch, are identified by a system of numerical addresses. By means of a number, then, we may tell the machine to refer to any information contained in electrostatic storage or to any component of the machine, provided only that we use the system to be described.

Electrostatic

The 2048 different locations for full words in electrostatic storage are identified by the negative even integers from −0000 to −4094. The 4096 possible locations for half-words in electrostatic storage are distinguished by the positive integers from +0000 to +4095. The relation between full and half-word addresses is as follows: if $-2n$ refers to a full-word location, then $+2n$ identifies the left half-word, and $+(2n+1)$ the right half-word, into which the full-word location may be split.

For example, if the full-word address is −1962, then the left half-word address is +1962 and refers to the sign position and positions 1 to 17 of the full word. The right half-word address is +1963 and refers to positions 18 to 35 of the full-word location, position 18 being the sign position of the right half word (Figure 1). If a full word is to be obtained from or supplied to electrostatic storage and, through error, a negative odd address is given (e.g., −1963), the result will be the same as if the next lower (in absolute value) negative even address (−1962) were given.

Magnetic Drums

As mentioned before, there are four magnetic drums on which information can be stored. Each drum is capable of storing 2048 full words of information. Each full word on a drum is identified by a system of addresses analogous to the system used for electrostatic memory, except that there is no provision for recognizing half-words. Thus, information must

be used or stored on a drum in units of full words. An address of −1962 may then refer to the full word stored in location 1962 in electrostatic storage or any one of the four drums. This address usually refers to a location in electrostatic storage. An address will refer to a drum location only under specific conditions. These conditions are described under *Input-Output Components.*

Magnetic Tapes

Information is recorded on magnetic tapes in blocks of full words. The size of this block of words is optional with the programmer and is limited only by the length of the tape itself. A series of these blocks is said to compose a file of information. By programming we can locate any particular full word in any unit record on any one of the four tapes. Usually, however, we are interested in obtaining a complete record, or even an entire file, at one time.

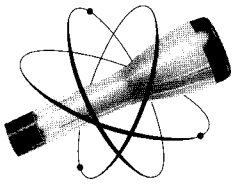
There is no way by which an address can be made to refer automatically to a particular location on tape, as was possible both in electrostatic storage and on the drums. This is because we normally use magnetic tapes to store a complete block of information at one time. If we want to refer to a particular word among a block of words, it is usually best to use the drum storage where this can easily be done. The exact method by which words are transferred to and from tape is discussed later under *Input-Output Components.*

COMPONENT IDENTIFICATION

THERE ARE four tape units, four drums, one card reader, one printer, and one card punch—all of which must be given identifying numbers. These identifying numbers are placed in the address part of an instruction whenever the programmer wants the machine to operate one of these units. Table I gives the system of addresses. Note in Table I that identifications coincide with those of certain electrostatic storage locations. Whether the address part of an instruction refers to electrostatic storage or to one of the components depends on the operation part. Some operations will make no sense if the address is interpreted as an electrostatic location; other operations make no sense if the address is interpreted as a component identification. Thus, an address is automatically interpreted by the machine in the light of what it is asked to do by the operation part of the instruction (see *Operations*).

TABLE I

	1	2	3	4
Tape Units	0256	0257	0258	0259
Drums	0128	0129	0130	0131
Printer	0512			
Card Reader	2048			
Card Punch	1024			



COMPUTING

CALCULATION is done by directing information to the computing section, causing operations to be performed on this information, and by storing the results of these operations in the memory. To understand these processes, we must first realize that the computing section is composed of three internal registers called (1) memory register, (2) accumulator register, and (3) multiplier-quotient register. Each of these registers is capable of holding a full word. Their exact capacities:

Memory register: 35 bits and sign
Accumulator register: 37 bits and sign
MQ register: 35 bits and sign

(The two extra positions of the accumulator register, called the overflow positions of this register, will be explained later; these positions are designated as P and Q.) Figure 2 shows a schematic representation of these registers.

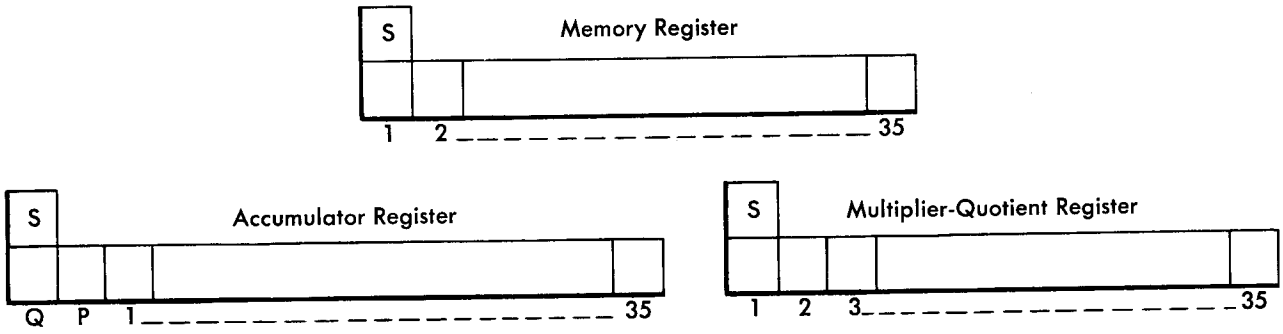


FIGURE 2

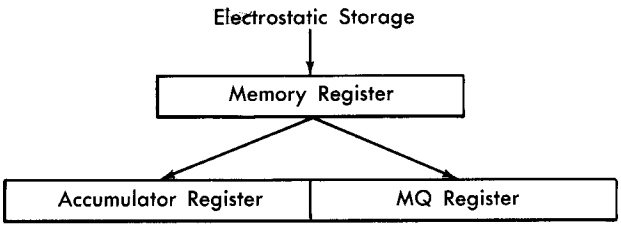


FIGURE 3

The flow of information from the electrostatic memory to these registers is shown in Figure 3. Note that all information must pass through the memory register before entering either of the other two registers. In this flow of information, three cases are to be explained (Figure 4):

1. If an instruction calls for a full word from electrostatic storage, the word first appears unchanged in the memory register before it goes to either of the other registers.

2. If an instruction calls for the left half-word of a full word, positions 1 through 17 of the full word and the associated sign are transmitted as shown. Note particularly that the least significant 18 bits of the memory register are set to zeros.

3. If an instruction calls for the right half-word of a full word, positions 19 through 35 of the full word are transmitted to the left half of the memory register. If there is a binary digit of 1 or 0 in the 18th position of the electrostatic location, the right half-word is recognized in the memory register as negative or positive, respectively. The 18 rightmost positions of the memory register are set to zero.

Once the word is in the memory register, it is a simple step for the word to be transmitted, bit for bit, to either the accumulator register or multiplier-quotient register. The programmer does not have to concern himself with the fact that the memory register is an intermediate step in the flow of words from electrostatic memory to the accumulator register or MQ (multiplier-quotient) register. It is of interest here, because at any time during machine calculation the operator can display the contents of all three registers on the operator's panel (see *Control*). For example, one instruction tells the machine to transmit a word from a given electrostatic location to the MQ register. Note that this is done with no explicit reference to the memory register.

There are also instructions that cause information to be transmitted *from* the accumulator register or

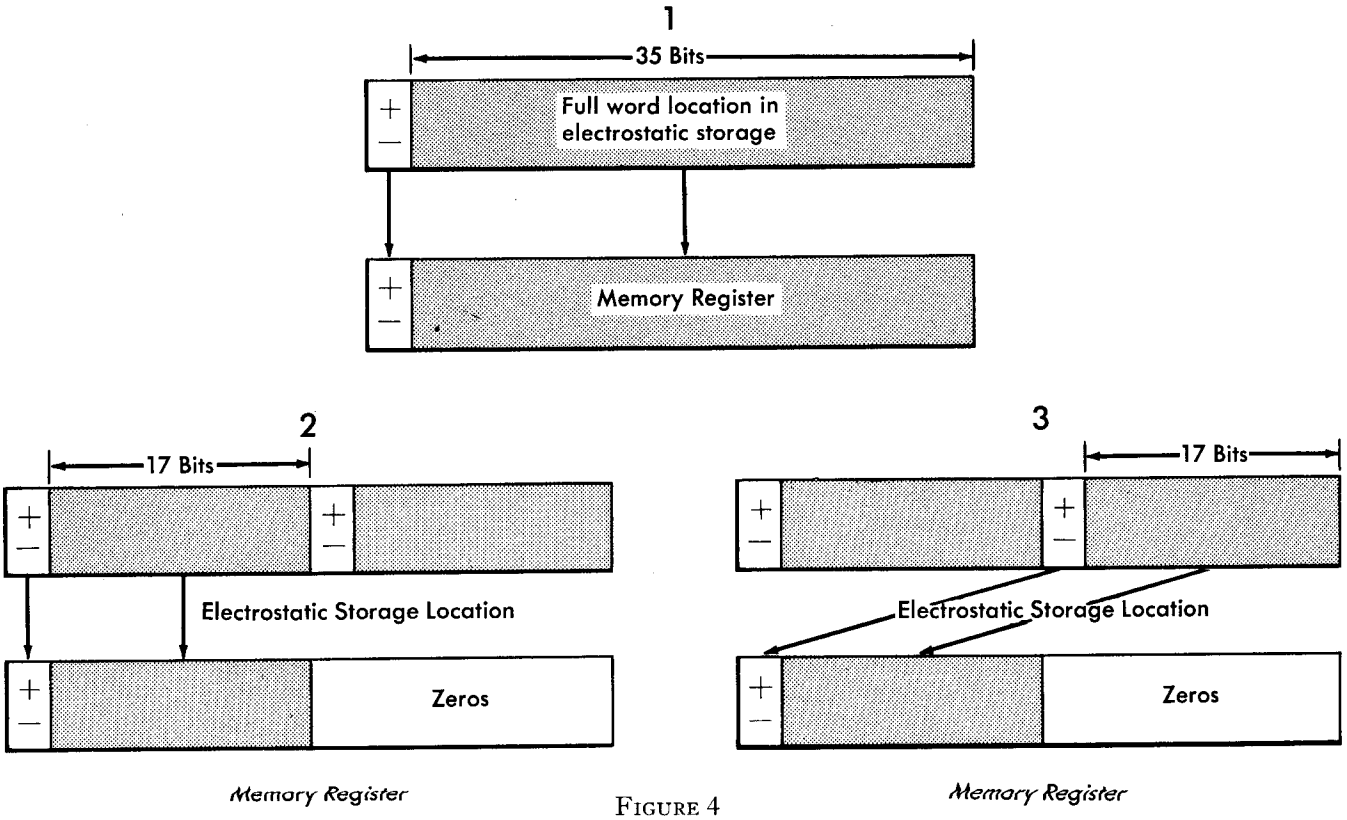


FIGURE 4

MQ register *to* an electrostatic location. In these cases, the memory register plays *no part whatsoever.*

When storing a result in a half-word location, only the sign and the first 17 bits of the accumulator register are stored. The remaining 18 bits on the right of the accumulator and the two overflow bits on the left are ignored. The same is true when storing the contents of the MQ register in a half-word location of electrostatic memory.

Before beginning a discussion of the arithmetic operations, it should be stated that all numbers in the 701 are expressed in the form of a magnitude and a sign. The results of any operation performed in the computing unit are always returned to this form. Results are never expressed as complements.

In the following paragraphs a description of possible operations will be stated in general terms. The actual methods and instructions necessary for performing these operations are explained later.

ACCUMULATION

A SCHEMATIC representation of the accumulator register was given in Figure 2. Note that there are the usual 35 positions to accommodate a full word, plus two overflow positions, P and Q. Note also that the sign is not located to the left of the bits as is done for both full and half-words in the electrostatic memory. The sign of a register is indicated separately and is schematically represented in Figure 2 by the block labeled S.

The accumulator register, together with the basic circuits for adding and subtracting, form what will be called the "accumulator." The accumulator is capable of adding a number (coming from memory via the memory register) to its contents, of subtracting an incoming number from its contents, of shifting its contents right or left, and of resetting itself to zero before entering a new word. The contents of the accumulator can also be stored in memory.

Accumulation is performed as follows:

Suppose, for example, we wish to calculate

$$A + B = C$$

where A and B may have either sign. A is first placed in the accumulator. Then B is called in from memory and is added to the contents of the accumulator. The sum, C, remains in the accumulator replacing A. Finally, C can be stored in memory or used for another operation in the accumulator.

A similar situation arises in subtraction, where the number to be subtracted is the one that comes from memory.

It is also possible to add to or subtract from the contents of the accumulator the absolute value of a number stored in memory, the sign of this number being ignored.

ROUNDING

WHEN the machine is instructed to round, the process is as follows:

> If a 1 is in the first position of the MQ register, the contents of the accumulator are increased by a 1 in the 35th position. If a zero is in the first position of the MQ register, the contents of the accumulator are unchanged. In either case the contents of the MQ register are unchanged.

Thus it will be seen how the rounding process, in conjunction with a shifting process to be described later, enables the programmer to round and truncate a number to any desired number of bits.

MULTIPLICATION

MULTIPLICATION in the 701 provides for the multiplication of two 35 bit factors to produce a maximum size product of 70 bits in one operation.

To multiply $A \times B$, we first must place the multiplier, A, in the MQ register. Then we simply call out B from memory and, at the same time, tell the machine to multiply. After the multiplication is complete, the most significant bits of the product are found in the accumulator, while the least significant bits are placed in the MQ. It should be noted that before any multiplication begins, the accumulator is automatically reset to zero. Also, the number in the MQ register is destroyed during the multiplication process. Both of these features are necessitated by the way in which the machine multiplies internally. Thus, it is seen that cumulative multiplication cannot take place in the accumulator, but the summation is very easily programmed.

Placing of the binary point in the factors is completely arbitrary. A simple familiar rule to remember with regard to placing the binary point in the resulting product follows:

RULE: Add the number of binary bits to the right of the binary point in each factor. This sum is the number of bits appearing to the right of the binary point in the product as defined above.

We can also tell the machine to multiply and round in one operation. In this case, an ordinary multiplication is followed by the rounding process. The result is a rounded product of 35 bits at most in the accumulator.

DIVISION

JUST AS multiplication may result in a 70 bit product, so division may start with a 70 bit dividend. The more significant half of the dividend is placed in the accumulator, and the less significant half is put into the MQ register. The divisor is called in from memory, and the machine is instructed to divide.

The quotient is developed in the MQ register, displacing the part of the dividend which was in that register. After the division, the accumulator contains the remainder. This remainder has the same sign as the dividend. Preservation of this remainder makes double precision division particularly convenient.

If only a full-sized 35-bit dividend is available, we may want to reset the MQ register to zero before division, because the contents of this register are considered to be part of the dividend. The magnitude of the number in the MQ register, in relation to the accumulator, however, is less than one in the 35th position of the accumulator. If this error is not tolerable, we must remember to reset the MQ register to zero before division.

These properties of the MQ register may be used conveniently to obtain a rounded quotient. With a 35-bit dividend the procedure is:

1. Place the divisor in the accumulator.
2. Shift it 36 places to the right into the MQ register, so that one-half the value of the divisor appears in the MQ register.
3. Place the dividend in the accumulator.
4. Divide.

The resultant quotient in the MQ register is then properly rounded to 35 bits. If a 70-bit dividend is to be used, the absolute value of the less-significant half must first be increased by one-half the absolute value of the divisor before shifting into the MQ register.

The effect of this procedure is the same as if a 36-bit quotient had first been developed, a 1 had been added to the 36th bit, and this bit had been dropped after any carries had been propagated.

As in multiplication, something must be said with regard to where the binary point is assumed to be in division. What will be called the "standard" case will be stated first followed by two rules necessary to determine the location of the binary point in any other case. The machine will perform division only if the divisor is larger than the dividend as defined below.

"STANDARD" CASE. Assume that the binary point of the dividend is located between the 35th position of the accumulator and the first position of the MQ register. Also assume that the divisor being called in from memory has its binary point located to the right of the 35th position. [CAUTION: *If, with the binary points assumed to be in these positions, the dividend is larger than or equal to the divisor (in absolute value), the machine will stop, and a division check light will turn on to warn the operator.*] With these assumptions, the 35-bit quotient developed in the MQ register will have its point located to the left of the first position. The remainder, if any, which is developed in the accumulator, has its binary point located between positions P and 1.

The following rules are based on changes of the binary points from the standard case.

RULE 1: A change in the binary point of the dividend results in a change equal in magnitude and in the same direction in the points of both the quotient and remainder.

RULE 2: A change in the point of the divisor results in a corresponding change in the opposite direction of the point of the quotient. The point of the remainder is unchanged.

SHIFTING

SHIFTING is a process by which the binary bits of a word may be moved to the right or left with respect to the positions of a register in the computing section. There have been references to these paragraphs in previous sections as a means of programming calculations so that the binary point may be arbitrarily located at the discretion of the programmer.

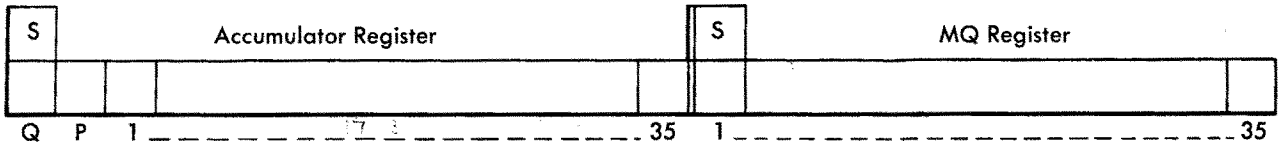


FIGURE 5

Two kinds of shifts are possible:

ACCUMULATOR SHIFT. The 37 bits that may be standing in the accumulator register can be shifted one or more places, either to the right or left. Digits shifted beyond the capacity of the register are lost. Vacated positions are filled with zeros.

LONG SHIFT. Both the multiplier-quotient register and the accumulator take part in the long shift. They behave as if the MQ register were connected to the right of the accumulator as shown schematically in Figure 5. For instance, a long right shift by 69 places causes the bit in position 1 of the accumulator to be shifted all the way over to position 35 in the MQ register, the intervening bits having dropped off the end. The process is similar for a long left shift. On a long *right* shift the sign of the MQ register is changed to the sign of the accumulator register. On a long *left* shift, the sign of the accumulator register is changed to the sign of the MQ register. A useful device is to specify a long shift of zero places which produces no actual shift but merely changes the sign of the MQ register or the accumulator as described above.

If the shift is far enough, with either the accumulator shift or the long shift, nothing will be left but zeros. Thus, shifting may be used to reset the accumulator or MQ register to zero. For convenience in certain programs, such as floating-point calculations, we can specify a shift of as many as 255 places in a single operation, although only zeros are produced by long shifts in excess of 71 places. However, no provisions have been made for permitting shifts by more than 255 places. An attempted shift of more than 255 places gives the results described under *Operations*.

It should be noted that the overflow positions of the accumulator participate in both kinds of shift. The signs of the registers, however, do not participate in the shift except as specifically noted above.

SIGN OF ZERO

IT IS POSSIBLE for a zero in this calculator to have either a plus or a minus sign. For instance, a negative number in the accumulator may be shifted so far to the right or left that all numerical bits are zeros. This still leaves the minus sign, so that technically the result is "−0". Arithmetic operations may also result in zeros of either sign. The arithmetic circuits are designed so that if the result of an addition or subtraction is zero, the sign of the result will be that of the number which was in the accumulator immediately before the addition or subtraction took place. In numerical work no distinction need be made between a +0 and a −0 result, because either zero can be used in further arithmetic operations.

This characteristic of the machine is sometimes very convenient; there are ways the machine can be made to recognize either type of zero. It is also possible for the machine to ignore the sign entirely and test to see only if the result is zero. Such controls are discussed in a following section.

OVERFLOW INDICATION

THIS SECTION covers the accumulator overflow positions, P and Q.

During such operations as adding, subtracting, and shifting left, it is possible for non-zero binary bits to enter into or be shifted completely through the overflow positions of the accumulator. This can happen by means of a left shift or as the result of a carry in addition or subtraction. Whenever a non-zero binary bit enters position P from position 1, an overflow indicator within the machine is turned on. Associated with the activation of this indicator is an overflow light on the operator's panel. This overflow indication occurs even in a shift that sends a binary bit completely through the overflow positions. Overflow may indicate an error in setting up the program for a given set of data. Frequently, however, operations are planned deliberately to produce overflow. Hence,

the machine will continue to operate after an overflow, but an instruction is available to test the condition of the overflow indicator and to program the desired action after an overflow. This may include stopping the machine on overflow, performing a special set of operations, or simply ignoring overflow. Testing the overflow indicator turns it off.

Examples of the operation of the overflow indicator follow. In these examples we assume the binary point to be to the left of position 1. For convenience and abbreviation only the first three bits to the right of the binary point are shown.

In the addition

$$\underbrace{+.100}_{\text{WORD FROM MEMORY}} \quad \underbrace{+\overset{\text{OVERFLOW POSITIONS}}{\overbrace{00}}.110}_{\text{ACCUMULATOR}} = \underbrace{+01.010}_{\text{ACCUMULATOR}}$$

there is a carry that produces an overflow and causes the overflow light to go on. If now the indicator is turned off (by use of the test instruction) and a second addition

$$+.100 + 01.010 = +01.110$$

is performed, the overflow indicator does not come on again, because there was no carry past the binary point this time. A further addition

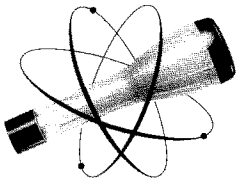
$$+.100 + 01.110 = +10.010$$

again gives an overflow indication, because a carry was propagated through the binary point.

If shifts or carries go beyond Q, the excess bits are dropped. Thus +00.010 shifted left by two places in the accumulator gives +01.000 with an overflow. But if +00.010 is shifted left by four places, the result is +00.000, again with an overflow indication. In the first case, the process may be reversed by shifting right, but in the second case, bit 1 has been lost, and any shifting to the right still gives a zero result.

The extra two bits in the accumulator enable the programmer to make full use of the 35 numerical bits in memory for such operations as double-precision arithmetic. When a possible overflow must be allowed for in a program, the extra bits make it considerably easier to shift and return the result to a standard form.

The overflow bits do not enter into multiplication, because the product of two 35-bit numbers cannot exceed 70 bits. Neither can they be part of a dividend, because the dividend cannot be greater than the divisor as defined earlier in the "standard" case.



CONTROL

STORED PROGRAM

COMPLETELY AUTOMATIC and flexible control of the calculator requires use of a stored program. The procedure generally used by 701 programmers with the stored program system is as follows:

1. The mathematician analyzes his problem and breaks its solution down into basic steps of which the 701 is capable.

2. By means of a number code, determined by the design of the computer, he translates these steps into a numerical form which can be interpreted by the machine. Each of these steps is then stored in the electrostatic memory. Each step, which later will be seen to consist of an operation part and an address part, will hereafter be referred to as an instruction to the machine.

3. Data necessary for solution of the problem are stored in the memory of the machine.

4. By means of a control, the programmer tells the machine in which memory location he has stored the first instruction to be executed. After receiving this information, the 701 is able to find all succeeding instructions and execute them automatically.

A complete analysis of the instruction system of the 701 follows.

Instruction Sequencer

The numerical representation of an instruction to the machine occupies the space of a half-word in the electrostatic memory. Instructions may temporarily be stored on drum, tape, or cards, 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, to produce a particular result. The instructions are ordinarily introduced into consecutively-numbered half-word locations of memory in the order in which they were written. The reasons for this follow.

Each time an operation is to be performed, the machine looks up the instruction in the 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 a unit known as the *instruction sequencer*. This unit contains a

counter known as the *instruction counter,* which contains the address of the instruction currently being executed. After each execution, the number in this counter is automatically increased by 1. Consequently, the machine will automatically take its next instruction from a location whose half-word address is one higher than the address 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 to be explained below. By means of these operations, any half-word location in electrostatic storage can be designated as the source of the next instruction. The address of this location is placed in the instruction counter by the transfer instruction; thereafter, execution of the program again proceeds sequentially.

An important observation with regard to this 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 in which they are interpreted by the machine. If for any reason the address of a half-word of data is entered into the program counter, the data will be interpreted as an instruction. Conversely, an instruction may be caused to enter the computing unit just as data are caused to enter the computing unit. Thus, one instruction may call for the modification of another instruction by directing the machine to compute a new address part or to substitute another operation part. One program may operate on itself and compute one of its own instructions. A program may choose between several alternatives, depending on results obtained in the course of the problem. The ability of the machine to modify and to relocate instructions at high speed lends great flexibility to its operation and enlarges the scope of its application.

Instruction Layout

Each operation the machine can execute—including arithmetic operations, shifting, rounding, reading, writing, and others—is assigned a numerical code. An operation in conjunction with the address of an appropriate operand constitutes an *instruction* and is written and stored as a single binary number. The two components of an instruction are referred to as the *operation part* and the *address part.* A schematic diagram of an instruction is shown in Figure 6.

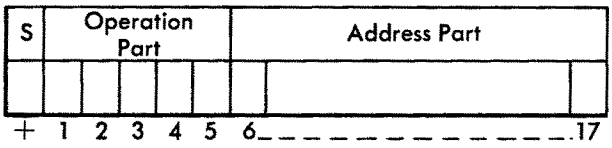


FIGURE 6

Note that the 17 bits of information and associated sign require exactly a half-word of storage space.

The operation part of an instruction determines the nature of the operation the machine is to execute. The numerical code for the operation is located in positions 1 through 5 of the instruction itself. The 701 is capable of performing exactly 32 distinct operations divided into four categories: (1) arithmetic operations, (2) logical operations, (3) input-output operations, (4) sense operations. These operations will be explained in detail later.

The address part of an instruction specifies a number that usually refers to a location in electrostatic storage. In such cases, the address will be a positive or negative integer as defined previously under *Address System.* The sign of the integer in positions 6 through 17 of the instruction is determined by the sign of the instruction itself. Thus, the sign of an instruction applies only to the address part. For some instructions, however, the address part designates a certain input-output unit; with shift instructions it indicates the number of places to be shifted. But the name "address part" for the 12 rightmost bits of an instruction is retained, although these bits do not always represent an address in the true sense of the word. It will be seen later that the sign of some instructions is immaterial.

The machine interprets the numbers in the operation part and the address part as integers. In other words, the binary point of the binary digits occupying positions 1 through 5 is considered to be immediately to the right of position 5. Similarly, the binary point of the digits occupying the address part is considered to be immediately to the right of position 17.

For example, an instruction which designates 05 as the operation and −0013 as the address, will actually look in binary form as follows:

OPERATION PART ADDRESS PART

−00101000000001101

OPERATOR'S PANEL

THE VARIOUS buttons, keys, switches, and signal lights by means of which the operator can control and communicate with the machine are assembled to form the operator's panel. The only control panels on the machine which require wiring are those for the card reader, printer, and card punch. These panels are discussed later in a separate section.

Description of the Panel

Components of the operator's panel are listed below with a brief explanation of their functions. Some of these functions will not be completely understood without reading later sections.

Power-On Button. Turns on the power for the entire calculator and automatically performs the resetting functions of the reset and clear-memory button.

Power-Off Button. Turns off the power for the entire calculator immediately, but leaves the blowers on for 10 minutes.

DC-Off Button. Turns off the direct-current power for servicing the machine.

DC-On Button. Turns the direct-current power on after it has been turned off with the DC-off button.

Power-On Light. Indicates that the main AC power is on.

Ready Light. Indicates that the AC and DC power is on and that the calculator is stopped but is ready to run. When the power is turned on, there is a delay before the ready light comes on.

Operating Light. Indicates that the calculator is running. When the calculator is started, the operating light goes on and the ready light goes off. When the calculator stops, the operating light goes off and the ready light goes on, if the power is still on.

Automatic-Manual Switch. When this switch is set to automatic, the calculator may be operated at full speed. When it is set to manual, the calculator may be operated manually by means of the half-step and multiple-step keys and the enter MQ, enter instruction, and memory display buttons; the load button is inoperative, and the start button cannot start the calculator. The calculator cannot be advanced manually while the stored program is using any input-output device.

Manual Light. Goes on when the automatic-manual switch is set to manual.

Start Button. If the automatic-manual switch is set to automatic, the start button resets the various machine interlocks and then starts the calculator. The program begins with the instruction whose address is contained in the instruction counter.

Reset Button. Resets the accumulator, MQ and memory registers, the instruction counter as well as certain internal input-output and check interlocks which are not discussed in this manual. It does not affect electrostatic storage.

Reset and Clear-Memory Button. Changes every bit in electrostatic storage to a 1 and, in addition, performs certain resetting functions not discussed in this manual.

Register Lights. Groups of small neon lights that indicate the contents of the following registers:

Memory register
Accumulator register
MQ register
Instruction counter
Instruction register
- (a) Sign register
- (b) Operation-part register
- (c) Address-part register

A light being on indicates a binary digit of 1 located in that position of the register. A light being off indicates a zero.

MQ-Entry Keys. These 18 keys are used to set up a half-word for manual entry into the MQ register. Depressing a key represents a binary 1.

Enter-MQ Button. If the automatic-manual switch is set to MANUAL, the enter-MQ button enters the half-word set up on the 18 MQ-entry keys in the left 18 positions of the MQ register. The rightmost 18 positions of the MQ register are reset to zero.

Instruction-Entry Keys. There are 18 instruction entry keys, consisting of a sign-entry key, five operation-part entry keys, and twelve address-part entry keys. They are used to set up instructions for manual entry of an instruction into the control section of the calculator.

Enter-Instruction Button. If the automatic-manual switch is set to manual, the enter instruction button enters the instruction set up on the instruction entry keys into the instruction register and causes this instruction to be executed. The calculator then stops. Instructions, pertaining to input-output devices which must follow each other at high speed, cannot be executed manually with the enter instruction button. If the stored program calls on an input-output device which must receive its instructions in rapid succession, the calculator will automatically go into high-speed operation until this section of the program is completed.

Memory-Display Button. If the automatic-manual switch is set to manual, the memory-display button causes the full word stored at the address set up on the address part of the instruction-entry keys to be displayed on the memory-register lights. Only full words are displayed in this way.

Half-Step Key. If the automatic-manual switch is set to manual, the half-step key advances the program one half-step at a time, provided the machine interlocks do not prevent the advance of the program. Half-steps are of two kinds: interpretation half-step, during which an instruction is interpreted, and execution half-step, during which the instruction is executed. Repeatedly pressing the half-step key causes the calculator to alternate between interpreting and executing an instruction. If a READ or WRITE instruction is executed, the calculator will go into automatic operation until the stored program is through using the particular input-output unit selected by the READ or WRITE instruction.

Multiple-Step Key. Holding down the multiple-step key is equivalent to pressing the half-step key repeatedly about ten times a second. Releasing the multiple-step key stops this action.

Machine Cycle Button. Advances the program one machine cycle at a time. It is intended only for servicing the machine and is not used by the operator. The half-step or multiple-step keys should be used to advance a program manually.

Load Selector Switch. Selects either the card reader, or the first tape unit (address 0256), or the first drum unit (address 0128), from which a unit record is to be read by means of the load button.

Load Button. If the automatic-manual switch is set to automatic, the load button initiates the reading of a unit record from the input unit selected by the load selector switch. It causes the first full word of the unit record to be read and to be stored at the address set up on the address-part entry keys. The calculator then starts automatically, using as the first instruction the left half-word at the same address. Pressing the load button is in effect the same as giving the following instructions:

READ	(Address of input-output unit specified by load selector)
SET DR 0000	(Relevant for drum only)
COPY	(Address set up on address-part entry keys)
TR	(Address set up on address-part entry keys)

Thus it is seen that the calculator will go into automatic operation starting with the instruction located at the address set up on the instruction entry keys.

Sense-Input Switches. There are 6 two-position sense-input switches, identified by addresses 0069 to 0074. They can be sensed by means of a SENSE instruction with the corresponding address part and used to cause the calculator to skip an instruction.

Sense-Output Lights. These four lights may be turned on individually by means of SENSE instructions, as explained above. Another SENSE instruction turns all four off together. The lights are used by the programmer to indicate the progress of a problem and to signal various conditions.

Instruction-Time Light. For half-step operation this light indicates that the calculator is ready to perform the next interpretation half-step.

Execution-Time Light. For half-step operation this light indicates that the calculator is ready to perform the next execution half-step.

Overflow Light. Turns on when the overflow indicator turns on. It is turned off by execution of a TR OV instruction or by the reset and clear-memory button.

Input-Output Light. Indicates that one of the input-output units is selected by the calculator.

Program-Stop Light. Indicates that the calculator has stopped as a result of executing a STOP instruction. It is reset by the start button, or the reset button, or the reset and clear-memory button.

Copy-Check Light. Indicates that the calculator has stopped because a COPY instruction was given at the wrong time. It is reset by the start button, or the reset button, or the reset and clear-memory button.

Tape-Check Light. Indicates that the calculator has stopped because of a discrepancy in the tape group count or the redundancy check while a tape was being read. It is reset by the start button, or the reset button, or the reset and clear-memory button.

Divide-Check Light. Indicates that the calculator has stopped because the dividend is not less than the divisor. It is reset by the start button or the reset button, or the reset and clear-memory button.

Calculator-Fuse Light. Indicates that a fuse for the main calculator unit, or electrostatic storage, or a tape or drum unit has burned out. After replacement of the fuse, the light is reset by the start button or the reset button, or the reset and clear-memory button.

Input-Output Fuse Light. Indicates that a fuse has burned out in the card reader, punch, or printer. After replacement of the fuse, the light is reset by the start button or the reset button, or the reset and clear-memory button.

Basic Manipulations

Entering Information into Storage

Information may be entered manually into electrostatic storage from the operator's panel one half-word at a time. The procedure:

1. Set up the automatic-manual switch to manual.
2. Set up the half-word on the MQ-entry keys.
3. Set up the instruction + STORE MQ XXXX on the instruction entry keys, where XXXX represents the address (in binary form) at which the half-word is to be stored.
4. Press the enter-MQ button to enter the half-word into the MQ register.
5. Press the enter-instruction button to execute the instruction (set up under 3), thus storing the half-word in memory.
6. Press the memory-display button to check that the half-word has been stored correctly.

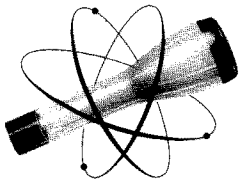
A full word can be entered into electrostatic storage from the operator's panel only by splitting it into two half-words and entering each separately.

Starting Machine with a Given Instruction

Assume the operator wants the machine to begin calculating with an instruction located in address XXXX of electrostatic memory.

1. Set automatic-manual switch to manual.
2. Set up the instruction TRANSFER XXXX on the instruction entry keys.
3. Press the enter-instruction button.
4. Set automatic-manual switch to automatic.
5. Press the start button.

If there is a program stored in memory, the machine will start calculating with the instruction located at XXXX and continue automatically.



OPERATIONS

ALL OPERATIONS of which the machine is capable will be explained in this section. For the sake of reference purposes and completeness, some operations will be discussed that will not attain their full significance until later sections are studied.

Operations are grouped under the general classifications of arithmetic operations, logical operations, input-output operations, and sense operations. Associated with each individual operation is a heading for its explanatory paragraphs. An example of such a heading:

Reset and Add
R ADD
10–1, 2, 3, 4

This heading has the following meaning: first is the name of the operation; the term in the second line is the abbreviation commonly used for the operation; the numerical code for the operation then appears on the third line followed by a dash and numbers of the examples in Part II that illustrate the operation by means of a simple program. If the word "none" appears after the numerical code, it means that the operation is either explained fully in other sections or needs no further explanation.

Moreover, each operation is assumed to refer to the address part x of some instruction, where x includes the sign of the instruction. The contents of electrostatic location x are *never changed* except as specifically stated. A half-word comes from memory into the memory register, and finally into the other registers, as described under *Computing*.

A complete list of all 32 operations, together with their numerical code, is given in *Summary of Machine Characteristics*.

ARITHMETIC OPERATIONS

OPERATIONS are grouped in this classification if they take place in the computing section. They include the ordinary arithmetic functions, shift instructions and store instructions.

Reset and Add
R ADD
10–1, 2, 3, 4
Contents of the accumulator register are replaced by contents of electrostatic location x.

Add
ADD
09–1, 2, 4, 10
The number in electrostatic location x is added to the number in the accumulator register.

Add Absolute Value
ADD AB
11–3
The absolute value of the number in electrostatic location x is added to the contents of the accumulator.

Reset and Subtract
R SUB
06–2, 12
Contents of the accumulator register are replaced by the negative of the contents of electrostatic location x.

Subtract
SUB
05–2, 11, 12
The number in electrostatic location x is subtracted from the number in the accumulator register.

Subtract Absolute Value
SUB AB
07–3
The absolute value of the number in electrostatic location x is subtracted from the number in the accumulator.

Store
STORE
12–1, 2, 3
The contents of electrostatic location x are replaced by the contents of the accumulator register, exclusive of the two overflow positions. When a full-word location is specified by the address x, the sign and the 35 bits to the right of the overflow positions of the accumulator are stored in the memory location. When a half-word address is specified, the sign and the bits in positions 1 through 17 of the accumulator are stored in the half-word memory location. The contents of the accumulator are not changed.

Store Number in MQ Register
STORE MQ
14–7
The contents of electrostatic location x are replaced by the contents of the MQ register. When a full-word address is specified, the entire contents of the MQ register, including sign, are stored in the memory location. When a half-word address is given, the leftmost 17 bits in the MQ register and the sign are stored in the half-word memory location. Contents of the MQ register are unchanged.

Load MQ Register
LOAD MQ
15–5, 6, 8, 11
Contents of the MQ register are replaced by contents of electrostatic location x.

Round
ROUND
19–3, 6, 8
The magnitude of the number in the accumulator register is increased by a one in position 35 if the bit in position 1 of the MQ register is a one. Otherwise the contents of the accumulator register are unchanged. In any case, the MQ register remains unchanged.

Multiply
MPY
16–6
The accumulator register is first reset to zero. Then the number in electrostatic location x is multiplied by the number in the MQ register. The most significant 35 bits of the product are produced in the accumulator; the least significant 35 bits are produced in the MQ register. Both registers have the sign of the product as determined by the two factors. The multiplicand, from the specified memory location, may be either a full-word or a half-word. If a half-word is specified, the instruction operates as though it were a full-word with 18 zeros on the right. The MQ register is always considered to contain a full-word, even though a half-word had been previously placed in it. Thus, if a half-word had been put into the MQ register and MULTIPLY were given with a half-word address, the product would have 34 significant bits and would appear in bit positions 1-34 of the accumulator. The remaining bit position 35 of the accumulator and the entire MQ register would be filled with zeros. When the product of a negative number by a positive number is such that either the accumulator part or MQ part result in all zeros, this zero will have a negative sign associated with it. Otherwise, a positive zero results.

Multiply and Round
MPY ROUND
17–5
This instruction produces the same result as a MPY instruction followed by a ROUND instruction.

Divide
DIV
18–7, 8
The possible 70-bit dividend placed in the accumulator and MQ registers is divided by the number contained in electrostatic location x. The quotient with proper sign appears in the MQ register. The remainder, if any, is developed in the accumulator and has the same sign as the dividend. The machine treats that part of the dividend placed in the MQ register as if it had the same sign as the accumulator. Half-word divisors behave as full-words with the rightmost 18 bits zero. If the divisor is not greater than the dividend, as previously defined, the machine stops, and the DIV CHECK light is turned on at the operator's panel.

NOTE: In all of the shift instructions that follow, the address x specifies the number of places to be shifted. The machine will recognize any number from 0 to 255 inclusive for this shift. If x is greater than 255, the actual number of places shifted is the difference between the given number, and the nearest integral multiple of 256 which is less than or equal to the given number. In other words, the number of places

shifted is x, modulo 256. For example, if $x = 258$, the extent of the shift will be two places. If $x = 512$, this is equivalent to giving the shift instruction with $x = 0$. Shifts up to 255 are permitted for convenience in certain uses of the machine such as floating-point programs.

Accumulator Right Shift
A RIGHT
23–4, 11
The contents of the accumulator register (not including the sign) are shifted right by x places. Emptied positions are filled with zeros. Bits shifted past position 35 are lost. Both overflow positions participate in the shift. (See note above.)

Accumulator Left Shift
A LEFT
22–1, 4, 8
The contents of the accumulator register (not including the sign) are shifted left by x places. Emptied positions are filled with zeros. If a non-zero bit is shifted from position 1 into position P at any time during the shifting operation, the overflow indicator is turned on. Bits shifted beyond the leftmost overflow position are dropped. (See note above.)

Long Right Shift
L RIGHT
21–3, 7, 11
The contents of the accumulator and the MQ register are considered as one number (not including signs) and are shifted right by x places. Bits shifted out of the accumulator appear at the left of the MQ register. For example, a bit initially occupying position 35 of the accumulator finally occupies position 1 of the MQ register on a long right shift of one. The sign of the MQ register is changed to agree with the sign of the accumulator register. (See note above.)

Long Left Shift
L LEFT
20–6, 8, 11
The contents of the accumulator and the MQ register (not including signs) are shifted left by x places. Bits shifted out of the MQ register appear at the right of the accumulator register in the opposite manner as described under *Long Right Shift* above. The sign of the accumulator register is changed to agree with the sign of the MQ register. If a non-zero bit is shifted from position 1 to position P of the accumulator register at any time during the shifting operation, the overflow indicator is turned on. (See note above.)

LOGICAL OPERATIONS

ANY OPERATION which may be used to alter selectively the course of a program is considered under this category. The address x, as before, is assumed to contain the sign of the instruction. The sign of an instruction in this group is entirely immaterial unless specifically mentioned.

Transfer
TR
01–3, 9, 11
This instruction causes the machine to take its *next* instruction from electrostatic location x. The program then proceeds sequentially from this new instruction.

Transfer on Plus
TR +
03–9, 12
If the sign of the contents in the accumulator register is plus, a transfer takes place exactly as described under *Transfer* above. If the sign is minus, the transfer is not executed and the program continues sequentially. The transfer is determined only by the sign of the word in the accumulator. Thus, "plus zero" is regarded as a positive word, and "minus zero" is regarded as negative.

Transfer on Zero
TR 0
04–9, 10, 11
If the contents of the accumulator are zero (including the overflow positions) a transfer is effected exactly as described under *Transfer* above. The sign of the accumulator has no significance. If, however, there is a binary 1 in any position of the accumulator, the transfer does not take place, and the program continues sequentially.

Transfer on Overflow
TR OV
02–3
If the overflow indicator is on as the result of a previous operation, a transfer of control takes place exactly as indicated under *Transfer* above, and the overflow indicator is reset to the off position. If, however, the overflow indicator is not on, no transfer is effected, and the program continues in its normal sequence.

Stop and Transfer
STOP
00–none
The calculator stops when this instruction is received. When the start button on the operator's panel is pressed, the calculator will start again, beginning with the instruction at electrostatic location x and will continue sequentially.

Store Address
STORE A
13–10, 12
The rightmost 12 bits of the half-word at electrostatic location x are replaced by the bits in position 6 through 17 of the accumulator register. Note that these positions

represent the address part of an instruction. The remaining bits (including the sign) at location x, are unchanged. The sign of this instruction must be positive.

Extract
EXTR
−13–none

The sign of this instruction must be minus. The extractor or "mask" is located anywhere in memory. The word from which we wish to extract any combination of bits is located in the accumulator. The EXTRACT order is then given. The extraction process (or logical multiplication) proceeds from the accumulator into the memory location specified by the address part of the extract instruction. At the completion of this operation, the result of the extraction has replaced the word in memory specified by the address part of the extract instruction.

In its simplest form, the extract operation can be understood by considering four cases. In each case assume corresponding bit positions in the accumulator and in memory.

Case	*Condition*	*Result*
I	ZERO in accumulator ZERO in memory	ZERO in accumulator ZERO in memory
II	ZERO in accumulator ONE in memory	ZERO in accumulator ZERO in memory
III	ONE in accumulator ZERO in memory	ONE in accumulator ZERO in memory
IV	ONE in accumulator ONE in memory	ONE in accumulator ONE in memory

From the above cases, note that the word in the accumulator remains unchanged, while the word in memory may change. The result of the extraction, therefore, is a word that has ones where both words have ones and zeros in every remaining position.

The EXTRACT order affects all 36 bits of a word; that is, the sign bit is treated in the same manner as any of the other bits.

The time for obtaining and executing the extract operation is the same as that for any STORE-type instruction—i.e., 60 microseconds, unless one of the previous 12 instructions was a multiplication, in which case the extract order will require 24 microseconds.

NOTE: By means of the conditional transfers described above, it is possible to program a "transfer on minus" and "transfer on non-zero." (See Example 9 in Part II.)

INPUT-OUTPUT OPERATIONS

THE OPERATIONS that follow will not be completely understood until the section on *Input-Output Components* has been studied. In fact, the *Operations* section and the *Input-Output Components* section complement each other. The sign of these operations is immaterial unless specifically stated otherwise.

Prepare to Read
READ
24–none

This instruction causes the calculator to prepare to read one unit record of information from an input mechanism (component) designated by the address x. If the address x identifies a tape unit, the MQ register is reset to zero. A selected tape unit must be in *read* or *neutral* status if the instruction is to be effective. (See *Magnetic Tapes* in next section.)

Prepare to Read Backward
READ B
25–none

This instruction causes the calculator to prepare to read one unit record of information from a tape unit in the backward direction. Thus, this instruction makes sense only if x refers to a tape unit. The MQ register is reset to zero as in the READ instruction. For the instruction to be effective, the selected tape unit must be in *read* or *neutral* status. (See *Magnetic Tapes* in next section.)

Prepare to Write
WRITE
26–none

This instruction causes the calculator to prepare to write one unit record of information on an output mechanism (component) designated by x. A tape unit must be in *write* or *neutral* status for this instruction to be effective. (See *Magnetic Tapes* in next section.)

Write End of File
WRITE EF
27–none

This instruction causes the machine to clear a section of the magnetic tape unit designated by x. This "gap" is then recognized, in any subsequent reading, as an end-of-file gap and may be used to control the program. The instruction is effective only if the selected tape unit is in a *write* or *neutral* status. (See *Magnetic Tapes* in next section.)

Rewind Tape
REWIND
28–none

The tape unit identified by x is rewound to its starting point, regardless of the tape status. The rewind, however, causes the tape unit to be placed in *neutral* status. (See *Magnetic Tapes* in next section.)

Set Drum Address
SET DR
29–none

The address x in this instruction specifies the location on the drum of the first word of a unit record that is to be read or written. It is given following a

READ or WRITE instruction that selects a drum as the input or output mechanism. Not giving a SET DR instruction is equivalent to giving the instruction with x equal to zero. Since only full-word locations may be referred to on a drum, the sign of the instruction is immaterial. However, x must be one of the *even* integers in the range 0000 to 4094.

Copy and Skip
COPY
31–none

The copy instruction is used following the READ, the READ BACKWARD, or the WRITE instruction. When COPY follows READ or READ B, its execution causes the word coming from the input mechanism to be placed in memory location x. When COPY follows a WRITE instruction, the word in memory location x is sent to the output mechanism to be written as part of a unit record.

While reading cards or tape, the machine will recognize the fact that the end of a unit record has been reached. If a COPY instruction is given subsequent to this recognition, the COPY will not be executed; instead, the machine will skip to the *third* instruction following the COPY instruction, and continue sequentially. This is known as an *end-of-record* skip.

Similarly, the machine will recognize that it has just completed the last unit record of a file if one extra READ instruction is supplied. In this case, a COPY instruction given after the extra READ instruction will not be executed, and the machine will skip to the *second* instruction following the COPY instruction. This is known as an *end-of-file* skip. In reading from a drum, or writing by means of *any* output device, the COPY instruction never causes a skip.

The MQ register is an intermediate step in the transmittal of words to or from electrostatic storage. Execution of a COPY instruction, therefore, destroys any number that may have been standing in the MQ register. In fact, this execution, for all input-output mechanisms (except tape), results in the MQ register containing the word just copied.

Since the input-output units are capable of handling units of full-words only, the COPY instruction normally is given a negative sign. If COPY is given with a positive sign while a unit record is being written, the *half-word* in the memory location actually specified by the COPY instruction, will be written as a *full-word* with zeros in the rightmost 18 bit positions. On reading a unit record, COPY given with a positive sign and an even address will cause the sign and leftmost 17 bits of the *full-word* being brought in to be stored in the *half-word* location actually specified by the COPY instruction. COPY given with a positive sign and an odd address will cause the rightmost 18 bits of the *full-word* being brought in to be stored in the *half-word* location specified. Note that this results in the most significant part of the number being lost.

SENSE OPERATIONS

THE NAME, abbreviation, and numerical code for a sense operation follows. The sign of the instruction has no significance.

Sense and Skip or Control
SENSE
30–none

The SENSE instructions provide a means by which electrical signals may be transmitted between the automatic control section of the calculator, on the one hand, and the printer, card punch, and operator's panel, on the other. The address x of a SENSE instruction is used to identify any one of several electrical terminals. These terminals are classified as either input or output terminals. The identifications of these terminals and the components with which they are associated are shown in Table II.

TABLE II

PRINTER		CARD PUNCH		OPERATOR'S PANEL	
INPUT	OUTPUT	INPUT	OUTPUT	INPUT	OUTPUT
0522	0512	none	1024	0069	0065
	0513		1025	0070	0066
	0514			0071	0067
	0515			0072	0068
	0516			0073	
	0517			0074	
	0518				
	0519				
	0520				
	0521				

Input Terminals

When a SENSE instruction refers to an input terminal by means of the address x, the action is as follows: if an electrical impulse is present on that terminal, the machine skips to the *second* instruction

following the SENSE instruction; if no impulse is present, the machine continues its instructions in normal sequence.

There are six input terminals on the operator's panel; these are known as the *sense-input switches* and are identified by the addresses 0069 through 0074. If the switch is pressed down, it is said to be in the ON position. Thus, when the SENSE instruction refers to that switch, there *will* be an impulse available, and one instruction will be skipped in the programming as described above. If the switch is not on, no impulse is available.

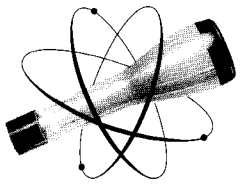
There are no input terminals on the card punch and only one on the printer. Use of the printer-input terminal is described later under *Control Panel Wiring.*

Output Terminals

When a SENSE instruction with an output terminal address is executed, an electrical impulse is made available on that terminal. No skipping of instructions takes place.

On the operator's panel are four output terminals that take the form of neon lights. These lights can be turned on individually by the program and are usually used to show the operator the progress of the problem. The addresses of these lights are 0065 through 0068. The output terminal whose address is 0064 simply turns out *all* the neon lights.

There are ten output hubs on the printer control panel, with addresses shown in Table II. By means of pulses made available by the SENSE instructions, it is possible to transfer selectors and effect mechanical control of the printer, such as double spacing, etc. The two output terminals of the card punch are used in a similar way. These output terminals actually take the form of hubs on the control panels of the components. Their use is explained under *Control Panel Wiring.*



INPUT-OUTPUT COMPONENTS

THE CARD READER, card punch, printer, magnetic tapes, and magnetic drums will all be loosely classified as input-output components of the machine, because they all share the common property of being able to receive information from, or transmit information to, electrostatic storage automatically. In fact, it must be remembered that, *whenever information is transmitted from one component of the machine to another, it* MUST *pass through the electrostatic memory.*

Any machine component capable of transmitting information *both* to and from electrostatic storage automatically (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 input-output system of this machine is outstanding both in versatility and speed. The computer has full automatic control over *all* input-output components. Even mechanical functions of components, such as the printer and punch, may be put under control of the program. Supplementing this mechanical control are the control panels for the printer, punch, and reader. This mechanical control is described under *Control Panel Wiring.*

The great versatility of the system is obtained by the automatic control of the components via the stored program. For example, a binary-to-decimal conversion can be programmed without slowing down the rate of printing. These programs are accomplished at the expense of some storage space, but the resulting usefulness of the machine makes it extremely profitable. Moreover, such programs as mentioned above need be written only once, and used over and over again. A few such basic sub-programs are given in Part II.

All of the input-output components will be discussed in this section to show how they operate under control of the stored program. Although timing considerations will be mentioned, exact data will be given in the *Timing* section. Also, wiring and manual control of components are discussed under *Control Panel Wiring* and *Manual Control of Components.*

In all instances it will be assumed that the components have been manually prepared for control by the calculator, such as the insertion of tapes or cards, etc.

PUNCHED CARDS

IN THIS MACHINE cards are intended to be the primary input medium because of their great flexibility and because of the availability of apparatus for key-punching, verifying, and duplicating. Errors are easily detected and corrected, input data may be readily prepared on several key-punches simultaneously, and the cards may be collected before entry into the computer. Cards are particularly desirable when one wants to have manual access to a file since they can be easily separated, and their contents may be printed on them. It should be emphasized that the punched card input and output may represent any alphabetic character or special symbol, provided only that a program exists to recognize the IBM code for this information. A program may also provide for quantities to be represented in any number system and read or punched accordingly.

Entering a program on cards may be done in such a way that instructions are punched, one to a card, in the form most desirable to the programmer (e.g., in decimal notation). The computer can then be supplied with a standard program to assemble the instructions in the desired order. Then, if errors are detected or if changes must be made, the wrong cards are removed, the correct ones (not necessarily the same number of cards) are added, and the computer prepares the new program. Note that there is no need to repunch any but the cards in question.

The card-feeding mechanism in the card reader is similar to that in the Type 402 Accounting Machine and includes two sets of 80 reading brushes. Correspondingly, there are 80 punching magnets and 80 punching brushes in the card punch. Only 72 columns of the standard IBM card, however, can be read into electrostatic storage, and only 72 columns can be punched from electrostatic storage (unless split-column wiring is used). Any 72 columns of the card can be selected through control panel wiring. For simplicity in the following discussion, it will be assumed that columns 1 to 72 of the card are used for both reading and punching.

Binary information is represented on a card as follows: each of the 12 rows of the card is split into two parts, the left half consisting of columns 1 to 36 and the right half of columns 37 to 72; each half row can be treated as a 36-bit word and read into a full-word location in electrostatic storage.

Figure 7 shows how the card is divided. In this particular example, the first 72 columns of the card are used. Each of the rows is split into half-rows of 36 columns each. Thus, the half-row identified by the circled 9 in Figure 7 is named the 5-row left. Similarly, the row identified by the circled 10 is named the 5-row right. Thus, there are 24 half-rows in the card. One full word of binary information can be punched

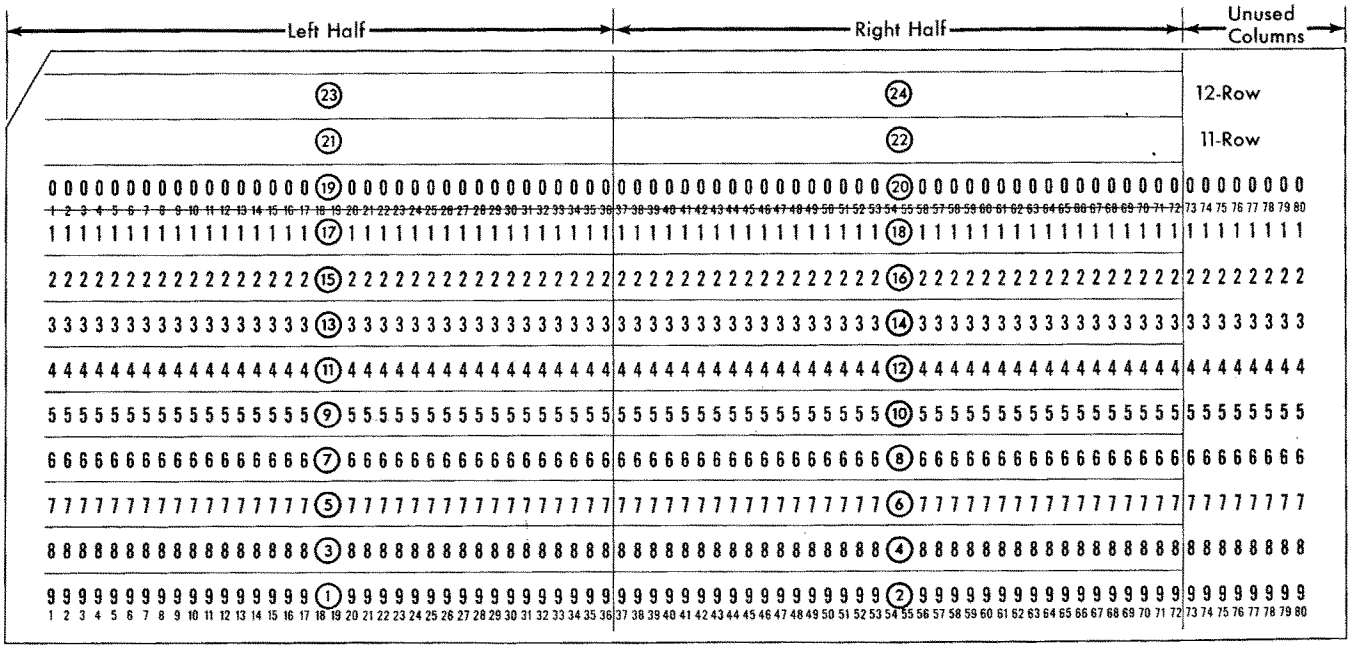


FIGURE 7

in any half-row (including sign). The machine regards any punched hole as a binary 1. "No punch" indicates a binary 0. Thus, an 8-punch in column 36 of the card is regarded by the machine as a binary 1 in the least significant position of the full binary word punched in the 8-row left. The leftmost position of each half-row is reserved for the sign bit of the full word. A binary 1 represents a negative sign, while a binary 0 represents a positive sign.

(NOTE: The exact position of a full word that each column represents is completely arbitrary according to how the particular control panel is wired. This example is the normal and simple case. The wiring discussed under *Control Panel Wiring* assumes that information is to be read into the calculator as described above.)

It should also be pointed out that this card representation of 24 binary words does not mean that the cards must always be punched with true binary information. The holes in the card can just as well be numerical punching in the standard decimal card code, alphabetic punching, or control punching. It is necessary only to provide a suitable program so that the computer can translate between the binary code in which it operates and the particular code used on the card. The translation to and from the decimal numerical code, for instance, can proceed simultaneously with reading and punching so that the over-all card-handling speed is not reduced below the standard rates of 150 cards per minute for reading and 100 cards per minute for punching.

Cards are fed face down, 9's edge first, in *both* the card reader and card punch. The internal card circuits are arranged so that the 24 half-rows of the card are read or punched in the sequence indicated by the circled numbers in Figure 7. The sequence of reading or punching full words is then as follows: 9-row left, 9-row right, 8-row left, 8-row right, and so on to 12-row left, 12-row right.

For reading and punching cards, a unit record is defined as the information contained in *one* card. A file will consist of any number of unit records and so will take the form of a deck of cards. Note that definitions of unit records and files will, in general, be different, depending on the particular input or output component being discussed. This has been done so that descriptions of input-output operations of the previous section could be made to apply, as nearly as possible, to all components.

Card Reader

For a program to cause the calculator to read all of the information punched on a card into electrostatic storage, it is necessary to give a READ instruction with an address of 2048 (card-reader identification) followed by 24 COPY instructions.

The READ instruction causes the card-feeding mechanism to start in motion. The program then is free to continue any operations until the 9-row of the card appears under the reading brushes. At this time, the program must provide a COPY instruction with an address, x. This instruction will cause the word punched in the 9-row left, to be read and stored in electrostatic location x. The program can then resume until the calculator is prepared to read information punched in the 9-row right. The program now must supply another COPY instruction to read this word into electrostatic memory. This procedure continues until all 24 half-rows have been read. Because of their functions, these COPY instructions are called 9 left COPY, 9 right COPY, etc. Another READ instruction must be given to read another unit record (card).

Of course, the READ instruction can be given, followed immediately by the 24 COPY instructions in succession, without any other operations being done between instructions. In such a case the calculator will automatically wait until a half-row is in position to be read before the COPY instruction is executed.

The intervals of time between these instructions which may be used for useful calculating, are definitely limited and are completely specified in the *Timing* section. If a COPY is given *after* the card reader is in position to read a given half-row, the machine will stop, and the copy check light will turn on at the operator's panel. The amount of calculating time available between the last COPY instruction for a given card and the READ instruction that initiates the reading of a succeeding card is unlimited. But if a READ instruction does not occur within a definite time limit, the card reader will stop and will start up only after the new READ instruction has been received. To keep the card reader in continuous motion and operating at its full speed of 150 cards per minute, the time limit discussed under *Timing* must be observed.

Programming experience shows that it is extremely convenient to have this time between input-output instructions in order to compute succeeding COPY addresses and to control the calculator. Calculator operation is such that during execution of a COPY in-

struction, the word read from a half-row of the card first enters the MQ register before being sent to the electrostatic memory. This, of course, destroys any information previously stored in this register.

If a 25th COPY instruction is given after a READ instruction, the card reader will already have set up what is known as an "end-of-record" condition (this denotes that all 24 half-rows of the card have been read). Under this condition, the 25th COPY is not executed, and the program skips to the *third* instruction after the COPY. In this way the program may transfer control to a section that will cause the succeeding card to be read.

When the hopper of the card reader becomes empty, the calculator stops. The start button on the card reader may then be depressed to allow the cards remaining ahead of the reading station to be read under control of the program. After the last card has been read in this way, and if another READ instruction followed by a COPY is given, the card reader sets up a condition known as an "end-of-file" condition. Under this condition the COPY instruction is not executed, and the program skips to the *second* instruction following the COPY. In this way, for example, control may be transferred to a particular section of the program that continues a calculation interrupted by the card-reading procedure.

The contents of the 24 locations of the electrostatic memory, into which the 24 half-rows have been read, is known as the *card image*. By a program that suitably manipulates this card image, decimal information punched in standard IBM code may be converted to binary information.

In reading cards it is not always necessary to follow a READ by 24 COPY instructions. The card reader will normally read half-rows for every following COPY instruction up to 24. If, however, after a few COPY instructions, another READ is given, the card reader automatically ignores any succeeding half-rows that have not been read and starts reading a new card. Thus, for instance, it is possible to read the first five words of a card and ignore the rest. It is not possible, however, to read the first five words, skip the sixth and seventh words, and continue on reading the card. A COPY instruction designed to accomplish any reading of *this* type will always result in a machine stop and a copy-check light. If successive READ instructions are given with no intervening COPY instructions, the net result is the feeding of cards through the machine with no words being read into storage.

Card Punch

The operation of punching information on a card is very similar to that of card reading. To make use of these similarities, it will be assumed that the programming necessary for card reading is understood.

Punching a card requires a WRITE instruction having an address of 1024 (card-punch identification) which sets the card-feeding mechanism of the punch in motion.

Following WRITE, a succession of COPY instructions is given, the address parts of which give the locations in electrostatic storage where the words to be punched in the half rows of the card are to be taken. To punch a full card, 24 COPY instructions must be given. These COPY instructions are called, as in card reading, 9 left COPY, 9 right COPY, etc. A separate WRITE instruction must be given for each card to be punched. Corresponding to card reading, a certain amount of computing can be carried out between WRITE and the first COPY instruction and between successive COPY instructions.

For example, binary to decimal conversion can be completed while a card is being punched. Each COPY instruction, however, must have been given by the time the corresponding half-row appears at the punching station. These time limits are specified under *Timing*. If time limits are exceeded, the machine stops, and the copy check light on the operator's panel signals the error. To keep the punch running at its full speed of 100 cards per minute (and if more than one card is to be punched) succeeding WRITE instructions must be given within a certain time interval. Otherwise, if a WRITE instruction is delayed too long, the punch will stop and will not start again until the WRITE is actually given. If these WRITE and COPY instructions are given in succession, the computer delays until a half-row is actually in position to be punched.

If fewer than 24 COPY instructions are given, the remaining half-rows on the card, for which there were no COPY instructions, are left blank.

Again, we must remember that the MQ register serves as an intermediate storage during a COPY instruction. Since the word to be punched first enters this register before being sent to the card punch, any previous information in the register is destroyed.

There are no end-of-record or end-of-file conditions in punching cards.

PRINTER

THE PRINTER, which is a modification of the printing unit on the IBM Type 407 Accounting Machine, is equipped with 120 rotary type-wheels. Each wheel has 48 characters, including Arabic numerals, alphabetic symbols, and special characters. By means of a proper program the machine can be made to print decimal numbers, binary numbers, or numbers to any other base. Titles and headings are also possible, since alphabetic characters and special symbols are provided on the type-wheels.

As in the standard Type 407, the type-wheels are positioned for printing by electrical pulses timed according to the print cycle itself. Thus, it will be recalled that if a print-wheel receives an electrical impulse during that part of the print cycle designated as 9-time, then the print-wheel will be positioned to print a 9. Also, if the print-wheel receives an impulse at 1-time *and* an impulse at 12-time, the machine will interpret this as the letter *A* (according to the standard IBM code) and will position the type-wheel to print this character. It is important to understand this timing principle of accounting machines. Reference may be made to the Type 407 Accounting Machine principles of operation manual.

A simple example is used to show how a series of nines might be printed in 72 positions of a line.

EXAMPLE: A WRITE instruction with an address of 0512 (printer identification) is programmed. This causes the printer to start a print cycle. As the print cycle progresses, it goes through points in the cycle known as 9-time, 8-time, 7-time, and so on to 11-time and 12-time. These times are analogous to the times designated for the standard Type 407, which operates in conjunction with a card-reading mechanism. Thus, in the standard Type 407, the above times are in exact coincidence with the time the 9-row is under the reading brushes, etc. As soon as the 701 printer reaches 9-time in its cycle, the program must furnish a COPY instruction that reads a full word from an electrostatic storage location. By means of control panel wiring (see *Control Panel Wiring*) this full word is directed to 36 type-wheels (one for each bit of information) of the printer. A second COPY instruction then follows; this causes another full word in electrostatic storage to be directed to 36 other type-wheels. These two COPY instructions are given close enough together (with respect to time) that the printer is still essentially at 9-time of its cycle. A binary digit of 1 in the full word will cause an electrical pulse to enter the printer and to impulse the associated type-wheel. This impulse will cause the type-wheel to be positioned for printing a 9, since the impulse arrived at 9-time of the print cycle. If now we assume that both full words mentioned in connection with the above two COPY instructions consist of a negative sign and 35 binary ones, the result will be 72 nines printing across the page. This assumes that any subsequent COPY instructions corresponding to 8-time, 7-time, etc., will not cause an additional impulse to a type-wheel and thus cause the wheel to be positioned in a different manner. A positive sign or a binary zero will not propagate a pulse to the type-wheels.

The general procedure in printing a line is to set up in electrostatic storage a card image similar in nature to the card image produced when a card is read by the card reader. A WRITE instruction then is followed by 12 *pairs* of COPY instructions; these cause the card image to send impulses to the type-wheels. The first pair of COPY instructions will cause 9-time impulses to be sent to the type-wheels as explained in the example above. The second pair will cause 8-time impulses to be sent to the type-wheels. This procedure continues until the 12 pairs of instructions are executed in accordance with the 12 distinct times of the print cycle. Subsequent WRITE and COPY instructions will result in a new print cycle and a new printed line. The full-word impulses, brought about by execution of the *first* COPY instruction of a pair, are available at the hubs labeled CALC EXIT LEFT on the printer control panel and may be directed to the selected type wheels by wiring. The impulses produced by the *second* COPY instruction of a pair are available at the hubs labeled CALC EXIT RIGHT. For various reasons described above, the 24 COPY instructions will be named in sequence, as follows: 9-left COPY, 9-right COPY, 8-left COPY, 8-right COPY, and so on to the 12-left COPY, 12-right COPY.

As in the card reader and card punch, it is possible to do useful calculating between the actual printing instructions. For example, the time required for the print cycle to start, and move to 9-time of its cycle, may be used for other calculations. Once the cycle has reached 9-time, however, the program must provide,

in succession, the pair of COPY instructions for impulsing the type-wheels. Useful calculating can also be performed between pairs of instructions and even between individual instructions of a pair. As before, there are definite time limits to be observed. These are precisely specified under *Timing*. If a COPY instruction arrives too late in the cycle, the machine will stop, and the copy check light will indicate the error. For the machine to print at its full rate of 150 lines per minute, the WRITE instructions for each print cycle must be given in the interval of time explained under *Timing*. If WRITE instructions do not follow each other within this time limit, the printer will stop and will start again only on receipt of the next WRITE instruction. If we do not want to do calculating between these input-output instructions, we can program these instructions in immediate sequence. Under these conditions, the calculator will automatically wait until the printer reaches the proper point of its cycle before executing the instructions. Also, it is not always necessary to give a full set of 24 COPY instructions for each line of print. If a full set is not given, the action will be similar to the card reader; namely, the print cycle will continue without any more impulses to the type-wheels, and a following WRITE instruction will start a new cycle.

Again, the MQ register is used as an intermediate storage for a full word passing from electrostatic memory to the printer; so the execution of any COPY instruction destroys information previously standing in the register.

The previous paragraphs give the procedure for printing *without* checking. Checking is possible because the printer is capable not only of receiving print pulses from the calculator to set up the type-wheels for printing, but also capable of sending to the calculator "echo pulses" generated by the type-wheels according to what character the wheels are in position to print. Printing with checking, however, requires a somewhat more complicated program, but it can be done without reducing printing speed. The timing for this combination is *roughly* as follows:

The first half of the print cycle is used to position the type-wheels by means of words in electrostatic storage. The second half is used for reading the "echo pulses" generated by the type-wheels and for placing them in electrostatic storage for verification via a programmed check.

When it is desired to check, the echo pulses are read in such a way as to form a card image when received by the calculator. Thus, the calculator can both write the original card image for printing and read a corresponding card image, at a later time in the same print cycle, from the echo pulses. If the two card images do not agree exactly, as determined by a suitable program, then an error must have occurred. Only numerical information can be checked this way.

Printing with checking is programmed as follows. First a READ instruction (note this difference), with the address of the printer, is given, followed by *46* COPY instructions in a specified sequence. Twenty-four of these COPY instructions refer to printing, and cause words to be sent from electrostatic storage to the printer. The other 22 COPY instructions refer to checking. They require words to be read from the printer into electrostatic storage. During part of the print cycle, the two kinds of COPY instructions must alternate in pairs. Note, then, that the READ instructions will cause *both* writing information from storage to printer, *and* reading the echo impulses into storage.

Exact sequence of the 46 COPY instructions for printing with checking is described below. There are two sets of codes for plus and minus signs, as follows: with one set, used for printing *without* checking, 12 is the code for plus, and 11 for minus; with the other set, used for printing *with* checking, the combination of 8 and 3 is the code for plus, and 8 and 4 the code for minus.

The first 18 COPY instructions are to supply impulses to the printer from the left and right halves of rows 9 through 1 of the card image. The 19th and 20th COPY instructions are for storing the echo impulses received from the minus sign (code 8, 4). The 21st and 22nd COPY instructions send impulses to the printer from the zero row of the card. The 23rd and 24th COPY instructions store echo impulses received from the plus sign (code 8, 3). The 25th and 26th send the 11-row of the card image to the printer. The 27th and 28th are for checking the 9-row. The 29th and 30th send the 12-row of the card image to the printer. Finally, the 31st through 46th COPY instructions form the check images of the 8-row through the 1-row. The fact that no checking is provided for the 0, 11, and 12 rows explains the two separate codes for plus-and-minus signs. The exact sequence of instructions and the allowable time between them are fully explained under *Timing*.

Through use of selectors or column splits on the printer control panel, more than 72 type-wheels can

be activated from electrostatic storage. For example, seven 10-digit numbers with signs can be printed. Additional characters can also be printed by means of impulses emitted on the control panel. Alternatively, the control panel can be wired so that up to 120 characters originating from electrostatic storage can be printed on each line at the rate of 75 lines per minute, two cycles being required for each line of printing.

The printer has an IBM tape-controlled carriage, details of which are found in the manual of operation for the IBM Type 407 Accounting Machine. Functions of the carriage (such as changing or suppressing line spacing, selecting the channel on the punched tape to control skipping, or sensing sheet overflow) may be controlled through sense output or input hubs on the control panel; these hubs are activated by appropriate SENSE instructions in the stored program.

MAGNETIC TAPES

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. Libraries of standard sub-programs, which can be called on to reduce the amount of programming required for each problem, can also be stored on tape.

There are four tape units; each contains a magnetic tape of any length up to 1400 feet. The tape itself is one-half inch wide, oxide-coated, and non-metallic. After the tape has been placed in motion, it can read or write information at the rate of 1250 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. Six groups recorded serially on a tape are needed to store one binary word of 36 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 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 the tape-check light on the operator's panel is turned on. If the number of 1's is odd (as it should be when correct), the machine continues the reading and writing process. It should be emphasized that operation of this seventh channel is *completely* automatic and is brought out to indicate the function of the tape-check light on the operator's panel.

A *schematic* diagram of how a word of 36 bits is recorded on tape is shown in Figure 8. Each X denotes a binary 1 or 0 recorded in that position on the tape. The 36 bits recorded in the six recording channels represent the full word. When the tape moves in the direction of the arrow, the group numbered 1 will contain the sign and first five bits of the word. The remaining five groups contain the following bits of the word in groups of six. Thus, group 2 contains bits 6 through 11 of the word, etc.

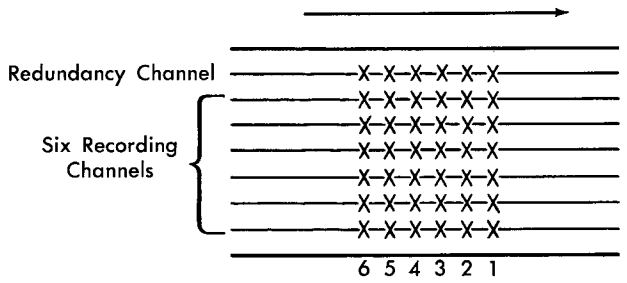


FIGURE 8

Writing

The general procedure for writing a file of information on a tape is as follows: A WRITE instruction is given with an address designating a particular one of the four tapes (see *Address System*). This instruction causes the selected tape to be started in motion and prepares it to record a unit record of information from electrostatic storage.

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

If the WRITE instruction is given when the tape unit is in the rewound position (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 there results a blank portion of tape called the beginning-of-file gap.

As soon as the beginning-of-file gap has been written the program must supply a COPY instruction. This instruction takes the full word stored in the electrostatic storage location specified by the address part of the COPY instruction, and puts it into the MQ register. The selected tape unit then takes this word from the MQ register in groups of six bits each, and records them on the tape in the manner shown in Figure 8. By the time all six groups of bits have been recorded, the program must furnish another COPY instruction for recording the second word of the record. The COPY instructions may continue in this way until the entire unit record is written. In all cases where an instruction is furnished by the program before the tape unit is ready for it, the execution of the instruction is automatically delayed.

Note that the length of the unit record is variable and depends only on the number of COPY instructions following the WRITE instruction. If the tape unit is ready for a COPY instruction (which means it has finished recording the last word), but the program fails to supply this instruction, 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 will be called an end-of-record gap; these gaps are extremely convenient when reading words back into electrostatic memory. When the programmer wants to end a unit record, he simply stops supplying the calculator with COPY instructions. If a COPY instruction is given after the tape unit disconnects, the entire computer stops, and the copy-check light signals the error.

The first unit record of the file has now been written. To write a second unit record, the same procedure (WRITE instructions followed by COPY instructions) is programmed. In this way a series of records is recorded. Note again that the records may be of variable size if desired. A series of WRITE instructions with no COPY instructions will result in the tape moving through the machine and being erased. If a new WRITE instruction is given while the machine is still recording the last word of the previous record, the program is automatically delayed until the tape has erased an end-of-record gap, and is prepared to accept the first word of the new record.

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 file, there may be bits left over from a previous use of the tape. These residual bits of information may not be properly spaced with respect 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 *Write End of File* (abbreviated to WRITE EF) 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.

Figure 9 shows *schematically* how a typical file of information is recorded on tape. The arrow desig-

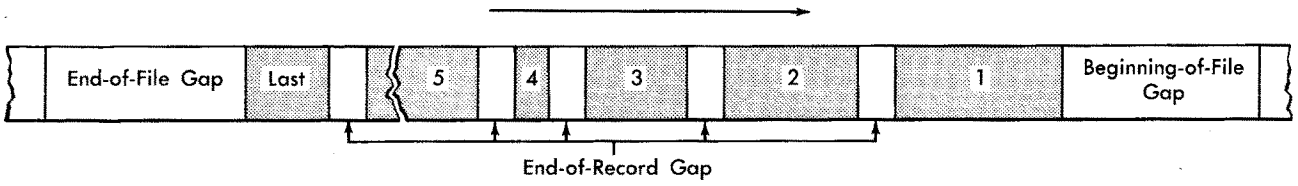


FIGURE 9

nates 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 the intervening end-of-record gaps. Note that these gaps are of a fixed length regardless of the length of the unit record itself. Finally, an end-of-file gap appears after the last unit record. Note, too, that the machine operates so that lengths of the two gaps at each end of a file are equal to each other, but longer than end-of-record gaps. All of these items will play an important role when a program is required to read this information back into storage.

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 stop and start between records if desired.
3. The still longer gaps at the ends of the file.

As in other input-output devices, it is possible to do useful calculating (such as the computation of the addresses of succeeding COPY instructions) between the WRITE instruction and first COPY instruction and between successive COPY instructions. Again, this time is limited and is specifically discussed under *Timing*. It will be recalled, however, that the MQ register is in continuous use after each COPY instruction, because the word is recorded six bits at a time on tape. This observation means that between COPY instructions no calculating can be done that requires use of the MQ register. This restriction specifically excludes the following operations: STORE MQ, LOAD MQ, MPY, MPY R, DIV, ROUND, L LEFT, and L RIGHT. (In devices previously discussed, the MQ register is used *only* during execution of the COPY instruction and simply destroys the number previously stored there. The MQ register could be used between COPY instructions in those instances.)

After the last COPY instruction for writing a unit record has been given, the MQ register is in use as before. Consequently, the programmer must be sure that a certain amount of time elapses (while the last word is going from the MQ register to tape) before programming any instructions involving the MQ register. There is an instruction available to the programmer for delaying the program until the MQ register is free for use. By using this delay instruction after each unit record, the programmer need not worry about the delay. The delay instruction is "WRITE 2052." Note that this instruction, if interpreted in the usual way, is nonsense, since 2052 does not identify any component of the machine. However, the machine circuits are so set up that it will recognize this particular instruction as a delay instruction, and the program is automatically held up until the MQ register is not being used.

Note that the MQ register is not in use with the tape unit between the WRITE instruction and the first COPY instruction for a unit record. Consequently, there are no restrictions on use of the MQ register by the program when using the calculating time available between these instructions. Note, also, that the MQ register is not required by the WRITE EF instruction, and the program can continue unrestricted in this case.

Reading

Reading information stored on tape is similar in many respects to the writing process. Because of this similarity, the description of reading a file of records will be more abbreviated than the preceding paragraphs.

Assume that the tape is in position to read the first unit record of a file. A READ instruction, with an address to select the particular tape to be read, causes the tape to start in motion. *This instruction also causes the* MQ *register to reset to zero* (a READ instruction causes this *only* when the address designates a tape unit). Subsequent COPY instructions will cause a word to be loaded six bits at a time into the MQ register, and from there the whole word is sent to the electrostatic location specified by the address part of the COPY instruction. When the tape unit comes to the first word of the unit record, it places it, six bits at a time, into the MQ register. When the word is completely assembled in the register, the program must supply a COPY instruction that results in a transfer of the word to the electrostatic memory. The COPY instructions can then follow, and the tape continues to move as the record is read word by word.

But eventually the tape comes to the end-of-record gap. The machine recognizes this gap, the tape stops, and an end-of-record condition is set up. If, under this condition, the program supplies *another* COPY instruction, the machine will not execute it, but will skip to the *third* instruction following the COPY instruction. This allows the program to follow a different course after finding the end of a record on read-

ing. It is not necessary to know the number of words stored in a unit record, because the tape will automatically stop after reading the entire record.

However, it is not necessary to read and store every word of a record. After reading the number of words desired, the program may go on to something else, and there will not be another COPY instruction waiting the next time the MQ register is filled with a word from the tape. This causes the tape unit to disconnect itself from the MQ register. The MQ register can then be used by the program for other purposes; but the tape itself *does not stop until it comes to the end-of-record gap.* As explained in the previous section for writing on tape, the delay instruction (WRITE 2052) should follow the last COPY instruction to delay the program and avoid an overlap between the next word of the record (which will enter the MQ register before the tape unit discovers the absence of a corresponding COPY instruction) and any instruction that may require the MQ register. But the delay instruction *is not necessary* when the *entire* unit record is read.

In general, the programmer has four alternatives for disposing of the words of a unit record after giving a READ instruction:

1. He may give a series of COPY instructions to store every word of a unit record; and he may continue to do so until the end of the record is signaled by the end-of-record gap. Often this is the simplest way of reading the entire record, and it is particularly useful when the exact number of words in the record is not available and need not be known. The delay instruction is not necessary.
2. He may know the number of words in the unit record and give only as many COPY instructions as there are words in the record. He would then continue without waiting for an end-of-record signal. In this case the delay instruction need not be given.
3. He may read and store fewer words than there are in the unit record. When a sufficient number of COPY's have been given, he may give WRITE 2052 to delay the program and force the tape unit to disconnect. Then he may continue with a different program and ignore the rest of the unit record.
4. He need not give any COPY instructions. In that case the tape unit will put the first word into the MQ register. When it does not find a COPY instruction waiting at the end of this word, the tape unit will disconnect immediately. The tape itself, however, continues to move through the entire unit record until it reaches the end-of-record gap. Nothing will have been stored in memory. This device makes it possible to skip through one or more unit records to the desired one without storing the unwanted records in memory. (If there is any danger of overlap in the use of the MQ register, then the delay instruction should be used here, too.)

In any case, the tape unit continues to run after a READ instruction until it comes to the end of the unit record, regardless of how much of the record is actually being read. If two READ instructions follow each other too closely, the second is held up until the tape unit has reached the end of the previous unit record and is ready to read another, because every READ instruction applies to a new unit record.

All of the unit records in a file can be read as described above. After the last unit record in the file is read, however, the tape is positioned at the beginning of the end-of-file gap. If, at this time, another READ instruction is given, the tape unit starts up and attempts to read another unit record. Instead of a unit record, it finds the end-of-file gap, and after sufficient searching for a new unit record, the tape unit again stops and sets up an end-of-file condition within the machine. If, under this condition, the program supplies a COPY instruction (as if it were going to read the first word of the new unit record), the COPY instruction will not be executed, and the program will skip to the *second* instruction following the COPY. The programmer may then take advantage of this automatic skip to go into a new program. This end-of-file condition 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 WRITE EF instruction.

The end-of-file, of course, can be taken into account by giving the same number of READ instructions as there are unit records. But the automatic end-of-file feature is very convenient when the number of unit records is not known directly or when it is of no interest, such as when reading through a library of programs on tape. Although the end-of-file gap may

not be used, it is *always* necessary to give a WRITE EF instruction after *writing* a file (see above section).

As in writing, useful calculation can be done between the input-output instructions. Again, the time is definitely limited and is discussed below under *Timing*. If a COPY instruction is given after the specified time interval has elapsed, the machine stops and the copy-check light turns on to signal the error. The restrictions on use of the MQ register between COPY instructions is the same as for writing. As with writing, calculating may be done between a READ instruction and the first COPY instruction of a unit record, but there is one *important difference*. The MQ register is reset to zero by the READ instruction. This is done in preparation for the following COPY instruction which assumes that the MQ register *is actually zero*. Thus, the MQ register *should not be used* for calculating during this interval.

NOTE: The programmer may use the MQ register during this interval if he takes extreme care to reset this register to a *positive zero* (via the program) *before* the tape unit starts to put the first word into the MQ register. This is not good practice, however, and should be done only if *absolutely necessary*. Note in particular that the program must reset the MQ register to zero *before* the tape unit comes to the first word and *not* simply before the first COPY instruction. This requires a thorough understanding of tape timing.

A tape can be read backward one unit record at a time by giving the instruction called *Prepare to Read Backward* (abbreviated to READ B) in place of READ. This instruction differs from READ only in that it causes the tape to move in the reverse direction.

There are two uses of READ B. One is to backspace a tape by a given number of unit records. This is done simply by giving the appropriate number of READ B instructions. Also, by means of the READ B instruction, the tape can actually be read in reverse order and part or all of a unit record stored in electrostatic storage. This is done by giving READ B, followed by a suitable number of COPY instructions. Under these circumstances, it is evident that the words will be read in the reverse order in which they were written. Although the words are read in reverse order, the machine automatically arranges the bits within the word into their correct order before placing the word in the MQ register.

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

In general, the procedure and controls available when reading backward are the same as when reading forward. Restrictions on use of calculating time between instructions are identical to those imposed when reading forward.

Rewinding

The REWIND instruction is used to cause the tape specified by the address part of the instruction to return to the starting point of its file of records.

A REWIND instruction may be given in the course of reading a file, or after the end-of-file condition has been sensed, or while the tape is disconnected. If REWIND is given while the tape is being read, the tape will go to the end of the unit record before rewinding. When writing a tape, a REWIND instruction should not be given until after a WRITE EF instruction has been used to terminate the file properly. Information cannot be read from nor written on the tape during rewinding. While this instruction is being executed, the program may continue, and may use any other input-output unit; but if it calls for a tape unit still in the rewinding process, the program is held up until the tape unit has stopped.

Tape Status

It is not possible both to read and write on a single passage of the 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 completely before any new information can be written on it.

There are circuits associated with each tape unit that 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*.

Lights on each tape unit indicate the status of the unit. The reading light is on for read status, the writing light is on for write status, and both are off for neutral status.

When a tape unit is in read status, it may be used only for reading; when in write status, it may be used only for writing. If a WRITE or a WRITE EF instruction is given for a tape unit that is in read status, the instruction is not executed, and the tape remains stopped. If a COPY instruction follows the WRITE instruction, the calculator will stop and the copy-check light will turn on. Similarly, giving either READ or READ B instructions for a tape unit in write status results in the instruction being ignored; any subsequent COPY instruction would again be treated as an error.

Any READ or WRITE instructions will be executed in the normal manner if the tape is in neutral status upon which the tape unit will be set to read or write status, respectively. Any of the following actions will always restore a tape unit to neutral status, irrespective of its previous status:

1. Giving a REWIND instruction for the tape unit.
2. Opening the door of the tape unit (to insert a new tape).
3. Turning off the power.

A WRITE EF instruction will return a tape to neutral status only if it is originally in write status.

Table III shows the resulting status of the tape after an instruction has been given under various conditions of read, neutral, and write.

REWIND will be executed with the tape unit in any status. However, REWIND should not be given while the unit is in write status, unless an error has been detected and the operator simply wishes to write over the unfinished file from the beginning. Normally, WRITE EF, which resets the tape unit to neutral, must precede REWIND. A REWIND instruction must always be given before one can write, even after reading backward all the way to the beginning. Once a tape has been rewound, further REWIND 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 by means of a WRITE EF, the tape unit will be in neutral status. Then it must not be attempted to give a READ, WRITE, or another WRITE EF instruction which would move the tape further forward, although the neutral status would permit such meaningless actions. Only READ B or REWIND can follow WRITE EF for the same tape unit.

When a tape is being read in the forward direction and the end-of-file gap has once been sensed as previously described above, no further READ instructions should be given which would move the tape further forward. Only READ B or REWIND can follow the detection of the end-of-file gap. Similarly, after sensing the beginning-of-file gap and setting up an end-of-file condition on reading backward, or after giving a REWIND, one must not give any more READ B instructions.

MAGNETIC DRUMS

THE MAGNETIC DRUM STORAGE is divided into blocks of 2048 full words, each with addresses consisting of the even integers from 0000 to 4094. Each block will be referred to simply as a drum. These drums provide an auxiliary memory that is more accessible than tapes or cards. Individual words on the drum can be selectively altered at any time. Drums are used to a large extent for storing tables of data and sections of

TABLE III

INSTRUCTION	RESULTING TAPE STATUS IF ORIGINAL STATUS WAS:		
	READ	NEUTRAL	WRITE
READ	Read	Read	Write*
READ B	Read	Read	Write*
WRITE	Read*	Write	Write
WRITE EF	Read*	Neutral	Neutral
REWIND	Neutral	Neutral	Neutral

*Instruction treated as no operation.

long programs that may not fit into the electrostatic memory. It should be noted that words on a drum cannot be split into half words as in electrostatic storage.

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

Words are transmitted between electrostatic storage and drum storage via the MQ register just as in the card reader and card punch.

Writing and reading on a drum are so similar that they will be discussed together. The following paragraphs show how to write a unit record on a drum. The reading process is identical, except that a READ instruction is given in place of a WRITE instruction.

A WRITE instruction with an address specifying a particular one of the four drums is given. This instruction causes the specified drum to be connected to the calculator for writing purposes. This instruction is then followed by a *Set Drum Address* instruction (abbreviated to SET DR), whose address specifies the location where the first word of the record is to be written. If the address part of a SET DR instruction is an odd number, the address part is treated as the next lower integer. The sign of the instruction is immaterial. A series of COPY instructions (one for each word of the record) follows. The first COPY instruction causes the calculator to be held up until the drum location specified by the address part of the preceding SET DR instruction passes under the drum heads. The first word is then written from the electrostatic storage location specified by the first COPY instruction into the specified location on the drum. From then on, consecutively even-numbered drum locations appear under the drum heads at regular intervals; each time this happens, a COPY instruction must be supplied if another word is to be written or read. Thus, the second COPY instruction will cause the number stored in the electrostatic location given by the address part to be recorded on the drum at the address two higher than the first word.

This continues by means of COPY instructions, with each word being automatically written at consecutive-numbered even addresses of the drum. If the new consecutive drum location appears under the drum heads and no COPY instruction is supplied by the program, the drum disconnects itself from the computing unit. So when the programmer wants to end a unit record, he simply stops giving COPY instructions. If a COPY instruction is supplied after the drum disconnects and before other WRITE and SET DR instructions are given, the machine stops, and the copy-check light signals the error.

This means that successive COPY instructions must be given within certain time intervals (given later under *Timing*). Calculations may be performed between the input-output instructions; the only restriction on use of the MQ register is to remember that its contents are destroyed upon execution of a COPY instruction.

The division of the drum storage into unit records is quite arbitrary. Hence, if only a single word is to be read or written, it is treated as a one-word unit record. WRITE or READ selects the drum, SET DR specifies the drum location, and a single COPY gives the memory location of the word.

There are no end-of-record or end-of-file conditions on the drums. These are not necessary, because any particular word on a drum may be located by means of the SET DR instruction, and the length of the records is variable.

The method of reading a block of words that begins at any drum location is almost identical to writing a record. Again, the above explanation may serve for reading by replacing the word *write* by *read* in every instance.

The following limitations must be observed in programs for reading or writing on drums:

1. There is no limitation on the length or location of a unit record, except that if it is attempted to read or write beyond drum address 4094, the next address will be 0000. Nothing prevents a unit record from being started at address 0000.

2. Omitting the SET DR instruction is equivalent to giving SET DR with address part 0000.

3. Any number of instructions *(except input-output instructions)* may intervene between READ or WRITE and SET DR and between SET DR and the first COPY instruction. Successive COPY instructions, however, must follow each other within a definite time limit. If a COPY arrives too late, the drum will have been disconnected, and the calculator will stop with the copy-check light turned on.

SUMMARY

PRINCIPAL RULES governing the use of the card reader, card punch, printer, tape units, and drums are summarized below. This summary repeats information already given in the preceding pages.

1. Only one input-output component can be used at a time for reading or writing a unit record.

2. In using any input-output component, COPY instructions must follow one another within a definite time limit. If a COPY instruction is not available within this limit, the device in use will be disconnected from the computing unit, and no more information in that unit record can pass to or from electrostatic storage. Note, however, that the card reader, card punch, or printer will continue in motion until the end of the card (or print) cycle and that a tape unit will continue in motion until the end-of-record gap is reached.

3. If a COPY instruction is given when there is no read-write component connected to the computing unit (e.g., if no READ or WRITE instruction has been given, or if a COPY has been given too late), the calculator stops and a copy-check light turns on, indicating an error in the program.

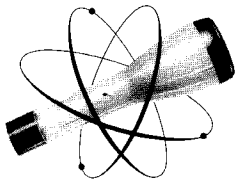
4. In using any of the read-write components except drums, the first COPY instruction must be given within a definite time limit following the READ, READ B, or WRITE instruction, as the case may be. After the unit record has been written or read, the next READ, READ B, or WRITE instruction must follow the last COPY instruction within a definite time limit, if the device in use is to be kept in uninterrupted motion.

5. The following rules apply to tapes only:

   (a) During the reading or writing of information on a tape, the MQ register cannot be used for any other purpose.

   (b) It is not possible to read a tape after writing on that tape unless an intervening WRITE EF or REWIND instruction is given. (Both may be given if desired.)

   (c) It is not possible to write on a tape after reading that tape unless an intervening REWIND instruction is given.



TIMING

OPERATIONS

THE FUNDAMENTAL machine cycle of the 701 is 12 microseconds; the time required to execute an instruction, or a sequence of instructions, is an integral multiple of this cycle.

Table IV shows every operation of which the machine is capable, together with the basic number of fundamental cycles required for its execution. The times shown specifically include obtaining, interpreting, and executing the instruction. The actual number of cycles required, however, is subject to modification under certain conditions. The third column of Table IV lists the class of modification, if any, pertaining to each operation. The four types of modifications are explained below.

TYPE I. The instruction will be executed in two cycles less if a multiplication or division has been performed in the preceding 12 instruction executions.

TYPE II. The instruction will be executed in three cycles less if a multiplication or division has been performed in the preceding 12 instruction executions.

TYPE III. The instruction will be executed in four fundamental cycles provided the extent of shift is 24 places or less, and provided a multiplication or division has not been performed in the preceding 12 instruction executions. Each additional eight places, or portion thereof, require another cycle. If a multiplication or division has been performed in the preceding 12 instruction executions, then the instruction will be executed in two cycles for shifts of eight places or less. Each additional eight places, or portion thereof, require another cycle.

TYPE IV. The execution of this instruction *may* be delayed an indefinite length of time after its interpretation, depending on the status of the input-output components. For example, if the execution of one tape READ instruction is fol-

TABLE IV

OPERATION	BASIC NUMBER OF FUNDAMENTAL CYCLES	MODIFICATION TYPE*
Stop and Transfer	4	I
Transfer	4	I
Transfer on Overflow	4	I
Transfer on Plus	4	I
Transfer on Zero	4	I
Subtract	5	I
Reset and Subtract	5	I
Subtract Absolute Value	5	I
No Operation	4	I
Add	5	I
Reset and Add	5	I
Add Absolute Value	5	I
Store	5	II
Store Address	5	II
Store Contents of MQ Register	5	II
Load MQ Register	5	II
Multiply	38	none
Multiply and Round	38	none
Divide	38	none
Round	4	I
Long Left Shift	4	III
Long Right Shift	4	III
Accumulator Left Shift	4	III
Accumulator Right Shift	4	III
Prepare to Read	4	I and IV
Prepare to Read Backward	4	I and IV
Prepare to Write	4	I and IV
Write End of File	4	I and IV
Rewind Tape	4	I and IV
Set Drum Address	4	I
Sense and Skip or Control	4	I
Copy and Skip	5	IV

*NOTE: A description of modification types may be found on page 47.

lowed by the interpretation of a second tape READ instruction (for the same tape), the execution of the second READ instruction will be delayed until the first record has been passed over. In the case of the COPY instruction, the electronic and mechanical equipment must be synchronized, and short delays may result for this synchronization to take place. In general, any execution delays of this type are of varying lengths, because they depend to a great extent on the programming.

CARD READER

CARDS ARE READ at the rate of 150 cards per minute. In continuous card reading, 292 milliseconds of the card cycle of 400 milliseconds are available for useful calculating. The difference, 108 milliseconds, is required for the execution of the COPY and READ instructions and appropriate time-margins for safe synchronization of mechanical and electronic components.

The maximum safe times available for useful calculating between executions of COPY instructions are indicated in Figure 10. For example, after execution of the 9-left COPY instruction, 540 microseconds are available before the 9-right COPY instruction must be given; and after execution of the 9-right COPY instruction, 15 milliseconds are available before the 8-left COPY execution. The READ instruction must be given in the hatched portion for continuous operation of the card reader. After the 12-right COPY execution of a card, however, it takes 20 milliseconds before the machine can execute a READ instruction for the next card. If the READ instruction, then, is given t milliseconds after the 12-right COPY, and if t is less than 20, the machine will compute (i.e., it will proceed with any intervening programs) for these t milliseconds. Upon receiving the READ instruction, however, the program will be delayed until 20 milliseconds (from the 12-right COPY) have elapsed. If the READ instruction is given after the interval of 20 milliseconds, the program will not be delayed. So to be able to compute for *all* of the available time between cards, *and* to keep the card reader in continuous motion, it is necessary to give the READ instruction between 20 and 70 milliseconds after the 12-right COPY instruction.

If the card reader is not in motion and a card READ instruction is given, the average elapsed time between the READ instruction execution and the first 9-left COPY instruction execution will be 270 milliseconds. However, only 50 milliseconds are available for calculation after the READ instruction execution.

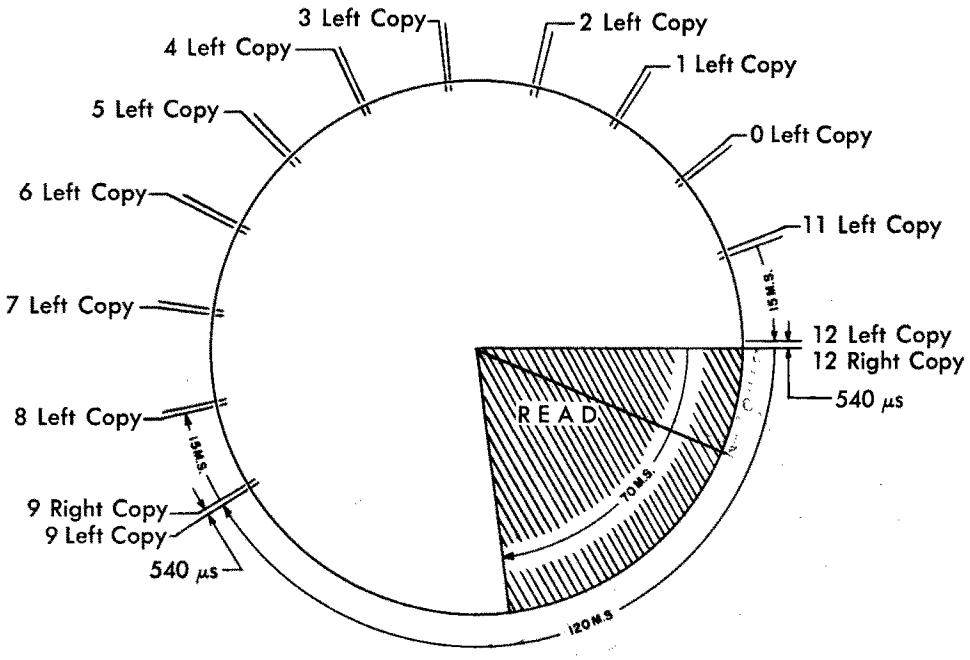


FIGURE 10

CARD PUNCH

CARDS ARE PUNCHED at the rate of 100 cards per minute. In continuous card punching 442 milliseconds of the card cycle of 600 milliseconds are available for useful calculating. The difference, 158 milliseconds, is required for the execution of the COPY and WRITE instructions and appropriate time-margins for safe synchronization of mechanical and electronic components.

The maximum safe times available for useful calculating between executions of COPY instructions are indicated in Figure 11. For example, after execution of the 9-left COPY instruction, 540 microseconds are available before the 9-right COPY instruction must be given; and after execution of the 9-right COPY instruction, 31 milliseconds are available before the 8-left COPY execution. The WRITE instruction must be given in the hatched portion for continuous operation of the punch. After the 12-right COPY execution of a card, however, it takes 25 milliseconds before the machine can execute a WRITE instruction for the next card. If the WRITE instruction, then, is given t milliseconds after the 12-right COPY, and if t is less than 25, the machine will compute for these t milliseconds. Upon receiving the WRITE instruction, however, the program will be delayed until 25 milliseconds (from the 12-right COPY) have elapsed. If the WRITE instruction is given after the 25-millisecond interval, the program will not be delayed on this account; but the punch will already have disconnected, and a delay results. Thus, to be able to compute for *all* of the available time between cards, *and* to keep the punch running at full speed, it is necessary to give the WRITE instruction at exactly 25 milliseconds after the 12-right COPY instruction.

If the card punch is not in motion, and a card WRITE instruction is given, the average elapsed time between the WRITE-instruction execution and the first 9-left COPY-instruction execution will be 400 milliseconds. However, only 70 milliseconds are available for calculation after the WRITE-instruction execution.

Not more than 24 COPY instructions can be given per card cycle.

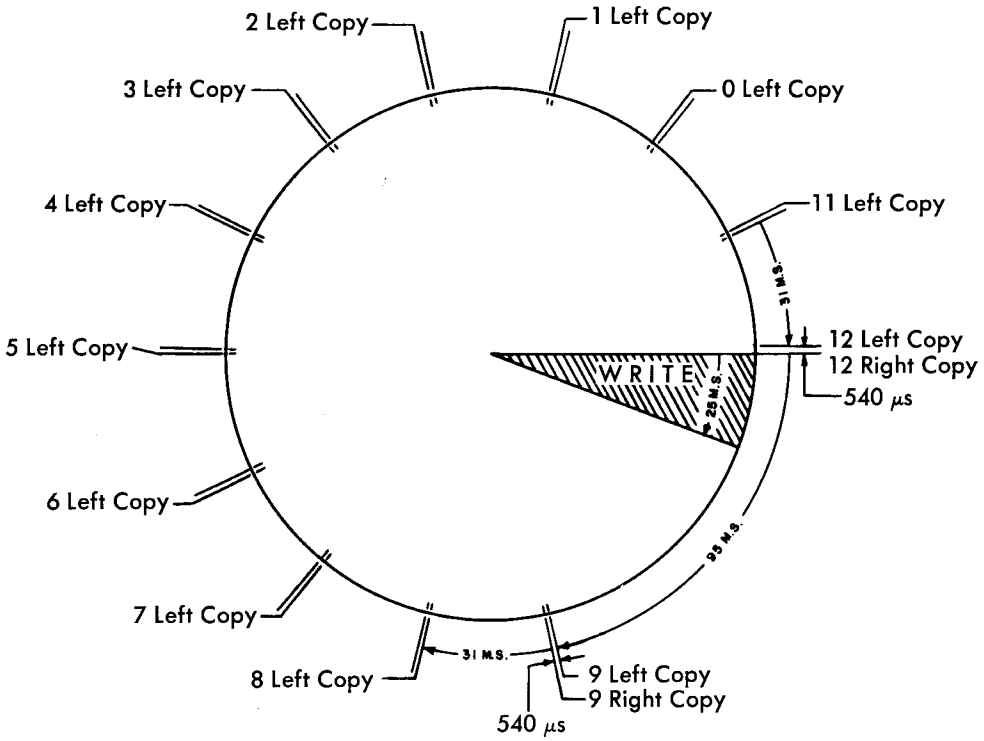


FIGURE 11

PRINTER

Without Echo Checking

Information is printed at the rate of 150 lines per minute. In continuous printing, 322 milliseconds of the print cycle of 400 milliseconds are available for useful calculating. The difference, 78 milliseconds, is required for the execution of the COPY and WRITE instructions and for appropriate time-margins for safe synchronization of mechanical and electronic components.

Maximum safe times available for useful calculating between executions of COPY instructions are indicated in Figure 12. For example, after execution of the 9-left COPY instruction, 540 microseconds are available before the 9-right COPY instruction must be given; and after execution of the 9-right COPY instruction, 13 milliseconds are available before the 8-left COPY execution. The WRITE instruction must be given in the hatched portion for continuous operation of the printer. After the 12-right COPY execution of a print cycle, however, it takes 16 milliseconds before the machine can execute a WRITE instruction for the next cycle. If the WRITE instruction, then, is given t milliseconds after the 12-right COPY, and if t is less than 16, the machine will compute for these t milliseconds. Upon receiving the WRITE instruction, however, the program will be delayed until 16 milliseconds (from the 12-right COPY) have elapsed. If the WRITE instruction is given after the 16-millisecond interval, the program will not be delayed. So to compute for *all* of the available time between print cycles, *and* to keep the printer running at full speed, it is necessary to give the WRITE instruction between 16 and 115 milliseconds after the 12-right COPY instruction.

If the printer is not in motion, and if a printer WRITE instruction is given, the average elapsed time between the WRITE-instruction execution and the first 9-left COPY-instruction execution will be 280 milliseconds. However, only 58 milliseconds are available for calculation after the WRITE-instruction execution.

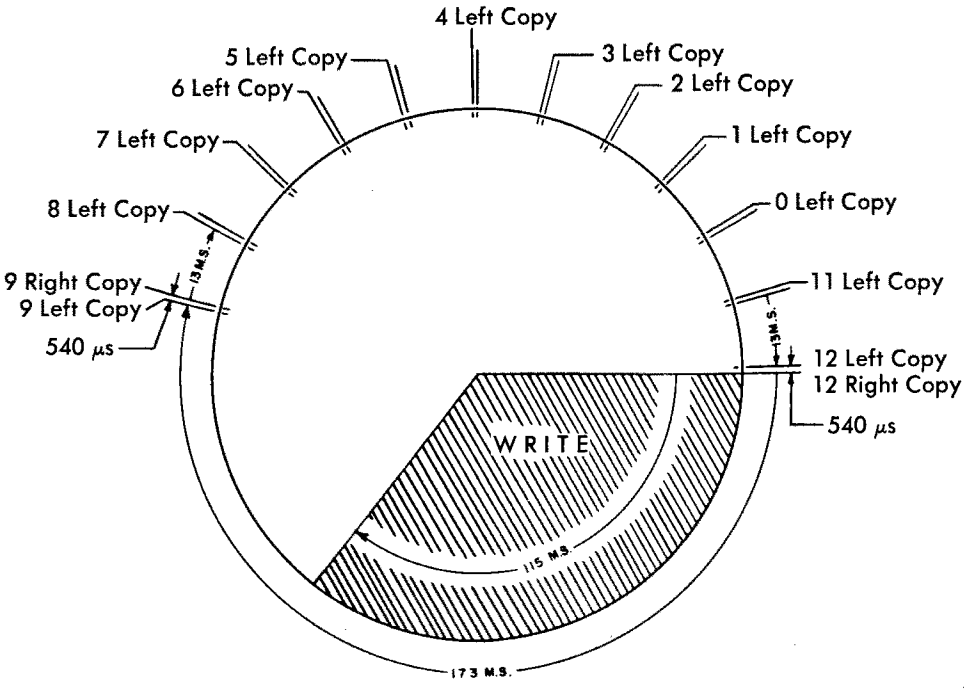


FIGURE 12

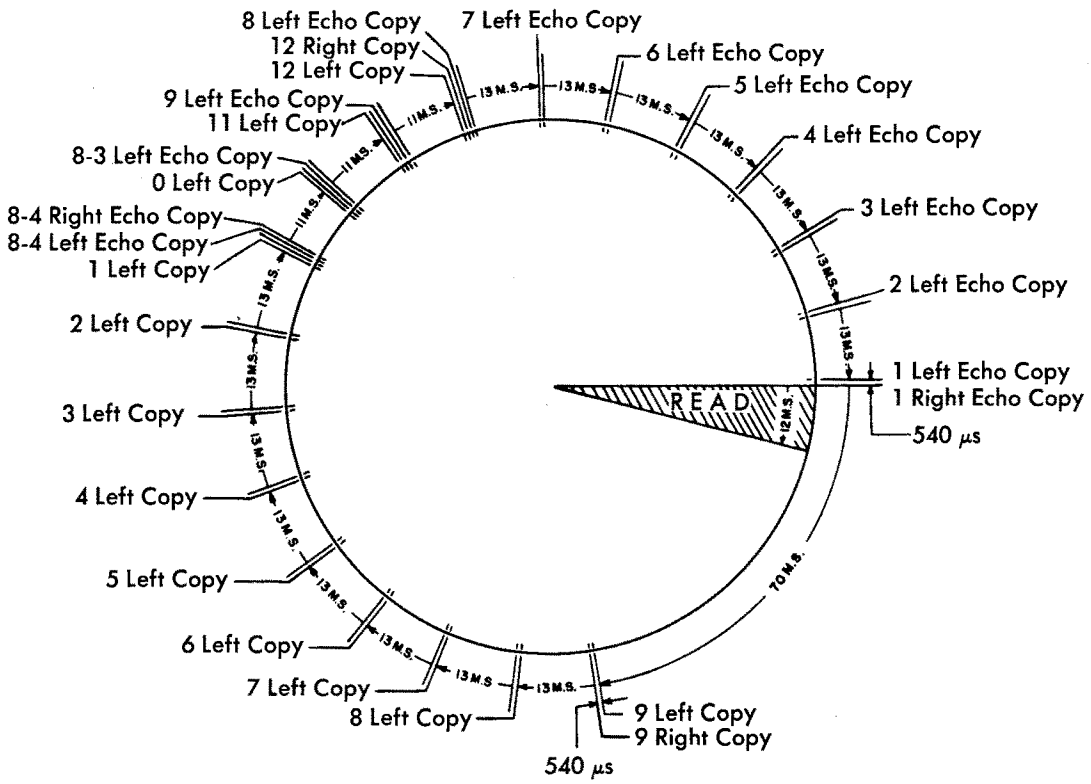


FIGURE 13

With Echo Checking

In continuous printing with checking, 313 milliseconds are available for useful calculating. Appropriate times are given in Figure 13. The READ instruction must be given in the hatched portion for continuous operation of the printer after the 1-right echo COPY execution of a print cycle; however, it takes 12 milliseconds before the machine can execute a READ instruction for the next cycle. If the READ instruction, then, is given t milliseconds after the 1-right echo COPY, and if t is less than 12, the machine will compute for these t milliseconds. Upon receiving a READ instruction, however, the program will be delayed until 12 milliseconds (from the 1-right echo COPY) have elapsed. If the READ instruction is given after the 12-millisecond interval, the program will not be delayed on this account; but the printer will already have disconnected, and a delay results. So to compute for *all* of the available time between print cycles, *and* to keep the printer running at full speed, it is necessary to give the READ instruction at exactly 12 milliseconds after the 1-right echo COPY instruction.

If the printer is not in motion and if a printer READ instruction is given, then the average elapsed time between the READ execution and the first 9-left COPY-instruction execution will be 280 milliseconds. However, only 58 milliseconds are available for calculation after the READ-instruction execution.

MAGNETIC TAPES

SUCCESSIVE full-words are written on, or read from, the magnetic tapes at a rate of about 1250 words a second.

Writing

The maximum safe calculating time between successive COPY executions is 700 microseconds. If the tape is not in motion and if a tape WRITE instruction is given, 6 milliseconds are available for calculating between the WRITE execution and the first COPY execution. If the WRITE instruction for a second unit record is given t milliseconds after the execution of the last COPY instruction of a first unit record, and if $t \leqq 6$, then $12 - t$ milliseconds are available for calculating between the WRITE and the first COPY executions of the second record (Figure 14). If $t > 6$, then 6 milliseconds are available. After tape rewinding, the maximum safe computing time between a tape WRITE execution and the first COPY execution is 500 milliseconds.

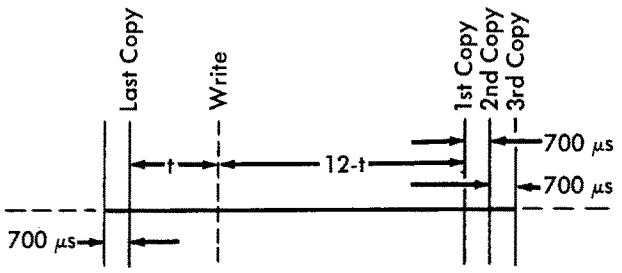


FIGURE 14

Reading

The maximum safe calculating time between successive COPY executions is 540 microseconds. If the tape unit is not in motion, and if a tape READ instruction is given, 4 milliseconds are available for calculating between the READ execution and the first COPY execution. If the READ instruction for a second unit record is given t milliseconds after the execution of the last COPY instruction of a first unit record, and if $t \leqq 4$, then $8 - t$ milliseconds are available for calculating between the READ and first COPY execution for the second record (Figure 15). If $t > 4$, then 4 milliseconds are available. After tape rewinding, the maximum safe computing time between a tape READ execution and the first COPY execution is 300 milliseconds.

These allowable computing times also hold for reading backward. When a reversal of tape motion is made (reading forward, followed by reading backward) the available computing time between the READ and COPY executions stated above may be increased by 7 milliseconds. After the writing of an end-of-file gap, the maximum safe computing time between a tape READ BACKWARD execution and the first COPY execution is 500 milliseconds.

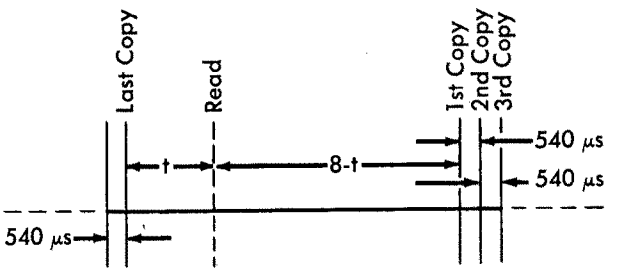


FIGURE 15

Summary

Note that the multiplier-quotient register must not be used in the calculating between the READ and the first COPY executions, or between successive tape COPY executions or for 1 millisecond following the execution of the last COPY instruction of a record. The delay instruction (WRITE 2052) may be used to insure the latter condition.

The following information may be used to find the over-all time required for the execution of a tape-reading or writing program. It should be kept in mind that these times should be used *only for estimations.* The times for safe calculating described above must be *strictly observed.*

1. If a tape is not in motion and if no reversal of previous motion is specified, 15 milliseconds is the maximum elapsed time between the execution of the READ or WRITE instruction and the execution of the first COPY instruction. The average elapsed time will be 10 milliseconds. If a reversal of the previous motion is specified, then the maximum time is increased to 27 milliseconds, and the average time is increased to 20 milliseconds.

2. If tape writing of several records on a single tape is in process, the maximum elapsed time between the last COPY execution of a given record and the first COPY execution of the next record is 25, or $15 + t$ milliseconds whichever is greater, where t is the time between the last COPY execution of the given record and the WRITE execution of the next record.

3. If tape reading of several records (with no reversal of tape motion) is in progress, the maximum elapsed time between the last COPY execution of a given record and the first COPY execution of the next record is 18, or $8 + t$ milliseconds, whichever is greater, where t is the time between the last COPY execution of the given record and the READ execution of the next record.

4. For small records (less than five words) 1 millisecond is a maximum elapsed time between COPY executions. For records of more than five words the average time between COPY executions approaches 800 microseconds.

5. After tape rewinding, the maximum elapsed time between a tape READ execution and the first COPY execution is 1.5 seconds. Average elapsed time is 0.9 seconds.

6. After tape rewinding, the maximum elapsed time between a tape WRITE execution and the first COPY execution is 1.2 seconds. Average elapsed time is 0.9 seconds.

7. After the writing of an end-of-file gap, the maximum elapsed time between a tape READ BACKWARD execution and the first COPY execution is 1.2 seconds. Average elapsed time is 0.9 seconds.

MAGNETIC DRUMS

SUCCESSIVE full-words are written on, or read from, the magnetic drums at the rate of about 800 words a second. More precisely, the average time of transmitting one word is 1.28 milliseconds. The maximum safe computing time between successive COPY executions is 1 millisecond.

Any number of instructions (other than input-output instructions) can be executed between the drum READ or WRITE and SET DRUM executions, and between the SET DRUM and the first COPY executions.

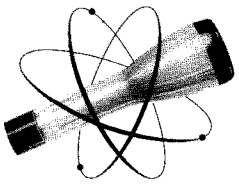
The execution of the first COPY instruction is delayed a length of time that depends on the drum address and on the elapsed time t between the READ or WRITE execution and the first COPY interpretation. The time delay T in milliseconds between the READ or WRITE execution and the first COPY execution may be determined approximately in this way:

Divide the drum address by 32, forming an integer quotient Q and a remainder R. Let $\bar{t}$ be 30 or t, whichever is greater. Then

$$T = \bar{t} + d + \frac{5R}{8} + \frac{Q}{100}$$

where d is a time delay depending on the orientation of the drum. The d is not predictable but will not exceed 20 milliseconds, and averages 10 milliseconds.

Thus, addresses 0, 32, 64, . . . are the most accessible addresses, and addresses 30, 62, 94, . . . are the least accessible addresses. For example, the formula above yields $T = 40$ milliseconds as the average access time to drum address 0000 if the COPY instruction is given (by the program) 30 milliseconds or less after the READ or WRITE instruction. If the programmer is careful to store information starting at an address that is a multiple of 32, the average access time is very close to 40 milliseconds. The average *random* access time is 50 milliseconds.



CONTROL PANEL WIRING

THE CARD READER, card punch, and printer of the 701 installation are modifications of the IBM Types 402, 521, and 407, respectively. The following descriptions take advantage of similarities to the standard machines. All new functions and features will be described in detail. Questions on how standard features operate can be answered by referring to the appropriate principles of operation manual.

In relation to the card reader and punch, note that a given COPY instruction can read or set up punching for only 36 columns of a given card row, because a COPY instruction can handle no more than a full word of 36 bits. The control panels for the card machines are supplied with 72 entries to (or exits from) the calculator, corresponding to two full words in memory or two half-rows on the card. Pulses from the card machine, synchronized with the passage of the card rows under the read brushes (or punch magnets), will cause first the left 36 entries (or exits) and then the right 36 entries (or exits) to be activated for each card row as the row passes under the read brushes (or punch magnets). The card columns can be wired to the calculator entry (or exit) hubs in any order. This section will assume that the first 72 columns of the card are used and that the entries (or exits) from the calculator are used in a normal left to right order.

It is important to remember when reading this section that familiarity with wiring of the standard machines, of which the 701 components are modifications, is assumed.

CARD READER

FIGURE 16 shows the hubs on the control panel of the card reader and the wiring necessary to provide for a direct transfer of information from cards in the card reader to the calculator when the proper set of instructions are provided by programming.

Hubs not described in the standard manuals or hubs with different names will now be discussed.

CONTROL BRUSHES: Equivalent to the second reading brushes of the Type 402.

READ BRUSHES: Equivalent to the third reading brushes of the Type 402.

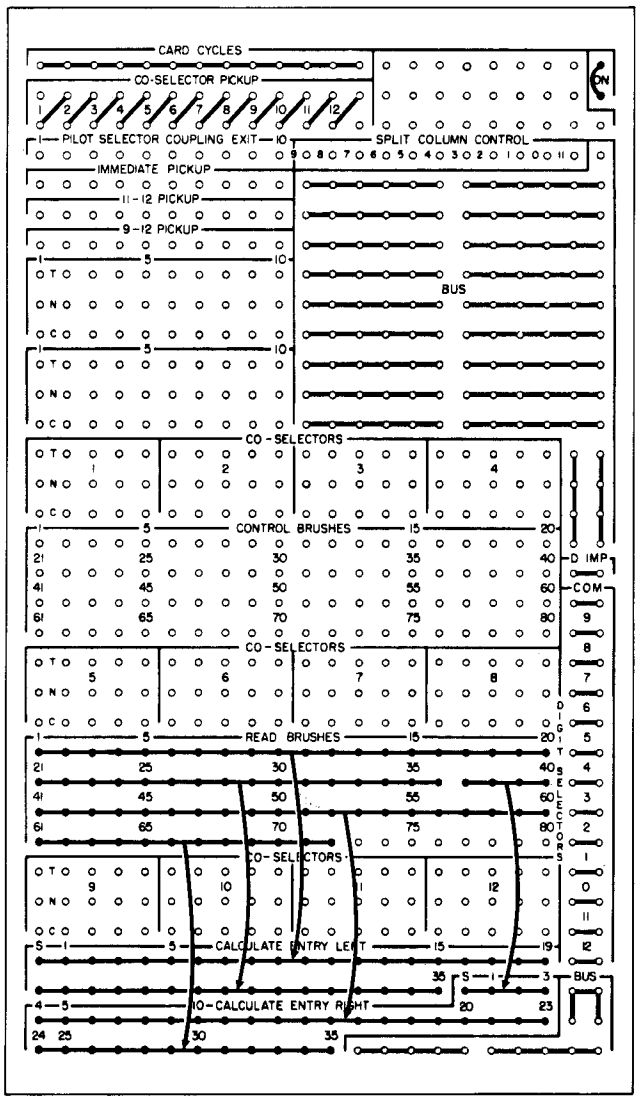


FIGURE 16

CALC ENTRY LEFT and **CALC ENTRY RIGHT:** Entry hubs for pulses entering electrostatic storage from rows of the card. Successive COPY instructions are associated alternately (by internal circuits) with first the left hubs then the right hubs, etc. Thus, with the wiring shown in Figure 16 it is seen that successive COPY instructions will cause half-rows of the card to be read in the sequence described under *Input-Output Components* (i.e., 9-row left, 9-row right, etc.). The S hub for each of these groups accepts a pulse to determine the sign of the full word being entered, while the remaining hubs numbered 1-35 correspond to the other 35 bits of the full word.

ON: These hubs must be connected for the card reader to operate as a component of the 701.

Conditional transfer of information is possible through the use of selectors. The pilot selectors (which work in the same way as the Type 402 pilot selectors) have three sets of pickup hubs: immediate pickup (equivalent to the *pickup* function of the immediate pickup and coupling exit of the 402); 11-12 pickup (equivalent to the *X* pickup of the 402); and the 9-12 pickup (digit pickup in the 402).

The pickups can be activated directly from punching in the card by wiring from control brushes to the appropriate pickup, as described in detail in the Principles of Operation of the Type 402 under the heading *Pilot Selectors*. In addition, each type of pickup can be activated by an impulse from a given set of the hubs described below. In most cases, whether a given pickup will be activated by a given hub can be decided on the basis of whether the equivalent 402 pickup would be activated. Cases that cannot be decided in this way will be described in detail under the appropriate heading (below).

By wiring the pilot selector coupling exits (similar to the coupling function of the immediate pickup and coupling exits of the 402) to appropriate co-selector pickups, one or more co-selectors can be picked up in unison with the pilot selector when the pilot selector is picked up with either the 9-12 pickup or the 11-12 pickup. In this case, the co-selector, once picked up, will hold through the next cycle in the same manner as the pilot selector.

The hubs that can activate the pickup hubs of selectors are, as on the 402, digit selector, split column control, and card cycles. These hubs emit pulses in exactly the same manner as the Type 402.

CARD PUNCH

FIGURE 17 shows the hubs on the card punch control panel and the wiring required for punching information directly from the calculator into a card, when the proper sets of instructions are executed by the program. The example provides that information from the calculator be punched in columns 1-72 of the card; but, as in the card reader, it is possible to punch *any* arrangement of not more than 72 columns by means of appropriate wiring. The hubs involved in wiring of Figure 17 will be explained.

PUNCH MAGNETS: These hubs have the same function as the corresponding hubs on the type 521.

CALC EXIT LEFT and **CALC EXIT RIGHT:** These are exit hubs for information being transmitted from electrostatic storage to the punch magnets. In relation to programming, the full word specified by the first COPY instruction following the WRITE instruction is available at the CALC EXIT LEFT hubs. The full word specified by the second COPY instruction is available at the CALC EXIT RIGHT hubs. The left and right hubs then alternate for the following COPY instructions. Because cards are fed 9's edge first, it is evident from the wiring in Figure 17 that the half-rows of the card will be punched in the sequence discussed under *Input-Output Components* (i.e., 9-row left, 9-row right, etc.).

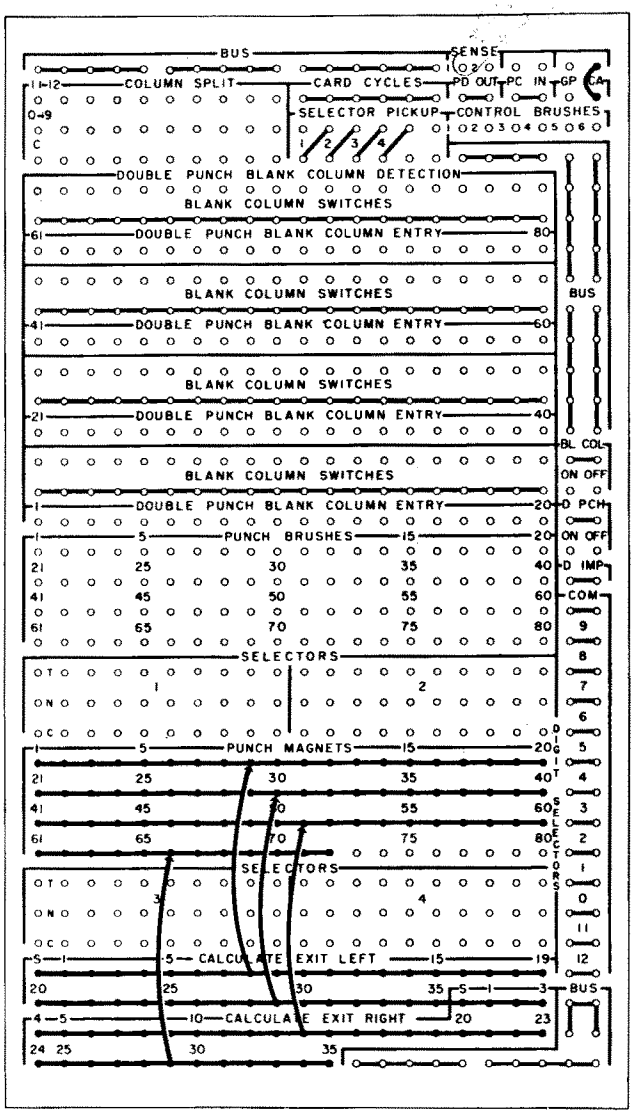


FIGURE 17

CA (calculate): These hubs must be wired to enable the punch to operate as a component of the 701.

The four selectors (two standard) of 10 positions each are picked up by means of the corresponding selector pickups, 1-4. Activation of the pickups directly from the control brushes will be discussed later, using gang punching for illustration purposes. The selectors can also be picked up by electrical impulses available at the two SENSE output hubs in the upper-right section of the control panel.

SENSE output hubs: Wiring a sense output hub to a selector pickup will allow a programmed SENSE instruction with the proper address, to effect the transfer of a selector. With reference to Figure 11, a SENSE instruction given at any time after the first COPY of a given cycle, and at least *32* milliseconds before the first COPY of the *next* cycle, will cause the selector to be transferred shortly after the instruction is given (between 15 and 30 milliseconds later). The selector will then stay transferred until 20 milliseconds after the last COPY of that next cycle. In normal usage the SENSE instruction is given soon after the WRITE instruction that initiates the cycle in which the selector is to be picked up. When all rows of a card are not being punched after a WRITE instruction, any gang punching or emission of digits directed through selectors, which in turn are controlled by sense exits, requires special precautions to insure that the selector is transferred *only* during the cycles desired. As pointed out in Table II, the address of sense output hub number 1 is 1024, while number 2 is 1025.

All other hubs on the panel operate in the same manner as on the standard Type 521.

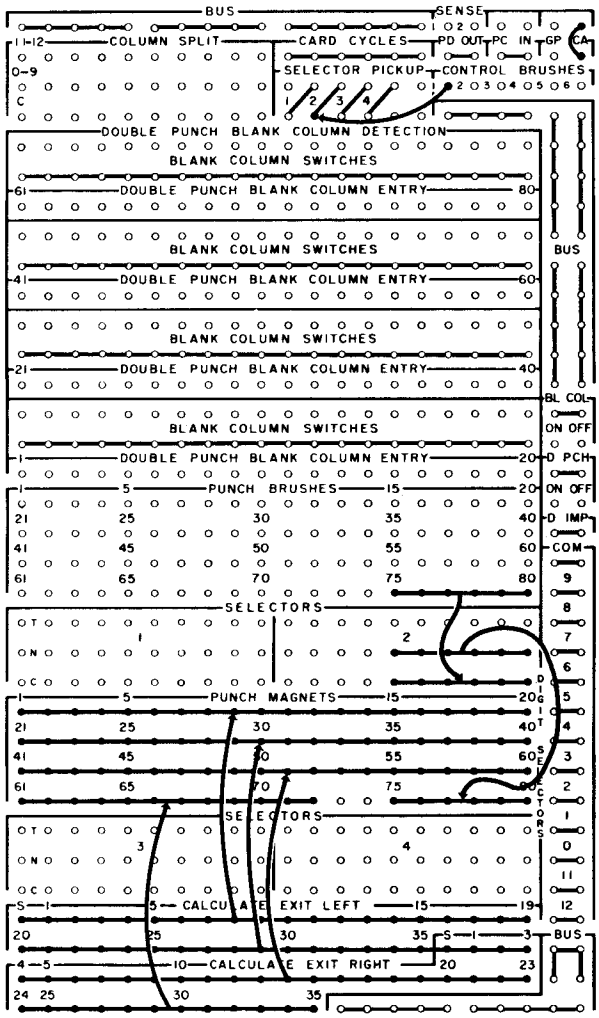


FIGURE 18

The card punch can be used for gang punching cards under calculator control as well as independently of it. Figure 18 shows the wiring necessary for gang punching with interspersed master cards under calculator control (for this example the information to be gang punched is assumed to be in columns 75-80 of the card).

The control panel wiring as shown in Figure 18 is:

1. Hubs S-35 of the calculator exit left hubs followed by hubs S-35 of the calculator exit right are wired to hubs 1-72 of the punch magnets.
2. Columns 75-80 of the punch brushes are wired to five consecutive common hubs of selector 2. The corresponding normal hubs are wired to columns 75-80 of the punch magnets.
3. The hub for control brush 1 (positioned at the column in which the control punch will be) is wired to selector pickup 2.
4. The CA hubs are connected.

If it is desired to gang punch cards independently of the calculator, the GP (gang punch) hubs should be connected instead of the CA hubs.

If the program should call for the card punch to operate with the GP hubs wired, the calculator will stop, because no control by the program is possible with these hubs wired.

Offset gang punching can be done in the normal way by wiring the appropriate control brush to the PC (punch control) hub and wiring the PD (punch delay) hub to the selector pickup. To prevent punching in the master cards when offset gang punching, two selectors are necessary: one wired as above to provide for offsetting; the other wired to prevent punching the master card. For a more detailed account, see the section on *Offset Gang Punching* in the manual for the Type 521 reproducing punch.

PRINTER

THE PRINTER is used to print information contained in electrostatic storage. It should be emphasized that this printed material can be any combination of several characters. Some of the characters that can be made to print by means of impulses from electrostatic storage are decimal digits, letters of the alphabet, punctuation marks, dollar signs, etc. The IBM code for printing these characters is given in Table V. For example, a dollar sign ($) may be printed by arranging impulses (from electrostatic storage) to arrive at

TABLE V

CHARACTER CODE FOR THE 701 PRINTER

ZONE / DIGIT	No (N) Zone	12 (Y) Zone	11 (X) Zone	0 Zone
No. Digit	*	+	—	0
1	1	A	J	/
2	2	B	K	S
3	3	C	L	T
4	4	D	M	U
5	5	E	N	V
6	6	F	O	W
7	7	G	P	X
8	8	H	Q	Y
9	9	I	R	Z
8-3	+	.	$	,
8-4	—	□	*	%

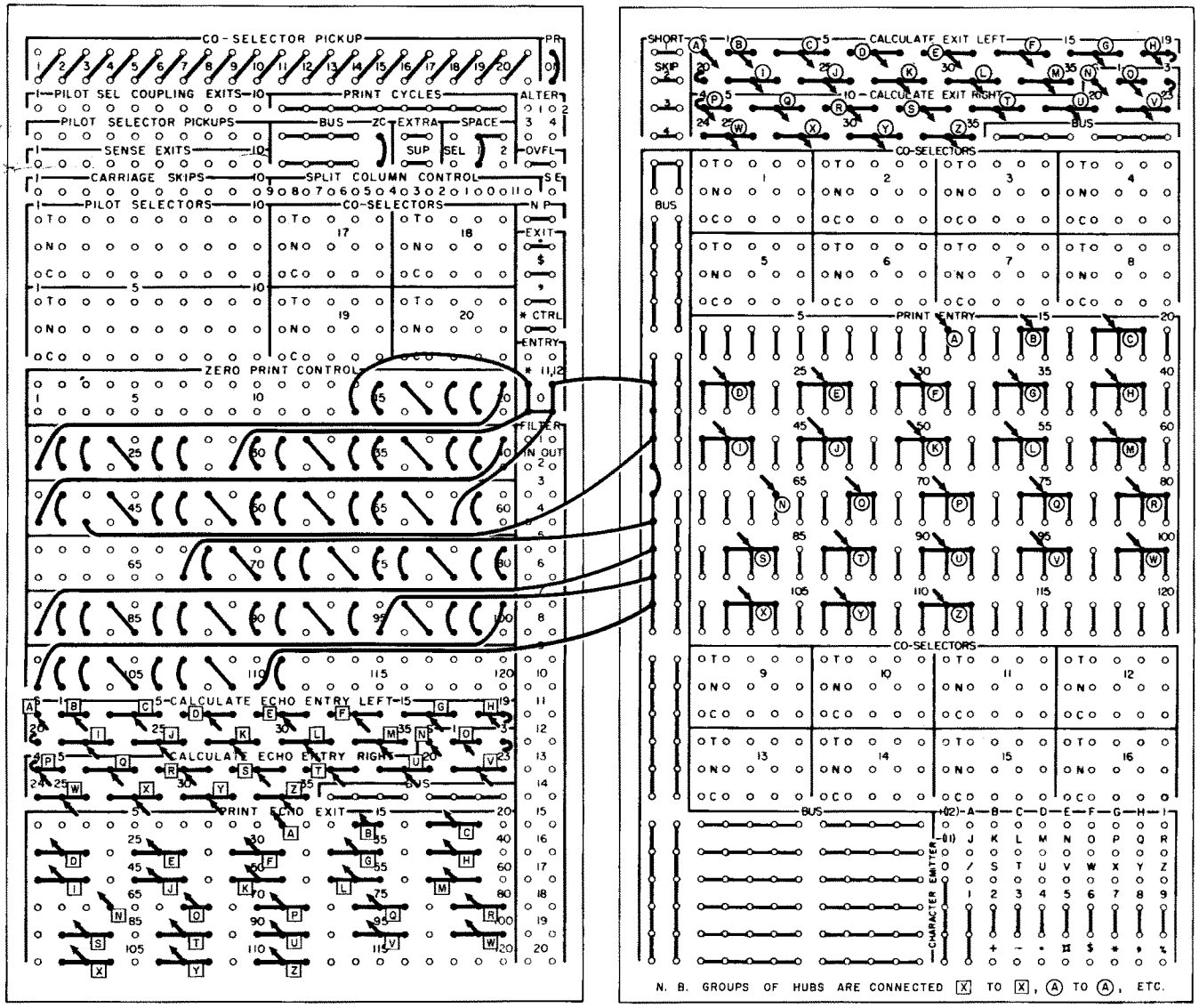


FIGURE 19

a print wheel at 11-time, 8-time, and 3-time of the print cycle.

Figure 19 shows the control panel wiring for the printer, with the wiring for an example to be explained later. The hubs used for this example will now be explained in general terms.

CALC EXIT LEFT and **CALC EXIT RIGHT:** These hubs are exits for words being sent from electrostatic storage to the printer by COPY instructions. These two sets of hubs alternate with the COPY instructions in the same way as similar hubs on the card punch.

PRINT ENTRY: These are entry hubs for impulses to the individual type-wheels from electrostatic storage.

PRINT ECHO EXIT: These hubs are exits for the echo pulses generated by the type-wheel according to the character they are positioned to print.

CALC ECHO ENTRY LEFT and **CALC ECHO ENTRY RIGHT:** These are hubs that can accept the echo impulses generated by the type-wheels. The COPY instructions in the stored program then direct these impulses to electrostatic locations in preparation for a programmed check. The left and right hubs alternate similar to the card reader.

PR, ON: These hubs must be connected if the printer is to be used as a component of the 701.

ZC: If these hubs are wired together, the zero print control function behaves exactly as in the 407. If these hubs are not wired, the type-wheels will print *only* if impulsed through the print entry hubs.

Printing Control

EXAMPLE: The control wiring of Figure 19 provides for printing any digit (including zero), letter, or special character that has been coded in the card image being copied. The wiring also provides for echo checking of the digits 9 through 1 in all positions except 11 and 64. In positions 11 and 64, provision is made for checking the special codes corresponding to the plus and minus signs (8, 3 and 8, 4). Printing from the card image will occur when the calculator executes a program similar to the one explained under *Input-Output Components*. Although this wiring is valid for printing any character, assume that binary numbers will be printed here.

In the specific example, characters being printed are separated into groups of three each to facilitate translation from the binary to the octal system. Any other arrangement could be accomplished.

The control panel wiring shown in Figure 19 is as follows:

1. The calculator exit hubs (two sets of hubs labeled S, 1-35; the left half-row and the right half-row, respectively) are wired to the print entry hubs in this order:

	CALC EXIT	PRINT ENTRY
	S	11
	1, 2	14, 15
Left half-row	3, 4, 5	17, 18, 19
	6, 7, 8	21, 22, 23
	.	.
	.	.
	.	.
	33, 34, 35	57, 58, 59
	S	64
	1, 2	67, 68
Right half-row	3, 4, 5	70, 71, 72
	.	.
	.	.
	.	.
	33, 34, 35	106, 107, 108

2. The wiring from the calculator echo entry hubs (two sets of hubs labeled in the same way as the calculator exit hubs) to the print echo exit is the same as that described in paragraph 1 above, if the words "calculator echo entry" and "print echo exit" are substituted for "calculator exit" and "print entry," respectively. Use of echo pulses is explained below.

3. The ZC hubs are wired. This allows the zero print control to operate in the same way as on the 407. Note that *all* of the zero print control wiring, described in the next paragraph, could be eliminated if the ZC hubs were not wired. The zero print control is brought into play here only to illustrate appropriate wiring.

4. The pairs of zero print control hubs (described below) are numbered to correspond to the print entry hubs. All pairs of zero print control hubs corresponding to the print entry positions enumerated in paragraph 1 will be connected (i.e., the upper hub at a position is wired to the lower hub at the same position) *except* the pairs that are at the first of a subgroup (the hubs corresponding to positions 11, 14, 17, 21, . . . 57, 64, 67, 70, . . . 106). The lower hubs at positions 14 and 67 will be wired to the zero entry. A *further exception* to the system described above occurs because the zero print control hubs should not be wired in groups of greater than 10 pairs of hubs. To separate the zero print control wiring into independent groups of appropriate size, the lower hubs of some positions are wired to zero entry. The remainder of the zero print control hubs at the first of a group will be connected diagonally from the lower hub of the pair to the upper hub immediately at the left (this corresponds to a blank position in the printing).

5. The PR and ON hubs are coupled.

6. The space hub is wired to 1 to provide single spacing between lines of print.

To check the printing of a number and its sign, the print echo exits corresponding to the print wheels that printed the number should be wired to the calculator echo entries corresponding to the calculator exits from which the information was originally taken. It is then possible, by means of programming, to read these impulses into electrostatic storage and perform a programmed check on the original information.

In general the program for this checking relies upon the fact that the echo pulses occur in a given order: 8-4, 8-3, 9, 8, 7, 6, 5, 4, 3, 2, 1. Each print wheel emits an echo pulse timed to indicate the sector within which it was set up to print. Since no provision

is made for checking the zones within these sectors, the checking is restricted to numerical printing. For example, at 8-echo time, the print echo exits, corresponding to the print wheels set up to print in the 8 sector, will emit a pulse. The program will copy these pulses into memory in the form of a binary word which can then be compared with the word in the card image corresponding to the 8-row.

ZERO PRINT CONTROL: By means of the zero print control hubs on its control panel, the 701 printer can provide any one of a number of responses to zeros or any symbol not having a digit pulse. Each pair of zero print control hubs corresponds to a print entry position; the manner in which the pair of hubs is wired to its neighbors controls the printer's response. Zero print control functions only during zone (0, 11, 12) time and *N* (no-zone) time and can have no effect upon the printing in a position that has received a digit impulse (1 through 9) during a given print cycle. Thus, the only special characters that can be controlled are those consisting only of zone pulses (the zero, check-protecting asterisk, plus sign, and minus sign) or those emitted from special hubs on the control panel (the dollar sign, period, and comma). The specially emitted symbols provide for setting the print wheel to the proper sector without use of the usual combination digit punching (as an 8 and a 3 in the case of the dollar sign, period and decimal point, and comma). The dollar sign, period, and comma can be printed with the usual combination punching, of course, but in this case the symbol cannot be controlled by means of zero print control.

Note that zero print control hubs cannot operate correctly when used in groups of more than ten at a time. Groups larger than this should be split and wired independently.

Examples and applications of zero print control are given under the heading *Zero, Comma, Decimal and Dollar Symbol Control* in the Type 407 principles of operation manual.

FILTER IN-OUT: These hubs permit the passage of an impulse in only one direction—into the IN hub and out of the OUT hub. The OUT hub of one filter should not be wired to the IN hub of another filter, either directly or indirectly.

The printer has ten pilot selectors, each of which is picked up by an associated one of the pilot selector pickups (these pickups are similar to the IMMEDIATE PU hubs on the Type 407). In addition, there are 20 co-selectors, each with two identical co-selector pickups. The co-selectors can be picked up in unison with given pilot selectors by wiring the appropriate pilot selector coupling exits to the co-selector pickups, or the co-selector can be picked up independently by direct wiring from other hubs. The pilot selectors and co-selectors are similar, *in action,* to the pilot selectors and co-selectors of the card reader. Hubs that provide pulses to activate the selectors through their pickups are:

ALTERATION SWITCHES: These switches function in the same way as the 407 alteration switches, by emitting a pulse every machine cycle when the corresponding toggle switch on the printer has been turned on. These pulses can be used to pick up either pilot selectors or co-selectors.

SPLIT-COLUMN CONTROL: These hubs perform the same function as the 407 split column control. The numbers on either side of a given hub of the split column control refer to the corresponding print times. A selector pickup wired from a given one of these hubs (the hub between numbers 8 and 7) will cause the selector to be transferred between the corresponding print times (the selector would be transferred after 8-time and before 7-time).

PRINT CYCLES: These hubs are similar in use to card cycles of the Type 407. A pulse is emitted from these hubs during every machine cycle of the printer.

SENSE EXITS: Exits 1 through 10 are addressed by the numbers 0512 through 0521, respectively (Table II). By programming a SENSE instruction with one of these addresses, an impulse is made available at the corresponding exit hub. This pulse can then be used to pick up pilot selectors. If the exit is wired in this way, the normal usage is as follows: if the SENSE instruction is given during the hatched portion of Figure 12, the pilot selector will be transferred for the duration of the cycle initiated by the WRITE instruction that also is given during the hatched portion of Figure 12. SENSE instructions given at times in the machine cycle other than those specified above may have no effect. (Additional uses of sense exits are discussed below under *Carriage Controls*).

Carriage Control

The carriage of the printer is usually controlled by means of a punched paper tape (the control tape) used in combination with the ten sense exit hubs.

(For some simple applications, such as line-by-line printing, the carriage can be directly controlled without the use of the control tape.) In brief, the control tape is utilized as follows:

The tape is cut to the length of the form to be used (and later glued into a loop to provide for repetitive operation); punched holes in the tape thus correspond to positions on the form. When the carriage is in operation, the tape advances in synchronism with the form. An impulse to a given carriage skip hub (number 4) will cause the form to skip until the control tape—and consequently the form—reaches the position where there is a punched hole in the channel (column) corresponding to the impulsed hub (channel 4).

For a detailed description of the above points and for a further description of the carriage manual controls (restore key, space key, etc.), see the principles of operation manual for the Type 407.

The hubs listed below, when impulsed from the carriage control hubs, activate the various carriage skips and spacings. It should be kept in mind, when reading the descriptions of these hub functions, that an automatic space is *initiated,* but *not* automatically terminated, before each line of printing, except before printing the first line immediately after skipping. Before the first line of printing or in the cycle immediately after a skip, no normal spacing takes place.

CARRIAGE SKIPS: These hubs are similar to the D hubs of the Type 407 carriage skips. For example, a pulse to carriage-skips hub 1 will initiate a form skip that terminates when the first punch in channel 1 passes under the control-tape read brushes. In general, a hole punched in a given channel of the tape stops the form at a predetermined position after a pulse to the corresponding carriage-skips hub has started a form skip. The ten channels in the tape (in conjunction with the ten corresponding carriage skips) provide for an almost unlimited number of combinations of such skips. Because tape length and form length are equal, it is easy to make the punches in the tape correspond to the predetermined positions on the form.

By wiring sense exits to carriage skips and by punching the various channels in the tape to correspond to various sequences of printing on the forms, it becomes possible for stored programs (in electrostatic memory) to maintain an extremely flexible control over the printed output.

SHORT SKIP: These hubs are similar to the short-skip hubs of the Type 407. The short-skip hubs provide for skipping with no interruption of printing. The hubs can be used wherever there occurs an overflow of less than one inch or a regular skip of less than two inches. Any impulse used to initiate a short skip (e.g., a sense-exit impulse used to cause a skip of less than two inches) should be wired first to a short-skip hub and from there to its ultimate destination (a carriage-skip hub). As a result of such wiring, printing will continue at the normal rate of 150 lines per minute during short skips.

SPACE-SEL (selective space): These hubs are similar to the selective space hubs of the Type 407. When connected, the two selective space hubs allow spacing of less than seven lines to be selectively controlled by punches in channel 11 of the control tape. The action is such that, before each line of printing, spacing is automatically started; a punch in channel 11 then stops the spacing. For spacing of less than three lines it is necessary only to connect the selective space hubs and punch the desired positions in the control tape. For spacing of more than three lines (but less than seven lines), it is also necessary to impulse the space suppress and extra space hubs from print cycles (space suppress and extra space are discussed below).

EXTRA (extra space): These hubs are similar to the extra-space hubs on the Type 407. These hubs are usually used in conjunction with the space 1 or the space 2 hub. When space 1 is wired and an extra-space hub is impulsed (by a print cycles pulse, for instance), an extra single space results. With space 2 wired, an extra double space results.

SUP: Similar to the space suppress hubs of the Type 407. If one of these hubs is impulsed (by a print cycles for instance), all normal spacing will be suppressed during the cycle in which the hub was impulsed.

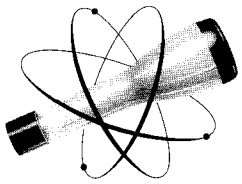
NP: Similar to the non-print hubs of the Type 407. The NP hubs will suppress both printing and spacing, regardless of other control panel wiring, for the cycles in which they are impulsed by a print cycles pulse.

The three types of hubs to be described below may be used to control the carriage. It should be noted that impulses available from these hubs may be directed through various selectors to provide controlled variations in form spacing from print cycle to print cycle.

SENSE EXITS: A given one of these hubs emits a pulse when a SENSE instruction, with the appropriate address, is executed. (See the previous discussion of sense exits in this section.) These hubs can be wired to carriage skip hubs, thus providing the stored program in the calculator with a means of controlling form spacing. When the sense exits are to be used for this purpose, the corresponding sense instruction should be given *immediately* after the WRITE instruction that starts the print cycle.

OVFL: Similar to the Type 407 overflow hub. This hub emits a pulse whenever a punch in channel 12 of the control tape passes the control tape reading brushes. The pulse emitted by this hub lasts (Figure 12) through the hatched portion of the cycle, after the cycle during which overflow has occurred (this fact will be of use in discussion of the sense-entry hub below). The overflow hub is often wired to a carriage-skips hub, thus providing a skip from the position at which channel 12 was punched to the position of the first punch encountered in the channel corresponding to the carriage skips hub. Such wiring is usually used to overflow to another form when the bottom of a given form is reached.

SE (sense entry): The sense-entry hub is an input hub on the printer control panel that can be sensed by a SENSE instruction with address 0522 (see Table II). If during the execution of such a SENSE instruction, the sense-entry hub is being impulsed, the SENSE instruction will skip over the next instruction in the stored program. If the hub is *not* being impulsed when the SENSE instruction is executed, the stored program will continue without skipping. The sense entry is intended primarily to be used with the overflow hub to inform the stored program when a form overflow is occurring. To accomplish this, the overflow hub is connected to the sense entry hub and a SENSE instruction is given sometime within the portion of the print cycle in which the overflow hub is emitting a signal (see above paragraph).



MANUAL CONTROL OF COMPONENTS

CARD READER

To PREPARE the card reader for control by the calculator, once the control panel is in place, it is necessary only to fill the hopper with cards and hold the start button until the ready light goes on. Figure 20 shows the card path through the card reader, and indicates the relative locations of the card levers, contacts, and reading brushes as cards move through the reader under control of the stored program. After the card reader has been prepared for calculator control and the ready light is on, there will be two cards in the reader, and all three card contacts (upper-card lever, lower-card lever, and hopper contact) will be closed.

Buttons and Lights

Start Button: Serves to run-in cards initially and to turn control of the card reader over to the calculator. The button is operative only if: the power is on, no fuses are blown, there is no card-feed failure, the stacker is not full, the control panel is in place, and the control panel calculator switch is wired ON.

If there is no card waiting ahead of the read brushes, pressing the start button causes the card feed to operate for one or more card cycles until either the button is released or until the card enters the station just ahead of the read brushes. When the first card reaches the station ahead of these brushes, the start button causes control to be turned over to the calculator and the ready light to go on.

If there is a card waiting ahead of the read brushes, pressing the start button merely turns control over to the calculator, and the ready light is turned on.

If there are no cards in the hopper or in the card feed ahead of the read brushes, pressing the start button turns on the running light and allows the calculator to set up an end-of-file condition.

While the ready light is on, the start button cannot be used to feed cards.

Stop Button: Causes the calculator to lose control over the card reader, and turns off the ready light. If a card is being read at the time the stop button is pressed, the action is delayed, and the card reader does not stop until the end of the current card cycle. The calculator will then be held up on the next copy instruction that refers to the card reader.

Feed Button: Permits cards to be run out of the card feed manually when the card reader is *not* under control of the calculator.

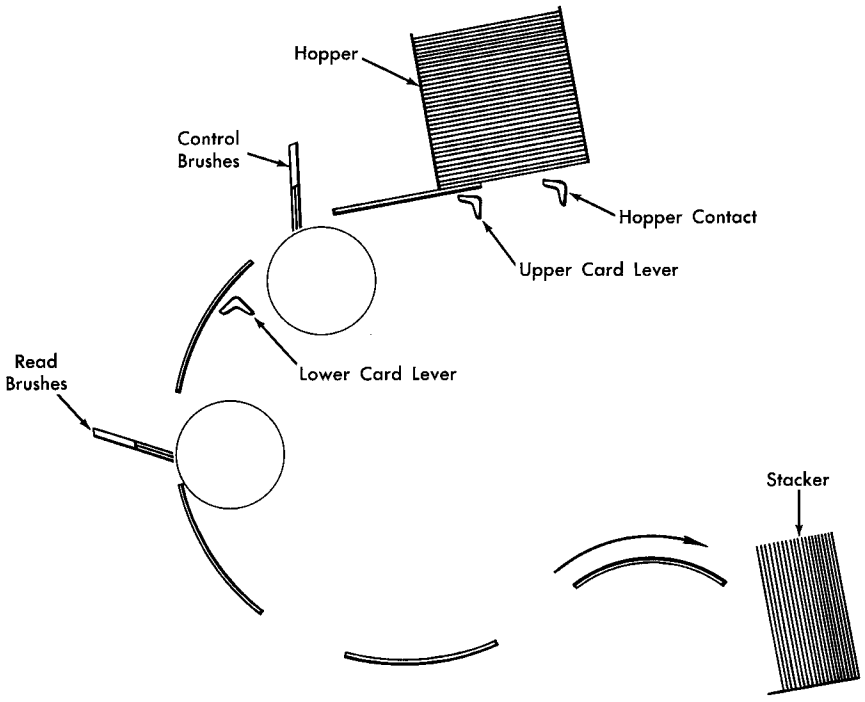


FIGURE 20

If the power is on, no fuses are blown, the stacker is not full, and the ready light is off (indicating that the calculator does not have control), pressing the feed button causes the card feed to operate for one or more card cycles until the button is released.

While the ready light is on, the feed button is inoperative. The stop button may be used to turn off the ready light in order to operate the feed button.

Ready Light: Indicates that the card reader is under control of the calculator. The ready light is turned on by the start button. It is turned off as follows:

1. By the stop button.
2. When the lower card lever is open at the end of a card cycle.
3. When the hopper contact opens at the end of a card cycle (after which it may be turned on again by means of the start button).
4. When there is a card-feed failure.
5. When a fuse is blown.
6. When the power goes off.
7. When the control panel is removed.
8. When the stacker is full.

The hopper contact opens when the hopper runs out of cards. This turns off the ready light and stops the card reader. The card reader can be started again by pressing the start button, regardless of whether more cards meanwhile were placed in the hopper.

Select Light: Goes on when the calculator gives a READ instruction for this card reader. The light goes off when the card cycle called for by the READ instruction has been executed.

Card-Feed Stop Light: Is on whenever there is a card-feeding failure.

Power-on Light: Indicates that the DC power is on in the card reader.

Fuse Light: Indicates a blown fuse, if the main power is still on.

Card-Feed Failure

When a card-feed failure occurs, the card-feed stop light is turned on. The start button is inoperative until a certain procedure is accomplished. The procedure:

1. Remove all cards from the hopper. (Note that this opens the hopper contact and turns off the ready light.)
2. Run out the cards in the feed by means of the feed button.
3. Then press the stop button.

The last card in the stacker will not have been read.

The stop button will not reset the card-feed stop light if there are still cards in the hopper or in the feed ahead of the read brushes.

End-of-Cards Procedure

When the last card in the hopper is fed, the hopper contact (Figure 20) opens, the calculator stops, and the READY light is turned off.

If, at this point, there are more cards for the card reader to read, it is necessary only to reload the hopper and press the start button. The calculator will then read the cards in the hopper as if they were a continuation of the previous sequence of cards.

If, on the other hand, the card hopper is left empty when the start button is pressed, it is an indication that the end of the card file has been reached. In this case, pressing the start button will again return control to the calculator, but as the last of the remaining cards passes the read brushes, the calculator sets up an end-of-file condition; this provides a means of control as described under *Input-Output Components.*

CARD PUNCH

BUTTONS and lights on the punch are similar to the corresponding controls on the card reader. Consequently, the discussion of punch controls will be of a general nature. Details peculiar to the punch, however, will be explicitly discussed. Note particularly that there is no end-of-file condition on the punch.

To turn control of the card punch over to the calculator (once the control panel is in place with the calculator hubs connected) the hopper is filled with cards, and the start button is held until the ready light goes on. There will now be one card in the punch; the hopper contact and the die-card-lever contact (Figure 21) will be closed. When an appropriate program is executed by the calculator, the first card that was in the hopper will now be punched with information from electrostatic storage.

The path of the cards through the punch is shown in Figure 21, along with the relative locations of the card levers, brushes, and punches. If, for any reason, one of the card contacts is not closed, the ready light will be turned off. The ready light is also turned off by any of the following conditions in the card punch: power off, blown fuse, full stacker, control panel not inserted, gang-punch switch wired, or a double punch or blank column if the panel is so wired.

If the control panel has been so wired, the double punch or blank column in any of the columns being checked will turn on the double-punch blank-column light and turn off the ready light. To reset the double-punch blank-column light, press the stop button. Control can then be returned to the calculator by pressing the start button.

If the gang-punch (GP) switch on the control panel is wired, the ready light cannot be turned on; so the calculator will have no control over the card punch. The punch can now be used as a gang punch in the usual way.

If the control panel has been wired for gang punching with the calculator switch on the control panel wired, the operation will depend upon the condition

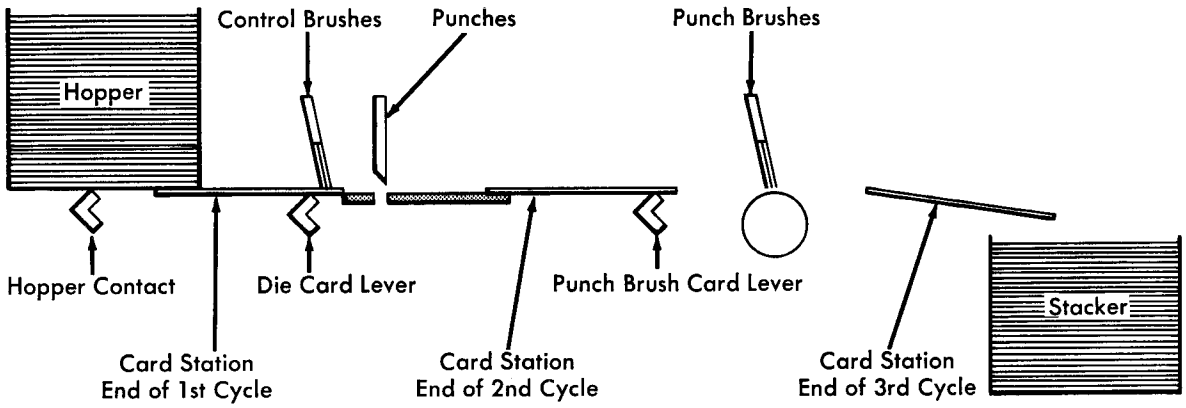


FIGURE 21

of the ready light. If the ready light is *on*, then each card fed under calculator control will be gang punched in accordance with the panel wiring. (The feed button under this condition is inoperative.) If the ready light is *off*, then gang punching will take place as long as the feed button is depressed; the feed button must be held down, for as soon as the feed button is released, the punch stops at the end of the next card cycle.

Should the card punch run out of cards, the hopper contact opens, and the ready light is turned off.

PRINTER

BUTTONS and lights on the printer are similar to those on the card reader, with the following exceptions:

1. The feed button is replaced by a print-cycle button.
2. The card-feed stop light is replaced by a form-stop light.
3. The STOP BEFORE PRINTING, TEST, and FORM STOP switches are added.

Consequently, a general discussion of printer controls will be discussed with special emphasis on the different features.

To prepare the printer for control by the calculator —once the control panel is in place—it is necessary only to hold the start button until the ready light goes on. The ready light will be turned off by any of the following conditions: test switch on; a form stop, indicated by the form-stop light, when the form-stop switch is on; depressed stop button on the carriage; depressed stop button on the printer; power off; blown fuse. (The test switch is discussed below.) If the form-stop switch is on, the form-stop light goes on when the printer runs out of paper. Other carriage controls are similar to controls on the Type 407 carriage.

As in the card reader and card punch, the printer stop button gives the operator a means of holding up printing whenever he so desires. The carriage stop button has an equivalent effect.

Turning the test switch on causes the ready light to go off.

The print cycle button will start a print cycle only under the following conditions:

1. When the ready light is off (as when the test switch is on).
2. When the ready light is on, the stop-before-printing switch is on, and the program supplies a READ or WRITE instruction for the printer.

With the test switch *on*, the ready light is off. Depressing the print cycles button will cause the printer to go through print cycles until the button is released. If it is desired to switch control back to the calculator, the test switch must be turned off and the printer start button pressed.

With the STOP BEFORE PRINTING switch on, the printer, after being selected by a READ or WRITE instruction in the program, is held up at the first COPY instruction. Depressing the print cycles button will cause the printer to print one line and the calculator to proceed with the program until the next group of output instructions for the printer are ready to be executed.

MAGNETIC TAPE UNITS

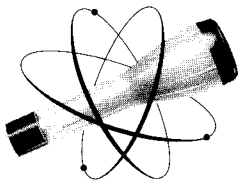
AFTER a new tape reel has been inserted, the forward direction button is pressed. (This button sets up the tape unit to run forward on manual operation of the load-unload-rewind button or the feed button.) Then pressing the load-unload-rewind button causes the tape to advance to the proper starting point for automatic operation; at this point the tape stops automatically. Closing the door of the tape unit permits the operator to turn control of the tape unit over to the calculator.

To unload a tape after it has been used, the door is opened (this prevents the calculator from interfering with manual operations) and the BACKWARD direction button is pressed. If the tape is not already rewound, this can be done by use of the load-unload-rewind button that is set up for rewinding or unloading by the BACKWARD button. After the tape has been rewound to the unload position, the tape can be removed from the tape unit.

As a safety precaution, the tape drive is completely disabled while the tape stop bar is depressed. This prevents any accidental operation of the buttons.

The door of the tape unit should not be opened while the tape unit is in automatic operation. This may wreck the problem being run, especially when the tape unit is in WRITE status, as indicated by the writing light being on. Unless the file being written is already in error and cannot be used anyway, one should *never open the door when the writing light is on.*

The stop bar on the tape unit may be used for an emergency stop.



SUMMARY

MACHINE CHARACTERISTICS

General

Parallel operation.
Binary notation internally.

Word Size

Either 36 bits or 18 bits, including sign.

Instructions

Single address system.
32 distinct operations.
Instructions are 18-bit words.

Computing Speed

More than 2000 multiplications per second on full-word factors.

Electrostatic Storage

Capacity: 2048 words of 36 bits each.
Each full-word location may be split into two half-word locations with the capacity of 18 bits.

Magnetic Drums

Four drums, each with the capacity of 2048 words of 36 bits each.
Average access time to first word of block: 40 milliseconds.
Rate of reading or writing: 800 full words per second.

Magnetic Tapes

Four magnetic tape units.
Material: Oxide-coated non-metallic tape, ½ inch wide.
Maximum length per reel: 1400 feet.
Recording in 7 parallel channels, 6 channels for information and one for redundancy checking.
Tape may be written forward, read forward, or read backward, under control of the program.
Density within a unit record: 200 words per foot.
Distance between unit records: one inch.
Average time of acceleration to reading or writing speed: approximately 10 milliseconds.
Rate of reading or writing a unit record: 1250 full words per second.

Page Printing

One printer.
Speed: 150 lines per minute with up to 72 computed characters per line.
Printing is possible in any number system. Conversion accomplished by programming simultaneously with printing.
Alphabetic and special symbols may also be printed.
Stored program may be used to control printer.

Card Reading and Punching

One card reader and one card punch.
Read or punch up to 72 card columns.
Reading rate: 150 cards per minute.
Punching rate: 100 cards per minute.
Cards punched in standard IBM decimal code can be read at full speed and converted simultaneously by means of a program.
Can read or punch in binary code with 24 full words to a card at full speed.
Stored program may be used to control punch.

Manual Control

Operator's panel.
Control panels for each component.
Buttons and switches on components.

Checking

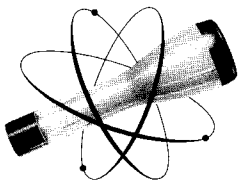
Tape reading and writing checked by redundancy check.
Printing checked by echo signals.
Decimal card punching checked by double-punch blank-column detection.
Calculation duplication by programming.
Mathematical and physical checks by programming.

INSTRUCTIONS

Decimal Code	Abbreviation	Name of Operation
00	STOP	Stop and Transfer
01	TR	Transfer
02	TR OV	Transfer on Overflow
03	TR +	Transfer on Plus
04	TR 0	Transfer on Zero
05	SUB	Subtract
06	R SUB	Reset and Subtract
07	SUB AB	Subtract Absolute Value
08	NO OP	No Operation
09	ADD	Add
10	R ADD	Reset and Add
11	ADD AB	Add Absolute Value
12	STORE	Store
13	STORE A	Store Address
14	STORE MQ	Store Contents of MQ Register
15	LOAD MQ	Load MQ Register
16	MPY	Multiply
17	MPY R	Multiply and Round
18	DIV	Divide
19	ROUND	Round
20	L LEFT	Long Left Shift
21	L RIGHT	Long Right Shift
22	A LEFT	Accumulator Left Shift
23	A RIGHT	Accumulator Right Shift
24	READ	Prepare to Read
25	READ B	Prepare to Read Backward
26	WRITE	Prepare to Write
27	WRITE EF	Write End of File
28	REWIND	Rewind Tape
29	SET DR	Set Drum Address
30	SENSE	Sense and Skip or Control
31	COPY	Copy and Skip

PART II

programming and examples



PROGRAMMING

THIS SECTION is intended primarily to give the relative beginner an understanding of some of the basic techniques of programming. It is impossible to cover *all* aspects of programming, because machine versatility permits a programmer to handle a problem in a variety of ways, one of which is most efficient. The conventions and techniques to be described have been in use with a 701 installation, but do not necessarily represent the most desirable or most efficient methods. It is hoped, however, that these paragraphs will point to a greater understanding of 701 programming and its simplicity.

CONVENTIONS AND SYMBOLISM

A PORTION of memory capable of retaining a word is termed a location or a cell. As mentioned in Part I, these locations are identified by individual addresses that range from 0000 to 4095 as far as electrostatic memory is concerned. Consequently, the twelve bits of the address part of an instruction are interpreted to be an integer. The five bits of the operation part of an instruction are also regarded as an integer. With the address part represented by an integer y where

$$0 \leqq y < 2^{12}$$

and the operation part represented by an integer x where

$$0 \leqq x < 2^{5}$$

the entire instruction is represented by an integer i

$$i = \pm (x \cdot 2^{12} + y)$$

In other words, the address part alone is thought of as an integer. Similarly, the operation part, when thought of by itself, is regarded as an integer. And finally, the instruction as a whole is thought of as an integer. Thus, when modifying instructions by simple additions, the binary point is assumed to be between accumulator positions 17 and 18.

It is convenient to express addresses as decimal integers since the decimal system is most familiar. For ease in reading programs, alphabetic abbreviations are used as operation parts. For example, the instruction

— ADD 1492

means add the contents of full-word location 1492 to the contents of the accumulator. Of course, the decimal addresses and the operation abbreviations must be converted to binary information before interpretation by the 701. Decimal-to-binary conversion is readily accomplished by the 701 itself, however, and may be made a part of a standard library of programs.

Examples of programming will be given in a standard form. An instruction given in this form is shown below.

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
+1066	—	R ADD	10	1492	Place the contents of cell −1492 in the accumulator

In the example above, +1066 is the cell in which the instruction is stored. The minus sign pertains to the sign of the instruction itself; a plus sign is implied if this column is left blank. The operation part consists of the alphabetic abbreviation and decimal code for the operation. The address part contains the address of the operand. Thus, the number actually appearing in cell +1066 is

$$-101492.$$

Because, in describing a program, it is frequently necessary to refer to a particular cell (or location) in memory, the phrase "cell a" is used to mean "the cell whose address is the integer a." Thus, the phrase "1/10 is retained in cell −1492" means "1/10 is retained in the cell identified by the address −1492." This type of phrase frequently will be further abbreviated to

$$L(x) = a.$$

by which is meant "the location of the quantity x in memory is the cell whose address is a." So, using the same example,

$$L(1/10) = -1492.$$

Similarly, it is frequently necessary to refer to the quantity or word stored in a particular location. The quantity or word stored is usually referred to as "the contents of the cell" (e.g., "the contents of cell −1492 is 1/10"). The notation

$$C(a) = x$$

means "the contents of cell a is the quantity x," e.g.,

$$C(-1492) = 1/10$$

The symbols L and C are reciprocal. By the definitions, then,

$$L[C(a)] = a$$

and

$$C[L(x)] = x.$$

Instructions and remarks may be written with this notation:

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
1066	—	R ADD	10	L(1/10)	Place 1/10 in the accumulator
1067	—	STORE	12	1494	C(−1494) is replaced by 1/10

Note in the above program that the actual array of binary digits that represent 1/10 is not precisely specified, since position of the binary point is not specified.

However, the use of the L and C symbols in connection with instructions or instruction modification is precise because of the convention to regard instructions as integers. Thus, in the program

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
1063		R ADD	10	L(0)	
1064		STORE A	13	1068	
1065		READ	24	0256	
1021→1066		R ADD	10	1068	
→1067		SUB	09	L(2)	
-1068	—	COPY	31	[ ]	Modify the copy instruction
1069		STORE A	13	1068	
1070		TR	01	1067	
→1071					

the execution of instruction 1067 increases the address part of the instruction in the accumulator by 2, or equivalently, by a binary 1 in accumulator position 16. The symbol $L(2)$, used in the above program, represents an application of the following convention.

Assume the symbol $\pm L(x)$ where the + or − designates either a half or full word.

RULE 1. If $x \geqq 1$ the binary point of the word is assumed to be to the right of the rightmost position of the word.

RULE 2. If $x < 1$ the binary point of the word is assumed to be to the left of the leftmost position of the word.

Examples:

The contents of $+L(-2)$ are
−00000000000000010

The contents of $-L(+1)$ are
+00000000000000000000000000000001

The contents of $-L(+2^{-2})$ are
+01000000000000000000000000000000

The contents of $+L(+2^{-3})$ are
+00100000000000000

The contents of $-L(+2)$ equals the contents of $-L(2^{-34})$ equals
+00000000000000000000000000000010

When an instruction or address part is modified by computation, the computed *instruction* or *instruction part* is enclosed by square brackets. (See instruction 1068 above.)

Arrows are drawn to the left of the instructions to indicate the action of the transfer instructions, as in the program above. Dotted arrows are drawn to indicate the action of COPY or SENSE skips. If it is not possible to draw lines to indicate a transfer (such as may happen when a program is on two or more pages) the notation above on instruction 1066 is used. The number and arrow to the left of 1066 mean that the instruction in cell 1066 can be executed as a result of a transfer instruction in cell 1021.

For simple exposition it frequently is not convenient to write actual addresses, as 1492 or 1066, above. A symbol, such as a Greek or Roman letter, may also be used to represent an address where the correspondence to a particular address is arbitrary or not yet assigned. For example, consider the following program:

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
a	—	R ADD	10	1492	Duplicate the contents of cell
$a+1$	—	STORE	12	1494	−1492 in cell −1494

With this notation it is possible to refer to the instruction a (i.e., the instruction stored in some cell a) without actually specifying where the program is to be located in memory.

A subscript is used to indicate the base of a number when the context is not clear: Thus

$$(1101)_2 = (15)_8 = (13)_{10}$$

represent the same number to the bases 2, 8, and 10, respectively.

In text involving card input the following notation has been adopted. The word obtained by the first COPY execution is designated by 9_L to indicate the left portion of the 9's row; the word obtained by the second COPY execution is designated by 9_R, and so on; the last word of the card image is designated by 12_R for the right portion of the 12-row.

SUB-PROGRAMMING DEVICES

THE DEVICES to be described illustrate some of the general techniques in programming. They represent, however, only a few of many such devices.

Basic Linkage

This method has, in general, been found to be most efficient for entering a sub-program routine from a main program and later returning to the main program.

The terms "main program" and "sub-program" are purely relative. For example, in calculating the function z where

$$z_i = \sqrt{x_1} + \sqrt{x_2} + \sqrt{x_3} + \ldots + \sqrt{x_i}$$

it is seen that the square root of a number must be computed i times. It would be wasteful of storage space to write a program that consisted of i separate square-root programs. The obvious thing to do is to use the same square root program for every x. So the main program can be thought of as the one that calculates z_i by using a square-root sub-program.

Now if we define a function Z such that

$$Z_n = z_1 \cdot z_2 \cdot z_3 \ldots z_{n-1} \cdot z_n$$

then the main program can be thought of as the one that calculates Z_n by means of a sub-program that calculates z_i; and finally the program for z_i again uses the square-root program as a sub-program. Thus any given program may be thought of as a sub-program in relation to a more extensive program, while it may be considered a main program in relation to a program it uses repeatedly.

With these definitions, assume that two programs A and B are given, where A is the main program and B is a sub-program of A. Also suppose that program B is to be executed after instruction $a - 1$ of program A has been executed, and the execution of program A is to be continued after the execution of program B. Let the instruction stored in cell b be the first instruction of program B. Suppose that program B requires no information other than the address of the instruction to be executed after its completion. To provide information to compute this address, two instructions are stored at locations a and $a + 1$.

$a - 1$	...	...	
a	R ADD	a	Part of basic linkage
$a + 1$	TR	b	
$a + 2$	...	...	

The instruction in cell $a + 2$ is to be executed after the completion of program B. Instruction a places the numerical representation of R ADD a in the accumulator. The next instruction transfers control to b, which is the first instruction in program B.

b	ADD	$L(2)$	Part of basic linkage
$b + 1$	STORE A	$b+k$	
.	.	.	Execution of the sub-program
.	.	.	
.	.	.	
$b + k$	TR	[ ]	

Instruction b adds 2 to the contents of the accumulator which consequently contains the numerical representation of +R ADD $a+2$. Instruction $b+1$ stores $a+2$ into the address part of instruction $b+k$. The sub-program is then executed and finally comes to its last instruction, $b+k$; this transfers control back to $a+2$ of program A. The transfer into and out of the sub-program has now been completed. If program B was written to calculate the function sin x, for example, it naturally must assume that x is located in some predetermined cell, say +1021. So a main program must place any pertinent information in locations where the sub-program can find it. This is done in the main program before transferring to the sub-program. Thus, program A must place the desired argument in a place where program B can find it, namely cell +1021. This, of course, must be done before the actual basic linkage is executed.

Note in the above technique that instructions b and $b+1$ could have been placed between instructions a and $a+1$. However, this would require writing two additional instructions in program A for *each* time it is desired to execute program B. This could result in a considerable amount of extra storage space for instructions if the sub-program is called on at quite a few different points in the main program.

Occasionally, some of the information program B requires is not computed. For example, information may be specified by the programmer, such as the number of times program B is to be repeated, or the address of a quantity to be used by program B. Let w be a half-word of such information. The w can be stored in program A in the cell $a + 2$.

$a - 1$	...	...
a	R ADD	a
$a + 1$	TR	b
$a + 2$		w
$a + 3$	...	...

Instruction $a + 3$ is to be executed after the completion of program B. For program B to use the quantity w it must be supplied with the location of w. Suppose instruction $b + j$, of program B, refers to w—that is, the address part of instruction $b + j$ is to be $a + 2$, the location of w. The first four instructions of program B now become

b	ADD	$L(2)$
$b + 1$	STORE A	$b + j$
$b + 2$	ADD	$L(1)$
$b + 3$	STORE A	$b + k$

Instructions b and $b + 1$ compute $a + 2$ and store the quantity as the address part of instruction $b + j$. Instructions $b + 2$ and $b + 3$ compute $a + 3$ and store that quantity as the address part of the last instruction of program B. This last instruction to be executed in program B is again a transfer instruction that returns control to program A. Any amount of information could be supplied in a similar manner.

All of the linkages described above result in an unconditional transfer to program B after the execution of instruction $a + 1$ of program A. Conditional transfer is readily obtained by using the MQ register to retain the address a. The following program shows the transfer to program B only if the accumulator contents are zero.

$a - 1$	...	...
a	LOAD MQ	a
$a + 1$	TR 0	b
$a + 2$	...	...
b	L LEFT	35
$b + 1$	ADD	$L(2)$
$b + 2$	STORE A	$b + k$
.	.	.
.	.	.
.	.	.
$b + k$	TR	$[a + 2]$

Branches and Forks

The execution of the conditional transfer instruction

a	TR +	b

will be followed by execution either of the instruction in cell b or the instruction in cell $a + 1$. Instruction b will be executed if the accumulator is positive, and instruction $a + 1$ will be executed if the accumulator is negative. Thus the conditional transfer operation provides the possibility of executing either the instructions in cells b, $b + 1$, $b + 2$ . . . or the instructions in cells $a + 1$, $a + 2$, $a + 3$, . . . , the choice based on the condition of the accumulator. The sequence of instructions in cells b, $b + 1$, $b + 2$ . . . is called a branch, as is the sequence of instructions in cells $a + 1$, $a + 2$, ... Each branch of a program is ordinarily a different computational procedure. For example, the instructions in cells a, $a + 1$, $a + 2$, . . . might be designed to evaluate $y = \sin x$ for x in the interval 0 to $\pi/2$, and the other branch, the instructions in cells b, $b + 1$, $b + 2$, . . . , might be designed to evaluate the same function for x in the interval $\pi/2$ to π. The conditional transfer instruction could be used to select the appropriate branch for any argument x in the interval 0 to π. That part of a program which provides the possibility of executing one of several branches is called a fork. A conditional transfer instruction is a fork. However, a fork may be more than one instruction. Consider the pair of instructions

a	TR 0	b
$a + 1$	TR +	c

This pair of instructions is a three-branch fork. The instruction in cell b will be executed if the number in the accumulator is zero. Instruction c will be executed if the contents of the accumulator are positive and *not* zero. Instruction $a + 2$ will be executed if the accumulator contents are negative and not zero. If c is set equal to b, then the branch of instructions starting at cell b will be executed if the number in the accumulator is positive *or* zero. By appropriately specifying the address parts of this pair of instructions, any of the conditions equal to, greater than, less than, greater than or equal to, or less than or equal to may be obtained.

In the preceding example the choice of the branch to be executed is determined by the condition of the accumulator. The following four-branch fork shows that this is not always the case. Let b, c, d, and e be the addresses of the first instruction of four branches. Suppose the criterion for the branch selection is a parameter Δ which may take on the values 0, 1, 2, and 3. For $\Delta = 0$ the instruction in cell b is to be executed; for $\Delta = 1$, instruction c is to be executed; and so on. The selection can be done by computing the

address part of an unconditional transfer instruction as follows:

a	R ADD	$L(\Delta)$
$a+1$	ADD	$L(a+4)$
$a+2$	STORE A	$a+3$
$a+3$	TR	$[a+4+\Delta]$
$a+4$	TR	b
$a+5$	TR	c
$a+6$	TR	d
$a+7$	TR	e

The address part of instruction $a + 3$ is computed by instructions a, $a + 1$ and $a + 2$. Instruction $a + 3$ provides the selection of instruction $a + 4 + \Delta$ which is an unconditional transfer to the appropriate instruction b, c, d or e.

Occasionally it is convenient not to store the first three instructions of this fork adjacent to the unconditional transfer instructions, but rather to separate the sets of instructions:

a	R ADD	$L(\Delta)$
$a+1$	ADD	$L(a+k+1)$
$a+2$	STORE A	$a+k$
.	.	.
.	.	.
.	.	.
$a+k$	TR	$[a+k+1+\Delta]$
$a+k+1$	TR	b
$a+k+2$	TR	c
$a+k+3$	TR	d
$a+k+4$	TR	e

Calculations may then be performed after the selection of the branch but before executing the selected branch. This fork is then called a preset fork.

Alternators

The purpose of an alternator is to supply a program with two exits that alternate each time the program is executed. The following program illustrates a method for doing this. In this program let x be a number that is positive the first time instruction $a + 1$ is executed.

a	...	...
$a+1$	R SUB	$L(x)$
$a+2$	STORE	$L(x)$
$a+3$	TR +	b
$a+4$	TR	c

Instructions $a + 1$ and $a + 2$ result in the sign of x being changed in storage. The contents of the accumulator are negative; this results (by $a + 3$ and $a + 4$) in a transfer to the instruction located in cell c. Now, if program $a + 1$ is again reached during the problem, x is a negative number. The expressions $a + 1$ and $a + 2$ change x to a positive number and leave the accumulator positive. Instruction $a + 3$ then results in a transfer to the instruction in cell b. Further returns to $a + 1$ will result in repetition of the two cycles mentioned above. In this way the program alternates between two branches of instructions.

BINARY INPUT

PROGRAMS that transmit binary information into electrostatic storage are presented in the following sections. These programs were chosen because they:

1. Illustrate many programming techniques.
2. Are relatively simple.
3. Are useful programs that may be used directly by many installations.
4. Have been completely checked out on a 701 installation.

These programs do not necessarily represent the most efficient method of programming and, in addition, programs may be written that are specifically adapted to the problems of a particular installation.

Self-Loading Technique

Often it is necessary to load a program into electrostatic storage without using any information already contained in the electrostatic memory. This happens when the power has been turned off or when one programmer does not know, or does not care to know, what a previous programmer has left in electrostatic storage. For these occasions programs can be designed that are capable of loading themselves into the machine. The underlying principle of all such programs is the following. One instruction, which is only one half-word, can cause a full word to be placed in electrostatic storage. This full word may be composed of two instructions, each of which can put a full word into electrostatic storage. Thus, the two instructions can store four more instructions in storage. In this way a surplus of instructions is rapidly built up; this may constitute a program capable of loading any desired number of half-words into electrostatic storage.

Two examples of self-loading programs are given below. Both programs are assumed to be punched in cards. The first program, which consists of only six instructions, will load itself and an additional 42 half-words (or 21 full words) punched in the same card. The six instructions that produce the self-loading feature are punched in the first three half-rows of the card. The last 21 half-rows of the card contain the program which it is desired to load. The six self-loading instructions are:

+0000	— COPY	0002
+0001	+ R ADD	0003
+0002	+ ADD	0000
+0003	— COPY	[0004]
+0004	+ STORE A	0003
+0005	+ TR	0002

Figure 22 schematically shows the half-rows where the respective instructions are punched.

Loading of the program is initiated from the operator's panel. First, set the load selector switch to CARDS, and set the address entry keys to zero. Then depress the load button; this automatically starts the loading of the program by causing the calculator to execute a READ instruction (with the address of the card reader as its address part) followed by the instruction "—COPY 0000," causing the word in the left half of the 9-row to be stored at location —0000 in electrostatic storage. This word is composed of the first two instructions of the program (numbered +0000 and +0001, above). Instruction number +0000 is then executed and results in instructions +0002 and +0003 (the right half of the 9-row) being stored at —0002.

The execution of instructions +0001 and +0002 causes the sum of the contents of half-word locations +0000 and +0003 to be formed in the accumulator. By reference to the *Operations* section in Part I, the numerical code for the operation part of a COPY instruction is found to be 31 or, in binary, 11111. Therefore, the contents of the accumulator are:

S	QP	1←→17	18←→35
—	00	11111000000000100	0 0
—	00	11111000000000010	0 0
—	01	11110000000000110	0 0

as a result of the addition of the two COPY instructions. Thus, bit positions 6 to 17 of the accumulator contain 0006. Since a binary 1 has carried over into the P overflow position of the accumulator, the overflow indicator will be turned on.

Next, instruction +0003 is executed, causing instructions +0004 and +0005 to be stored in full-word location —0004. The execution of instruction +0004 replaces bit positions 6 to 17 of half-word location +0003 with the contents of bit positions 6 to 17 of the accumulator register. Thus, instruction +0003 is modified and now becomes "—COPY 0006." Control is then transferred back to instruction +0002 when instruction +0005 is executed.

During the succeeding repetition of the loop, the first two half-words of the program to be loaded into electrostatic storage are copied from the right half of the 8-row and entered in full-word location —0006. Instruction +0003 is modified and becomes "—COPY 0008." Instruction +0005 again transfers control back to instruction +0002. This sequence of

Row No.	Left Half-Rows (Columns 1-36)		Right Half-Rows (Columns 37-72)	
12				
11				
0				
1				
2				
3				
4				
5				
6				
7				
8	+ Store A 0003	+ TR 0002		
9	— Copy 0002	+ R Add 0003	+ Add 0000	— Copy 0004

FIGURE 22

events continues until an end-of-record signal is received, indicating that there are no more half rows of the card to be copied. The end-of-record condition then causes the COPY instruction not to be executed and transfers control to +0006, which is the first instruction of the desired program. Thereafter the calculator continues with the execution of this program. Observe that when the instruction stored at +0006 is executed, the overflow indicator is on, since it was turned on the first time instruction +0002 was executed, and no TR OV instruction has been given since. The indicator must be turned off if overflow indication is to be used later for conditional control of the program itself.

The foregoing method may be used only to load programs comprising not more than 42 instructions. The following self-loading program may be used to load any number of instructions punched on any number of cards, to within the capacity of electrostatic storage:

+0000	— COPY	0002
+0001	— COPY	0004
+0002	+ R ADD	0005
+0003	+ ADD	0000
+0004	+ STORE A	0005
+0005	— COPY	[0004]
+0006	+ TR	0002
+0007	+ TR	0010
+0008	+ READ	2048
+0009	+ TR OV	0005

These instructions are punched in the first card in the locations shown in Figure 23. The rest of the first card, starting with the right half of the 7-row, contains the first 38 instructions of the program to be loaded. Later instructions are placed in additional cards.

Loading of the program is initiated from the operator's panel by the same procedure described above. The COPY instruction, executed automatically when the load button is depressed, causes instructions +0000 and +0001 to be entered in full-word location —0000 of electrostatic storage. Execution of instructions +0000 and +0001 results in instructions +0002 to +0005 being stored in full-word locations —0002, —0004.

Execution of instructions +0002 and +0003 results in the sum of the contents of half-word locations +0005 and +0000 being formed in the accumulator. Just as in the previous example, bit positions 6 to 17 of the accumulator contain the number 0006 and, since a binary 1 has carried over into the P overflow position, the overflow indicator is turned on. On the execution of instruction +0004, the contents of bit positions 6 to 17 of half-word location +0005 are replaced with the contents of bit positions 6 to 17 of the accumulator register. Thus, instruction +0005 is modified and becomes "—COPY 0006."

Next, the COPY instruction +0005 is executed, causing instructions +0006 and +0007 (punched in the right half of the 8-row) to be entered into full-word location —0006. Execution of instruction +0006 transfers control back to instruction +0002. On the second repetition of the loop, instruction +0005 is modified and becomes "—COPY 0008." Then when instruction +0005 is executed, instruc-

Row No.	Left Half-Rows (Columns 1-36)		Right Half-Rows (Columns 37-72)	
12				
11				
0				
1				
2				
3				
4				
5				
6				
7	+ Read 2048	+ TR OV 0005		
8	+ Store A 0005	— Copy 0004	+ TR 0002	+ TR 0010
9	— Copy 0002	— Copy 0004	+ R Add 0005	+ Add 0000

FIGURE 23

tion +0008 and +0009 (punched in the left half of the 7-row) are loaded into full-word location −0008. Again, instruction +0006 transfers control back to instruction +0002, and the loop is repeated.

On the next repetition, instruction +0005 becomes "−COPY 0010," and the right half of the 7-row (containing the first two instructions of the program whose presence in electrostatic storage is desired) is loaded into full-word location −0010. The loop continues to be repeated until all the half-rows of the first card have been copied and an end-of-record signal is received. Then, when the COPY instruction +0005 is given, it is not executed; instead, control is transferred to instruction +0008.

When the READ instruction in +0008 is executed, another card feeds in the card reader. Since the overflow indicator was turned on the first time instruction +0003 was executed and has not been turned off by a TR OV instruction, it is still on when instruction +0009 is given. Consequently, when instruction +0009 is executed, control is transferred back to the COPY instruction (+0005) whose execution was previously prevented by the end-of-record condition. This COPY is now executed, and the left half of the 9-row of the second card is loaded into electrostatic storage.

The loop consisting of instructions +0002 to +0006 is repeated until all 24 half-rows of the second card have been loaded. Then the end-of-record condition again causes control to be transferred to the READ instruction, and the next card is fed. This sequence of events continues until all cards have been read. Then the last READ instruction, followed by the COPY instruction +0005, sets up an end-of-file condition. The COPY instruction is not executed, and control is transferred to instruction +0007; this in turn transfers control to the first instruction of the desired program that was stored in +0010. The calculator proceeds with the execution of this program. Note that the overflow indicator is off at the time instruction +0010 is executed, since it was turned off by instruction +0009.

Thus the desired program is loaded, as punched, into consecutive electrostatic locations beginning with +0010.

Binary Reading Program, L 05

A program will now be described that can load itself into storage and later load blocks of binary information into any section of electrostatic storage.

The self-loading techniques described above require that every program to be loaded carry its own self-loading sequence and that all loading be done using the consecutively-numbered electrostatic storage locations immediately following the locations occupied by the self-loading sequence itself. It is evident that these restrictions need not apply in general, since a self-loading sequence of six instructions may be used to enter a more general loading program. The following program (L 05) is an example of such a general program.

L 05 is contained in a single card which also includes six instructions that allow it to be entered by means of the load button. It makes use of electrostatic storage locations 0000 through 0047. L 05 may be used to enter binary information from any number of punched cards. This information may be stored in any desired consecutively-numbered electrostatic-storage locations, except those already reserved for L 05. Further, the reading and storing operations are checked by use of a check sum.

A block of information to be loaded by L 05 must be preceded by a check sum, half-word count, and an initial loading address. These and other quantities will now be defined.

Card Check Sum: This is calculated by summing the absolute values of all half-words in the block, including the half-words that specify the half-word count and initial loading address. The check sum itself is specifically excepted. To this sum is added 2^{17} times the number of *negative* half-words. The sum is then doubled, and a minus sign is attached. This card-check sum requires a binary full word and is punched in the left half of the 9-row of the first card in the group containing the block of information to be loaded.

Half-Word Count (V): This is the number of half-words to be loaded into consecutively-numbered locations. This count *does not* include the four half-words that make up the card-check sum, the half-word count, and the initial loading address. This count requires a binary half-word and is punched in the sign position and positions 1-17 of the right half of the 9-row of the first card.

Initial Loading Address (R): This specifies the locations into which the following block of information is to be loaded. The V half-words of the block are then loaded into electrostatic storage locations $R + 2$ through $R + V + 1$. Locations R and $R + 1$ are reserved for the storage check sum (see below). R must be a positive even integer. It requires a binary half-word and is punched in positions 18-35 of the right word of the 9-row of the first card.

Storage Check Sum: This is calculated by adding the quantity $2(V + R)$ to the card-check sum. The card-check sum is thereby reduced in absolute value.

The following things should be noted about these quantities:

1. Both V and R are included in the calculation of the check sum contained in the card.
2. Since L 05 occupies electrostatic storage locations 0000 through 0047, R must be greater than or equal to 0048.
3. If more than one block of information is to be loaded, then the following two requirements must be observed:
   a. The V half-words of each block must be loaded into consecutively numbered electrostatic-storage locations, and
   b. Each block must start on a new card whose 9-row contains the card check sum, the half-word count, and the initial loading address for that block.
4. If V is odd, location $R + V + 2$ is set to zero.

The output of the program consists of $V + 2$ half-words for each block of information entered. These half-words are stored in locations R through $R + V + 1$, the first two of these half-word locations being reserved for a full-word *storage* check sum computed from the card check sum by the program itself.

Instructions for L 05 follow. This program is in turn followed by a general description of its operation.

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
0000	—	COPY	31	[0002]	Load L 05 card
0001		R ADD	10	[0003]	
0002	[	ADD	09	0000]	
0003	—	COPY	31	[0004]	
0004		STORE A	13	0003	
0005		TR	01	0001	
0006		READ	24	2048	Select card reader
0007	—	COPY	31	0002	Card check sum
0008		TR	01	0010	Not end of file
0009		STOP	00	0001	End of file stop
0010	—	COPY	31	0004	Loading information
0011		R ADD	10	0005	Obtain loading address for storage check sum and for COPY sequence
0012		STORE A	13	0022	
0013		STORE A	13	0028	
0014		ADD	09	0045	
0015		STORE A	13	0033	Reset address to initial value
0016		ADD	09	0004	Set up end of COPY index and end of ADD index
0017		STORE A	13	0001	
0018		SUB	05	0045	
0019		STORE A	13	0000	Modify card check sum to give storage check sum
0020		A RIGHT	23	17	
0021	—	ADD	09	0002	
0022	—	STORE	12	[ ]	
0023	—	STORE	12	0002	

PROGRAM (continued)

→0024		R ADD	10	0028	}
0025		SUB	05	0045	
0026		STORE A	13	0028	
0027		SUB	05	0000	} COPY sequence
→0028	—	COPY	31	[ ]	
—0029		TR +	03	0024	
—0030		TR	01	0033	}
→0031		READ	24	2048	Select card reader
—0032		TR	01	0028	Continue copying
0043→0033		R ADD	10	[ ]	}
0034		STORE	12	0047	
0035		R ADD	10	0033	
0036		ADD	09	0009	
0037		STORE A	13	0033	
0038		SUB	05	0001	} ADD sequence
0039	—	LOAD MQ	15	0046	
0040		L LEFT	20	36	
0041	—	ADD	09	0002	
0042	—	STORE	12	0002	
0033←0043		TR OV	02	0033	}
0006←0044		TR 0	04	0006	Test unit record sum
0045		STOP	00	0002	Error stop
0046		STOP	00	0000	} Contribution to unit record sum
0047	[	STOP	00	0000]	}

A detailed description of each individual instruction of the above program is not included, but a general discussion of the operation of the program follows.

Put the card containing L 05 at the front of the file of cards containing one or more blocks of binary information to be loaded. Place this file of cards in the card reader, restore the address entry keys to zero, and depress the load button. The rest of the loading operation takes place automatically under calculator control.

The first six instructions of L 05 load the remainder of this program. Control is then transferred to location 0006 as a result of the end-of-record skip. Instructions 0006, 0007 and 0010 read and copy the 9-row of the first card whose information is to be entered into electrostatic storage. This row contains the card check sum and the quantities V and R. These words are stored in locations −0002, +0004, and +0005 and replace four instructions of the loading sequence, which is no longer required. The card check sum, V, and R are then used by instructions 0011 through 0023 to reset various address parts and to calculate the storage check sum stored both in locations $-R$ and −0002.

The remaining V half-words of the block are then read and stored by instructions 0024 through 0032. When this information has been entered, control is transferred to location 0033. Instruction 0033 is the first of the loop, 0033 through 0043; this verifies that the newly-entered information is stored correctly. This verification is performed by the addition of successive half-words, whose signs are treated as described above, to the storage check sum stored in location 0002.

After the V half-words have been added to the storage check sum, instruction 0044 tests this new sum for zero. If no error has been made, this test for zero is successful, and control is transferred to location 0006. The calculator is now ready to read a second block of information. This process continues as long as no error is made and cards remain to be read.

After the reading of the last group of cards whose binary contents are to be stored, the program verifies the correctness of the newly-entered information and again transfers control to location 0006. At this point, however, no cards remain to be read. So the end-of-file skip transfers control to location 0009, which contains a STOP instruction. This signals the successful

completion of the loading operation. Should an error be detected in the summing of a block, the calculator stops at instruction 0045.

L 05, once loaded, may be used any desired number of times without reloading simply by placing the cards to be read in the card reader and starting the calculator at instruction 0006.

CHECKING

THE VALIDITY of a problem solution involves several distinct aspects. Initial data must be correct. Usually input data are not susceptible to machine analysis for verification and must be guaranteed by the problem sponsor. Of course, transcribing the data onto cards may be verified by standard machine methods. The programming of the problem must be shown to represent the actual sequence of operations required for the solution. The introduction of the data and program into the machine, the performance of the machine during the solution, and the recording of the results should be verified. In all these checking operations, the problem sponsor and programmer should determine the extent of the checking. The design of the 701 allows a large variety of methods for checking its operation and, as well, provides the programmer with a wide choice of the *extent* of the checking.

To ascertain the correctness of the program, experience has shown that the program should be executed by the machine in a test cycle, and the results compared with expected results. One method is to interrupt instruction executions manually and examine the effects of the executions through the indicators on the operator's panel. Specific devices have been incorporated to facilitate this technique of program verification and the correction of errors. Lights and buttons on the operator's panel provided for this purpose are the register lights, half-step key, multiple step key, memory display button, instruction entry keys, MQ entry keys, enter instruction button, and enter MQ button.

An alternative method (called "tracing") has been used successfully on a 701 installation. In this method, the machine prints a record of the contents of the accumulator register, multiplier-quotient register, and the status of the overflow indicator for each instruction-execution of the problem program; the machine also prints the instruction itself. We can examine the resultant recording at leisure without consuming machine time. The special program for this is usually termed a tracing program.

Error-detection mechanisms have been included in the machine; these suspend calculation and signal an error when, because of programming errors, a COPY or DIVIDE instruction is interpreted under circumstances which, if executed, always would produce an erroneous result. These mechanisms include the copy check light and the divide check light. Checks for the introduction and recording of information are print echo pulses, double-punch blank-column detection, and the tape check light.

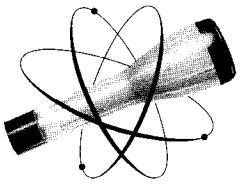
Because there is the possibility of the machine itself making an error, the performance of the machine during the solution of a problem must be checked. Such checks are usually embodied in the problem itself and should be regarded as part of the general procedure of efficient programming. The method of checking a particular problem is determined by the character of the problem and the degree of confidence in the results the programmer desires. Thus, some problems have an inherent mathematical check in their solution, and for some a mathematical check can easily be designed. A calculation can be checked by duplicating it and comparing the two sets of results. Another check is by duplicate calculation using complements of the original numbers, or by duplicate calculation performed in the residue-class ring of integers modulo some number p.

Consider programming of checking from the viewpoint of efficient operation. If a transient error should be made by the machine it is an advantage for the machine to repeat the calculation automatically; because in the time required for the operator to reach the operator's control panel, the machine could have performed many operations. This is normally arranged for by dividing the problem into parts, at natural mathematical breaking points. The programming is so arranged that calculation, check calculation and comparison of results are done for one part and, if the results are correct, calculation of the next part is started. If, on the other hand, an error is detected, the calculation for that part is repeated.

This section points out some techniques for checking solutions and input-output operations. There are many possibilities, and individual problems merit individual consideration.

Certain checking procedures have been continuously inserted in the preceding text. The previous section, *Binary Reading Program L 05,* showed how proper operation of the program is assured according to a predetermined card check sum. An example of a mathematically designed check is a square-root program checked by squaring the result and comparing it with the original argument. This is not necessarily the best method for checking a square-root program in a particular problem, but it does show one of the possibilities for this kind of check.

It cannot be emphasized too strongly how important it is to include appropriate checks in a program. The extent and method of checking is *entirely* dependent upon the degree of confidence in the results, which is determined by the problem's importance and the needs of the problem sponsor.



EXAMPLES

THIS SECTION gives several examples of programming that progressively use more and different operations and so become lengthier and more complicated. These examples can be studied in sequence to understand how different operations work together to create a meaningful program. We have already referred to some of these examples in the *Operations* section of Part I. Input-output operations are not illustrated here because they have been covered under *Input-Output Components.*

Some of the examples cover such operations as forming a rounded 35-bit quotient, transferring control on non-zero, transferring control on minus, vector addition, and square root. Most of the examples use programming conventions described in the earlier *Programming* section.

1. The following program will: add a 35-bit number A found in electrostatic location -1492 to a 35-bit number B located at -1588; double the sum; and store the result in location -1812. Assume that the result does not carry over into the overflow positions of the accumulator.

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
1066	—	R ADD	10	1492	Place A in the accumulator
1067	—	ADD	09	1588	Form $A+B$
1068		A LEFT	22	1	Form $2(A+B)$ by shifting one binary place left
1069	—	STORE	12	1812	Place $2(A+B)$ in ES location -1812

2. Two sets of programming are given; either one may be used to compute $A - B$.

LOCATION	INSTRUCTION			REMARKS
	±	OPERATION PART	ADDRESS PART	
1066	—	R ADD 10	L(A)	Place A in accumulator
1067	—	SUB 05	L(B)	Place $-B$ in accumulator forming $A-B$
1068	—	STORE 12	x	Place result in location x

If $A = B$, the result of computing $A - B$ is zero with the sign of A.

LOCATION	INSTRUCTION			REMARKS
	±	OPERATION PART	ADDRESS PART	
1066	—	R SUB 06	L(B)	Place $-B$ in accumulator
1067	—	ADD 09	L(A)	Form $-B+A$
1068	—	STORE 12	x	Place result in location x

If $A = B$ in this programming, the result of computing $-B + A$ is zero with the sign of $-B$.

3. Suppose $C + |A| - |B|$ is to be calculated with the result stored in x. Also assume that an overflow into position P *might* occur; in this case the sum should be rounded to 35 bits before it is stored in x.

LOCATION	INSTRUCTION			REMARKS
	±	OPERATION PART	ADDRESS PART	
1066	—	R ADD 10	L(C)	Place $+C$ in accumulator
1067	—	ADD AB 11	L(A)	Form $+C+\|A\|$
1068	—	SUB AB 07	L(B)	Form $+C+\|A\|-\|B\|$
1069		TR OV 02	1071	Transfer to rounding procedure if overflow occurred
1070		TR 01	1073	Transfer to storing procedure if overflow did not occur
1071		L RIGHT 21	1	Shift sum right one place
1072		ROUND 19		Round
1073	—	STORE 12	x	Place sum in location x

The long right-shift instruction causes the combined contents of the accumulator and MQ registers to shift one place to the right. Thus, the bit in the rightmost overflow position enters the first position of the accumulator register; the bit in the 35th position of the accumulator enters the first position of the MQ register. Since the store instruction only reads out the accumulator register, the bit remaining in the MQ register after rounding is dropped.

4. To truncate without rounding and without disturbing the contents of the MQ register, the A RIGHT and A LEFT operations may be used. Assume in this program that the least significant 5 bits of the sum $A + B$ are to be dropped.

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
1066	—	R ADD	10	L(A)	Place A in accumulator
1067	—	ADD	09	L(B)	Forms $A+B$
1068		A RIGHT	23	5	The 5 least significant bits of $A+B$ are shifted out of the register and lost
1069		A LEFT	22	5	The number is returned to its original position with the 5 low order digits now zero
1070	—	STORE	12	x	Places the truncated sum in location x

5. Multiplication of B by A to form a 35-bit rounded product can be done as follows:

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
1066	—	LOAD MQ	15	L(A)	Place A in the MQ register
1067	—	MPY R	17	L(B)	Form AB and round
1068	—	STORE	12	x	Place rounded product in location x

The multiply-round instruction forms the 70-bit product AB and rounds the product to 35 bits. The rounded product is then stored from the accumulator.

6. To shift the product before rounding, use the following procedure:

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
1066	—	LOAD MQ	15	L(A)	Place A in the MQ register
1067	—	MPY	16	L(B)	Form AB
1068		L LEFT	20	3	Shift AB three places left
1069		ROUND	19		Round
1070	—	STORE	12	x	Place rounded answer in location x

The long left-shift instruction causes the combined contents of the accumulator and MQ registers to be shifted left three places. After rounding, the contents of the accumulator register are stored.

7. Suppose that $|A| < |B| < 1$ and that it is desired to find the 35-bit rounded quotient of A by B. The following programming will accomplish this result.

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
1066	—	R ADD	10	L(B)	Place B in the accumulator
1067		L RIGHT	21	36	Place $2^{-36}B$ (with respect to the entire dividend) in the MQ register
1068	—	R ADD	10	L(A)	Form $A+2^{-36}B$
1069	—	DIV	18	L(B)	Form rounded quotient
1070	—	STORE MQ	14	x	Place rounded quotient in location x

The accumulator and MQ registers together serve as a 70-bit dividend register during division. The dividend $A + 2^{-36}B$ is formed by shifting B into the MQ register from the accumulator register and then placing A in the accumulator register. During division the sign of the entire 70-bit dividend is taken to be the sign of the accumulator register (i.e., the sign of A); the sign of the MQ register is ignored. Thus, in effect, the dividend has the modulus (absolute value)

$$|A| + 2^{-36}|B|$$

and the sign of A. So the quotient on division by B has the modulus

$$\frac{|A|}{|B|} + 2^{-36}$$

and the sign of A/B. Consequently, after division the MQ register contains the quotient A/B rounded to 35 bits.

8. A shifted and rounded quotient may be obtained as follows. Assume that the half-word location +0055 contains the number zero; also that the quotient is to be shifted four places to the right and then rounded.

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
1066		LOAD MQ	15	0055	Reset MQ register to zero
1067	—	R ADD	10	L(A)	Place A in the accumulator register
1068	—	DIV	18	L(B)	Form A/B
1069		R ADD	10	0055	Clear remainder from accumulator register
1070		L LEFT	20	31	Shift quotient into accumulator register
1071		ROUND	19		Round shifted quotient
1072		A LEFT	22	4	Returns number to proper relative location in register
1073	—	STORE	12	x	Store rounded, truncated quotient in x

The LOAD MQ instruction resets the MQ register to zero. Note that to reset all 35 bits of the MQ register, it is not necessary to have a 35-bit zero in storage, because when a half-word is loaded into the MQ register the 18 rightmost bits automatically become zero. A similar remark applies to the clearing of the remainder from the accumulator after division. The 31 leftmost bits of the quotient are shifted into the accumulator register, leaving the four rightmost bits in the MQ register. The truncated (31-bit) quotient is then rounded. Instruction 1072 then shifts the quotient to its proper relative location in the register, and finally the STORE instruction writes the rounded 31-bit quotient from the accumulator register in full-word location x. The four bits left in the MQ register are dropped.

9. Transfers on non-zero and minus can be effected by these two devices:

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
1066		TR 0	04	1068	Transfer to 1068 if accumulator contains zero
1067		TR	01	x	Hence, transfer to x if accumulator is non-zero

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
1066		TR+	03	1068	Transfer to 1068 if accumulator is plus
1067		TR	01	x	Hence, transfer to x if accumulator is minus

Other more complicated discriminations are also possible.

10. The transfer operations can be used to make the machine choose among several programs. The following example is almost identical to a program explained above under *Programming,* except that no symbolism will be used here.

Assume there are four possible programs whose initial instructions are located in electrostatic storage:

Program	*Address of Initial Instruction*
I	+0051
II	+0351
III	+0651
IV	+0951

Also assume that the decision as to which of the four programs to be used has already been made in the course of the problem and that the result of this decision is indicated by the presence of one of the four numbers 1, 2, 3, 4 in the accumulator register. Suppose further that the number 1068 is stored at location +1812 in electrostatic storage. The following set of instructions will direct the machine to choose the appropriate one of the four programs:

LOCATION	INSTRUCTION			REMARKS
	±	OPERATION PART	ADDRESS PART	
1066		ADD 09	1812	Compute 1068+k, where k = 1, 2, 3, or 4
1067		STORE A 13	1068	Address part of instruction at +1068 becomes 1068+k
1068		TR 01	[1068+k]	Program transferred to +1069, +1070, +1071, or +1072
1069		TR 01	0051	Transfer to Program I
1070		TR 01	0351	Transfer to Program II
1071		TR 01	0651	Transfer to Program III
1072		TR 01	0951	Transfer to Program IV

11. As an example of the use of the arithmetic and logical instructions, a square-root program will now be presented. The program is intended only for illustration and does not necessarily represent the most efficient method of obtaining square root. The argument A, whose square root is desired, is assumed to lie in the range $0 < A < 1 - 2^{-35}$ and to be stored at location −1588. The square root is computed by the well-known iteration formula

$$y_n = y_{n-1} + \tfrac{1}{2}\left[\frac{A}{y_{n-1}} - y_{n-1}\right]$$

Assume that the first approximation is the number

$$y_0 = 1 - 2^{-35}$$

stored at the full-word location −1492 in electrostatic storage; the final result is to be stored at −1812. Also the contents of half-word location +0055 are assumed to be zero.

LOCATION	INSTRUCTION			REMARKS
	±	OPERATION PART	ADDRESS PART	
1066	—	R ADD 10	1492	Place y_0 in the accumulator
1067	—	STORE 12	1812	Store y_0 in −1812
1068		LOAD MQ 15	0055	Reset MQ register to zero
1069	—	R ADD 10	1588	Place A in the accumulator
1070	—	DIV 18	1812	Compute A/y_{n-1}
1071		A RIGHT 23	37	Reset accumulator to zero
1072		L LEFT 20	35	Put A/y_{n-1} in accumulator
1073	—	SUB 05	1812	Compute $\frac{A}{y_{n-1}} - y_{n-1}$
1074		L RIGHT 21	2	Compute $\frac{1}{2}\left[y_n - y_{n-1}\right] = \frac{1}{4}\left[\frac{A}{y_{n-1}} - y_{n-1}\right]$
1075		TR 0 04	1080	Transfer to end if $y_n = y_{n-1}$ to 34 places
1076		L LEFT 20	1	Compute $y_n - y_{n-1} = \frac{1}{2}\left[\frac{A}{y_{n-1}} - y_{n-1}\right]$
1077	—	ADD 09	1812	Compute $y_n = y_{n-1} + \frac{1}{2}\left[\frac{A}{y_{n-1}} - y_{n-1}\right]$
1078	—	STORE 12	1812	Store y_n in −1812
1079		TR 01	1068	Repeat cycle of iteration
1080				Start termination procedure

The first two instructions of the program set up the calculation by placing y_0 in location −1812. These two instructions are used only once. The succeeding instructions compute the new approximation y_n from the old approximation y_{n-1} and test for convergence. As soon as $y_n - y_{n-1} = 0$ to 34 binary places, the iteration stops. The transfer-on-zero instruction causes the rest of the iteration to be skipped and the next instruction to be taken from location +1080; this location may contain the initial instruction of a program for terminating the calculation. For instance, if A is sufficiently far from zero so that one more cycle of iteration will give the accuracy desired, the termination program might perform an additional cycle carried to extra precision and then round the result to the desired number of places.

12. Another sample program, showing the technique of computing addresses, is the following program for adding two n-dimensional vectors. Assume that a vector

$$X = (x_1, x_2, \ldots, x_n)$$

is to be added to a vector

$$Y = (y_1, y_2, \ldots, y_n)$$

to form the vector

$$Z = (z_1, z_2, \ldots, z_n) = (x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n) = X + Y$$

Note the following:

(a) Components of X are stored in consecutive full-word locations -0502, $-0504, \ldots, -(0500 + 2n)$.

(b) Components of Y are stored in $-1002, -1004, \ldots, -(1000 + 2n)$, respectively.

(c) The calculation assumes that Z is to replace X in electrostatic storage—i.e., $z_i = x_i + y_i$ replaces x_i in $-(0500 + 2i)$, for $i = 1, 2, \ldots, n$.

(d) The half word "—ADD $(1000 + 2n + 2)$" is stored at location $+2000$.

(e) The constant 500 is stored at location $+2001$. In other words, $L\ (+500) = +2001$.

(f) The constant 2 is stored at location $+2002$. In other words, $L\ (2) = +2002$.

(g) The symbol I_i is used to represent the half word "—ADD $(1000 + 2i)$."

Thus,

I_{i+1} stands for "—ADD $(1000 + 2i + 2)$"

I_{n+1} stands for "—ADD $(1000 + 2n + 2)$"

The program:

LOCATION	INSTRUCTION				REMARKS
	±	OPERATION PART		ADDRESS PART	
1500		R SUB	06	2001	Resets addresses of operating instructions to proper starting values
1501		SUB	05	2002	
1502		STORE A	13	1506	
1503		STORE A	13	1508	
1504		SUB	05	2001	
1505		STORE A	13	1507	
1506	—	R ADD	10	[0500+2i]	Operating instructions which calculate and store $z_i = x_i + y_i$
1507	—	ADD	09	[1000+2i]	
1508	—	STORE	12	[0500+2i]	
1509		R ADD	10	1506	These instructions modify the addresses of the operating instructions in preparation for calculating z_{i+1}
1510		SUB	05	2002	
1511		STORE A	13	1506	
1512		STORE A	13	1508	
1513		R ADD	10	1507	
1514		SUB	05	2002	
1515		STORE	12	1507	
1516		SUB	05	2000	A comparison test to see if the last component of Z has been calculated
1517		TR +	03	1506	

The first six instructions of the program (located at $+1500$ to $+1505$) restore the three operating instructions ($+1506$, $+1507$, $+1508$) to their initial status

($i = 1$). Thus, after the execution of instruction +1505, the operating instructions read:

1506	— R ADD	0502
1507	— ADD	10C2
1508	— STORE	0502

So no matter what the addresses of the operating instructions originally were, they are reset to their proper starting values.

The working part of the program consists of the three operating instructions located at +1506 to +1508. These instructions add the ith component of Y and store the sum (i.e., the ith component of Z) at $-(0500 + 2i)$. The succeeding seven instructions (located at +1509 to +1515) serve to modify the three operating instructions by adding 2 to the magnitude of each address part.

The two instructions located at +1516 and +1517 test whether all n components of the two given vectors have been added. If $i < n$, then since

$$I_{i+1} - I_{n+1} > 0,$$

the TR + instruction transfers control back to instruction +1506 and x_{i+1} is added to y_{i+1} to form z_{i+1}. If $i = n$, then since $I_{i+1} - I_{n+1} = -0$, the TR + instruction does not transfer control and the next instruction is taken from +1518. Note that I_{n+1} (stored at +2000) is actually a constant of the program, although it has the form of an instruction. By modifying this constant, the program can be altered to accommodate vectors of varying dimension.