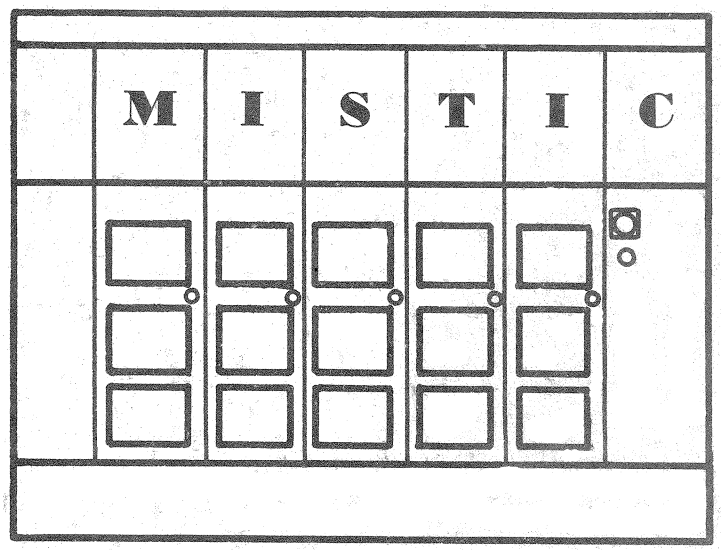


PROGRAMMING MANUAL

Computer Laboratory
MICHIGAN STATE UNIVERSITY
East Lansing, Michigan

MISTIC PROGRAMMING MANUAL

Second Edition
September 15, 1959

Prepared by the Staff of the
COMPUTER LABORATORY
MICHIGAN STATE UNIVERSITY
EAST LANSING, MICHIGAN

PREFACE

Perhaps one of the most significant innovations of the century has been the design and actual construction of electronic digital computers. It is true that designs for computers had been considered as early as the first quarter of the nineteenth century, but it is equally true that the then existing means of operating such a computer were insufficient. It was only in the period following 1940 that advancing electronic techniques made an automatic computer feasible. Among the earliest conceived computers was the one at the Institute for Advanced Study at Princeton University, followed by the initial phases of design of the ILLIAC at the University of Illinois in 1948. The Illinois computer, modeled to a certain extent after the Princeton machine, was completed in 1952.

In the decade following 1945, the number of digital computers in this country increased from at most a dozen to more than a thousand. The applications of such instruments have reached into almost every scientific field. As a result, it was decided by Michigan State University, in order to provide access to such an important aid to research, that a digital computer should be constructed on this campus. With the generous assistance of the staff of the Computer Laboratory at the University of Illinois, work on a duplicate of the ILLIAC was begun in the summer of 1956. The machine, called MISTIC, went into operation on October 18, 1957.

This second edition of the MISTIC Programming Manual has been up-dated and corrected following nearly two years of MISTIC operation. In particular, information concerned with the use of punched cards and the core memory has been added.

The manual is organized for use as a general reference for the experienced programmer. It is used also as a text for programming courses taught on the Michigan State University campus. It can also be used by those with little or no background in the use of computers as an instruction manual. The index will aid the experienced in finding detailed answers to specific questions. The use of the manual by those working on their own remains to be explained.

For this latter group of users who are getting little or no outside help, the following comments and outline may be useful. In particular, it should be emphasized that the manual has been written with a view towards being all-inclusive and a lot of worthwhile use can be made of the computer without knowing all the

details presented herein. Thus any chapters, or parts thereof, which present difficulty should be skipped until later experience dictates their use, without any fear of missing something important.

The following outline has been used successfully in individual situations:

A. 1. Chapter 2 - Read casually, refer back as necessary.
   2. Chapter 3 - Read, then use as a reference as necessary.
   3. Chapters 4 and 5 - Study in detail as a unit, with reference to Chapter 3 when required.

B. 1. Chapter 8 - Read thoroughly first, then reread occasionally and note for reference.
   2. Chapter 9 - Read and study as required.
   3. Chapter 6 - Read and study as required.

C. 1. Chapter 2 - Read before starting tape preparation.

Several simple problems should be programmed and used on the computer during the process of completing this outline. Individual needs will dictate succeeding study and use of the manual.

CONTENTS

CHAPTER 1

NUMBER SYSTEMS

Computers are machines which can add, subtract, multiply by certain numbers, and divide by certain numbers; they can make decisions between possible courses of action and, once started, can operate under their own control for a period of time. Computers operate upon data which is expressed in numerical form. Some computers, such as analogue computers, do not operate directly on numbers. Analogue computers derive their name from the fact that problems are solved in such a computer by setting up and solving an analogous problem in electric circuits. If, however, the computer operates on numbers, then it is called a digital computer. MISTIC is a digital computer, and it performs operations on numbers expressed in a different fashion from that which one is accustomed to using, for purposes of both speed and economy. The system used is the binary number system, which is discussed here.

1.1. Positional Notation

A moment's reflection will reveal that when one uses the decimal number system, one has really used a representation for a sum. Thus, for example, the number 41,978 really represents 40,000 + 1,000 + 900 + 70 + 8. Furthermore, one can see that 41,978 = 40,000 + 1,000 + 900 + 70 + 8 = 4 x 10,000 + 1 x 1,000 + 9 x 100 + 7 x 10 + 8. Finally, if one observes that $10,000 = 10^4$, $1,000 = 10^3$, $100 = 10^2$, and $10 = 10^1$, then

$$41,978 = 4 \times 10^4 + 1 \times 10^3 + 9 \times 10^2 + 7 \times 10^1 + 8 \times 10^0$$

where $10^0 = 1$. Likewise, for a decimal fraction, say .02796,

$$.02796 = 0 \times 10^{-1} + 2 \times 10^{-2} + 7 \times 10^{-3} + 9 \times 10^{-4} + 6 \times 10^{-5}.$$

The notation in which only the coefficients of the powers of ten are preserved (the 41,978 of the above example) is called a positional notation. The number 10 is called the base of the decimal system. It is easy to convince oneself that the base 10 is not the only base that could be used to represent numbers as a sum of powers. Indeed, it can be shown that any integer N can be represented as a sum of powers of any whole number r greater than one as

1) $N = a_n r^n + a_{n-1} r^{n-1} + \ldots + a_1 r + a_0$

where each a_i is an integer in the range $0 \leq a_i < r$,

2) For any fraction M,

$$M = b_1 r^{-1} + b_2 r^{-2} + \ldots + b_m r^{-m}$$

where the b_i are integers in the range $0 \leq b_i < r$.

A digital computer in representing numbers provides a place to hold each coefficient of the power-sum representation for that number. If the computer operated in decimal, each such coefficient position would have to be able to assume one of ten states--namely, states corresponding to 0, 1, 2, ..., 9. It is not difficult to imagine that it would be easier to represent fewer states, so that an optimum arrangement would be one in which only two states need be represented in each coefficient position. This means then that the a_i and the b_i of representations 1 and 2 need to assume only the values 0 and 1 and that r is 2. Such a system is called the binary number system.

Some examples of numbers in the binary system follow.

$25 = 16 + 8 + 1 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$502 = 256 + 128 + 64 + 32 + 16 + 4 + 2 = 1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

$.25 = 0 \times 2^{-1} + 1 \times 2^{-2}$

If positional notation is used for the binary numbers on the right side of the equation, then

$$25_{10} = 11001_2$$
$$502_{10} = 111110110_2$$
$$.25_{10} = .01_2$$

(Notice that the subscript in the last three equations denotes the base used.)

The MISTIC operates in binary, and each number is assumed to be a fraction and to be exactly 40 binary digits (or bits) long.

1.2. Conversion Between Binary and Sexadecimal

Since a number 40 bits long represents the same information as one 12-digit decimal number at most, it is apparent that though binary may be economical for a computer to use, it is not convenient for the human being to use. It would be better to abbreviate the binary number in some way, preferably an easy way. The device used with the MISTIC is to start at the binary point (binary point in binary is the equivalent of the decimal point in decimal) and group bits into sets of four, working to the left on the integer part and to the right on the fractional part. Thus, if the number M is given by

M = 10110110001101010011.0110111011011

then M = 1011 0110 0011 0101 0011 . 0110 1110 1101 1000

Give each group of four bits its equivalent value in decimal. Since

1011 = 11

0110 = 6

0011 = 3 etc.,

then M = 11, 6, 3, 5, 3 . 6, 14, 13, 8

If the commas were removed, confusion as to the meaning of 11, 6, for example, would arise. If, instead, it were possible to guarantee that each set of four bits translates to a single digit, such confusion would not arise. Since the maximum number representable by four bits is $1111 = 15_{10}$, single symbols for 10, 11, 12, 13, 14 and 15 are needed. The symbols used are

10 = K

11 = S

12 = N

13 = J

14 = F

15 = L

To complete the example, M converts to

M = S6353.6FJ8

To convert from this kind of notation to binary one needs simply to replace each symbol by its four bit representation. To illustrate, if M = 4F6L.21K, then

M = 0100 1110 0110 1111 . 0010 0001 1010

It is to be observed that the system using the 16 distinct symbols, 0, 1, . . . , L, is nothing more than a base 16 system, which is called the sexadecimal system. That is, r = 16 and each a_i ranges from 0 to L.

1.3. MISTIC Representation of Binary Numbers--2's Complement

MISTIC represents every number by a 40 bit number assumed to be a fraction. Since numbers are either positive or negative, some provision must be made within the 40 bits to allow for the sign. There are actually several methods of sign representation available in addition to the way which the human being is accustomed to seeing. For certain engineering reasons, the method selected for the MISTIC is the so-called 2's complement system.

Definition 1.3.1. Every MISTIC 40 bit word is made up of one sign bit and 39 bits representing a fraction in the format $n_0 n_1 n_2 \ldots n_{39}$, where n_0 is the sign bit, n_1 represents the coefficient of 2^{-1}, n_2 represents the coefficient of 2^{-2}, and in general, n_i represents the coefficient of 2^{-i}, in the binary representation of a number.

Definition 1.3.2. The 2's complement of a 39 bit fraction N is 2 + N.

The sign bit actually leads a double life, since it is interpreted as a sign but is really a number. Thus, if the sign of a number is plus, the sign bit is 0; if the sign if minus, the sign bit is 1. This is the case since the number 2 in binary is 10.000..., while the computer, according to definition 1.3.1, can represent only the 0.00... of 10.000... Consequently, 2 + N for a positive

fraction N where $N = .n_1n_2n_3\ldots$ is

$$2 + N = 10.0000\ldots + .n_1n_2n_3\ldots$$

$$= 10.n_1n_2n_3\ldots$$

which in the computer would appear as $0.n_1n_2n_3\ldots$ If, on the other hand, N is a negative fraction, a borrow must be made from the 1 in 10.000..., thus placing a 1 in the 2^0 position.

Example 1.3.1. If N = .11010...0 = 13/16,

then in the complement form,

$$\begin{aligned} 2 + N &= 10.0000....0 \\ &\quad + \underline{\quad .11010...0} \\ &= 10.11010...0 \end{aligned}$$

But in the computer the first 1 does not appear so that the 2's complement representation of 13/16 is 0.11010...0

But if N is negative, say N = -.11010...0 = -13/16, then

$$\begin{aligned} 2 + N &= 10.0000....0 \\ &\quad - \underline{\quad .11010...0} \\ &= \quad 1.00110...0 \end{aligned}$$

The machine representation then for -13/16 is 1.00110...0

It is to be observed that the 2's complement of a negative fraction N can be formed by the following steps:

1) Represent |N| in its 40 bit form.

2) Change each zero to a one and each one to a zero.

3) Add 1 in the least significant position.

Example 1.3.2. Let N = -13/32

Step 1. $|N| = |-13/32| = 13/32 = 0.01101000\ldots0$

Step 2. Change 0's to 1's and 1's to 0's:

$$1.100101111\ldots1$$

Step 3. Add 1 to the last place: 1.10011000...00

Hence, -13/32 is represented by 1.100110...0

To change a negative fraction represented by a 2's complement number, M, to sign and absolute value form, use the following method:

1) Subtract 1 from the least significant digit of M.

2) Change 0's to 1's and 1's to 0's.

3) Place a minus sign in front of the result.

Example 1.3.3. Suppose it is desired to change 1.01101010...0 to a signed decimal fraction, following the above steps:

Step 1.
$$\begin{aligned} M &= 1.01101010\ldots0 \\ &\quad - \underline{\qquad\qquad 1} \\ &\quad 1.01101001\ldots11 \end{aligned}$$

Step 2. 0.10010110..00

Step 3. -.10010110 = -75/128

1.4. Arithmetic in the Binary System

Since binary and sexadecimal are essentially equivalent, one should try to learn how arithmetic is performed in one of these systems. Since the machine operates in binary, arithmetic in that system is presented here by means of the following table.

0 + 0 = 0	0 - 0 = 0	0 x 0 = 0	0 ÷ 0 = **
0 + 1 = 1	0 - 1 = 1*	0 x 1 = 0	0 ÷ 1 = 0
1 + 0 = 1	1 - 0 = 1	1 x 0 = 0	1 ÷ 0 = **
1 + 1 = 10	1 - 1 = 0	1 x 1 = 1	1 ÷ 1 = 1

*indicates a borrow from the next bit
**meaningless

Table 1.4.1.

An example of addition is: 01010011 + 01011101. Using the same form for addition as in the decimal system, one has, starting from the right

```
 01010011
 01011101
----------
 10110000
```

Starting from the right, 1 + 1 = 10--that is, 0 and 1 to carry. The next column gives 1 + 0 = 1, and then adding the carry from the previous column, 1 + 1 = 0 and 1 to carry, etc.

An example of subtraction is:

```
 01011011
-00101010
----------
 00110001
```

Here, the right column gives 1 - 0 = 1. The next gives 1 - 1 = 0; the next, 0 - 0 = 0, and so on. In the third column from the left there is 0 - 1 = 1 and 1 to borrow. Then the second column is 1 - 0 = 1, then less the one borrowed, 1 - 0 - 1 = 0.

An example of multiplication is:

```
    010110
  x 001101
----------
    010110
   000000
  010110
 010110
000000
000000
----------
00100011110
```

The only difficulty encountered in multiplication is adding several 1's together.

This may be alleviated if one adds the bits in decimal; e.g., 1 + 1 + 1 + 1 + 1 = 5. Then correct the sum to binary; i.e., 5 = 101; and write the right-most 1 of 101 as the sum of the five 1's and the 10 as the carry. Thus the carry can affect several columns at once.

An example of division is:

```
          101010 + remainder of 100/101
101 /11010110
    101
     110
     101
      111
      101
       100
```

1.5. Conversion Between Sexadecimal and Decimal

Since it is often necessary to convert fractions or integers from sexadecimal to decimal, methods of conversion are introduced here. In general, if N is an integer expressed in the base r system as

$$N_r = a_n r^n + a_{n-1} r^{n-1} + \ldots + a_1 r + a_0,$$

where $0 \leq a_i < r$ for $i = 0, 1, \ldots, n$; and it is desired to represent N in the base s system as

$$N_s = b_m s^m + b_{m-1} s^{m-1} + \ldots + b_1 s + b_0,$$

the following procedure applies:

Divide N by s. (This means divide N_r by s. Hence, in order for this division to mean anything, s must be expressed in the base r system. For example, if N is given in the sexadecimal as N_{16}, and N_{10} is wanted, divide N_{16} by 10, which when expressed in the sexadecimal is $10_{10} = K$.) Since, except for notational difference, $N = N_r = N_s$, then

$$\frac{N}{s} = \frac{N_r}{s} = \frac{N_s}{s}$$

and $\frac{N_r}{s} = I_1 + \frac{R_1}{s}$ and $\frac{N_s}{s} = b_m s^{m-1} + b_{m-1} s^{m-2} + \ldots + b_1 + \frac{b_0}{s}$

where I_1 and R_1 are integers, R_1 being the remainder. It can be shown that N/s produces a unique quotient and a unique remainder for a given N and s. Hence, R_1, the remainder of N_r/s, must be the same as b_0, the remainder of N_s/s. Hence, the least significant digit in the representation of N in the base s system is then the first remainder obtained by dividing N_r by s.

Since

$$I_1 = b_m s^{m-1} + \ldots + b_2 s + b_1$$

a repitition of the division process gives

$$\frac{I_1}{s} = I_2 + \frac{R_2}{s} = b_m s^{m-2} + \ldots + b_2 + \frac{b_1}{s}$$

so that the next significant digit of N_s is the remainder of I_1/s--namely, R_2. Since m is finite, this process can be repeated a finite number of times to produce N_s.

Example 1.5.1. Convert 1069_{10} to sexadecimal (and then to binary). Divide 1069 by 16 expressed in decimal. But 16 in decimal is 16. Hence,

$$\frac{1069}{16} = 66 + \frac{13}{16}$$

$$\frac{66}{16} = 4 + \frac{2}{16}$$

$$\frac{4}{16} = 0 + \frac{4}{16}$$

Since 13 = J, it follows that $1069_{10} = 42J_{16} = 010000101101_2$.

Example 1.5.2. Convert $3LF_{16}$ to decimal. It is necessary to divide by 10 expressed in sexadecimal as K. In order to divide by K, a multiplication table in sexadecimal for K is useful.

K x 1 = K	K x 5 = 32	K x 9 = 5K	K x J = 82
K x 2 = 14	K x 6 = 3N	K x K = 64	K x F = 8N
K x 3 = 1F	K x 7 = 46	K x S = 6F	K x L = 96
K x 4 = 28	K x 8 = 50	K x N = 78	

Then we divide as follows:

$$\begin{array}{r} 66 \\ K\overline{)3LF} \\ 3N \\ \hline 3F \\ 3N \\ \hline 2 \end{array} \qquad \begin{array}{r} K \\ K\overline{)66} \\ 64 \\ \hline 2 \end{array} \qquad \begin{array}{r} 1 \\ K\overline{)K} \\ K \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ K\overline{)1} \\ 0 \\ \hline 1 \end{array}$$

Hence, $3LF_{16} = 1022_{10}$.

Finally, a method for converting fractions from one base to another is also available. The method is so similar to that for converting integers that only the end result is given here.

To convert a base s fraction to a base r fraction, multiply the base s fraction by r; the integer part of the product is the most significant digit of the base r number. Then repeat this process with the fractional part of the product.

Example 1.5.3. Convert $.0765_{10}$ to sexadecimal and then to binary. Multiply by 16, getting

$$.0765 \times 16 = 1 + .2240$$
$$.2240 \times 16 = 3 + .5840$$
$$.5840 \times 16 = 9 + .3440$$
$$.3440 \times 16 = 5 + .5040, \text{ etc.}$$

so that

$$.0765_{10} = .1395\ldots_{16} = .0001\ 0011\ 1001\ 0101\ldots_2$$

Example 1.5.4. Convert $.K57_{16}$ to decimal. Multiply by K:

$$.K57 \times K = 6 + .766$$
$$.766 \times K = 4 + .9LN$$
$$.9LN \times K = 6 + .3J8$$
$$.3J8 \times K = 2 + .670$$
$$.670 \times K = 4 + .060$$
$$.060 \times K = 0 + .3N0, \text{ etc.}$$

Then, $.K57_{16}$ can be written as

$$.K57_{16} = .646240\ldots_{10}$$

Of course, one could notice that for short numbers, a direct conversion is always possible, particularly from sexadecimal to decimal. This conversion is based on the expanded sum of powers form for any number. For instance, let the problem be to convert $0L5_{16}$ to decimal. This number could be written

$$0L5_{16} = 0 \times 16^2 + 15 \times 16^1 + 5 \times 16^0 = 240 + 5 = 245$$

CHAPTER 2

A DESCRIPTION OF THE MISTIC

The MISTIC, like most digital computers, is composed of five sections: input, storage or memory, arithmetic, control and output. A knowledge of certain characteristics of these sections is indispensable to the programmer. A collection of such characteristics is presented here.

2.1. Input

The input section of the MISTIC is an electro-mechanical device for translating information presented to it into the language of electrical pulses accepted by the computer. Since the computer operates on binary numbers, the input accepted is binary. The input media now available are perforated tape and punched cards. The former is a long tape in which holes have been punched by an operator using a perforator, a machine similar to a typewriter. Across the width of the paper there is room for six holes, one of which is smaller than the other five. This small hole is a sprocket hole used for pulling the tape along through the tape punch and for providing the tape reader with the information that a character is present. Only the five larger holes have any information content as far as the programmer is concerned. There are ten such vertical lines per inch. The information on the tape is translated to the computer by means of a tape reader. A tape reader operates on the following principle: A bright light is allowed to shine above the tape. If there is a hole in the tape, the light shines through to a light sensitive surface. The configuration of spots of light corresponding to the holes in one line (called a character or frame) perpendicular to the length of the tape is then translated into electrical pulses which in turn are equivalent to the number represented by that frame. The reader pulls the tape under the beam of light, thus translating each frame as it passes by. The reader available for use with the MISTIC has a maximum speed of approximately 300 characters translated per second.

As mentioned above, there are five information bearing spaces per frame. Each space can be in one of two states, perforated or not. This implies that, used singly, these spaces could represent binary coefficients, which is indeed the way they are used for MISTIC. The binary numbers representable by five bit positions are 00000 to 11111, or 0 to 31. If only four of these five bit positions were used, there would result a number range from 0000 to 1111, or 0 to 15. But these are exactly the sexadecimal coefficients, so that if the tape preparation equipment had

keys for the 16 sexadecimal coefficients, one could perforate an entire frame by the use of a single key. Furthermore, ten such frames constitute one MISTIC 40 bit number. Finally, the fifth bit space could be used to change the meaning of the other four bit spaces, thus, for example, allowing alphabetic information to be prepared for computer input. For many convenient reasons, then, it is commonly the case that only four of the five bit spaces are used for numerical input.

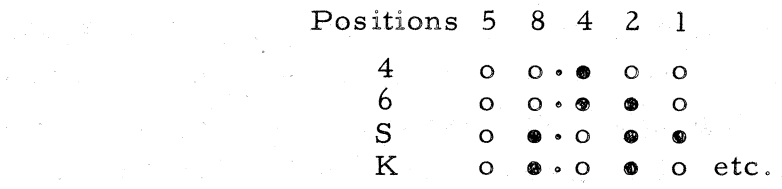
Example 2.1.1. If one wished to punch a tape which would input

0100 0110 1011 1010 0000 0000 1000 0010 1101 1111

it would be necessary to strike the keys labeled

4, 6, S, K, 0, 0, 8, 2, J, L

in that order. These would then appear on tape as



where the blackened circles indicate perforations and the unblackened ones indicate no perforation. The small perforations are the sprocket holes. The various columns are given the names fifth hole, eight hole, four hole, two hole and one hole.

When a tape has been prepared, it is often desirable to have its contents printed out for inspection. Likewise, tapes which have been prepared by the computer (which will be discussed in the section on output) are frequently printed out. The printing device is similar to a typewriter and requires format controls such as carriage return, line feed, space, delay. etc. These printer format controls are also punched on the tape, and are all characterized by having a fifth hole. Furthermore, under the standard form of input, such frames with a perforated fifth hole are ignored by the reader and never sent to the computer. As a result, one can generally use characters with perforated fifth holes as he needs. The use of the tape preparation equipment is discussed in detail in Chapter 12.

The punched cards used are IBM 80 column cards, with 12 rows. The top row is called the 12 row, the next row is called the 11 row, the next row is called the 0 row, and then on to the last row, which is called the 9 row. Used numerically, each column represents one decimal digit, indicated by a perforation in one of the corresponding rows 0 through 9.

The 11 and 12 rows are used to change numerical coding into alphabetic or control data, and also for algebraic sign data.

The structure and use of punched cards is presented in greater detail in Chapter 14.

2.2. Storage or Memory

Information is transmitted from the input reader to a register in the arithmetic section. From there the number assembled can be sent to storage.

Storage or memory is that part of the computer which holds information for a period of time, to be used by other parts of the computer. Theoretically, information will remain in storage indefinitely, but actually the reliability of data diminishes as the length of time that it is stored increases.

Memory is composed of 40 cathode ray tubes and associated circuitry. Cathode ray tubes are similar in appearance to television picture tubes, except that they have faces only a couple of inches in diameter. On the face of each tube is a square grid or coordinate system, composed of 1024 points, 32 points on a side. Each spot on the face of one tube represents one bit of the number in each memory position. The 40 tubes taken together then give the 40 bits of every number in memory. Since there are 1024 points on the face of each tube, then there are 1024 memory positions each capable of holding 40 bit numbers. As we shall see, each number may possess several meanings, roughly categorized as numbers which represent numbers and numbers which are codes for non-numeric information such as orders. Due to the double use of numbers, it has become the custom to speak of each 40 bit number as a word, to indicate that it may represent non-numeric data. In this terminology, the memory can hold 1024 forty bit words.

Each grid point on the face of a tube is assigned a number from 0 to 1023, with corresponding points on the other 39 tubes being assigned the same number. These numbers are called the addresses of memory locations. When the computer, under the direction of the programmer, wishes to obtain a word stored at a given memory location, it calls for the contents of a memory location with a given address. Thus, if at memory location 146 (the memory location whose address is 146), is stored the number L0 495 40 205 (in sexadecimal), then the contents of 146 is L0 495 40 205. The contents of a memory register can be acquired for use in about 18 microseconds (i.e., 18 one-millionths of a second).

Since the computer operates in binary, the addresses it uses are expressed inside the computer in binary (though as the reader will see later, the programmer thinks of these addresses as decimal). A moment's reflection will suffice to show that $1023_{10} = 1111111111_2$. Consequently, the addresses range from 0 to 1111111111, so that if all addresses are 10 bits long, then every address is representable. For convenience, these addresses are referred to outside the computer in sexadecimal, to give addresses ranging from 000 to 3LL in that system. By means of a certain mode of input, addresses will be considered decimal.

2.3. Arithmetic

The arithmetic section is that part of the computer in which addition, subtraction, multiplication, division and certain logical operations are performed. The MISTIC's arithmetic section is composed of three 40 bit registers and an adder. The registers are the accumulator, called the A register; the quotient register, called the Q register; and a third register, called R. (Also, A is called RI, Q is called RII, and R is called RIII, but this terminology will not be used here.) Each bit position of each of these three registers is composed of a flipflop, a certain two-stable-state device, and associated circuitry. Each register has specific functions to perform.

Words coming from memory to the arithmetic section have direct access to the R register and to the Q register. Transmission is parallel, meaning all 40 bits in a memory location are copied into R simultaneously, and similarly for Q. It is important to note that whenever a word is sent from a memory location to an arithmetic register, the contents of the memory location are not altered. Thus, at the end of such a transmission, both the arithmetic register and the memory register contain the same number. The Q register can be directly called into use while the R register is only an intermediate storage register over which the programmer has only indirect control. Actually, the programmer can ignore the R register for purposes of programming.

To send a word from the R register to the accumulator, the word must pass through a set of complement gates, whose function is to change the sign of the number passing through or to leave the sign unchanged depending upon the operation concerned. (The change of sign is accomplished by taking the 2's complement.) From there the number is added into the adder together with the contents of A, and the result is sent to A. If it is desired to send a word from memory into A, it is necessary to clear A first, meaning that A is made to contain 40 zeros. This is accomplished automatically by calling for the proper order (orders are discussed in Chapter 3).

The A and Q registers can be made to act like a 79 bit shifting register, called AQ. A left shift of one place means that the set of bits

$$a_0 a_1 a_2 a_3 \cdots a_{37} a_{38} a_{39} \qquad q_0 q_1 q_2 q_3 \cdots q_{37} q_{38} q_{39}$$

is replaced by

$$a_1 a_2 a_3 a_4 \cdots a_{38} a_{39} q_1 \qquad q_0 q_2 q_3 q_4 \cdots q_{38} q_{39} 0$$

A left shift of n places means that AQ is shifted one place, and that result shifted one place, until a total of n shifts has been completed. At that time a_{39} will have shifted over to the a_{39-n} position, and all the other bits will be associated in order with a_{39-n}.

A right shift of one place means that

$$a_0 a_1 a_2 a_3 \cdots a_{38} a_{39} \qquad q_0 q_1 q_2 \cdots q_{38} q_{39}$$

is replaced by

$$a_0 a_0 a_1 a_2 \cdots a_{37} a_{38} \qquad q_0 a_{39} q_1 \cdots q_{37} q_{38}$$

An n place right shift sends a_1 to a_{1+n} (provided n is less than 38).

Note that: (1) q_0 is unaffected by a shift, (2) right shifts propagate a_0, and (3) left shifts generate zeros in the right-most bit of Q.

This ability to shift is used as another means of transmitting words to the arithmetic section. Words are sent, four bits at a time, from the input reader directly to $a_{36} a_{37} a_{38} a_{39}$ in the accumulator. Then, if more bits are to be sent to A from the reader, the accumulator is left shifted four places and the new four bits sent to $a_{36} a_{37} a_{38} a_{39}$.

The only ways, then, that words can be sent into the arithmetic section are to R, to Q and to A from the input tape reader, and to AQ from the card reader.

When a word is to be sent from the arithmetic section, it usually leaves from A. Five such exits are possible.

1) For output to the tape punch (see the section on output), bits are sent four at a time from $a_0 a_1 a_2 a_3$ followed by a four-place left shift.

2) An entire forty bit word can be transmitted in parallel to any memory location.

3) The contents of bit locations $a_{10} \cdots a_{19}$ can be sent to the corresponding bit positions of any memory location.

4) The contents of bit locations $a_{30} \cdots a_{39}$ can be sent to the corresponding bit positions of any memory location.

5) The contents of A and Q can be punched on a card.

In transmitting words from A to memory, the contents of A are left unchanged.

Some of the uses of the various registers in arithmetic operation are listed here.

A Register

1) Holds one summand in an addition.
2) Holds the minuend in a subtraction.
3) Holds part of the dividend in division.
4) Receives the sum in addition.
5) Receives the difference in subtraction.
6) Receives part of a product.
7) Receives a "remainder" in division.

Q Register

1) Holds one factor in multiplication.
2) Holds half the dividend in division.
3) Receives the quotient in division.
4) Receives part of a product in multiplication.

The R register holds the remaining parts for each operation.

It is interesting to notice that an addition requires 100 microseconds and a multiplication 1000 microseconds.

Each arithmetic operation is called for by the control section.

2.4. Control

The entire computer must be given explicit directions as to how to perform, and the control section is that part of the computer which must accept, interpret and execute these directions. The directions that the control section uses are called orders or instructions, and the collection of orders (called a program) which a programmer wants the control to act on are available to control in the memory section. The orders are of four types: arithmetic, decisions, logical and manipulative. These orders are discussed in detail later, but it is important to know that each order is coded as a number. Some typical orders would be 40 205, L5 143, 26 065, and so on. There is no special place in memory reserved for orders, so that the only distinction made between orders and data is relative position, which is known to the programmer. However, once control knows where the first order of a set of orders is located, control will acquire each order from successive locations. If the programmer has written his orders well, the control will not try to acquire a data word as an order.

As will be seen later in this manual, orders are stored two to a word. Each order pair is transferred in its turn to the control section for execution. While in the control section, the pair of orders being executed is in a 40 bit flipflop register called the Instruction Register or R_3. The control section executes the left-hand order first, then the right-hand order. When a pair of orders has been performed, the control section must discover where the next order pair is located. The address of the next order pair to be acquired is in a ten bit flipflop register called the control counter. These two control registers are very important to the programmer in code checking a problem, and for that reason, their contents are displayed on the computer operator's console.

It is important to understand that the computer will do nothing until it has an order pair in its instruction register and an address in the control counter. This means that nothing can be input unless an order pair in the instruction register directs input to send in data. The dilemma of how to start the machine is

resolved, however, in the following manner: There is a white three-way switch on the control panel which has the settings FETCH, EXECUTE and RUN. There are also two push buttons, one labeled CONTROL COUNTER, the other, SET R_3. By setting the white switch to FETCH, and then depressing the two push buttons mentioned, the order pair 80 028 40 000 is automatically placed in the instruction register and the address 000 is put in the control counter. If a tape is inserted in the reader and the white switch moved to EXECUTE and then to RUN, the first order pair on tape is brought into the computer ready to be executed. This method of starting the computer is called a "bootstrap start" about which more will be said later.

2.5. Output

When the control section has executed a number of orders which have caused, for example, input, storage in memory and the execution of certain arithmetic steps, then it may be desired to send some of the information out of the computer. The section which translates from the machine language of electrical pulses representing binary digits to some mechanical medium is called the output section. There are three kinds of output available now for MISTIC. The first is a paper tape perforator which punches a tape which is entirely similar to, and in fact useful as, an input tape. The second kind of output is the Teletype printer. The tape punch operates at 60 characters per second. (Since these characters also include printing format controls, this is not necessarily 60 information characters per second.) The printer operates at the rate of 10 characters per second. The third kind of output available is a card punch. Maximum speed for this is 100 cards per minute. See Chapter 14 for details on the use of card equipment.

CHAPTER 3

INTRODUCTION TO PROGRAMMING--ORDER CODE

Programming for a computer is a two-fold operation. The first step in programming is the selection of the mathematical procedure to be used. This phase of programming is essentially a problem in numerical or statistical analysis and not within the scope of this manual.

The second step in programming, perhaps more properly called "coding", is that of translating the numerical method selected into machine language. It is this coding which is the concern of this manual.

3.1. Orders and Numbers in Machine Language

Information enters MISTIC in 40 bit units called words. A word may be intended as a number on which some arithmetic operation is to be performed. This use of a word is illustrated in Chapter 1.

Another use of a word is to contain instructions or orders for the various arithmetic operations to be performed.

The binary form of a word is exactly the same whether the word is to be interpreted as a number or as an instruction. Whether the machine acts on the word as a number or as an order depends only upon the way in which the word is encountered--i.e., as one of a sequence of instructions being obeyed or as an operand for an instruction.

3.2. The Make-up of Orders

An instruction in a digital computer involves the following five factors:

1) where to find the first operand,
2) where to find the second operand,
3) what to do with the two quantities,
4) where to put the result, and
5) where to go for the next instruction.

Thus, each instruction must contain, either explicitly or by implication, four addresses and an operation order.

MISTIC is a sequential machine; i.e., the next order to be obeyed is usually the next order in sequence. Hence, the fifth factor is automatically determined and need not be supplied in the instruction.

MISTIC is a single address machine; i.e., each instruction carries only one address (memory location identification number), which is that of the first

operand. The location of the second operand and where to put the answer must, therefore, be carried by implication in the instruction itself. In MISTIC these two locations are always one of the two registers, A and Q. The part of the instruction which specifies what operation to perform also specifies, by implication, in which register the second operand is found and in which register the answer is to be left. An instruction may say, for example, "Add the number in memory location 10 to the contents of A and leave the sum in A."

An order for MISTIC contains two symbols chosen from the decimal digits 0, 1, 2, ..., 9 and the letters (sexadecimal digits and hereafter called simply digits) K, S, N, J, F, L and one decimal number n in the range $0 \leq n \leq 1023$. The two symbols form what is called the operation code (op code). The number is an identification number (address) for one of the 1024 electrostatic memory locations in MISTIC; or, in some cases, a number is used for some purpose such as to indicate the number of times an operation is to be performed, to change addresses, as a counter, etc. The two-digit op code carries the information needed for factors 2, 3 and 4; the number carries the information for factor 1. Thus the order

L4 120

says: "Add the number in memory location 120 to the number in A, leaving the result in A."

Each of the 16 digits 0, 1, ..., 9, K, S, N, J, F, L can be represented by four bits since K, S, N, J, F, and L can be taken as representing 10, 11, 12, 13, 14 and 15. The addresses range from 0 to 1023 and hence are less than $1024 \equiv 2^{10}$ An order, therefore, requires at most 18 bits. Since MISTIC has 40 bits per word, a word can contain two such orders, or an order pair.

The two orders of an order pair are referred to as the left-hand order and the right-hand order and are obeyed in the sequence left to right. The digit or bit make-up of an order pair is as shown in Figure 3.2.1.

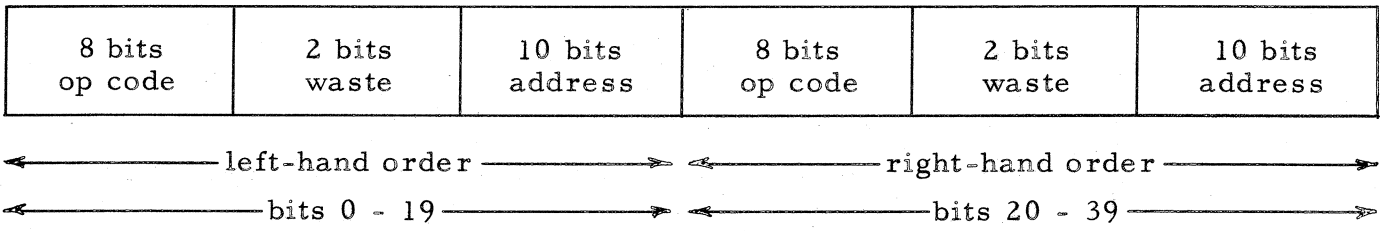


Figure 3.2.1.

3.3. The Order Code

A particular combination of two of the 16 digits 0, 1, 2, ..., 9, K, S, N, J, F, L was defined as an op code or simply a code, and the totality of such combinations which are accepted by the computer is called the order code of the machine.

MISTIC op codes are formed in such a way that all orders with a common first code letter are related. The format of the tabulated order code is, therefore, the following:

1) first code letter and descriptive title of orders;
2) codes and definitions;
3) explanatory discussion and special comments.

The letter n represents the memory location identification number unless specifically defined otherwise. In all cases, n is called an address. The letters A and Q stand for the accumulator and quotient registers, respectively. The symbols (n), (A), etc., are to be read "contents of memory location n", "contents of accumulator register", etc.

In all cases, "put (n) into A", "put (A) into n", etc., leaves the contents of n or A, respectively, undisturbed; i.e., it merely duplicates the contents of n in A, the contents of A in n, respectively.

The total number of possible combinations in the MISTIC order code is 256. A detailed discussion of each of these is presented in Chapter 10.

More than 100 out of the 256 possible codes will be accepted by MISTIC as legitimate instructions. These codes are explained in detail on the following pages. Following the detailed listing is a one page condensation of all usable order pairs, which is satisfactory for quick reference after the details of the orders have been learned.

Although there are over 100 instructions which MISTIC will obey, many of these are just duplications or minor variations of others and in some cases are not useful at all, occurring only as a by-product of the design logic necessary to provide a more fundamental order. Consequently, most problems can be programmed using as few as 20 op codes. Table 3.3.1 contains a list of op codes which will provide the user with a good fundamental set of orders, sufficient for writing most programs and a good basis on which to build a more complete working set of orders.

OP CODE	FUNCTION
26	Transfer Control
36	Conditional Transfer Control
40	Store (A) in Memory
50	Load Q from Memory
66	Divide
7J	Multiply
S5	Transfer (Q) to A
L0	Subtract
L4	Add
L5	Load A

Table 3.3.1.

0 AND 1

0--Left Shift (Multiplication by 2)

00 n Left shift contents of double register AQ n places

01 n Clear A, then execute 00 n

09 n Clear A, insert 1/2 in A, then execute 00 n

Left shift replaces the contents of AQ

$a_0\ a_1 \cdots a_{39}\ q_0\ q_1 \cdots q_{39}$

by

$a_1\ a_2 \cdots a_{39}\ q_1\ q_0\ q_2 \cdots q_{39}\ 0$

leaving q_0 unchanged.

1--Right Shift (Division by 2)

10 n Right shift contents of double register AQ n places

11 n Clear A, then execute 10 n

19 n Clear A, insert 1/2 in A, then execute 10 n

Right shift replaces the contents of AQ

$a_0\ a_1 \cdots a_{39}\ q_0\ q_1 \cdots q_{39}$

by

$a_0\ a_0\ a_1 \cdots a_{38}\ q_0\ a_{39}\ q_1 \cdots q_{38}$

leaving q_0 unchanged and propagating a_0.

The operation is repeated n times where n is interpreted modulo 64; i.e., if $n \geq 64$, n is replaced by the remainder when n is divided by 64. If n = 0, the machine will stop.

2 AND 3

2--Unconditional Control Transfer

20 n Stop. Transfer control to right-hand order in memory location n after start. Stop can be ignored by black switch setting.

22 n Transfer control to right-hand order at n.

24 n Stop. Transfer control to left-hand order in memory location n after start. Stop can be ignored by black switch setting.

26 n Transfer control to left-hand order at n

21 n, 23 n, 25 n, 27 n } Clear A, then execute { 20 n, 22 n, 24 n, 26 n

29 n, 2S n, 2J n, 2L n } Clear A, then insert 1/2 in A, then execute { 20 n, 22 n, 24 n, 26 n

3--Conditional Control Transfer

30 n If (A) $\geq$ 0, execute 20 n
If (A) $<$ 0, take next order in sequence.

32 n If (A) $\geq$ 0, execute 22 n
If (A) $<$ 0, take next order in sequence.

34 n If (A) $\geq$ 0, execute 24 n
If (A) $<$ 0, take next order in sequence.

36 n If (A) $\geq$ 0, execute 26 n
If (A) $<$ 0, take next order in sequence.

31 n, 33 n, 35 n, 37 n } Execute { 21 n, 23 n, 25 n, 27 n

39 n, 3S n, 3J n, 3L n } Execute { 29 n, 2S n, 2J n, 2L n

See Sections 10.5--10.8 for discussion of starting after stop in 20 n, 24 n, 30 n, and 34 n.

4 AND 5

4--Store

40 n Replace (n) by (A)

41 n Replace (n) and (A) by 0

49 n Replace (n) and (A) by 1/2

42 n Replace address digits of right-hand order at n by corresponding digits of A

43 n Clear A, then execute 42 n

4S n Replace (A) by 1/2 and the address digits of right-hand order at n by 0

46 n Replace address digits of left-hand order at n by corresponding digits of A

47 n Clear A, then execute 46 n

4L n Replace (A) by 1/2 and address digits of left-hand order at n by 0

5--Memory to Q

50 n Replace (Q) with (n)

51 n Clear A, then execute 50 n

59 n Put 1/2 in A, then execute 50 n

6 AND 7

6--Divide

66 n Divide (AQ) by (n), put rounded quotient in Q, residue in A
67 n Clear A, then execute 66 n
6L n Put 1/2 in A, then execute 66 n

After dividing, the least significant bit in Q (the quotient) is always 1 for round off. The residue is in A.

If $|(A)| > |(n)|$, the MISTIC will stop after dividing.
If $|(A)| = |(n)|$ and $(A) \geq 0$, the MISTIC will stop after dividing.
If $|(A)| = |(n)|$ and $(A) < 0$, the MISTIC will not stop after dividing.

7--Multiply

	Negative Multiply		Positive Multiply
70 n	Put $-(n).Q + 2^{-39}$ A in AQ	74 n	Put $(n).(Q) + 2^{-39}$ A in AQ
71 n	Clear A, then execute 70 n	75 n	Clear A, then execute 74 n
72 n	Put $-\|(n)\|.(Q) + 2^{-39}$ A into AQ	76 n	Put $\|(n)\|.(Q) + 2^{-39}$ A into AQ
73 n	Clear A, then execute 72 n	77 n	Clear A, then execute 76 n
79 n	Put 1/2 in A, execute 70 n	7J n	Put $(n).(Q) + 2^{-40}$ in A
7S n	Put 1/2 in A, execute 72 n	7L n	Put 1/2 in A, execute 76 n

The product is in AQ, the least significant 39 bits in Q with $q_0 = 0$.

Orders 71 n and 75 n are used as "unrounded multiply" orders.

Orders 79 n and 7J n are used as "rounded multiply" orders.

8--Input-Output

80 n Shift AQ four places left and replace a_{36}, a_{37}, a_{38}, a_{39} by the binary digits corresponding to a sexadecimal character on tape. This is repeated n/4 times

81 n Clear A, then execute 80 n

82 n Punch the digits a_0, a_1, a_2, a_3 as one sexadecimal character and shift AQ four places left. This is repeated n/4 times

83 n Clear A, then execute 82 n

8S n Put 1/2 in A, then execute 82 n

The address n is the number of bits to be read or punched and must be a multiple of 4.

The 80 orders by-pass any character with perforated fifth hole.

All 80 orders change the contents of A and Q.

9--Special Input-Output

91 4 Five hole input. Clear A, shift AQ four places right, replace a_{36}, a_{37}, a_{38}, a_{39} by the binary digits corresponding to the four least significant holes in one tape character. Place the contents of the fifth hole in position a_0

92 n Letter output. Punch on tape or print a character depending upon the address digits n

94 n Card input. Each of the 80 bits of A and Q are replaced by 1 if the corresponding card column(determined by the plugboard)of the row being read is punched, otherwise by a 0. The address n must be zero. See Chapter 14.

96 n Card output. Each of the 80 columns of the card now under the punch head is punched if a 1 is in the corresponding bit in AQ (determined by plugboard), otherwise there is no punch. The address n must be zero. See Chapter 14.

91, 92 orders change A and Q. A 96 order leaves both A and Q unchanged.

The following tabulation gives the complete orders for output of letters, numbers, symbols, and print format characters.

92 963F	Space
92 131F	Carriage return and line feed
92 515F	Delay--2 hole
92 3F	Delay--1 hole
92 707F	Numbers shift
92 259F	Letters shift

	After letters shift	After numbers shift
92 387F	A	)
92 195F	B	(
92 835F	C	:
92 67F	D	tab
92 194F	E	3
92 898F	F	F
92 579F	G	=
92 771F	H	apostrophe
92 514F	I	8
92 834F	J	J
92 642F	K	+
92 962F	L	L
92 643F	M	.
92 770F	N	N
92 578F	O	9
92 2F	P	0
92 66F	Q	1
92 258F	R	4
92 706F	S	-
92 322F	T	5
92 450F	U	7
92 323F	V	,
92 130F	W	2
92 451F	X	/
92 386F	Y	6
92 899F	Z	x

To output any character k times, add 4(k - 1) to the address of the appropriate order, $k \leq 16$.

K--Increment Add from Q

K0 n	Add $-(Q)-2^{-39}$ to (A)
K1 n	Put $-(Q)-2^{-39}$ in A
K2 n	If $(Q) \geq 0$, add $-(Q)-2^{-39}$ to (A) If $(Q) < 0$, add $(Q)+2^{-39}$ to (A)
K3 n	If $(Q) \geq 0$, put $-(Q)-2^{-39}$ in A If $(Q) < 0$, put $(Q)+2^{-39}$ in A
K4 n	Add $(Q)+2^{-39}$ to (A)
K5 n	Put $(Q)+2^{-39}$ in A
K6 n	If $(Q) \geq 0$, add $(Q)+2^{-39}$ to (A) If $(Q) < 0$, add $-(Q)-2^{-39}$ to (A)
K7 n	If $(Q) \geq 0$, put $(Q)+2^{-39}$ in A If $(Q) < 0$, put $-(Q)-2^{-39}$ in A
K9 n	Put $-(Q)-2^{-39} + 1/2$ in A
KS n	If $(Q) \geq 0$, put $-(Q)-2^{-39} + 1/2$ in A If $(Q) < 0$, put $(Q)+2^{-39} + 1/2$ in A
KJ n	Put $(Q)+2^{-39} + 1/2$ in A
KL n	If $(Q) \geq 0$, put $(Q)+2^{-39} + 1/2$ in A If $(Q) < 0$, put $-(Q)-2^{-39} + 1/2$ in A

F--Increment Add

F0 n	Add $-(n)-2^{-39}$ to (A)
F1 n	Put $-(n)-2^{-39}$ in A
F2 n	If $(n) \geq 0$, add $-(n)-2^{-39}$ to (A) If $(n) < 0$, add $(n)+2^{-39}$ to (A)
F3 n	If $(n) \geq 0$, put $-(n)-2^{-39}$ in A If $(n) < 0$, put $(n)+2^{-39}$ in A
F4 n	Add $(n)+2^{-39}$ to (A)
F5 n	Put $(n)+2^{-39}$ in A
F6 n	If $(n) \geq 0$, add $(n)+2^{-39}$ to (A) If $(n) < 0$, add $-(n)-2^{-39}$ to (A)
F7 n	If $(n) \geq 0$, put $(n)+2^{-39}$ in A If $(n) < 0$, put $-(n)-2^{-39}$ in A
F9 n	Put $-(n)-2^{-39} + 1/2$ in A
FS n	If $(n) \geq 0$, put $-(n)-2^{-39} + 1/2$ in A If $(n) < 0$, put $(n)+2^{-39} + 1/2$ in A
FJ n	Put $(n)+2^{-39} + 1/2$ in A
FL n	If $(n) \geq 0$, put $(n)+2^{-39} + 1/2$ in A If $(n) < 0$, put $-(n)-2^{-39} + 1/2$ in A
FF n	Stop. Used to indicate failure in program check.

$-(n) - 2^{-39}$ is the digitwise complement of (n).

The term "increment add" means the right-hand address of n is increased by 1 before the "add" order is executed.

N--Change Memory Banks

N0 n Go to next order

N2 n Change the bank number from which operands are coming to that indicated by the last two bits of n

N4 n Change the bank number from which orders are coming to that indicated by the last two bits of n

N6 n Change both bank numbers from which operands and orders are coming to that indicated by the last two bits of n

N1, N3, N5, N7 } Clear A, then execute { N0, N2, N4, N6

N9, NS, NJ, NL } Clear A, insert 1/2 in A, execute { N0, N2, N4, N6

L--Add

L0 n Subtract (n) from (A)
L1 n Put -(n) in A
L2 n Subtract |(n)| from (A)
L3 n Put - |(n)| in A
L4 n Add (n) to (A)
L5 n Put (n) in A
L6 n Add |(n)| to (A)
L7 n Put |(n)| in A
L9 n Put 1/2 - (n) in A
LS n Put 1/2 - |(n)| in A
LJ n Put 1/2 + (n) in A
LL n Put 1/2 + |(n)| in A

S--Add from Q

S0 n Subtract (Q) from (A)
S1 n Put -(Q) in A
S2 n Subtract |(Q)| from (A)
S3 n Put - |(Q)| in A
S4 n Add (Q) to (A)
S5 n Put (Q) in A
S6 n Add |(Q)| to (A)
S7 n Put |(Q)| in A
S9 n Put 1/2 - (Q) in A
SS n Put 1/2 - |(Q)| in A
SJ n Put 1/2 + (Q) in A
SL n Put 1/2 + |(Q)| in A

J--Extract

J0 n If corresponding digits of (n) and (Q) are both 1, put 1 in that place in Q; otherwise, put 0

J1 n Clear A, execute J0 n

J9 n Put 1/2 in A, execute J0 n

J0 forms the logical product of (n) and (Q).

00 Shift AQ left n $<$ 63 places.
01 A=0, shift AQ left n places.
09 A=1/2, shift AQ left n places.
0F Final stop

10 Shift AQ right n places.
11 A=0, shift AQ right n places.
19 A=1/2, shift AQ right n places.

20 STOP, transfer control to n'.
21 STOP, A=0, execute 20.
22 Transfer control to n'.
23 A=0, transfer control to n'.
24 STOP, transfer control to n.
25 STOP, A=0, execute 24.
26 Transfer control to n.
27 A=0, transfer control to n.
29 STOP, A=1/2, execute 20.
2S A=1/2, transfer control to n'.
2J STOP, A=1/2, execute 24.
2L A=1/2, transfer control to n.

3V A $\geq$ 0, do 2V A $<$ 0, go on.

40 (A) $\rightarrow$ N(n).
41 0 $\rightarrow$ A and N(n).
42 Replace address of n' by A.
43 A=0, execute 42.
46 Same as 42 except L.H. order.
47 A=0, same as 43 except L.H.
4S A=1/2, 0 to n' address digits.
4L A=1/2, 0 to n address digits.

50 N(n) $\rightarrow$ Q.
51 A=0, N(n) $\rightarrow$ Q.
59 A=1/2, N(n) $\rightarrow$ Q.

66 Divide AQ by N(n), rounded
67 Divide AQ by N(n), not rounded
6L A=1/2, execute 66

70 $-N(n)\ Q + 2^{-39}\ A \rightarrow AQ$.
71 $-N(n)\ Q \rightarrow AQ$.
72 $-|N(n)|\ Q + 2^{-39}\ A \rightarrow AQ$.
73 $-|N(n)|\ Q \rightarrow AQ$.
74 $N(n)\ Q + 2^{-39}\ A \rightarrow AQ$.
75 $N(n)\ Q \rightarrow AQ$.
76 $|N(n)|\ Q + 2^{-39}\ A \rightarrow AQ$.
77 $|N(n)|\ Q \rightarrow AQ$.
79, 7S, 7J, 7L same as 70, 72, 74, 76 except A = 1/2.

N2 Change OPERAND bank.
N4 Change ORDER bank.
N6 Change BOTH banks.

80 Read n/4 numbers from tape.
82 Punch n/4 numbers on tape.
81, 83 A=0 then 80, 82.
8S A=1/2 then 82.

91 Read 1 tape character.
92 Punch 1 tape character.
94 Read from card.
96 Punch on card.

K0 $-Q - 2^{-39} + A$ to A
K1 $-Q - 2^{-39}$ to A. (complement)
K4 $Q + 2^{-39} + A$ to A.
K5 $Q + 2^{-39}$ to A.
K9 $-Q - 2^{-39} + 1/2$ to A.
KJ $Q + 2^{-39} + 1/2$ to A.
K2, K3, K6, K7, KS, KL same as K0, K1, K4, K5, K9, KJ if Q $\geq$ 0. Same as K4, K5, K0, K1, $\overline{\text{KJ}}$, K9 if Q $<$ 0.

S0 A - Q to A.
S1 -Q to A.
S4 A + Q to A.
S5 Q to A.
S9 1/2 - Q to A.
SJ 1/2 + Q to A.
S2, S3, S6, S7, SS, SL same as S0, S1, S4, S5, S9, SJ except use |Q|.

J0 Logical Q. N(n) $\rightarrow$ Q.
J1 A=0, execute J0.
J9 A=1/2, execute J0.

F0 $-N(n) - 2^{-39} + A$ to A.
F1 $-N(n) - 2^{-39}$ to A.
F4 $N(n) + 2^{-39} + A$ to A.
F5 $N(n) + 2^{-39}$ to A.
F9 $-N(n) - 2^{-39} + 1/2$ to A.
FJ $N(n) + 2^{-39} + 1/2$ to A.
F2, F3, F6, F7, FS, FL same as F0, F1, F4, F5, F9, FJ, if N(n) $>$ 0. Same as F4, F5, F0, F1, $\overline{\text{FJ}}$, F9 if N(n) $<$ 0.

L0 A - N(n) to A.
L1 -N(n) to A.
L4 A + N(n) to A.
L5 N(n) to A.
L9 1/2 - N(n) to A.
LJ 1/2 + N(n) to A.
L2, L3, L6, L7, LS, LL same as L0, L1, L4, L5, L9, LJ except use |N(n)|.

CHAPTER 4

DECIMAL ORDER INPUT

In the description of the make-up of orders, the "address" was described as a decimal number. Since MISTIC is a binary machine, obviously this decimal address must be converted to binary before entering the machine. This can, of course, be done by the programmer by converting all decimal addresses to sexadecimal, each of which is read as four bits by MISTIC.

Fortunately, this is not always necessary, due to a special input program called the Decimal Order Input (DOI).

The Decimal Order Input is a program which, when read into and stored in MISTIC, reads in any program written in a prescribed format, changes the orders to ordinary MISTIC format, and stores this program in specified locations in the machine.

4.1. Format for Words to be Input by the DOI

The prescribed format in which a program must be written to be accepted by the DOI involves the following:

1) Every half word must contain a two-digit op code whether the word is an order pair or a number.

2) Every address (second part of each half word) must be a decimal number, of zero or more digits.

3) Every half word must terminate in one of the six symbols: K, S, N, J, F or L.

The required format for a word is thus

2 digit op code	decimal digits	1 digit term.	2 digit op code	decimal digits	1 digit term.

Figure 4.1.1

To illustrate:

```
L5 150F   L4 200L
50     F  26 200F
```

are order pairs in acceptable DOI format. The termination symbols are not stored but are used by the DOI.

4.2. Directives

A directive is a half word (one order) which is read by the DOI and executed but not stored in memory.

Terminations N and K indicate that the order is a directive. These orders must follow a right-hand order. The first order following a K directive is treated as a left-hand order. A waste order must be inserted if necessary to insure that the directive does follow a right-hand order.

A directive of the form

op code m K

causes the following to occur:

1) The next order pair read from tape by the DOI is stored at address m, with the following words stored in sequence.

2) In following orders which are terminated in L, the address is increased by m before the word is stored.

Both the sequential storing and the address increase will continue until a new directive (K or N termination) is encountered. The particular K-terminated order usually is 00 mK although any op code will work.

A directive terminated by N will usually have an unconditional transfer order (20, 22, 24 or 26) as the op code, the address associated being the address of the first order to be obeyed in the program. This directive indicates the end of the input and transfers control from the DOI to the program now stored in the computer.

4.3. Word Assembly by the DOI

The format of orders permits the DOI to separate words into order pairs, since each decimal address is terminated by a letter and since each op code contains exactly two digits.

A word is assembled by the DOI in the following manner: two digits (op code) are read from tape, shifted to the right-hand order op code position (bits 20 through 27), and placed in location 1. The decimal digits are read and converted to binary. These digits are stored in location 0 during and at the end of the conversion process. When the terminating symbol is read, the address in location 0 is added to the op code in location 1, the whole stored in 1. At this point, the next step depends on the terminating symbol. The effect of the various terminating symbols is given in tabulated form in Table 4.3.1.

It should be noted that during order assembly, when a left-hand order has been assembled and stored, it is stored in the right half of 1 and location n. Location 1 always contains two orders, but it is only after a right-hand order has been assembled that the two orders in location 1 constitute a proper order pair. Also, only after a right-hand order has been assembled does the memory location being loaded contain a complete word or order pair. If an N directive follows a left-hand order, control is sent to the right-hand side of 0 which contains the

address of the directive with a 00 op code. Hence if the left-hand order in 1 does not stop the computer, control will then pass to the right-hand side of 1. However, the order preceding the N directive will not be stored in the correct place.

Terminating Symbol	Procedure
K	Address is stored in location 2 and next order read.
F	Address is added to op code, whole stored in location 1 and also in the desired address n. This address is determined by the address in the last K directive plus the number of orders between this directive and the present order. If the order is a left-hand order, the word in location 1 is shifted 20 places left, the next order is read and assembled and added to the contents of 1, the whole word stored in the proper location. The store address is raised by 1 at the end of each order pair.
L	The address is increased by the number in location 2, determined by the last K directive, and the procedure under F followed.
N	The order in the right side of 1, which is the order terminated by N, will be executed. This is the order which transfers control from the DOI to the stored program.
J	The general procedure under F is followed in this case except that the address is multiplied by $2^{39}/10^{12}$ before adding to op code and storing. This is a decimal fraction input and is discussed in Section 4.5.
S	The termination S is always followed by a digit--one of the digits 3, 4, . . . , K, S, N, J, F, L. This termination causes the contents of the location whose address is the particular terminating digit to be added to the order address before storing. This termination provides for preset parameters and is discussed in detail in Section 6.4.

Table 4.3.1

4.4. Fixed and Relative Addresses

The F and L terminations in the DOI format are used to distinguish between a fixed address and a relative address.

A fixed address is an address which is a definite memory location or a fixed number and is designated by an F termination.

A relative address is one which is not a memory location number but which is transformed into a memory location number during the read-in of the program. In particular, the relative addresses in a program written for DOI input, specified by a terminating L, refer to the order number in the program starting with 0L for the first order, 1L for the second order, etc.

For example, consider the following sequence of orders, where the addresses underlined are the ones different for the different methods of writing the program.

```
100)  L5 200F
      L4 300F
101)  40 400F
      L5 100F
102)  L4 105F
      40 100F
```

The same program beginning at location 14 would be:

```
14)   L5 200F
      L4 300F
15)   40 400F
      L5 14F
16)   L4 19F
      40 14F
```

Written using relative addresses this program would be:

```
0)    L5 200F
      L4 300F
1)    40 400F
      L5 0L
2)    L4 5L
      40 0L
```

(The zeros in orders 1 and 2, i.e., L5 0L and 40 0L, need not be inserted. L5 L and 40 L suffice.)

An obvious advantage of relative addresses is that it is unnecessary to decide where a program or part of a program is to be stored in the computer before the program is written. Relative addresses also facilitate writing a program in parts, to be combined later into a complete program.

4.5. Input of Decimal Fractions and Integers

The DOI forms an address from the decimal digits on tape and adds that address to the op code. This address could be as large as $2^{39} - 1$ without getting into the sign digit. Thus, any positive integer can be input by the DOI as $n \times 2^{-39}$ by letting the left-hand order and the right-hand op code be zero and the right-hand address be n, with terminations F on both orders. Thus,

```
00 F   00 520F
```

will cause 520×2^{-39} to be input.

Since numbers are input as $n \times 2^{-39}$, a 12-digit right-hand address smaller

than $2^{39} \cong 5.5 \times 10^{11}$ can be converted to a decimal fraction by multiplying it by $2^{39}/10^{12}$. The terminating symbol J causes this multiplication to be effected. For example,

00 F 00 2765 0000 0000J

causes 0.2765 to be placed in the memory in binary form.

Since $2^{39} \cong 5.5 \times 10^{11}$, positive fractions not greater than .5 can be input by the DOI in this manner. To input decimal fractions in the range $.5 < d < 1$, replace the left-hand op code by 40. Thus,

40 F 00 d J

will cause 1/2 + d to be input.

To input negative decimal fractions, the op codes 80 and N0 may be used. Since 80 F represents -1 and N0 F represents -1/2, the orders

80 F 00 d J

and

N0 F 00 d J

will cause -1 + d and -1/2 + d, respectively, to be input. Thus, -.2376 is input by

N0 F 00 2624 0000 0000 J

-.7862 is input by

80 F 00 2138 0000 0000 J

and .7765 is input by

40 F 00 2765 0000 0000 J

Both op codes must be included and terminating zeros, but not leading zeros. Thus, the number .003769 will be input by

00 F 00 37 6900 0000 J

Table 4.5.1 shows the relationship between the range of the number, the op code, and the right-hand address digits.

If range of decimal number B is	The DOI format is	Where w is found by
$0 \leq B < 1/2$	00 F 00 wJ	$w = B \cdot 10^{12}$
$1/2 \leq B < 1$	40 F 00 wJ	$w = (B-1/2) \cdot 10^{12}$
$-1 \leq B < -1/2$	80 F 00 wJ	$w = (1+B) \cdot 10^{12}$ or $(1-\lvert B\rvert) \cdot 10^{12}$
$-1/2 \leq B < 0$	N0 F 00 wJ	$w = (1/2+B) \cdot 10^{12}$ or $(1/2-\lvert B\rvert) \cdot 10^{12}$

Table 4.5.1

Use of the J termination to input decimal fractions is not recommended for more than a few numbers. Special input programs are available when large

amounts of data are needed. (See Chapter 6).

The use of the remaining termination, S, will be discussed in Chapter 6.

CHAPTER 5

NON-ITERATIVE AND ITERATIVE PROGRAMMING

Use of the order code to construct a program unit is a relatively simple matter. The best way to describe the process of program construction is by examples.

5.1. Non-iterative Programming

Consider the problem of constructing a program to compute the quantity

$$ab + c = d$$

Written first in terms of the quantities a, b and c rather than addresses, this program would be

```
50  a      Put a into Q
7J  b      Form a·b in AQ (rounded product)
L4  c      Add c to most significant part of AQ in A
40  d      Store d
```

If the quantities a, b and c are now assumed to be in locations 100, 101 and 102 and the answer is to be placed in 103, the program in DOI format will be

```
0) 50  100F    (100)≡a ⟶ Q
   7J  101F    a·b ⟶ AQ
1) L4  102F    a·b + c ⟶ A
   40  103F    a·b + c ⟶ 103
```

This program, written in DOI format, is numbered with the first order given the number 0. The actual address of these orders in MISTIC would be determined by a K directive.

As a further example, consider the same problem--ab + c = d--where a = .3276, b = .8542, and c = -.2864. These numbers are to be input as part of the program, the whole to be written in DOI format to be stored at memory location 96 and following. The answer is to be stored in 200. The program is as follows:

```
   00  96K
0) 50  3L  . . . . . . . . . . . a ⟶ Q
   7J  4L  . . . . . . . . . . . ab ⟶ AQ
1) L4  5L  . . . . . . . . . . . ab + c ⟶ A
   40  200F . . . . . . . . . . . ab + c ⟶ 200
2) 0F  F   . . . . . . . . . . . Stop
   0F  F   . . . . . . . . . . . Waste
3) 00  F  00 3276 0000 0000J .  }
4) 40  F  00 8542 0000 0000J .  } Constants
5) N0  F  00 2136 0000 0000J .  }
   26  96N . . . . . . . . . . . Stops input, starts execution
                                  of program
```

Table 5.1.1

5.2. Iterative Programming--No Changing Addresses

A somewhat more difficult but much more useful type of programming is made possible by the control transfer orders. Use of these control transfer orders permits the construction of loops, by means of which one set of orders may be used for any number of iterations, with or without alteration.

Such an iterative program involves a counter, which may be either a constant entered for the purpose or an order which has a changing address. In either case, a constant, usually a 1 in either the 19th or 39th bit position, is required for decreasing or increasing the counter. As an illustration, consider the following examples.

Example 5.2.1. Construct a program which will form $\sum_{x=1}^{10} a^x$, where a, a number in the range $-.1 \leq a \leq .1$, is in address 100. The result is to be stored in address 101.

Solution: Constants 8×2^{-39} and 1×2^{-39} are input for counting. The program is as follows:

1)	L5	100F	$a \longrightarrow A$
	40	101F	$a \longrightarrow 101$
2)	40	F	$A \longrightarrow 0$ (working space)
	50	100F	$a \longrightarrow Q$
3)	7J	F	$a \cdot a^{i-1} \longrightarrow AQ$
	40	F	$a^i \longrightarrow 0$
4)	L4	101F	$a^i + \sum_{x=1}^{i-1} a^x \longrightarrow A$
	40	101F	$\sum_{x=1}^{i} a^x \longrightarrow 101$
5)	L5	8L	Counter to A
	L0	9L	Decrease counter
6)	40	8L	Restore counter
	32	2L	Loop if counter is ≥ 0
7)	0F	F	Stop if counter is < 0
	0F	F	Waste
8)	00	F 00 [8]F	8×2^{-39}--counter
9)	00	F 00 1F	1×2^{-39}--constant

Table 5.2.1

In order to assure that the counter is properly set at the beginning of the loop, it is good practice to make all loops self-restoring. In the above program, this will involve adding an order at the beginning and a constant at the end, namely,

0) L5 10L
40 8L

and

10) 00 F
008 F

Another counting method, using the order F5 n, is illustrated in Table 5.2.2,

which is another program for the problem of example 5.2.1.

0) 41	9L	Clear counter
L5	100F	$a \longrightarrow A$
1) 40	101F	$a \longrightarrow 101$
40	F	$a \longrightarrow 0$
2) 50	100F	$a \longrightarrow Q$
7J	F	$a \cdot a^{i-1} \longrightarrow AQ$
3) 40	F	$a^{i} \longrightarrow 0$
L4	101F	$a^{i} + \sum_{x=1}^{i-1} a^{x} \longrightarrow A \longrightarrow 101$
4) 40	101F	
F5	9L	$c_i + 1 \longrightarrow 9$
5) 40	9L	
L0	8L	$c_i + 1 - (9) \longrightarrow A$
6) 36	7L	Transfer control to 7L
26	2L	Reloop if $c_i + 1 < 9$
7) 0F	F	Stop when $c_i + 1 = 9$
0F	F	Waste
8) 00	F 00 9F	
9) 00	F 00 F	

Table 5.2.2

5.3. Iterative Programming--Changing Addresses

A second type of loop is one in which some addresses are changing. In the following example, the loop has been made self-restoring and a changing address is used as a counter.

Example 5.3.1. Construct a self-restoring program which will evaluate the polynomial $\sum_{i=0}^{12} a_i x^i$, where the a_i and x are assumed to be scaled so as to assure that the sum lies always in machine range, i.e., between -1 and +1. The a_i's are stored in 101, ..., 113, x in 100, and the final answer is to be in 99. Use a nested form of factoring.

Solution: Using the form

$$\sum_{i=1}^{12} a_i x^i = [---[([(a_{12}x + a_{11})x + a_{10}]x + a_9)x + a_8]--- + a_1]x + a_0$$

$$= \sum_{j=0}^{12} (S_{12-j}x + a_{11-j})$$

where $S_{12} = a_{12}$

and $S_{12-j-1} = S_{12-j}x + a_{11-j}$

for $j = 0, 1, \ldots, 11$

the desired program might be the following:

0) L5	9L	Preset order 3
40	3L	
1) L5	113F	$a_{12} \rightarrow 0$, working space
40	F	
2) 50	F	$S_{12-j} \rightarrow Q$
7J	100F	$S_{12-j}{}^{x} \rightarrow AQ$
3) L4	[112]F	$S_{12-j-1} \rightarrow A \rightarrow 0$
40	F	
4) L5	3L	Decrease address in 3
L0	8L	
5) 40	3L	
L0	7L	Test for end
6) 36	2L	Loop if $(A) \geq 0$
0F	F	Stop if $(A) < 0$
7) L4	101F	Test constant
40	F	
8) 00	1F	Constant
00	F	
9) L4	112F	For presetting order 3
40	F	

Table 5.3.1

In this program, the order pair at 3L with a decreasing address was used as the counter with the test constant in 7L. Using a changing address in this manner eliminates at least one order pair from the program.

When writing a program which has several changing addresses, it is sometimes difficult to write the program with the "housekeeping" or program-restoring orders at the beginning of a program without upsetting the order numbers. This difficulty may be avoided by making the address-setting orders into a separate program part, with its own 00 mK directive and with the last order a control transfer to the main program. The N directive will then send control to the address-setting program or to the first of a sequence of orders in the main program which, in turn, sends control to the address-setting routine.

CHAPTER 6

SUBROUTINES

A problem to be programmed may often be broken into several semi-independent steps, each of which can be programmed separately. Thus, the programmer is able to concentrate on one step at a time and to test each part of the program separately before incorporating it in the total program.

Since certain computations are common to many problems, programs can be prepared for these computations and a program used whenever a particular computation is called for. Such programs are called routines, or subroutines when used as part of a larger routine or program. The collection of routines and subroutines which is available for use on the MISTIC is called the Library of routines. A complete listing of MISTIC library routines is included at the end of this chapter.

6.1. Types of Routines in the MISTIC Library

The Routine Library for MISTIC consists of four types of routines: complete programs, open subroutines, closed subroutines, and interpretive routines.

A complete program is just what the name implies--a program which will perform a complete set of operations and for which the user need supply only a data tape and perhaps a parameter tape.

An open subroutine is a set of orders which performs a certain operation and which may be inserted at an appropriate place in the program.

A closed subroutine is also a set of orders which performs a specified operation but which is not inserted in the program but placed at some arbitrary section of memory. Control is transferred to the closed subroutine by a control transfer order in the main program or master routine.

The closed subroutine is probably the most useful since control may be transferred to the subroutine many times in a particular program, either as one part of a loop or by separate instructions.

6.2. Entry into and Exit from a Closed Subroutine

To be useful, a closed subroutine must be written in such a way as to be applicable in a variety of programs. The closed subroutines in the MISTIC Library are entered by means of two half-word orders, called a standard entry. These two orders are:

```
p   )  any
       50 p
p+1)   26 m     m is the address of the first order
       any      in the subroutine
```

The first order puts the order 50 p in the right-hand half of Q; the second order transfers control to the subroutine. These two orders must be located in the right-hand half of one word and the left-hand half of the following word even if a waste order is required to do so.

Any variation of this standard entry in a library subroutine is explained in the program description.

Control is automatically transferred back to the master routine by the subroutine. This control transfer is usually accomplished by the two orders in the subroutine

```
K5    F
42  n L
```

where n is the number of an order in the subroutine which contains, in the right-hand half, a control transfer order

```
22 (  )
```

The K5 order puts 50 p+1 in the right-hand half of A; the 42 n order puts p+1 in the address part of the 22 order in the right-hand half of m+n (m is the location of the first order of the subroutine). This process is called "planting the link".

6.3. Placing the Argument

Since a subroutine usually performs an operation on one or more quantities, these quantities must be provided by the master routine. If only one such quantity is required, it is usually placed in A before entry into the subroutine. In this case, the K5 order is preceded by a 40 q order to preserve the quantity in A, which would otherwise be destroyed by the K5 order.

6.4. Parameters--Program and Preset

Many subroutines require certain parameters. For example, a routine which finds the nth root of a number, where n is any positive integer, requires that the number n be supplied by the master routine. Or a subroutine may operate on more than one quantity, so that either the number of such quantities or the address of the first of a sequence of such quantities must be supplied. There are two standard ways of supplying parameters.

A program parameter is specified in one of the orders in the master routine. If only one such parameter is required, it is specified in the first half of the word containing the 50 order in the standard entry to a subroutine. For example, Library Routine R2 is a closed routine for replacing (A) by its nth root. The entry to R2 is

```
p   )  50 n      program parameter
       50 p
p+1)   26 m
```

A second method of supplying parameters leads to what are called preset parameters.

A preset parameter is one which is usually set during the input of the routine. The S termination symbol of the DOI is used for this parameter setting. The parameter, which may represent either a constant or an address, is placed by the master routine in one of the addresses 3, 4, ..., K, ..., L. The subroutine is written with kS3, kS4, ..., as the address of the orders which use the constant or address thus placed. As the program is read in, the kS3, kS4, ... will be replaced by the constants at address 3, 4, ... plus k. Thus, the order L5 S3 will go into the computer as L5 100 if (3) is 100, whereas the order L5 6S3 will go into the computer as L5 106 if (3) is 100.

6.5. Examples of Closed Subroutines

Example 6.5.1. Write a closed subroutine which will form $\sum_{i=0}^{n} a_i b_i$ in A.

Solution: Let the address of a_0 be specified by S3, and the address of b_0 by S4. The program parameter is n. On entry to the subroutine, 50 n 50 p is in Q. The subroutine is

0)	K5	F	
	42	9L	Link
1)	46	12L	Program parameter--for test
	L5	10L	
2)	40	3L	Preset order 3L
	41	F	Clear 0--working space
3)	50	S3	$a_i b_i$
	7J	S4	
4)	L4	F	$\sum_{j=0}^{i-1} a_j b_j + a_i b_i \longrightarrow 0$
	40	F	
5)	L5	3L	
	L4	11L	Raise address in 3L
6)	40	3L	
	L5	12L	
7)	L0	13L	Test constant
	40	12L	
8)	36	3L	
	L5	F	$\sum_{i=0}^{n} a_i b_i \longrightarrow A$
9)	22	9L	Waste
	22	( )F	Set by 0L
10)	50	S3	
	7J	S4	For preset 3L
11)	00	1F	
	00	1F	Constant
12)	00	(n)F	
	00	F	Set by 1L
13)	00	1F	
	00	F	Constant

Table 6.5.1

The following must be on the program tape at some point preceding the subroutine:

```
00  3K
00  F   00 m1F      m1 is address of a0
00  F   00 m2F      m2 is address of b0
```

Example 6.5.2. Write a program to compute the cube root of each of a sequence of six numbers:

+.127632
-.027543
-.987653
+.002764
+.756532
-.000376

The numbers are to be read in by Library Routine N2. Library Routine P1 is to be used to print out the numbers and the answers. Numbers are to be printed as six digits with a space after the first three. The answers are to be printed as nine digits with a space after the first digit. The print format is to be

number 2 spaces answer
number 2 spaces answer, etc.

Descriptions of the three library routines needed are given at the end of the chapter.

Solution:

The Tape Format

```
DOI
00  10K
Master Routine
00  150K
N2
00  176K
P1
00  205K
R2
24  10N
+127632
-027543
-987653
+002764
+756532
-000376N
```

Table 6.5.2

```
   Master Routine

 0) L5  16L
    42  6L        Preset changing orders
 1) 42  3L
    92  131F      Carriage return
 2) 50  100F
    50  2L        Entry to N2
 3) 26  150F
    L5  [100]F
 4) 52  63F
    50  4L        Print number
 5) 26  176F
    92  967F      Print 2 spaces
 6) 22  6L        Waste
    L5  [100]F
 7) 50  3F        Program parameter
    50  7L
 8) 26  205F      Entry to R2
    26  9L        Waste
 9) 52  91F
    50  9L        Print answer
10) 26  176F
    92  131F      L.F. and C.R.
11) 92  515F      Delay
    F5  3L
12) 40  3L        Increase address in 3L and 6L
    42  6L
13) L0  15L
    32  14L       Test for end
14) 22  3L        Loop
    0F  F
15) 26  150F
    L5  106F      Test constant
16) 26  150F
    L5  100F      Constant for preset changing orders
```

Table 6.5.3

Notice the use of the 92 131F order in 1L. This assures that the carriage is in the proper position at the beginning of print-out. In address 11L, the order 92 515F after the 92 131F is used to allow time for the carriage return to be completed before the next print-out starts.

6.6. Interpretive Routines

The fourth type of routine in the program library is the interpretive routine.

An interpretive routine is a routine which reads an order which is not in MISTIC format, then interprets and executes the order before reading the next order. The orders, or pseudo-orders since they are not orders found in the ordinary MISTIC order code, may be either on tape or in memory. The operations performed in executing the pseudo-order are performed by obeying certain blocks

of orders in the interpretive routine which are orders in the regular MISTIC order code.

An interpretive routine is commonly entered by a standard entry. Following the entry the master routine must have the pseudo-orders to be read by the interpretive routine. A special pseudo-order will transfer control back to the master routine.

Interpretive routines may be used in programs which involve actual operations on elements which are not numbers stored in the usual way but may be numbers stored in some special form. The most common application is to numbers stored in "floating point" and to complex numbers.

Interpretive routines are also used to permit programs written for one computer to be read and executed by a different computer.

6.7. Library Routine N2

TITLE . . . . . Input a Sequence of Decimal Fractions

TYPE . . . . . Closed with one program parameter

NUMBER OF WORDS 26

PURPOSE . . . . . To read a set of signed fractions and store them in successive locations

TEMPORARY STORAGE . . 0, 1, 2

ACCURACY . . . . . $\pm 2^{-40}$

DURATION . . . . . 4 ms per digit (input time)

DESCRIPTION . . . . . This routine at location p is called into use by the orders

q	-- nF
	50 qF
q+1	26 pF

n is the first of the sequence of locations at which the decimal fractions are stored.

It reads a sequence of decimal fractions from the tape. Each fraction is punched on the tape as sign, K(+) or S(-), followed by up to 12 decimal digits; the final fraction in the sequence is terminated by an N, J, F or L. When this subroutine reads one of these terminating symbols, it returns control to the main routine. Upon leaving this routine, 0×2^{-39}, 1×2^{-39}, 2×2^{-39}, or 3×2^{-39} is left in the accumulator according to whether the terminating symbol is N, J, F or L.

This program works by bringing in the decimal fraction $a_0, a_1, a_2, \ldots, a_p$ one digit at a time. Now this number $a_0, a_1, a_2, \ldots, a_p = N_p/D_p$ where a_0 is the is the sign of the number and $p \leq 12$.

When the i+1 digit is read, we have stored $D_i/2 = D_{i-1}/2 \times 10$ in location 1 and $M_i = N_i - D_i/2$ in location 0 where $N_i = 10\, N_i - 1 + a_i$. $D_0/2 = 5$, $M_0 = a_0 - 5$, $N_0 = a_0$.

6.8. Library Routine P1

TITLE . . . . . Print One Number Fractional or Integer in a Manner Determined by a Program Parameter

TYPE . . . . . Closed with one program parameter. If the routine starts at x, then it is called into use by the orders

p	XY dF 50 pF
p+1	26 xF

PURPOSE . . . . . To print decimal integers or fractions, with or without sign, to a specified number of places with simulated decimal point

NUMBER OF WORDS 28

TEMPORARY STORAGE . . 0, 1, 2

ACCURACY . . . . . Will print a fraction to 11 significant figures

DURATION. . . . . Determined by punch speed

DESCRIPTION . . . . . When the subroutine is called into use, it punches one space and then A in a manner determined by XY dF.

If X = 5, a minus sign is punched if A is negative, a space is punched if A is positive.

If X = J, a space is punched.

If Y = 0, the integer A x 2^{-39} is punched.

If Y = 2, the fraction A is punched correctly rounded off.

The number of digits punched and the location of extra space (decimal point) is determined by d = q x 10 + s.

$1 \leq s \leq 10$ $\qquad$ $1 \leq q \leq 11$

q is the number of digits punched. s is the number of digits punched before the extra space; e.g., 52 95F would cause the fraction in A to be punched to 9 figures with a space after the 5th figure. A minus sign would be punched if necessary.

J0 105F would cause the integer in A to be punched to 10 figures with a space after the 5th figure.

J0 33F would cause the integer in A to be punched to 3 figures followed by one space.

6.9. Library Routine R2

TITLE . . . . . Integral Root $A^{1/p}$

TYPE . . . . . Closed with one program parameter.

PURPOSE . . . . . To calculate the pth root of a fraction when p is an integer.

NUMBER OF WORDS 24

TEMPORARY STORAGE . . 0 - 3

ACCURACY . . . . . $\pm 2^{-39}$

PARAMETER . . . . . If the "Integral Root" subroutine starts at location t, then it is entered (with A in the accumulator) by the following:

s	50 pF
	50 sF
s + 1	26 tF

DURATION. . . . . Negligible for the special cases A = 0 and $|A| > p \times 2^{-39}$ A table of typical times (in milliseconds) follows:

p/A	.1	.2	.3	.5	.8
2	35	30	35	25	30
3	55	45	40	40	30
4	55	60	50	50	40
10	120	105	120	105	90

For large p and small A the times are considerably greater.

DESCRIPTION . . . . . This routine computes the pth root (p, a positive integer, $2 \leq p \leq 1023$) of a 39 binary digit real argument A, $-1 \leq A < 1$. If $|A| > 1 - p \times 2^{-39}$, then $\pm(1-2^{-39})$ is taken as $A^{1/p}$. Another special case is A = 0, in which case $A^{1/p} = 0$. Otherwise, $A^{1/p}$ is found by Newton's iteration method in which

$$x_0 = 1 - 2^{-39}$$

$$x_{n+1} = x_n + 1/p[(A/x_n^{p-1}) - x_n]$$

Convergence of x_n to $A^{1/p}$ is assumed when

$$1/p\,[(A/x_n^{p-1}) - x_n] \geq 0.$$

If p is even, of course, A must be non-negative and in this case the non-negative real pth root is found. At the end of the routine the accumulator contains the signed pth root of A.

6.10. MISTIC Library Categories

Programmed Arithmetic

A. Floating Point
B. Other programmed arithmetic

Code Checking

C. Post Mortem checks
D. Dynamic code checks

Integration

E. Quadrature
F. Ordinary differential equations
G. Partial differential equations

Operations on Functions

H. Zeros and minima
I. Interpolation
J. Operations on polynomials and power series
K. Approximations and statistics

Linear Algebra

L. Simultaneous linear equations
M. Other operations on matrices and vectors

Input and Output

N. Number input
O. Scope output
P. Printing and punching

Mathematical Logic

Q. Mathematical logic

Particular Functions

R. Roots and fractional powers
S. Logarithmic, exponential and hyperbolic functions
T. Trigonometric functions
V. Other special functions

Organizational

W. Counting, sorting and selecting
X. Program preparation

Miscellaneous

Z. Miscellaneous complete programs

6.11. MISTIC Library Index

Programs in the MISTIC library are derived from three sources. The origin of a particular program is indicated by the program label as follows: a) an M suffix, Michigan State University, b) an S suffix, SILLIAC library, University of Sydney, and c) no suffix, ILLIAC library, University of Illinois. Some programs have been obtained from University of Illinois sources other than the ILLIAC library itself and have been submitted to the MISTIC library, and these carry an M suffix also.

The number in parentheses following the title indicates the number of words in the program.

Complete descriptions of any of these programs are available upon request through the Computer Laboratory.

LABEL	TITLE
A1	Floating Decimal Arithmetic Routine (168)
A1M	Standard Entry Floating Point Arithmetic (86)
A2M	Floating Point Error Analyses (A1M) (600)
A3	Convert a Floating Decimal Number to Standard Form (27)
A3M	Interpretive Routine Entered as a Closed Subroutine (524)
A4	1.7 Precision Floating Binary Arithmetic (280)
A4M	Simply Programmed Computer for Amateurs (complete)
A5	Complex Number Arithmetic (248)
A5M	Floating Point Complex Arithmetic (160)
A6	Floating Decimal Routine and Auxiliaries (467)*
A7S	Semi-interpretive Floating Binary Arithmetic Routines (100)
B2	Complex Number Operations (54)
C1	Post Mortem Version of the DOI (37 + 25 of DOI)
C1M	Plugboard Wiring Check (complete)
C2M	Sexadecimal Post Mortem (complete)
C3	Print Signed Fractions and their Locations (31)
C3M	Sexadecimal Input (Miniature Bootstrap) (4)
C4	Print Sets of Decimal Integers and their Locations (38)
C4M	Sexadecimal Input Routine (6)
C5	Print Sets of Order Pairs and their Locations (32)
C5M	Print Out Decimal Order Input
C6	Address Search Routine (23)
C6M	Sexadecimal Card Post Mortem (37)
C9	Print Floating Decimal Numbers and Locations (36)
C12S	Post Mortem Version of Sum Checking DOI (X12S) (74)
CA1M	Print Set of Floating Point Numbers (A1M) (90)
D1	Check Point Code II*
D1M	Dynamic Trace (180)*
D3	Sequence Checking Routine (30)
D4	Control Transfer Check Routine (41)
D5	Iteration Counter (35)
DA1M	Check Point Print Out (A1M)*

*See Description

LABEL	TITLE
E1	Integration of f(x) for Equal Increments of x (45)
E2	Integration by Simpson's Rule (Tabulated Values) (21)
E3	Integration by Simpson's Rule (Function Values) (38)
E4	Numerical Differentiation with Interpolation (103)
E5	Quadrature by Gauss' Method*
E6	Integration Over a Single Interval (32)
E6S	Inverse La Place and Mellin Integration (64)
EA1M	Floating Point Integration (A1) (27)
EA2M	Integration in Floating Point (A1M) (67)
F1	Solution of a System of Ordinary Differential Equations (41)
F2	Solution of a System of Differential Equations (126)
F3	Integration Second Order Differential Equations (40)
F4	Numerical Solution of the Schroedinger Equation (complete)
F5	Integration of a System of Ordinary Differential Equations (60)
F6	Integration -- Differential Equations -- Control of Interval (129)
FA1	Second Order Linear Differential Equation (A1)(101)
FA2	Solution of a System of Ordinary Differential Equations (45)
G1	La Place's Equation -- Liebmann Method (42)
G2	Poisson's Equation, Liebmann-Frankel Method (59)
H1	Inverse Interpolation, a Real Root of f(x) = 0 (33)
H2	A Search for the Real Roots of f(x) = 0 (80)
H3	Minimization of a Function of Two Variables (46)
H4	Minimization of a Function of Four Variables (75)
H5	Minimization of a Function of N Variables (89)
H6	Minimization of a Function of N Variables (46)
H7S	Minimization of a Badly Conditioned Function (99)
HF1	Zero of a Solution of a Differential Equation (16)
I1	Interpolation (51)
J2	Roots of a Polynomial (complete)
J3M	Plot of P (jw)/Q(jw) (complete)
J4M	Partial Fraction Expansion (complete)
K1M	Frequency Distribution (complete)
K2	Correlations, Means, Standard Deviations, Covariances (complete)
K2M	Multiple Correlation (complete)
K3	Least Squares (complete)
K3M	D Statistic (complete)
K4	Intercorrelation of Scores Based on Paired Comparisons (complete)
K4M	Analysis of Variance for Completely Randomized Design (complete)
K5	Autocorrelations (complete)
K5M	Correlation, Means, Standard Deviation, Variance, Card Input
K6	Chi-squared (23)
K6M	Chi-square for k x ℓ Tables (complete)
K7	Iterative Estimation of Communalities (complete)
K7M	Correlations in Logarithmic Scale, Card Input (complete)
K8M	Sequential Multiple Range Test (complete)
K9	Product Moment Correlations, in Logarithmic Scale (complete)
K10	Calculate Autocorrelation of a Time Series (85)
K11	Quartimax Orthogonal Rotation of Factors (complete)

*See Description

LABEL	TITLE
L1	Solution of Simultaneous Linear Algebraic Equations (83)
L1M	Solution of a Set of Linear Equations (complete)
L3	The Complete Linear Equation Solver (complete)
L4	Half-precision Solution of a Set of n Simultaneous Equations (159)
L5	Half-precision Automatic Linear Equation Solver (complete)
L7S	Solution of Linear Simultaneous Equations (complete)
M0	Eigenvalues and Eigenvectors of a Symmetric Matrix (113)
M1M	Matrix Row and Column Deletion (complete)
M2	Automatic Inversion of a Symmetric Matrix (complete)
M2M	Mass Production Matrix Multiplication with Rescaling (complete)
M3	Automatic Eigenvalue-Eigenvector Program (complete)
M3M	Form Square Matrix from its Triangular Representation (complete)
M4	Closed Eigenvalues-Eigenvectors (163)
M4M	Multiplication of a Matrix and its Transpose (complete)
M5	Solution of Determinantal Equation $A - \lambda\beta = 0$ (complete)
M5M	Triangular Representation of a Square Matrix (complete)
M6	Matrix Multiplication (complete)
M6M	Add Columns to a Matrix (complete)
M7	Principal Axes Factor Analysis (complete)
M7M	Matrix Transposition (complete)
M8	Sums of Squares of Rows and Columns (complete)
M8M	Matrix Element Scaling (complete)
M9	Square Root Factorization (complete)
M9M	Form Matrix of Signed Numbers from Unsigned Numbers (complete)
M10	Estimation of Communalities by Maximum Likelihood
M10M	Square Root of the Elements of a Matrix (complete)
M11	Matrix Multiplication (77)
M11M	Linear Equation Solver; Error Analysis Routine (complete)
M12	Matrix Triangularization and Determinant Computation (complete)
M12M	Complex Determinant Calculator (94)
M13	Linear Matrix Equation Solver and Matrix Inversion (complete)
M13M	Rescale M13 Output (complete)
M14	Linear Matrix Equation Solver and General Matrix Inversion (115)
M14M	Real Determinant Calculators (74)
M15	Linear Programming (complete)
M15M	Simplex Codes for Linear Programming Modified for C_j (complete)
M16	Matrix Multiplication for Large Common Dimension (complete)
M16M	Simplex Code for Linear Programming Modified for b_i (complete)
M17	Post Multiplication of a Matrix by a Vector (45)
M17M	Characteristic Polynomial; N^4 Method (complete)
M18M	Characteristic Polynomial; N + 1 Points Method (complete)
M19M	Form a Diagonal Matrix of Square Roots (complete)
M20M	Iterating Eigenvalue - Factor Loading Program (complete)
M21M	Replace Diagonals of a Symmetric Matrix and Scale Output (complete)
MA1	Matrix Multiplication Floating Decimal Auxiliary (A1) (26)
MA1M	General Matrix Multiply in Floating Point (A1M) (35)
N1	Input One Number from Tape, Integer or Fraction (19)
N1M	Integer or Fraction Card Input, Variable Format (161)
N2	Input a Sequence of Decimal Fractions (26)
N3	Decimal Number Sequence Input Routine (21)
N3M	Card Input: Fixed Format, Variable Field Length (187)
N4	Input a Sequence of Integers (17)
N4M	Card Input: Fixed Format, Uniform Fields (173)

LABEL	TITLE
N5M	Integer Card Input, Fixed Format (X2M) (185)
N8	Read One Number from Tape, as Integer or Fraction (22)
N9	Matrix Modifying Input (69)
N10	Input a Sequence of Integers with Sum Check (36)
N11	Input a Sequence of Fractions or Integers (25)
N12	Infraput (39)
N13	Input Sequence of Fractions, Having Same Number of Digits (21)
N13S	Input a Sequence of Fractions or Integers (49)
N14	Input a Sequence of Integers (18)
NA1M	Input a Sequence of Numbers for (A1M) (107)
NA2M	Input One Floating Point Number (A1M) (58)
NA26S	Input a Sequence of Decimal Numbers (A7S) (39)
NP1M	Fraction Card to Tape Conversion in N2 Format (complete)
P1	Print One Number Fraction or Integer (28)
P2	Print Fraction with or without Sign to n Places (18)
P3	Print Integer with or without Sign (20)
P3M	Card Output: Fixed Format (155)
P4	Zero Suppression Integer Print (27)
P4M	Print Fraction or Integer (34)
P5	Print One Number in a Parameter Set Layout (24)
P5M	Print Sequence of Fractions or Integers (45)
P6	Single Column Print (14)
P7	Letter Printing (32)
P8	Tape Lettering (110)
P9	Decimal Order Print Routine (23)
P10	Print Headings Interlude (8)
P11	Rounded or Unrounded Print (18)
P12	Single Column Print
P13	Combined Integer Print (35)
P15	Multiple Precision Integer Conversion (32)
P16	Infraprint (56)
P17	Maximum Speed Fraction Print to Twelve Places (59)
P26S	Floating Decimal Print for Floating Binary Numbers (A7S) (57)
P27S	Single Column Prints from Successive Locations (18)
PA2M	Output a Sequence of Numbers for (A1M) (60)
Q1	Logical Algebra Subroutine (87)
Q2	Logical Input Variable Subroutine (33)
Q3	Complete Circuit Analyzer (complete)
Q4	Single Circuit Analyzer
R1	Square Root $(9)_{1/P}$
R2	Integral Root $A^{1/P}$ (24)
R3	Fractional Power Routine (18)
R4	Cube Root (15)
R5	Reciprocal (17)
R6S	Fast Square Root (19)
RA1	Floating Decimal Square Root Auxiliary (A1) (16)
RA1M	Square Root in Floating Point (31)
RA4S	1.7 Precision Floating Point Square Root (A7S) (17)
S1	Natural Logarithm (45)
S2	Exponential (33)
S3	Logarithm (14)
S4	Exponential (21)

LABEL	TITLE
S5	1/32 Natural Logarithm (36)
S10S	Fast Low Accuracy Logarithm Routine (20)
S11S	Fast Exponential Routine (28)
S12S	Integral of Exponential (65)
S13S	Fast Low Accuracy Exponential Routine (28)
SA2	Exponential Auxiliary for Floating Decimal (A1) (26)
SA2M	Hyperbolic Sine and Cosine Auxiliary (A3) (18)
SA3	Natural Logarithm (A1) (30)
SA3M	Natural Logarithm in Floating Point (A1M) (83)
SA5M	Exponential (A1M) (84)
SA6M	Sinh and Cosh in Floating Point (A1M) (17)
SA6S	Floating Binary Point Natural Logarithm (A7S) (31)
SA7S	Floating Binary Point Exponentials (A7S) (39)
T1	Sine, Cosine Subroutine (30)
T3	Arctangent in Degrees (15)
T4	Arctan X Subroutine (25)
T5	Sine-Cosine Routine (21)
T6	Fast Arcsine (10)
T6S	Fast Sine-Cosine Routine (30)
T7S	Inverse Tangent (36)
TA1	Sine Auxiliary for Floating Decimal (A1)(26)
TA1M	Sine in Floating Point (A1M) (50)
TA2	Arc Tangent Auxiliary (A1) (48)
TA2M	Arctangent Auxiliary (A1M) (85)
V1	Legendre Polynomials, $P_n(x)$ (25)
V2	Tchebyscheff Polynomials (15)
V3	Generate a Sequence of Random Numbers (13)
V4	Fourier Analysis (52)
V5	Spherical Bessel Functions (59)
V5S	Fourier Analysis (complete)
V6	Associated Legendre Functions (80)
V7	Provide Sets of Random Numbers from 1 to N (complete)
V8	Ordinary Bessel Functions (257)
V9	Generate 40-bit Random Numbers (32)
W1	Loop Cycling Control (21)
W1M	Decimal Scaling (32)
W2	Word Sorting According to an Ordering Relation (25)
X1	Decimal Order Input (25)
X1M	The Symbolic Address Input Converter (196)
X2M	650 Interpretive Routine (358)
X2S	Decimal Order Input with Sum Check (37)
X3	Constant-listing Auxiliary (21)
X3M	Clear Memory (6)
X4	Standard Subroutines (155)
X4M	Decimal Order Input for Core Memory (32)
X5M	Decimal Order Input Format for DOI Core
X6M	Data Tape Check for K3M or M9M (complete)
X7	Sum Check (11)
X7M	Card Bootstrap
X8	Assemble Blocks (4)
X9	Program Interruption Routine (without Sum Check) (34)
X10	Program Interruption Routine with Sum Check (49)

LABEL	TITLE
X11	Data Tape Checking for K2 and K9 (complete)
X11S	Periodic Sum Check (15)
X12S	Sum Checking DOI (41)
X13S	Tag Address Decimal Order Input (46)*
X15S	Decimal Data Input (19)
X16S	Sexadecimal Program Dump with Sum Check (24)
XA1	Constant Listing Auxiliary (A1) (18)
Y1M	Paper Tape Memory (30)*
Z2M	Frequency Response of an R-C Coupled Amplifier
Z3M	Ladder Network Analysis
Z4M	Root Locus Plotter
Z5M	Pulse Response of an R-C Coupled Amplifier (complete)
Z6M	General Network Analysis for Passive Networks (complete)

*See Description

CHAPTER 7

SPECIAL CODING TECHNIQUES

This chapter deals with some elements of programming which were not covered in earlier chapters as well as some special techniques which result from particular orders or combinations of orders in the MISTIC order code. The chapter is therefore made up of a number of relatively unrelated sections having to do with operations which frequently arise in programming.

7.1. Interludes

An interlude is a computation performed during the input of a program, the input being interrupted and then resumed.

A tape bearing a routine may contain an interlude which is placed in locations which may later be used to hold the routine or data. When the words of an interlude have been read, control is then directed to the interlude by means of an N directive, and the interlude is executed. Control is then transferred back to the input routine and the rest of the program input. The purpose of an interlude is usually to prepare some orders or constants required for the routine.

When the program is being read in by the DOI, input is resumed after the interlude by transferring control to the left side of location 999. The first word on tape after the interlude must be a K directive, or Q must contain, in the right-hand address position, the address at which the storage of the following routine is to start.

If, upon resuming input, it is desired to retain the last used K directive, control should be transferred to the right side of 1014 with $m \times 2^{-39}$ in A. The next words on tape will be placed at m, m + 1, ... retaining the previous relative address.

7.2. Some Special N Directives

The directive 20 1019N on a tape being input by means of the DOI causes the computer to stop and has no other effect. Upon being started, the tape will continue being read from where it stopped. This order can thus be used to stop the computer for inserting a new tape without a directive.

Other programmed stops are discussed in Section 8.5.

The N directive which stops input and starts the program is, in its simplest form, an order which transfers control directly to the desired order in the program. If the address to which control is to be sent is not known, as for example in a

subroutine which may be located arbitrarily, use can be made of the way in which the DOI operates to fix the address of the N directive. When a new order is read by the DOI and placed in the right-hand half of location 1, the left-hand half of 1 always contains the last previous order. If the last order pair before the N directive contains, as its right-hand order, a control transfer to the first order to be executed, written with relative address, the directive 26 1N will cause this control transfer to be obeyed. For example,

```
p)        ---                                        0)        ---
          OF F                                       :         :
p + 1)    22 p + 1      Waste        or              p + 1)    any
          26 L                                                 26 L
          26 1N                                                26 1N
```

will cause the order 26 L to be executed since the contents of 001 are 26L 26 1 at this point.

7.3. Bootstrap Input

The Decimal Order Input is used to input other programs, but nothing has yet been said about how this program itself is put in. The DOI is written in sexadecimal, occupies locations 999 to 1023, and is input by a bootstrap input routine.

The bootstrap is itself a program, on tape, which is used in conjunction with panel switches to put a three-word program into the machine. This three-word program in turn causes the DOI to be read and stored.

By panel switches, the control counter is cleared to zero and the sexadecimal equivalent of the order pair 80 40F 40 F is put in the order register; i.e., the "bootstrap start" described in Section 2.4 is executed. The bootstrap program is as shown in Table 7.3.1; decimal equivalents of addresses are in parentheses.

80 028	(40)
40 001	
80 028	(40)
40 002	
19 026	(38)
26 000	
80 028	(40)
40 [000]	
L4 001	
40 001	
80 028	(40)
40 [3F6]	(998)

Table 7.3.1

This program causes the three-word program shown in Table 7.3.2, with decimal addresses in parentheses, to be stored and control sent to location 1.

This three-word program, in turn, causes the DOI tape to be read and stored in location 999 and consecutive locations. The order pair

```
22  3LS     (1019F)
00  001     (1F)
```

is the last order on the DOI tape and is stored at location 0. This order sends control to the right side of 1019 and the DOI is started. If 20 3LS 00 001 is used in place of 22 3LS 00 001, the computer will stop on the 20 3LS. The black switch then causes the DOI to be started. With this variation, the DOI need not be on a program tape but may be read in separately.

```
0)  L4  001
    40  001
1)  80  028     (40)
    40  [3F7]   (999)
2)  19  026     (38)
    26  000
```

Table 7.3.2

The term bootstrap start is often used for tapes which are started by setting the order register to 80 028 40 000 and the control counter to zero.

7.4. Use of Input and Output Orders

If input orders are to be stored in memory by means of the DOI, the input orders are written and entered as part of the program. The characters to be read in response to 80 n or 81 n must be on the tape after the N directive which starts the program.

Characters read in by the 80 n or 81 n orders are interpreted as sexadecimal characters. Hence, if a number is being input by means of an 80 n or 81 n, that number must be in sexadecimal. Likewise, if an order pair is being input by means of these orders, the address as well as the op code must be in sexadecimal.

As noted in the tabulation of the 80 orders, these orders by-pass all characters with a fifth hole. Therefore, if a fifth hole character is to be read, the 91 4 order must be used.

The output orders in the 80 series (82 n, 83 n, 8S n) will output numbers from the computer as sexadecimal characters. These orders are convenient to output a small number of intermediate results, such as the results of a sum check, for example. The 92 n order, on the other hand, is used to output characters which are predetermined by the programmer.

It is particularly important to note that all 80 and 90 orders involve shifts of both A and Q. Hence, the contents of both A and Q are altered when these orders are

used and if either (A) or (Q) is needed later, it must be stored before the 80 or 90 orders are used.

Use of the 92 n orders is illustrated in example 6.5.2. Use of the 80 n orders is illustrated in the bootstrap discussion of Section 7.3. A second use of the 80 n order is as follows: if it is desired to make a 1, 2 or 3 place left shift of A without shifting Q into A, one can use the order 80 1F, 80 2F or 80 3F respectively. No input occurs and the input tape does not move.

One use of the 91 4 order is where a fifth hole character is used as a marker. Thus,

```
91 4F
36 kF
```

will read one character and test for fifth hole. If the character is a fifth hole character, a_0 will contain a 1; if the character is a sexadecimal digit, a_0 will contain 0. The 36 kF will thus transfer control to k if the character is a sexadecimal digit and take the next order if the character is a fifth hole character.

7.5. Summation of Products

It is often necessary to form sums of products, e.g., in matrix multiplications. Accuracy can be increased by performing such a summation either exactly or with one round-off. This can be done on MISTIC by the use of a 74 n order. To accomplish this summation by a 74 n order, it is necessary only to place the least significant half of the partially summed products into the accumulator before performing the 74 n order. This can usually be done by an S5 order because Q usually holds the least significant part of a summed product. Since 74 n gives $(n)\cdot(Q) + 2^{-39}$ A, a double length product is formed in AQ. To this, the most significant part of the previously summed product can be added by an L4 n order. If A is set to 1/2 before the first execution of the 74 n order, one rounding is effected.

The following example illustrates this procedure.

Example 7.5.1. Place the rounded sum

$$\sum_{i=1}^{30} a_i b_i$$

in location 99, where the a_i are in locations 100...129, the b_i in locations 130...159.

Solution

0)	41 99F	Clear 99 for sum
	26 1L	Waste
1)	50 8L	Put 1/2 in Q
	L5 11L	
2)	40 3L	Preset 3L
	S5 F	(Q) $\longrightarrow$ A
3)	50 [100]F	$a_i b_i$
	74 [130]F	
4)	L4 99F	$\sum_{j=1}^{i-1} a_j b_j + a_i b_i \longrightarrow 99$
	40 99F	
5)	L5 9L	Increase addresses of a_i and b_i by 1
	L4 3L	
6)	40 3L	
	L0 10L	
7)	32 2L	Test for end
	0F F	
8)	40 F	Round off constant = 1/2
	00 F	
9)	00 1F	Increment
	00 1F	
10)	J0 130F	Test constant
	74 160F	
11)	50 100F	Starting constant
	74 130F	

Table 7.5.1

7.6. Reversing the Control Transfer

Example 7.5.1 illustrates another coding trick. If the test constant in order 10 were

50 130F 74 160F

the result of order 6 would be negative until the end. This would necessitate a "double control transfer" order; i.e., a 32 nL 22 2L, since in this case the loop is re-entered on a negative. By adding -1 to the test constant, i.e., using

J0 130F 74 160F

the sign of the result of order 6 is reversed and the 32 2L order can be used.

7.7. Binary Switches

A program may involve two orders which are to be used alternately at a particular point in the computation. To accomplish this alternation, a number in memory which is alternately plus and minus, together with a conditional control transfer, can be used. The sign alternation is accomplished by the order pair: L1 m 40 m, where m is the address of the number; and the order alternation by the order pair: p) L5 m 36 k, where k is the address of one of the alternate orders, and p + 1 is the address of the second, and m is the address of the alternating number.

If an order or order pair is to take on two values alternately, this can be accomplished by storing the sum of the two orders at some location, m, and using the orders L5 m, L0 k, 40 k where k is the address of the order to be alternated. This technique can be used to alternate either addresses or op codes, or both together.

For alternating addresses only, the alternation can also be effected by the sequence of orders:

```
m)      50 (a+b)F
        L5 mF
m+1)    L0 kF
        46 kF
```

where k is the location of the order with alternating left-hand address and a + b is the sum of the two addresses. If the alternating address is a right-hand address, the two orders at m are reversed, and the right-hand order at m + 1 is changed to 42 kF. If the op code of the order at k which contains the alternating address is larger than L5 or if the op code is equal to L5 and the address in k is larger than m, a + b + 1 must be used in m in place of a + b to allow for overflow.

7.8. Special Tests

Because the only test orders are the 30 orders which take one of two alternatives for $(A) \geq 0$, and the other alternative if $(A) < 0$, two tests are usually required to test for a particular number value in memory. The numbers 0 and -1 can be tested for by using absolute values and a single test because of the special form of these two numbers. In machine language, 0 is the only number whose negative absolute value is positive and -1 is the only number whose positive absolute value is negative. Thus,

$|0|$ 0 = 0.0...0

$-|0|$ = comp. of 0.0...0=1.111...1111 + 0.00...001 = 0.0000...0000

which is positive, and

-1 = 1.000...000

$+|-1|$ = comp. 1.00...000=0.111...1 + 0.00...001 = 1.000...0000

which in machine language is negative.

The test for 0 is, therefore, accomplished by an L3 n order followed by a conditional control transfer. Likewise, the test for -1 is an L7 order followed by a conditional control transfer.

In a similar manner, a test for -2^{-39} and $1 - 2^{-39}$ can be made by using F3 and F7 respectively, followed by a conditional control transfer; i.e., F3 n gives $a_0 = 0$ for $(n) = -2^{-39}$, $a_0 = 1$ for any other value of (n) and F7 n gives $a_0 = 1$ for $(n) = 1 - 2^{-39}$, $a_0 = 0$ for (n) any other value.

When scaling numbers, it is very often necessary to test when numbers are larger in magnitude than one-half. This can be done by appropriate L or S orders. Thus, LL n followed by a conditional control transfer order forms $1/2 + |(n)|$ and tests for overflow.

7.9. Marking

It is often possible to use a marking technique in place of other counting processes. A sequence of decimal numbers can be terminated by one of the sexadecimal characters. This is common in decimal number input routines and in programs using sequences of decimal numbers, such as matrix programs and statistical correlation programs.

A fifth hole character in a certain position is sometimes used as a marker. For example, in the complex number routine A5, a fifth hole character is used to indicate the end of the exponent of a complex number.

A binary digit is sometimes shifted as a marker. The digit is so located that its shift into the sign bit indicates the end of a repetitive loop.

7.10. Masks

It is sometimes desirable to pick out certain of the digits in a particular word without including the rest of the digits. This can be accomplished by proper use of the J0 n order. The order J0 n puts in Q a 1 in each digit position in which (n) and (Q) both contain 1, 0 elsewhere. Thus, if certain digits of (n) are to be isolated, a word containing 1's in the positions of the desired digits, 0 elsewhere, is stored in memory. The orders

50 m	m is address of word some of whose digits are to be isolated
J0 n	n is address of stored word

will cause the desired digits of m to be generated in Q, with zeros in all other digit positions.

The word in n is called a mask.

CHAPTER 8

CODE CHECKING

After a program is written, it is necessary to find and remove all coding errors. While many of the errors can be found by a careful visual check, it is usually necessary to use the machine as a final check. There are several library routines designed for code checking.

8.1. Common Blunders

Many coding mistakes are familiar ones which are made over and over. Table 8.1.1 gives a list of some of these common blunders, to which each programmer will probably have other items to add from his personal experience.

8.2. Correcting Errors by Hand Punch or Reperforator

After a program has been punched on tape, it should be printed and carefully proofread. Some errors thus located can be corrected by means of the hand punch. Thus, an F can be changed to an L by punching the one hole, or extraneous characters can be turned into fifth hole characters by means of the hand punch though considerable care must be taken to insure a cleanly cut hole.

Errors located either before the program goes on the computer or as a result of a code check on the computer may be removed by reperforating. Correcting by reperforating involves first locating and marking errors on the original tape and punching the correct characters on separate tapes. The original tape is duplicated until a marked character is reached. The tape is then removed from the reperforator and the correction tape inserted and duplicated, after which the original tape is reinserted and duplication resumed.

The method of perforation is particularly well adapted to data tapes. In a program, only corrections which do not change the order numbering can be made by this method.

8.3. Tape Correction by Modification Tapes

Another method of making corrections is by means of program modification tapes. By this method, control is sent to the DOI at the end of read-in of the original program and a modification tape is read. This modification tape may simply replace incorrect orders by the correct ones. For example, if at location 125 the order pair L5 175F L4 4L should have been L5 175F L4 4F, a modification tape could contain

Typical Blunders

1. L5 orders used instead of L4 orders.
2. S5 orders omitted after divisions.
3. Orders terminated by L instead of F and vice-versa.
4. The renumbering of a code not completed after a modification has been made.
5. Rounded multiplication used when dealing with integers.
6. Control transfers to the wrong address or wrong order of an order pair.
7. Accumulating storage registers not cleared before a cycle of orders is entered.
8. The end condition for a cycle of orders not correct.
9. Allowing temporary storage of a subroutine to erase useful data.
10. Using a 46 order instead of a 42 order and vice-versa.
11. Omitting directives and starting orders on the program tape.
12. Incorrectly remembering the specifications of a subroutine.
13. Forgetting to reset addresses when coming back to a cycle of orders.
14. Making corrections incorrectly.
15. Using the same relative address on correction words although the preceding directive is different from that of the program.
16. Overlooking the digits shifted from the quotient register to the accumulator on a left shift.
17. Attempting to convert fractions greater than one-half by using the J termination symbol.
18. Failure to make an L symbol relative to the correct address.
19. Failure to clear Q before a single length division.
20. Failure to store address or counter after modification.
21. Use of letter O instead of the number zero.

Table 8.1.1

```
00  125K
L5  175F
L4  4F
24  100N
```

where 100 is the location of the first order to be obeyed in the program.

If the correction involves addition of orders, the modification tape must change one order in the original program to a control transfer to an address outside the program and place at the address the appropriate orders. For example, suppose a program which starts at location 100 has the following sequence of orders:

```
125)  L5  200F
      L4  13L
126)  7J  12F
      S5  F
```

where the orders 40 F 50 F should have been between these two order pairs. The modification tape in this case should be

```
      00  125K

 0)   L5  200F
      26  mF
      00  mK
 1)   L4  113F
      40  F
 2)   50  F
      26  126F

      24  100N
```

Note particularly that the 13L of order 125 is 113F in the modification tape, since an L on this tape refers to a different relative address base than that of the original tape.

After all corrections have been made and the program is working, one correct tape should be prepared. This final tape should have all unnecessary stops removed.

8.4. Returning Control to the DOI for Tape Modification

The modification tapes described in Section 8.3 are effective only if control has been transferred to the DOI at the end of the program read-in. A program which is at all complex, and hence likely to require modification, should be written with the directives

```
24 999N  26 kN
```

where k is the address of the first order to be executed in the program. When the computer stops on the 24 999N, the modification tape is inserted and the modification read. If no modification is necessary, a black switch start after the stop will cause the 26 kN to be executed. If this precaution has not been taken, it

is sometimes possible to change the address of the N directive to 999 by hand punch. Otherwise, a reperforation of the original tape can be made, changing the address of the N directive to 999 and including the modification tape.

Still another method of transferring control to the DOI for a modification is by a bootstrap start. The sexadecimal order pair 26 3F7 00 000 is punched at the head of the modification tape followed by a K directive. This tape is put into the reader when the computer stops on the N directive which must be a stop-control transfer order, the order register is set to 80 40F 40 F, and the control counter to zero, i.e., normal bootstrap start, without clearing the machine. The order 26 3F7 is then obeyed and the modification tape read in. This method may be useful during code checking but is not to be left in the final program whereas other methods may be part of a final complete tape.

8.5. Location of Errors--Programmed Stops

If a program runs but the answers are incorrect, an analysis of the results may yield some clue as to the fault. If such is not the case, the program will need to be examined by means of some of the various checking routines.

If the program hangs up, the cause may sometimes be located by an inspection of the order register and control counter. The order register contains the order on which the program hung up and the control counter contains a number one greater than the storage location from which the order came, thus locating the hang-up. Examination of the program in the light of this information may reveal the difficulty.

Stop-control transfer orders written into a program are a valuable aid in locating errors. The stop initiated by the N directives at the end of the program read-in, described in Section 7.2, serves to indicate that the program has been input and that any subsequent hang-up is not a program input hang-up.

Simple stop-transfer control orders at strategic locations in the program, for example at the end of a loop, aid in program checking since they serve to mark progress through the program execution and thus help pinpoint a program failure.

In general, after a program has been checked and is running, these stops are turned into non-stop-transfer control orders before production runs.

A special program stop order, the FF n order, is sometimes used to indicate the failure of a programmed check. For example, a sum check may be inserted at the end of a program read-in with an FF n order which causes the MISTIC to stop if the sum is incorrect. The FF n order may be used in connection with any programmed check the program writer may care to include in his program.

When the relatively simple methods of program checking fail to reveal the difficulty, it is necessary to resort to checking routines.

8.6. Post Mortem Routines

A post mortem routine is a routine which prints out the contents of certain specified memory locations after a program has stopped. This is the simplest type of checking routine to use, especially from the standpoint of the programmer, since it requires no preparation on the part of the programmer except a record of storage locations used by the program.

The post mortem routines C3, C4 and C5, in the MISTIC Library, are read into memory with a bootstrap input routine and are located in storage locations at the end of memory, overwriting the DOI. The longest of these occupies locations 986 to 1023, and all use locations 0, 1 and 2 as temporary storage.

The end of each of these post mortem tapes contains 100 two-decimal-digit numbers used to specify the location from which printing will occur. Routines C3 and C4 cause decimal fractions and integers, respectively, to be printed out of specified locations. Routine C5 causes order pairs to be printed.

Examination of the print out from these post mortem routines yields information such as whether numbers were properly stored and what addresses have changed.

Library Routine C1, which is the post mortem version of the DOI, is a very useful checking routine. It uses locations 962-1023 and locations 0, 1 and 2.

The C1 routine is input; then the original tape is inserted in the reader. The routine C1 compares the contents of memory with the contents of the input tape, printing out discrepancies in the following format: location of discrepancy, word from tape as an order pair, word in memory as an order pair.

Examination of the results of C1 gives such information as whether addresses and constants are changing as expected, whether links in closed subroutines are properly set, number of times certain loops have been executed, etc.

It should be noted that routine C1 restores the original program in MISTIC as it is executed so the program may be rerun immediately following the use of C1 without rereading it.

8.7. The Address Search Routine--Library Routine C6

A second useful type of checking routine is the address search routine, routine C6 in the MISTIC Library. This routine is read into MISTIC, overwriting the DOI. The address sought is then read into the machine as a three-character sexadecimal address. The routine then searches the memory for order pairs containing this address. When found, these order pairs are printed out, together

with their locations, all in sexadecimal.

Programs often fail because control has been transferred to a location which causes the machine to stop. By use of the routine C6, together with the address supplied by the control counter, the source of the offending control transfer order can be located.

Another use of routine C6 is to locate the order which causes a number to be incorrectly modified.

8.8. Sequence Checking Codes--Library Routines D2 and D3

Sequence checking routines are routines which control a program, order by order, and print out information about the execution of each order. This enables the action of a program to be traced, order by order. These routines use a blocking order technique which enables selected parts of a program to be checked. Because of the printing involved, these programs are very slow and should not be used indiscriminately.

Routine D2 prints in full each order that is obeyed, having one order pair per line of printing. After each store or address order, the number transferred to memory is also printed.

Routine D3 prints the op code of the orders which are actually obeyed, starting a new line of printing whenever a control transfer has been obeyed.

8.9. Other Check Routines--Library Routines D1 and D4

Library Routine D1 is a check point routine. This routine is designed to print out intermediate information about a program in store. It uses a blocking order principle, and the programmer prepares a specification tape to describe the kinds of information desired. It is possible to go through iterative loops and print out results on passage through the loops. Data can be obtained as an order pair, a right-hand address, a left-hand address, a 10-character sexadecimal word, a signed 12 decimal place fraction, or a signed three-decimal place fraction.

Routine D1 is very powerful because of the great latitude given the programmer in choosing the type and form of information to be obtained.

Library Routine D4 is a control transfer check. This routine causes a program to be obeyed order by order. Each transfer of control that is obeyed is placed in a list kept in a specified place. This list is cyclic in that later entries overwrite the earlier ones in a cyclic fashion. The final list printed out, therefore, shows how the program reached its final end. This routine requires no printing during execution and hence is much faster than the routines D2 and D3.

8.10. Resume of Code Checking Routines

The following tabulation gives the code checking routines in the MISTIC Library, their general form, and a brief resume of what each routine accomplishes.

C1--Post Mortem--Prints out discrepancies between original program and memory contents.

C3--Post Mortem--Prints decimal fractions at specified locations in memory.

C4--Post Mortem--Prints decimal integers at specified locations in memory.

C5--Post Mortem--Prints order pairs at specified memory locations.

C6--Address Search--Locates all orders which contain a specified address.

D1--Check Point---Prints out intermediate information as specified by programmer.

D2--Sequence Checking--Prints each order that is obeyed.

D3--Sequence Checking--Prints op code of each order that is obeyed.

D4--Control Transfer--Prints a list of control transfer orders which are obeyed.

Table 8.10.1

CHAPTER 9

FIXED POINT PROGRAMMING AND SCALING

Whenever an arithmetic operation is performed on two numbers, special attention must be given to the location of the decimal or binary points of the two original numbers. In addition and subtraction, the decimal points of the two numbers must be aligned with the decimal point of the sum or difference. For multiplication and division, alignment is of course no problem, but location of the decimal point in the result is a problem. In multiplication, the number of places following the decimal point is the sum of the number of places following the decimal point in the two factors. For division, the decimal point of the quotient must be first aligned with the decimal point of the dividend and then moved right as many places as there were behind the decimal point in the divisor.

The above needs are complicated in a computer since there is no decimal or binary point represented there, and even the sign bits are acceptable as numbers. The problem of decimal point location is then entirely the programmer's, who must anticipate every operation. In some cases it is not entirely possible to keep accurate knowledge of the decimal point location (such as in linear programming), so that the programmer must use a special form of programming called floating point programming. Such a program causes the computer itself to maintain the decimal point. More will be said about floating point operation. If, however, the programmer has anticipated every decimal point change by programming, then the program is called fixed point (even though the point may not remain fixed).

9.1. General Principles of Scaling

There are five problems connected with fixed point programming which arise because of the fractional representations used in the MISTIC. Since every number, N, in the MISTIC is in the range $-1 \leq N < 1$, then:

1) Every number, N, input to the computer must be in the range $-1 \leq N < 1$;
2) The result of every addition or subtraction must be in that range;
3) Quotients must be in the above range; and
4) The result of a left shift must be in the above range.

The fifth problem is not due to the number representation and in fact is a general problem for all computers.

5) The programmer must know the location of the binary point or decimal point at all times.

Problems 1, 2, 3 and 5 are generally disposed of by use of a process called scaling. Suppose a number, N, is in the range $2^{m-1} \leq N < 2^m$, or in the range $10^{n-1} \leq N < 10^n$ for positive integers m and n. In order to use this number in the computer, N could be represented by $\overline{N}$ where

$$\overline{N} = N \cdot 2^{-m}$$

or

$$\overline{N} = N \cdot 10^{-n}$$

respectively. Notice then that $1/2 \leq \overline{N} < 1$ or $1/10 \leq \overline{N} < 1$. Of course there are other representations for N which will fit in the computer, for if $\overline{N} = N \cdot 2^{-m}$ will fit into the computer, so also will $\overline{N} = N \cdot 2^{-r}$ where r is greater than m. The following definition can be used:

The machine representation of N with a binary scaling of m is a number $\overline{N} = N \cdot 2^{-m}$ where $-1 \leq \overline{N} < 1$.

The machine representation of N with a decimal scaling of m is a number $\overline{N} = N \cdot 10^{-m}$ where $-1 \leq \overline{N} < 1$.

Example 9.1.1. Give a machine representation of 407.98 with a binary scaling. Since $2^8 \leq 407.98 < 2^9$, some possible machine representations of 407.98

$$\overline{407.98} = 407.98 \times 2^{-9} = .7968359$$

or

$$\overline{407.98} = 407.98 \times 2^{-10} = .39841796, \text{ etc.}$$

Binary scaling of a large set of numbers becomes tedious since each representation must be obtained by dividing by a power of two. As a result, binary scaling is seldom desirable. It does have the advantage of greater accuracy, as will be seen.

Example 9.1.2. Give a machine representation of 407.98 with a decimal scaling. Since $10^2 \leq 407.98 < 10^3$, some possible machine representations are:

$$\overline{407.98} = 407.98 \times 10^{-3} = .40798$$

or

$$\overline{407.98} = 407.98 \times 10^{-4} = .040798, \text{ etc.}$$

Decimal scaling amounts to shifting the decimal point and does not involve any actual division.

It is generally desirable to choose the exponent or scale as small as possible. This is particularly true if one is seeking extreme accuracy in the computed result. Each increase in scale introduces more insignificant zeros at the most significant part of the number, at the same time causing digits to be lost off the least significant end.

9.2. Arithmetic with Scaled Numbers

If machine representations for two numbers, say a and b, have been selected as $\overline{a}$ and $\overline{b}$ where

$$\overline{a} = a \cdot 10^{-m}$$
$$\overline{b} = b \cdot 10^{-n}$$

then $\overline{a}$ and $\overline{b}$ can be multiplied to give

$$\overline{ab} = ab \cdot 10^{-(m+n)}$$

As a result of multiplication, the product has a scaling equal to the sum of the original scales. Similarly for division,

$$\overline{a}/\overline{b} = a/b \cdot 10^{-(m-n)}$$

Care must be exercised in this case, however, that $-1 \leq \overline{a}/\overline{b} < 1$.

In the case of an addition, $\overline{a} + \overline{b}$ has no useful connection with a + b since the exponents m and n may be unequal. Assuming this to be the case, and that m is greater than n, it will be found expedient to choose a new representation for a or b. Either the exponent n can be increased, or the exponent m can be decreased. If m is decreased, the new $\overline{a}$ might become greater than 1. It is usually safer to increase n to m, thus giving

$$\overline{a} = a \cdot 10^{-m}$$
$$\overline{b} = b \cdot 10^{-m}$$

and $\overline{a} + \overline{b} = (a+b) \cdot 10^{-m}$. The scaling m is satisfactory to this point provided $-1 \leq \overline{a}+\overline{b} < 1$; that is, provided $-10^m \leq a+b < 10^m$.

Example 9.2.1. Choose machine representations for the numbers a, b, c, d, e and f, which will be used to compute

$$(ab + c)d + ef + b$$

where

$$10 \leq a < 10^2$$
$$10^2 \leq b < 10^3$$
$$10 \leq c < 10^2$$
$$10 \leq d < 10^2$$
$$10^2 \leq e < 10^3$$
$$10^2 \leq f < 10^3$$

A first approximation to machine representations for these numbers is provided by

$$\overline{a} = a \cdot 10^{-2}$$
$$\overline{b} = b \cdot 10^{-3}$$
$$\overline{c} = c \cdot 10^{-2}$$
$$\overline{d} = d \cdot 10^{-2}$$
$$\overline{e} = e \cdot 10^{-3}$$
$$\overline{f} = f \cdot 10^{-3}$$

These representations will not be adequate, for when the machine representation of ab + c is formed, the decimal points of the summands are not aligned. That is,

$$\overline{ab} + \overline{c} = ab \times 10^{-5} + c \times 10^{-2}$$

This difficulty can be avoided by choosing a new machine representation for c, say $\overline{c} = c \times 10^{-5}$. Then

$$\overline{ab} + \overline{c} = (ab + c) \times 10^{-5}$$

However, $ab + c < 10^2 \cdot 10^3 + 10^2 = 10^5 + 10^2$, so that $\overline{ab} + \overline{c} < 1.01$, which might "overflow". By this is meant that ab + c may not remain in the range $-1 \leq \overline{ab} + \overline{c} < 1$. To avoid the overflow, it is necessary to change the scaling on a or b, and also on c. For example, the scaling can be changed to $\overline{b} = b \cdot 10^{-4}$ and $\overline{c} = c \cdot 10^{-6}$.

Then, the representation of (ab + c)d will be

$$(\overline{ab} + \overline{c})\overline{d} = (ab + c)d \times 10^{-8}$$

while that of ef is

$$\overline{ef} = ef \times 10^{-6}$$

In order to add the machine representation of (ab + c)d to that of ef, the representation of ef needs to be changed to $\overline{ef} = ef \times 10^{-8}$. This can be accomplished by changing the machine representation of e to $\overline{e} = e \times 10^{-5}$. Then,

$$(\overline{ab} + \overline{c})\overline{d} + \overline{ef} = [(ab + c)d + ef] \times 10^{-8}$$

This machine representation is not in danger of overflowing since

$$(ab + c)d + ef < (10^5 + 10^2)10^2 + 10^6 = 10^7 + 10^4 + 10^6$$

and

$$(\overline{ab} + \overline{c})\overline{d} + \overline{ef} < (10^7 + 10^4 + 10^6)10^{-8} = .1101$$

In order to add $\overline{b}$ to this machine number there are two options: either represent b by $b \times 10^{-4}$ and also by $b \times 10^{-8}$, or divide $\overline{b} = b \times 10^{-4}$ by 10^4 in the computer. In either case the result is

$$(\overline{ab} + \overline{c})\overline{d} + \overline{ef} + \overline{b} \times 10^{-4} = [(ab + c)d + ef + b] \times 10^{-8}$$

The final machine representations for a, b, c, d, e and f are

$$\overline{a} = a \times 10^{-2}$$
$$\overline{b} = b \times 10^{-4}$$
$$\overline{c} = c \times 10^{-6}$$
$$\overline{d} = d \times 10^{-2}$$
$$\overline{e} = e \times 10^{-5}$$
$$\overline{f} = f \times 10^{-3}$$

This example is a case in which the problem, once scaled, can be programmed with no further regard for scaling. This is not always the case, however. The following example illustrates a more complex situation.

Example 9.2.2. Find the positive real root accurate to ten decimal places of $f(x) = x^3 - x^2 - 1 = 0$, using the Newton Raphson iteration

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Since f(2) = 3 and f(1) = -1, there is a root between x = 1 and x = 2. For a

first approximation x_0 to that root select $x_0 = 1.2$. The problem then to be programmed is

$$x_{n+1} = x_n - \frac{x^3_n - x^2_n - 1}{3x^2_n - 2x_n}$$

$$= \frac{2x^3_n - x^2_n + 1}{3x^2_n - 2x_n}$$

$$= \frac{[(2x_n - 1)x_n]x_n + 1}{(3x_n - 2)x_n}$$

The root r being sought is in the range $1 < r < 2$, while the first approximation is $x_0 = 1.2$. This should indicate that $1 < x_n < 2$ at each iteration. Evidently, the machine representation of x_0 can be

$$\bar{x}_0 = x_0 \cdot 2^{-1} = .6$$

The numerator has an upper bound of 13, while the denominator has an upper bound of 8. This means that the final scaling for the numerator must be at least 2^{-4}, while for the denominator it must be 2^{-3}. If the routine is entered with $\bar{x}_n = x_n \cdot 2^{-1}$, notice that the result of one application of the Newton Raphson method with this scaling is $\bar{x}_{n+1} = x_{n+1} \cdot 2^{-1}$.

In forming $2x_n - 1$, if $\bar{x}_n = x_n \cdot 2^{-1}$, then $2\bar{x}_n = 2x_n \cdot 2^{-1} = x_n$, and the result can overflow. Instead of changing the scaling for x_n, notice that $\bar{x}_n = x_n \cdot 2^{-1} = 2x_n \cdot 2^{-2}$, so that if $\bar{1} = 1 \cdot 2^{-2}$, then

$$\bar{x}_n - \bar{1} = (2x_n - 1) \cdot 2^{-2}$$

Next

$$(\bar{x}_n - \bar{1})\bar{x}_n = (2x_n - 1) \cdot x_n \cdot 2^{-3}$$

and

$$[(\bar{x}_n - \bar{1})\bar{x}_n]\bar{x}_n = [(2x_n - 1)x_n]x_n \cdot 2^{-4}$$

Now if 1 is represented also by $\bar{\bar{1}} = 1 \cdot 2^{-4}$, the numerator becomes

$$[(\bar{x}_n - \bar{1})\bar{x}_n]\bar{x}_n + \bar{\bar{1}} = ([2x_n - 1)x_n]x_n + 1) \cdot 2^{-4} \qquad (9.2.1)$$

Making use of the two relationships

$$\bar{x}_n = x_n \cdot 2^{-1} = 2x_n \cdot 2^{-2}$$

$$\frac{1}{2}\bar{x}_n = x_n \cdot 2^{-2}$$

the denominator can be treated as follows.

Let $\bar{2} = 2 \cdot 2^{-2}$

Then $\frac{1}{2}\bar{x}_n - \bar{2} = (x_n - 2) \cdot 2^{-2}$

and $\bar{x}_n + (\frac{1}{2}\bar{x}_n - \bar{2}) = 2x_n \cdot 2^{-2} + (x_n - 2) \cdot 2^{-2} = (3x_n - 2) \cdot 2^{-2}$

(Notice that $\bar{x}_n + \frac{1}{2}\bar{x}_n = 3x_n \cdot 2^{-2}$, which can overflow. As a result, not even addition is necessarily commutative or associative in a computer.)

Finally, the denominator is given by

$$[(\tfrac{1}{2}\bar{x}_n - \bar{2}) + \bar{x}_n]\bar{x}_n = (3x_n - 2)x_n \cdot 2^{-3} \qquad (9.2.2)$$

The program for the above problem could be as displayed in Table 9.2.1. The numerator and denominator are formed using the left side of equations 9.2.1 and 9.2.2, respectively.

The result of this program will be an approximation to the real positive root of the equation $x^3 - x^2 - 1 = 0$, and this approximation is in the accumulator scaled by a factor of 2^{-1}.

In order pairs 4 and 7 in Table 9.2.1, the multiplication used was a 7J, while in order pair 8, the multiplication used was a 75. This suggests that a set of (not necessarily inflexible) rules can be stated for the use of the 7 orders, particularly the 74, 75 and 7J orders, as follows.

If a single length rounded product is desired, use the 7J order provided no shifting is needed to obtain the single length result. If, however, a shift is required following the multiplication, use the 74 order with a 1 in location a_{m+1} if an m place left shift is to follow the 74.

If a double length product is desired, particularly preceding a division, use the 75 order.

Table 9.2.1

0)	41	14L	$0 \longrightarrow 14L$
	50	14L	$0 \longrightarrow Q$
1)	L5	15L	$\bar{x}_n \longrightarrow A$
	10	1F	$\frac{1}{2}\bar{x}_n \longrightarrow A$
2)	L0	16L	$\frac{1}{2}\bar{x}_n - \bar{2} \longrightarrow A$
	L4	15L	$\frac{1}{2}\bar{x}_n - \bar{2} + \bar{x}_n \longrightarrow A$
3)	40	17L	$\longrightarrow 17L$
	50	15L	
4)	7J	17L	Rounded multiply: $(3x_n - 2)x_n \cdot 2^{-3}$
	40	17L	Denominator $\longrightarrow 17L$
5)	L5	15L	$\bar{x}_n \longrightarrow A$
	L0	18L	$\bar{x}_n - 1 \longrightarrow A$
6)	40	19L	$\longrightarrow 19L$
	50	15L	$\bar{x}_n \longrightarrow A$
7)	7J	19L	$(\bar{x}_n - \bar{1})\bar{x}_n \longrightarrow A$
	40	19L	
8)	50	15L	
	75	19L	$(\bar{x}_n - \bar{1})\bar{x}_n^{\,2} \longrightarrow AQ$
9)	L4	20L	$(\bar{x}_n - \bar{1})\bar{x}_n^{\,2} + \bar{\bar{1}} \longrightarrow AQ$
	66	17L	$\bar{x}_{n+1} \longrightarrow Q$
10)	S5	F	$\bar{x}_{n+1} \longrightarrow A$
	L0	15L	$\bar{x}_{n+1} - \bar{x}_n \longrightarrow A$
11)	40	19L	$\longrightarrow 19L$
	L7	19L	$\lvert\bar{x}_{n+1} - \bar{x}_n\rvert \longrightarrow A$
12)	L0	21L	$\lvert\bar{x}_{n+1} - \bar{x}_n\rvert - \frac{1}{2} \cdot 10^{-10} \longrightarrow A$
	36	22L	
13)	S5	F	Root in A
	0F	F	
14)	00F	00F	Zero
15)	[40	F	
	00	10000000000J	$\bar{x}_0 = 1.2 \times 2^{-1}$
16)	40F	00F	$\bar{2} = 2 \times 2^{-2}$
17)	[00F	00F]	Working space
18)	20F	00F	$\bar{1} = 1 \times 2^{-2}$
19)	[00F	00F]	Working space
20)	08F	00F	$\bar{\bar{1}} = 1 \times 2^{-4}$
21)	00F	0050J	$1/2 \times 10^{-10}$
22)	S5	F	$\bar{x}_{n+1} \longrightarrow A$
	40	15L	$\longrightarrow 15L$
23)	26	L	
	0F	F	

CHAPTER 10

THE REPERTOIRE OF THE MISTIC

It is the purpose of this chapter to present in detail the description of MISTIC orders, with explanations about their functions.

10.1. Order Codes

The control section, as was mentioned previously, provides the computer with directions, while the control section itself receives its own directions in the form of a routine composed of orders or instructions. Each instruction is found occupying one-half of a word location in memory; i.e., instructions are packed two per word. The instruction is composed of two sections, the order code and the address. The use to be made of the address depends upon the order code associated with it.

Each order code is an eight bit number, with the first four bits called the order type and the second four bits called the order variant. This provides 2^8 or 256 order codes for the MISTIC, all of which cause the MISTIC to perform operations, some of which may be undesirable for a given program.

10.2. Order Types

The meaning of each order type is given in Table 10.1.1, where the sexadecimal representation of the four bit order type is used.

In every order whose execution requires a shift, the shift is a left shift when the order type is an even number, and it is a right shift when the order type is an odd number. In particular, order types 1, 7 and 9 involve right shifts of AQ, while order types 0, 6 and 8 involve left shifts of AQ in their execution.

10.3. Order Variants

The order types listed above are further amplified by the order variants and the addresses listed with them. The order variant bits are each used to specify certain modifications of the order. Order variant bits are denoted by v8, v4, v2 and v1, representing the 2^3, 2^2, 2^1 and 2^0 bit positions. The bits v1 and v8 operate with all order types in the same way. If v1 = 1, the accumulator is cleared before the operation corresponding to the order type is executed. If v1 = 0, the accumulator is not cleared prior to execution of the order corresponding to the order type.

If v8 = 1, 1/2 is added to the contents of A prior to the execution of the rest of the order. If (A) is not zero, an overflow could easily occur when v8 = 1.

To avoid this possibility, the computer has been designed to stop when (A) is not cleared previous to the addition of the 1/2. That is, when v8 = 1 and v1 = 0, the computer stops without executing the order. When the computer is again started, the next order in sequence is executed. This means that any order code whose order variant is 8, K, N or F will cause the computer to stop. No special indication of the cause of stoppage is made on the operator's panel, other than that the instruction register contains an operation code with the order variant 8, K, N or F.

Order Type Digit	Function	Meaning of Address Associated with Order Type
0	Left shift AQ as a 79 bit register	Number of places to shift
1	Right shift AQ as a 79 bit register	Number of places to shift
2	Transfer control unconditionally	Address of next order pair to be executed
3	Conditional control transfer	Address of location in memory from which next order pair may be taken
4	Copy the contents of A into memory	Address of location in memory from which to copy
5	Copy the contents of a memory location into Q	Address of location in memory from which to copy
6	Divide	Address of location in memory of divisor
7	Multiply	Address of location in memory of the multiplicand
8	Four bit character input and output	Number of four bit characters input and output
9	Five bit character input and output	See the 9 order type discussion which follows
K	Increment add Q and A	The address is arbitrary and meaningless
S	Add Q and A	The address is arbitrary and meaningless
N	Bank control order	Number of memory bank to be used
J	Logical Product	Address of location in memory containing one factor
F	Increment add memory and A	Address of location in memory containing one summand
L	Add	Address of location in memory containing one summand

Table 10.1.1

The order variant bits v2 and v4 act in the way displayed in Table 10.3.1.

Order Type	v2	Effect of v2	v4	Effect of v4
0 or 1		None		None
2 or 3	0	STOP if black switch is set to OBEY	0	Select right hand order
	1	Do not STOP	1	Select left hand order
4	0	Store entire word	0	
			1	
	1	Store right hand address	0	
	1	Store left hand address	1	
5 or J		None		None
6		Must be 1		Must be 1
7, S or L	0	Use signed number	0	Complement the number added into A
	1	Use absolute value of number	1	Do not complement number added into A
8	0	Input	0	Tape
	1	Output	1	Hang-up
9	0	Input	0	Tape
	1	Output	1	Cards
K or F	0	Subtract (R^3) from (A)	0	
	0	Add (R^3) to A	1	
	1	Subtract $\|(R^3)\|$ from A	0	
	1	Add $\|(R^3)\|$ to A	1	

Table 10.3.1

Besides the special meanings assigned to the v2 and v4 bits by being associated with a given order type, v2 and v4 usually have the following meanings.

If v2 = 0, the number in R^3 is treated as a signed number. If v2 = 1, the number in R^3 is used in its absolute value. For example, the order L0 n causes (n) to be subtracted from (A), whereas L2 n causes |(n)| to be subtracted from (A).

If v4 = 0, whenever R^3 is used it is used in its complemented (or negative) form. If v4 = 1, whenever R^3 is used it is used in its original form. Thus, L1 n

causes -(n) to be put into A, whereas L5 n causes (n) to be put into A.

As a further example, compare the orders L7 n, L3 n, L5 n and L1 n. L7 n, with v4 = v2 = 1, causes (n) to be put in A; L3 n, with v4 = 0, v2 = 1, causes - (n) to be put in A; L5 n, with v4 = 1, v2 = 0, causes (n) to be put in A; and finally, L1 n, with v4 = v2 = 0, causes -(n) to be put in A.

However, if no arithmetic is being performed in a given order, these general meanings of v2 and v4 are not significant. Thus, order types 0, 1, 2, 3, 4, 5, 8, 9 and J do not involve arithmetic and the general meanings of v2 and v4 do not apply. However, in order types 6, 7, K, S, F and L the general meaning does apply. Desirable effects are achieved in these latter cases except for the 6 order type which is a divide order. More will be said about these orders in the following pages.

10.4. The MISTIC Repertoire of Instructions

In Chapter 3, listings of the most commonly used orders under each order type are given, along with a minimum of explanatory discussion. In many cases, the same results can be obtained by any one of several orders. In such cases, only one of the several possible orders appears in the listings in Chapter 3.

In this section, a detailed explanatory discussion of each order type is given. Each such discussion is followed by a complete table of orders of the given type.

While the material in Chapter 3 is adequate for most programming, this chapter presents a more technical aspect and makes possible a somewhat more sophisticated approach to programming.

Order Type 0--Left Shift and Final Stop

These are orders of the form 0v n where v is the order variant and n is the "address", which in this case is used to specify the number of shifts to occur. The count, n, is interpreted modulo 64.

If n = 0, the computer will stop with the 0v 0 order displayed in the instruction register. If not, repeat n times the operation which replaces the contents

$$a_0a_1a_2\cdots a_{38}a_{39} \quad q_0q_1q_2\cdots q_{38}q_{39}$$

of AQ by

$$a_1a_2a_3\cdots a_{39}q_1 \quad q_0q_2q_3\cdots q_{39}0$$

After an n place left shift, a_m has been placed in the position previously occupied by a_{m-n}, where if m-n is negative, a_m has been shifted off the end of A and lost. Likewise, q_m has been shifted to q_{m-n} if m-n is positive; when m-n is not positive, q_m has been moved into the position formerly occupied by a_{39+m-n}.

Here again a negative subscript on a_i means that the bit has been lost off the end of A. In every case, q_0 is not changed.

00 n 02 n 04 n 06 n	Left shift AQ n places.
01 n 03 n 05 n 07 n	Clear A, then left shift AQ n places.
09 n 0S n 0J n 0L n	Clear A, insert 1/2 in A, and then left shift AQ n places.
08 n 0K n 0N n 0F n	The computer will stop with the order in the instruction register.

Order Type 1--Right Shift

These are orders of the form 1v n where v is the order variant and n is the "address" which in this case is used to specify the number of shifts to occur. The count, n, is interpreted modulo 64.

If n = 0, the computer will stop with the 1v 0 order displayed in the instruction register. If not, repeat n times the operation which replaces the contents

$$a_0 a_1 a_2 \cdots a_{38} a_{39} \quad q_0 q_1 q_2 \cdots q_{38} q_{39}$$

of AQ by

$$a_0 a_0 a_1 \cdots a_{37} a_{38} \quad q_0 a_{39} q_1 \cdots q_{37} q_{38}$$

After an n place right shift, a_m has been placed in the position previously occupied by a_{m+n} if m+n is not greater than 39, whereas when m+n is greater than 39, a_m has been moved into the position previously occupied by q_{m+n-39}. Also, q_m has been shifted to q_{m+n} if m+n is not greater than 39, whereas when m+n is greater than 39, q_m has been shifted off the right end of Q and lost. In every case, q_0 is not changed.

10 n 12 n 14 n 16 n	Right shift AQ n places.
11 n 13 n 15 n 17 n	Clear A and then right shift AQ n places.
19 n 1S n 1J n 1L n	Clear A, insert 1/2 in A and then right shift AQ n places.
18 n 1K n 1N n 1F n	The computer will stop with the order in the instruction register.

Order Type 2--Unconditional Control Transfer

These are orders of the form 2v n where v is the order variant and n is the address of the next order pair to be put into the instruction register. By choosing the variant v in the 2v n order correctly, the right or left-hand order at n will be executed with an optional stop beforehand. The stop can be ignored by setting the black switch to IGNORE.

20	n	The computer will stop with this order in the instruction register. The first order after starting the computer with a black switch start will be the right-hand order at memory location n.
22	n	Transfer control to the right-hand order at location n.
24	n	The computer will stop with this order in the instruction register. The first order after starting the computer with a black switch start will be the left-hand order at at memory location n.
26	n	Transfer control to the left-hand order at location n.
21 n 23 n 25 n 27 n		Clear A, then do exactly as in 20 n, 22 n, 24 n, 26 n
29 n 2S n 2J n 2L n		Clear A, insert 1/2 in A, then do exactly as in 20 n, 22 n, 24 n, 26 n
28 n 2K n 2N n 2F n		The computer will stop with the order displayed in the instruction register and will not start with a black switch start.

See page 10.17 for starting after stops.

Order Type 3--Conditional Transfer of Control

These are orders of the form 3v n where v is the order variant and n is the address from which the next order may be selected.

If $a_0 = 0$, i.e., if $(A) \geq 0$, the next order to be executed will be taken from location n.

If $a_0 = 1$, i.e., if $(A) < 0$, the next order is taken in normal sequence.

By selecting v properly, the next order in the case of $a_0 = 0$ will be the left or right-hand order with a stop before the transfer or not. This stop can be ignored by setting the black switch to IGNORE.

30 n If $a_0 = 0$, i.e., if (A) ≥ 0, the computer will stop with this order in the instruction register. The first order after starting the computer by a black switch start will be the right-hand order at memory location n.
If $a_0 = 1$, the computer does not stop but goes on to the next order in sequence.

31 n Clear A and then the computer will stop with this order in the instruction register. The first order after starting with a black switch start is the right-hand order at location n.

32 n If $a_0 = 0$, i.e., if (A) ≥ 0, the next order will be the right-hand order at memory location n.
If $a_0 = 1$, the computer goes on to the next order in sequence.

33 n Clear A and transfer control to the right-hand order at memory location n.

34 n If $a_0 = 0$, i.e., if (A) ≥ 0, the computer will stop with this order in the instruction register. The first order after starting with a black switch start will be the left-hand order at memory location n.
If $a_0 = 1$, the computer does not stop but goes on to the next order in sequence.

35 n Clear A, then the computer will stop with this order in the instruction register. The first order after starting with a black switch start will be the left-hand order at memory location n.

36 n If $a_0 = 0$, i.e., if (A) ≥ 0, the next order will be the left-hand order at memory location n.
If $a_0 = 1$, the computer goes on to the next order in sequence.

37 n Clear A. The next order will be the left-hand order at memory location n.

39 n Clear A, insert 1/2 in A, and then the computer will stop with this order in the instruction register. The first order after starting with a black switch start will be the right-hand order at memory location n.

3S n Clear A and insert 1/2 in A. The next order will be the right-hand order at memory location n.

3J n Clear A, insert 1/2 in A, and then the computer will stop. The next order after starting with a black switch start will be the left-hand order at memory location n.

3L n Clear A and insert 1/2 in A. The next order will be the left-hand order at memory location n.

38 n
3K n — The computer will stop with the order in the instruction register and will not start with a black switch start.
3N n
3F n

Summary

1. Stop orders restarted by black switch
   a. Left-hand orders
      24 n 34 n
      25 n 35 n
      2J n 3J n
   b. Right-hand orders
      20 n 30 n
      21 n 31 n
      29 n 39 n

2. Non-stop orders
   a. Left-hand orders
      26 n 36 n
      27 n 37 n
      2L n 3L n
   b. Right-hand orders
      22 n 32 n
      23 n 33 n
      2S n 3S n

See page 10.17 for starting after stops.

Order Type 4--Store the Contents of A

These are orders of the form 4v n where v is the order variant and n is the address of the location in memory all of which is cleared and into which (A) is copied, or part of which is cleared and into which the corresponding part of (A) is copied, depending upon the digit v. As the result of a 4v order with even order variant digit, the contents of A are left unchanged. In the descriptions below, 2^{-10} and 2^{-30} should be replaced by 2^{-8} and 2^{-28} respectively when referring to the core memory.

40 n, 44 n — Copy (A) into memory location n.

41 n, 45 n — Clear A, then copy (A) into memory location n. This order sets (A) and (n) to zero.

42 n — Copy the right-hand address part of (A) into the right-hand address part of memory location n. The only bit positions affected are $2^{-30} \ldots 2^{-39}$.

43 n — Clear A, then copy the right-hand address part of (A) into the right-hand address part of memory location n. This order clears A and sets bit positions $2^{-30} \ldots 2^{-39}$ in n to zero.

46 n — Copy the left-hand address part of (A) into the left-hand address part of memory location n. The only bit positions affected are $2^{-10} \ldots 2^{-19}$.

47 n — Clear A, then copy the left-hand address part of (A) into the left-hand address part of memory location n. This order clears A and sets bit positions $2^{-10} \ldots 2^{-19}$ in n to zero.

49 n, 4J n — Clear A, insert 1/2 in A, and then copy (A) into memory location n. This order sets (A) and (n) to 1/2.

4S n — Clear A, insert 1/2 in A, and then copy the right-hand address part of (A) into the left-hand address part of memory location n. This order sets (A) to 1/2 and sets bit positions $2^{-30} \ldots 2^{-39}$ in memory location n to zero.

4L n — Clear A, insert 1/2 in A, and then copy the left-hand address part of (A) into the left-hand address part of memory location n. This order sets (A) to 1/2 and sets bit positions $2^{-10} \ldots 2^{-19}$ in memory location n to zero.

48 n, 4K n, 4N n, 4F n — The computer will stop with the order in the instruction register.

Order Type 5--Load Q from Memory

These are orders of the form 5v n where v is the order variant and n is the address of a memory location, the contents of which are copied into Q. The contents of location n are left unchanged by these orders.

50 n, 52 n, 54 n, 56 n — Copy (n) into Q.

51 n, 53 n, 55 n, 57 n — Clear A and copy (n) into Q.

59 n, 5S n, 5J n, 5L n — Clear A, insert 1/2 in A, then copy (n) into Q.

58 n, 5K n, 5N n, 5F n — The computer will stop with the order in the instruction register.

Order Type 6--Divide

These are orders of the form 6v n where v is the order variant and n is the address of a word in memory used as the divisor. The divide instruction divides the contents of AQ by the contents of n, leaving a quotient in Q and a "remainder" in A. The quotient is always a rounded quotient, the rounding being accomplished by setting q_{39} to 1.

The computer will stop after dividing if $|(A)| > |(n)|$, or if $|(A)| = |(n)|$ and $(A) \geq 0$.

An unrounded quotient can be obtained if it is possible to left shift AQ one place prior to division. Then after division, right shift AQ one place. The bit in a_{39} is a zero due to the left shift, so that Q is not affected by the right shift.

The divide order has been made so as to make 66 n work correctly. As a result, most of the other 6v n orders have little usefulness. The 66, 67 and 6L orders are listed here, followed by a general discussion of the 6v orders.

66	n	Divide (AQ) by (n), leaving a rounded quotient in Q and a remainder in A.
67	n	Clear A and then divide (AQ) by (n), leaving a rounded quotient in Q and a remainder in A.
6L	n	Clear A, insert 1/2 in A, then divide (AQ) by (n), leaving a rounded quotient in Q and a remainder in A.

The divide order in MISTIC. The 66 order operates in the following way. The divisor is sent to R^3. The sign of the dividend and the sign of the divisor are compared. If they agree, the complement gate is set to subtract throughout the division. If they do not agree, the complement gate is set to add throughout the division.

A temporary partial remainder (TPR) is formed by adding the divisor to or subtracting it from the quantity in A, according to the setting of the complement gate.

If the sign of the TPR agrees with the sign of the dividend, the TPR will be left in A. If not, the previous contents of A will be restored to A.

If the sign of the TPR agrees with the sign of the divisor, a 1 is sent to q_{39}; if they do not agree, a 0 is sent to q_{39}. Then AQ is left shifted one place and another subtraction takes place. This is done 40 times. The sign in q_0 is the first quotient digit.

In general, however, the division depends on the following quantities:

A--the sign bit of the dividend. Here A = 0 means +,
A = 1 means -.

B--the sign bit of the divisor

D--the v4 bit
E--the v2 bit
F--the sign bit of the TPR
U--the setting of the complement gate
If U = 0, the complement gate is set to add, and
if U = 1, the complement gate is set to subtract.
α--If α = 0, restore to A the contents of A prior to forming of a TPR
If α = 1, do not restore.
q_{39}--the last bit of Q

U, α and q_{39} are functions of A, B, D, E and F given by the following Boolean equation:

$$\overline{U} = \overline{A}BE + B\overline{D}E + A\overline{B}D + AD\overline{E}$$

$$\alpha = BE(ADUF + \overline{AD}\ \overline{U}\ \overline{F}) + \overline{EB}(\overline{AD}\ U\ \overline{F} + AD\overline{U}F)$$

and $$q_{39} = BF + \overline{B}\ \overline{F}$$

Thus, q_{39} is always selected in the same manner, regardless of the variant digit of the order (since q_{39} does not depend upon D and E). The setting of the complement gate and the decision about restoring (A), however, depend upon D and E and, therefore, change depending upon the variant digit. The formulas for U and α for each of the variant digits are presented here.

60 and 61	$U = 1, \alpha = \overline{F}$
62 and 63	$U = \overline{B}, \alpha = \overline{F}$
64 and 65	$U = \overline{A}, \alpha = \overline{A}\ \overline{F} + AF$
66 and 67	$U = AB + \overline{A}\overline{B}, \alpha = \overline{A}\ \overline{F} + AF$
69	$U = 1, \alpha = \overline{F}$
6S	$U = \overline{B}, \alpha = \overline{F}$
6J	$U = \overline{A}, \alpha = \overline{A}\ \overline{F} + AF$
6L	$U = AB + \overline{A}\overline{B}, \alpha = \overline{A}\ \overline{F} + AF$

Order Type 7--Multiply

These are orders of the form 7v n where v is the order variant and n is the address of a location in memory which contains the multiplier. These orders form $\pm(Q)(n) + (A) \cdot 2^{-39}$ or $\pm(Q)\ |(n)| + (A) \cdot 2^{-39}$ in AQ, the least significant 39 bits being in Q with $q_0 = 0$.

The orders can be considered to form the product (Q)(n) in AQ followed by adding the original contents of A (as it was just before the multiplication) into Q.

The orders 79 n, 7J n and 7L n are rounded multiplications since (A) = 1/2 in each case so that (A) x $2^{-39} = 2^{-40}$. But 2^{-40} added to AQ is a 1 added to the

q_1 position. If $q_1 = 1$ initially, the effect of adding 2^{-40} is to increase a_{39} by 1. If $q_1 = 0$ initially, there is no effect on A in adding 2^{-40} to AQ.

If it is desired to form a product followed by a left shift of n places and then to round to a_{39}, the initial contents of A should be 2^{-n-1}, and the multiplication order should be one of 70 n, 72 n, 74 n or 76 n.

70	n	Form $-(n)(Q) + (A) \times 2^{-39}$ in AQ
71	n	Clear A and then form $-(n)(Q)$ in AQ
72	n	Form $-\|(n)\|\,(Q) + (A) \times 2^{-39}$ in AQ
73	n	Clear A and then form $-\|(n)\|\,(Q)$ in AQ
74	n	Form $(n)(Q) + (A) \times 2^{-39}$ in AQ
75	n	Clear A and then form $(n)(Q)$ in AQ
76	n	Form $\|(n)\|\,(Q) + (A) \times 2^{-39}$ in AQ
77	n	Clear A and then form $\|(n)\|\,(Q)$ in AQ
79	n	Clear A, insert 1/2 in A and then form $-(n)(Q) + (A) \times 2^{-39}$
7S	n	Clear A, insert 1/2 in A and then form $-\|(n)\|\,(Q) + (A) \times 2^{-39}$ in AQ. This is a rounded multiplication.
7J	n	Clear A, insert 1/2 in A and then form $(n)(Q) + (A) \times 2^{-39}$ in AQ. This is a commonly used rounded multiplication.
7L	n	Clear A, insert 1/2 in A, then form $\|(n)\|\,(Q) + (A) \times 2^{-39}$ in AQ. This is a rounded multiplication.
78	n	The computer will stop with the order in the instruction register.
7K	n	
7N	n	
7F	n	

Order Type 8--Four Hole Tape Input-Output

These are orders of the form 8v n where v is the order variant and n has the meanings given below. These orders transfer only four hole characters to the A register and tape. The input orders cause all characters with a perforated fifth hole to be bypassed and the tape will advance until the appropriate number of four hole characters has been input.

Basic input step. Reduce the address n to n' where $n' \equiv n$ (modulo 64) and $0 \leq n' < 64$. Let

$$\frac{n'}{4} = q + \frac{r}{4} \quad \text{where } 0 \leq r < 4$$

Perform the following operation q times.

Left shift AQ four places. Transfer the four hole character being read from tape to $a_{36}a_{37}a_{38}a_{39}$. (This input is a "logical or" type; however, those four bits are cleared prior to input.)

Then left shift AQ r places. However, Q does not shift into A but rather performs an end off shift.

If $n' < 4$, no input takes place, but A and Q each shift left n' places, but Q does not shift into A. This is a means of obtaining a 1, 2 or 3 place shift of A without shifting Q into A.

If $n' = 0$, the computer will stop.

Basic output step. Let $\frac{n'}{4} = q + \frac{r}{4}$ as before. Do the following q times.

Punch on tape the four hole character corresponding to $a_0a_1a_2a_3$.

Left shift four places. Then if $r \neq 0$, punch on tape the four hole character corresponding to $a_0a_1a_2a_3$ and left shift AQ r places. If $r = 0$, stop after punching out the q characters. Q always shifts into A for output orders which do not stop the computer. If $n' = 0$, the computer will stop. If $0 < n' < 4$, one character is output corresponding to $a_0a_1a_2a_3$ and an n' place left shift follows, Q shifting into A.

80 n Perform the basic input step.

81 n Clear A, then perform the basic input step.

89 n Clear A, insert 1/2 in A, then perform the basic input step.

82 n Perform the basic output step.

83 n Clear A, then perform the basic output step.

8S n Clear A, insert 1/2 in A, then perform the basic output step.

84 n Left shift A and Q one place, but Q does not shift into A. Then the computer will stop.

85 n Clear A, then left shift A and Q one place, but Q does not shift into A. Then the computer will stop.

8J n Clear A, insert 1/2 in A, then left shift A and Q one place, but Q does not shift into A. Then the computer will stop.

86 n The computer will stop.

87 n Clear A, then the computer will stop.

8L n Clear A, insert 1/2 in A. Then the computer will stop.

88 n
8K n
8N n
8F n
— The computer will stop with the order in the instruction register.

Order Type 9--Five Hole and Card Input-Output

These orders are of the form 9v n and are concerned with reading and punching alphanumeric information and punched cards. Whether tape or cards are used is determined by the variant bit v4. That is, if v4 = 0, tape will be read or punched while if v4 = 1, cards will be read or punched.

Basic tape input step. Reduce n to n' where $n' \equiv n$ (modulo 64) and $0 \leq n' < 64$. Let $\frac{n'}{4} = q + \frac{r}{4}$ where $0 \leq r < 4$. Do the following q times:

Right shift AQ four places (bypassing q_0). Insert the four least significant bits of the character being input into $a_{36}a_{37}a_{38}a_{39}$ and insert the most significant bit into a_0.

Then right shift AQ r places. If q = 0 initially, i.e., if $n' < 4$, there is no input. If n' = 0, the computer will stop.

These inputs are of the "logical or" type. That is, if there is a one in any of $a_{36}a_{37}a_{38}a_{39}$ or in the four bits being read, then in the corresponding place in A will be a one.

Basic tape output step. Let $n = n_0n_1n_2n_3n_4n_5n_6n_7n_8n_9$ in binary. Punch on the tape a character whose fifth hole is n_9, and whose remaining four holes are $n_0n_1n_2n_3$ in that order; and punch that character $n_4n_5n_6n_7$ times if both n_8 and n_9 are zero and $n_4n_5n_6n_7+1$ times otherwise; right shift AQ $n_4n_5n_6n_7n_8n_9$ times.

One formula which will give n is

$$n = 64a + 4b + c - 2$$

where a is the four hole character to be punched, b is the number of times to punch the character, c is the fifth hole, and 4b + c - 2 is the number of right shifts of AQ. If $n' = n_4n_5n_6n_7n_8n_9 = 0$, the computer will stop.

Basic card input step. For this order, n must be zero. The 80 columns of a card row will then be read into the 80 bits of AQ, the exact location being dependent on the plugboard wiring, thus allowing columns to be read into AQ in whatever order desired. One row at a time is read and a check circuit in the computer precludes a card row going by without being read. If this should happen, the computer will hang up on the succeeding card read order. Since the first read order initiates a card cycle and all twelve rows will pass the reading station, read orders must occur in groups of 12 within a specified time, i.e., with less than 8.7 milliseconds between read orders. Additional information concerning card handling is included in Chapters 13 and 14.

Basic card output step. Here also n must be zero. When the punch order occurs, the contents of AQ are punched in the 80 columns of a card row, the exact column depending on the plugboard wiring. Again, the first punch order initiates

a card cycle and all twelve rows must be punched within a specified time, i.e., with less than 17.4 milliseconds between rows. Further discussion of the card punching procedure is included in Chapters 13 and 14.

90	n	Perform the basic tape input step.	tape input
91	n	Clear A, perform the basic tape input step.	
99	n	Clear A, insert 1/2 in A, perform the basic tape input step.	
92	n	Perform the basic tape output step.	tape output
93	n	Clear A, perform the basic tape output step.	
9S	n	Clear A, insert 1/2 in A, perform the basic tape output step.	
94	n	Perform the basic card input step.	card input
95	n	Clear A, perform the basic card input step.	
9J	n	Clear A, insert 1/2 in A, perform the basic card input step.	
96	n	Perform the basic card output step.	card output
97	n	Clear A, perform the basic card output step.	
9L	n	Clear A, insert 1/2 in A, perform the basic card output step.	
98	n	The computer will stop with the order in the instruction register.	
9K	n		
9N	n		
9F	n		

Order Type K--Increment Add from Q

These are orders of the form Kv n where v is the order variant and n is meaningless and can be any address. These orders make use of $\pm [(Q) + 2^{-39}]$. It is to be noticed that $-(Q) - 2^{-39}$ is represented in the computer by $2 - (Q) - 2^{-39}$ which is the one's complement of (Q). (Q) is left unchanged.

K0	n	Add $-(Q) - 2^{-39}$ to (A).
K1	n	Clear A and add $-(Q) - 2^{-39}$ to (A). This order puts the one's complement of Q in A.
K2	n	If $(Q) \geq 0$, add $-(Q) - 2^{-39}$ to (A). If $(Q) < 0$, add $(Q) + 2^{-39}$ to (A).
K3	n	Clear A. If $(Q) \geq 0$, add $-(Q) - 2^{-39}$ to (A). If $(Q) < 0$, add $(Q) + 2^{-39}$ to (A).
K4	n	Add $(Q) + 2^{-39}$ to (A).
K5	n	Clear A and add $(Q) + 2^{-39}$ to (A).
K6	n	If $(Q) \geq 0$, add $(Q) + 2^{-39}$ to (A). If $(Q) < 0$, add $-(Q) - 2^{-39}$ to (A).
K7	n	Clear A. If $(Q) \geq 0$, add $(Q) + 2^{-39}$ to (A). If $(Q) < 0$, add $-(Q) - 2^{-39}$ to (A).
K9	n	Clear A, insert 1/2 in A, then add $-(Q) - 2^{-39}$ to (A).
KS	n	Clear A, insert 1/2 in A. If $(Q) \geq 0$, add $-(Q) - 2^{-39}$ to (A). If $(Q) < 0$, add $(Q) + 2^{-39}$ to (A).
KJ	n	Clear A, insert 1/2 in A. Then add $(Q) + 2^{-39}$ to (A).
KL	n	Clear A, insert 1/2 in A. If $(Q) \geq 0$, add $(Q) + 2^{-39}$ to (A). If $(Q) < 0$, add $-(Q) - 2^{-39}$ to (A).
K8	n	The computer will stop with the order in the instruction register.
KS	n	
KN	n	
KF	n	

Order Type S--Add from Q

These are orders of the form Sv n where v is the order variant and n is meaningless and can be any address. The contents of Q are left unchanged.

S0	n	Subtract (Q) from (A).
S1	n	Clear A and subtract (Q) from (A). This order puts -(Q) into A.
S2	n	Subtract |(Q)| from A.
S3	n	Clear A and subtract |(Q)| from A. This order puts -|(Q)| into A.
S4	n	Add (Q) to (A).
S5	n	Clear A and add (Q) to (A). This order puts (Q) into A.
S6	n	Add |(Q)| to (A).
S7	n	Clear A and add |(Q)| to (A). This order puts |(Q)| into A.
S9	n	Clear A and insert 1/2 in A. Then subtract (Q) from (A). This order forms 1/2 - (Q) in A.
SS	n	Clear A and insert 1/2 in A. Then subtract |(Q)| from (A). This order puts 1/2 - |(Q)| into A.
SJ	n	Clear A and insert 1/2 in A. Then add (Q) to (A). This order puts 1/2 + (Q) into A.
SL	n	Clear A and insert 1/2 in A. Then add |(Q)| to (A). This order puts 1/2 + |(Q)| into A.
S8	n	The computer will stop with the order in the instruction register.
SK	n	
SN	n	
SF	n	

Order Type J--Logical Product or Extract

These are orders of the form Jv n where v is the order variant and n is the address of a word in memory. If two corresponding bits of (n) and (Q) are both 1's, put a 1 into that place in Q. Otherwise, put a 0 in that place in Q. This order gives the bit-wise logical product of (Q) and (n).

J0	n	Form the bit-wise logical product of (Q) and (n) in Q.
J2	n	
J4	n	
J6	n	
J1	n	Clear A, then form the bit-wise logical product of (Q) and (n) in Q.
J3	n	
J5	n	
J7	n	
J9	n	Clear A, insert 1/2 in A, and then form the bit-wise logical product of (Q) and (n) in Q.
JS	n	
JJ	n	
JL	n	
J8	n	The computer will stop with the order in the instruction register.
JK	n	
JN	n	
JF	n	

Order Type N--Bank Control Order

Orders of the form Nv n where v is the order variant are used to change the bank number (0, 1, 2, 3) from which ensuing orders and operands will be taken. The bank which is to be used is determined by the last two bits of the address n.

N0 n — Go to the next order in sequence.
N1 n — Clear A. Go to the next order in sequence.
N2 n — Change the bank number from which operands are coming to that indicated by the last two bits of n.
N3 n — Clear A. Change the bank number from which operands are coming to that indicated by the last two bits of n.
N4 n — Change the bank number from which orders are coming to that indicated by the last two bits of n.
N5 n — Clear A. Change the bank number from which orders are coming to that indicated by the last two bits of n.
N6 n — Change the bank numbers for both orders and operands to that indicated by the last two bits of n.
N7 n — Clear A. Change the bank numbers for both orders and operands to that indicated by the last two bits of n.
N9 n — Clear A, insert 1/2 in A. Go to the next order in sequence.
NS n — Clear A, insert 1/2 in A. Change the bank number from which operands are coming to that indicated by the last two bits of n.
NJ n — Clear A, insert 1/2 in A. Change the bank number from which orders are coming to that indicated by the last two bits of n.
NL n — Clear A, insert 1/2 in A. Change the bank numbers for both orders and operands to that indicated by the last two bits of n.

N8 n
NK n
NN n
NF n
— The computer will stop with the order in the instruction register.

Order Type F--Increment Add and Specially Used Stop

These are orders of the form Fv n where v is the order variant and n is the address of a word in memory. These orders make use of $\pm[(n) + 2^{-39}]$. It is to be noticed that $-(n) - 2^{-39}$ is the one's complement of (n).

F0 n — Add $-(n)-2^{-39}$ to (A).
F1 n — Clear A, then add $-(n)-2^{-39}$ to (A).
F2 n — If $(n) \geq 0$, add $-(n)-2^{-39}$ to (A). If $(n) < 0$, add $(n)+2^{-39}$ to (A).
F3 n — Clear A. If $(n) \geq 0$, add $-(n)-2^{-39}$ to (A). If $(n) < 0$, add $(n)+2^{-39}$ to (A).
F4 n — Add $(n)+2^{-39}$ to (A).
F5 n — Clear A, then add $(n)+2^{-39}$ to (A). This order is useful for increasing a right-hand address.
F6 n — If $(n) \geq 0$, add $(n)+2^{-39}$ to (A). If $(n) < 0$, add $-(n)-2^{-39}$ to (A).
F7 n — Clear A. If $(n) \geq 0$, add $(n)+2^{-39}$ to (A). If $(n) < 0$, add $-(n)-2^{-39}$ to (A).
F9 n — Clear A and insert 1/2 in A. Then add $-(n)-2^{-39}$ to (A).
FS n — Clear A and insert 1/2 in A. If $(n) \geq 0$, add $-(n)-2^{-39}$ to (A). If $(n) < 0$, add $(n)+2^{-39}$ to (A).
FJ n — Clear A and insert 1/2 in A. Add $(n)+2^{-39}$ to (A).
FL n — Clear A and insert 1/2 in A. If $(n) \geq 0$, add $(n)+2^{-39}$ to (A). If $(n) < 0$, add $-(n)-2^{-39}$ to (A).

F8 n
FK n
FN n
FF n
— The computer will stop with the order in the instruction register. The FF n order is used frequently to give an indication of why the computer stopped. The address, n, is assigned meanings by the programmer. Then when this stop occurs with a specific address, the programmer can determine the cause of the stop.

Order Type L--Add

These are orders of the form Lv n where v is the order variant and n is the address of a word in memory. Either ± (n) or ± |(n)| is added to (A). The contents of n are left unchanged.

L0 n Subtract (n) from (A).
L1 n Clear A and subtract (n) from (A). This order puts -(n) into A.
L2 n Subtract |(n)| from (A).
L3 n Clear A and subtract |(n)| from (A). This order puts - |(n)| into A.
L4 n Add (n) to (A).
L5 n Clear A and add (n) to (A). This order puts (n) into A.
L6 n Add |(n)| to (A).
L7 n Clear A and add |(n)| to (A).
L9 n Clear A and insert 1/2 in A. Then subtract (n) from (A). This order puts 1/2 - (n) into A.
LS n Clear A and insert 1/2 in A. Then subtract |(n)| from (A). This order puts 1/2 - |(n)| into A.
LJ n Clear A and insert 1/2 in A. Then add (n) to (A). This order puts 1/2 + (n) into A.
LL n Clear A and insert 1/2 in A. Then add |(n)| to (A). This order puts 1/2 + |(n)| into A.

L8 n
LK n
LN n
LF n
— The computer will stop with the order in the instruction register.

10.5. Starting After Stops

There are two kinds of stops in the MISTIC: those which the black switch can restart and those which the black switch cannot restart. The orders which cause the computer to stop which allow the computer to be restarted by the black switch are 20, 21, 24, 25, 29, 2J, 30, 31, 34, 35, 39 and 3J. The orders which cause a stop not restartable by the black switch are all orders with order variant 8, K, N or F; orders with order type 0, 1, 8 or 9 and an address congruent to zero modulo 64; orders 84, 85, 86, 87, 8J, 8L or 9J.

10.6. Starting After a Stop Transfer of Control

When the computer has stopped because of one of the above listed stop transfers of control, it is usually restarted by moving the black switch to START from which it automatically returns to OBEY. The stop occurs before the entire transfer order has been executed. If the black switch is set to IGNORE, the stop will not occur.

10.7. Starting by the White Switch

When the computer has stopped due to any of the orders listed above, it can be restarted by moving the white switch to FETCH (which advances the left-right count to the next order in sequence), then down to EXECUTE and RUN. If the stop

was a left-hand order, the next order obeyed will be in the right-hand position of that order pair. If the stop was a right-hand order, the next order obeyed will be the left-hand order from the next order pair. Transfer of control orders are ignored with one exception. If the stop was caused by a right-hand order which is one of the control transfer orders restartable by the black switch, and which transfers control to the right side of an order, then the next order obeyed upon setting the white switch to FETCH, EXECUTE and RUN will be the right-hand order of the next order pair instead of the left-hand order. It should be observed that 28 and 38 are not restarted by the black switch, so that this exception does not apply. For example, when the stop occurs at the right side of location n in the following example, control is sent to the right side of m by the black switch start, but to the right side of n+1 by a white switch start.

```
n)      ---
        20 m
n+1)    ---
        ---
```

10.8. Another Use of the White Switch Start

If it is desired after any stop to transfer control to some arbitrary location, n, a tape with 26 n 00 000 or 22 n 00 000, all in sexadecimal, should be inserted in the reader and a bootstrap start given. By a bootstrap start is meant setting the instruction register to 80 028 40 000 and the control counter to 000.

CHAPTER 11

ARITHMETIC IN THE MISTIC

Computers generally present a compromise between what is mathematically desirable and what is feasible engineering-wise. As a result, certain features of a computer may seem peculiar. This chapter will try to explain the peculiarities of the MISTIC.

11.1. Number Representation

Every word in the MISTIC is forty bits long with the first bit being a sign bit and the next thirty-nine being the representation of a fraction. If the bits are denoted $x_0x_1x_2\ldots x_{39}$, then x_0 is the sign bit and a binary point is assumed to lie between x_0 and x_1. To accommodate negative numbers without providing special handling for the sign, the MISTIC uses a 2's complement system. As a result, a number in the MISTIC format represents

$$-x_0 + \sum_{n=1}^{39} x_n \cdot 2^{-n}$$

This implies that all numbers x represented in the MISTIC must lie in the range $-1 \leq x < 1$. Any number manufactured by the computer outside that range is said to overflow.

11.2. The MISTIC Arithmetic Unit

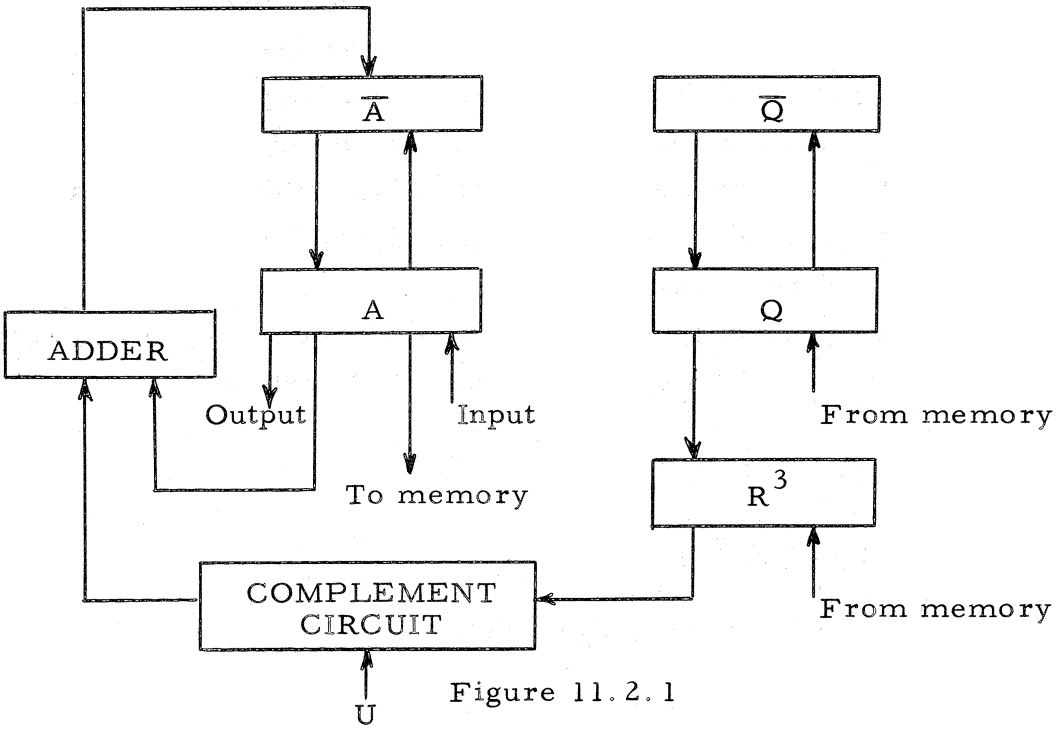


Figure 11.2.1

A is called the accumulator and Q, the quotient register. $\overline{A}$ is the temporary accumulator and $\overline{Q}$, the temporary quotient register.

A can transfer straight up to $\overline{A}$ (no shift), and Q, straight up to $\overline{Q}$. $\overline{Q}$ transfers down to Q either with a left shift of one place or a right shift of one place. $\overline{A}$ can shift straight down or with a one place left or right shift.

The complement circuit is set to complement if U = 1 (v4 = 0) and it is set to non-complement if U = 0 (v4 = 1).

Each of the above registers performs certain functions in the various operations of which the computer is capable.

11.3. Addition

An addition is called for by the order type L with an order variant whose v4 bit is 1. When an Lv n order is given, the state of v4 is sensed. If v4 is 1, the complement circuit is set to add (i.e., U = 0). Then the number in memory location n is copied into R^3, and is sent from there to the adder through the complement gate. Likewise, the number in A is sent to the adder, and the result is placed in $\overline{A}$, then in A.

11.4. Subtraction

A subtraction is called for by an L order type with an order variant in which v4 = 0. In this case, the complement circuit is set to subtract (U = 1). Then the rest of the operation is the same as the addition.

11.5. Multiplication

Initially, the multiplier y lies in Q while the multiplicand x is copied from memory location to R^3. The multiplication is then a series of additions and right shifts with a partial product left in A at the end of each step, as follows:

The bit in q_{39} is sensed. If $q_{39} = 1$, the sum of (A) and (R^3) is sent to $\overline{A}$. If $q_{39} = 0$, (A) is sent to $\overline{A}$. Then in either case $\overline{A}$ is sent down with a right shift to A, the right most bit of A being sent to q_1, and the rest of Q shifting right one place. This process is done 39 times.

If the ith partial product formed in A at the end of the ith iteration is denoted by p_i, i = 0, 1, ..., 39 where p_0 is the initial content of A, then

$$p_{i+1} = \frac{1}{2}(p_i + y_{39-i}x)$$

Then if i = 38,

$$p_{39} = \frac{1}{2}(p_{38} + y_1 x)$$

and

$$p_{39} = \frac{1}{2} y_1 x + \frac{1}{2}(p_{37} + y_2 x)$$

$$p_{39} = 2^{-39} p_0 + x \sum_{n=1}^{39} y_n 2^{-n}$$

Since $$y = -y_0 + \sum_{n=1}^{39} y_n 2^{-n}$$

as shown in Section 11.1, then

$$p_{39} = 2^{-39} p_0 + x(y_0 + y) \qquad (11.5.1)$$

The correct result in multiplying x by y should, of course, be xy. This is the result obtained in equation 11.5.1 provided $y_0 = 0$, i.e., provided y is non-negative. If, however, $y_0 = 1$ (when y is negative), then equation 11.5.1 is in excess of xy by xy_0. In this case, the control circuitry is made to subtract out the xy_0.

11.6. Division (66)

The divide algorithm for the MISTIC is designed so as to make the 66, 67, and 6L orders produce correct quotients. As a result, the other 6v orders produce predictable but generally useless results. This section will present the divide algorithm for the 6v order with v4 = v2 = 1.

The method used is the same as long division at least in the case when the dividend and divisor are positive. In that case the divisor is subtracted from the dividend. If the result is non-negative, a quotient digit of 1 is produced; if not, a quotient digit of 0 is produced. Then, if the difference is negative, the dividend is restored to its previous value. In either case, the dividend shifts left one place and the process is repeated. A total of 40 shifts takes place.

The dividend occupies AQ and the divisor, R^3. The quotient digits are formed in Q. Each left shift sends q_1 into q_0 and into a_{39}, thus leaving a sign bit in q_0. A remainder of sorts is found in A.

The precise algorithm for 6v n with v4 = 1 and v2 = 1 will be given after some definitions are made.

Let the dividend in AQ be called r_0 with a sign bit p_0. The divisor in R^3 is called y with a sign bit y_0. The partial remainder in AQ after the division starts is called r_n at the nth step. Then,

1) Compare y_0 and p_0.

If $y_0 = p_0$, set the complement circuit to subtract throughout the division. If $y_0 \neq p_0$, set the complement circuit to add. Define

$$\sigma = (-1)^{p_0 + y_0} \qquad (11.6.1)$$

Do the following 40 times:

2) Form a tentative partial remainder s_n with sign bit t_n by

$$s_n = r_n - \sigma y \qquad (11.6.2)$$

3) If $t_n = p_0$, send s_n to $\overline{A}$ from the adder. If $t_n \neq p_0$, send r_n to $\overline{A}$ from A. Notice that $t_0 \neq p_0$.

4) If $t_n = y_0$, send 1 to $\overline{q}_{39}$ (the last bit of $\overline{Q}$). If $t_n \neq y_0$, send 0 to $\overline{q}_{39}$. The quotient bit thus formed is called q_n and

$$q_n = 2^{-1}[1 + (-1)^{y_0+t_n}] \tag{11.6.3}$$

5) Send $(\overline{A})$and $(\overline{Q})$ to AQ with a one place left shift. Now the new partial remainder in A is

$$r_{n+1} = 2r_n - \sigma [1 + (-1)^{p_0+t_n}]y \tag{11.6.4}$$

6) Set q_{39} to 1 and go to step 2.

Using equation (11.6.4) with n = 38, it follows that

$$2^{-39} r_{39} = r_0 - \sigma y[(1-2^{-39}) + 2^{-1}(-1)^{p_0} \sum_{0}^{38} (-1)^{t_n} \cdot 2^{-n}] \tag{11.6.5}$$

The quotient which is being formed a bit at a time is

$$q = -q_0 + \sum_{1}^{38} q_n \cdot 2^{-n} + 2^{-39} = -2q_0 + \sum_{0}^{38} q_n \cdot 2^{-n} + 2^{-39} \tag{11.6.6}$$

The quantity in Q is the quotient of r_0/y according to the formula

$$\frac{r_0}{y} = q + \frac{s_{39}}{y} \cdot 2^{-39} \tag{11.6.7}$$

This will be proved by showing that

$$(r_0 - qy)\, 2^{39} = s_{39} \tag{11.6.8}$$

It is true that

$$(r_0 - qy) \cdot 2^{39} = 2^{39} r_0 - 2^{39} y[-2q_0 + 2^{-39} + \sum_{0}^{38} 2^{-1}\left\{1 + (-1)^{y_0+t_n}\right\} 2^{-n}] \tag{11.6.9}$$

Since $t_0 \neq p_0$, then

$$q_0 = 2^{-1}[1 + (-1)^{y_0+t_0}] = 2^{-1}[1 - (-1)^{y_0+p_0}] = 2^{-1}(1 - \sigma) \tag{11.6.10}$$

Then using this in equation (11.6.9), it follows that

$$(r_0 - qy) \cdot 2^{39} = 2^{39} r_0 - 2^{39} y[\sigma + 2^{-1}(-1)^{y_0} \sum_{0}^{38} (-1)^{t_n} \cdot 2^{-n}] \tag{11.6.11}$$

or

$$(r_0 - qy) \cdot 2^{39} = 2^{39} r_0 - 2^{39} \sigma y[(1-2^{-39}) + 2^{-1}(-1)^{p_0} \sum_{0}^{38} (-1)^{t_n} \cdot 2^{-n}] - \sigma y \tag{11.6.12}$$

Hence,

$$(r_0 - qy) \cdot 2^{39} = r_{39} - \sigma y = s_{39} \tag{11.6.13}$$

11.7. The General Case of Division

The algorithm for the 66 division is replaced in the general case by a different one which is presented here. The quotient obtained is a Boolean function of the signs of the dividend and divisor, the v4 and v2 bits of the order variant and of the complement gate setting, U, and finally of the sign of the tentative partial remainder.

Let:

A denote the sign bit of the dividend.

B denote the sign bit of the divisor.

D denote the v4 bit.

E denote the v2 bit.

F denote the sign bit of the tentative partial remainder.

U denote the setting of the complement gate.

If U = 0, the gate is set to add, and
if U = 1, the gate is set to subtract.

α indicate whether the tentative partial remainder is sent from the adder $\overline{A}$ ($\alpha = 1$) or whether (A) is sent to $\overline{A}$ ($\alpha = 0$).

The setting of the complement gate is given by the equation

$$\overline{U} = \overline{A}BE + B\overline{D}E + A\overline{B}D + ADE \qquad (11.7.1)$$

Then q_{39} is set to a 1 or a 0 after the formation of a tentative partial remainder according to the following equation:

$$\overline{q}_{39} = \overline{B}\,\overline{F} + BF \qquad (11.7.2)$$

Finally, α is given by

$$\alpha = BE(ADUF + \overline{AD}\,\overline{U}\,\overline{F}) + \overline{E}\overline{B}(\overline{ADU}\overline{F} + AD\overline{U}F) \qquad (11.7.3)$$

It can be noticed that q_{39} is independent of v4 and v2.

CHAPTER 12

TAPE AND CARD PREPARATION

Most of the information that is to be transmitted to or from MISTIC is handled on either punched tape or punched cards. The purpose of this chapter is to outline the procedures for preparing cards and tape for MISTIC use and for printing results from the computer.

12.1. Cards

In order to process punched cards on MISTIC, an IBM 528 Reproducer has been installed. The 528 provides for either reading or punching cards under control of the computer and therefore any card-handling procedure is carried out by means of a program. Several subroutines have been written for handling cards and are part of the MISTIC Library.

No other card equipment is maintained by the Computer Laboratory; however, upon request, any services such as key-punching, tabulating, etc., will be handled by the Research Division of the Tabulating Department, located on the fifth floor of the Electrical Engineering Building.

Information regarding the use of punched cards is contained in Chapter 14. Card users should refer both to this chapter and to the card subroutines in the MISTIC Library. The latter are available upon request through the Computer Laboratory office.

Anyone contemplating use of cards in conjunction with MISTIC would be well advised to examine these programs and/or discuss the problem with someone on the MISTIC staff, as correct organization of the problem originally can provide much cheaper and faster computation in many cases.

12.2. Care in Handling Cards

When using punched cards, some care should be exercised in handling them. In particular, they should be kept dry to prevent swelling and the edges should be kept free from mutilation. To assist the Computer Laboratory staff in handling cards, they should be submitted in a suitable box, free of rubber bands, paper clips, etc.

12.3. Tape

A first step in using punched tape is to learn to "read" the tape. MISTIC equipment uses a "5 level" or "5 hole" tape; that is, there are five positions across the width of the tape which can be punched to represent symbols. In

addition to these five positions, there is a sixth and smaller position near the center of the tape which is always punched. This is a sprocket hole or feed hole and is used by the readers to advance the tape.

Each combination of the five holes that is punched determines a particular character. Figure 12. 1 shows the code used by the MISTIC. Of particular interest are the code combinations for the sexadecimal digits and for the format control characters.

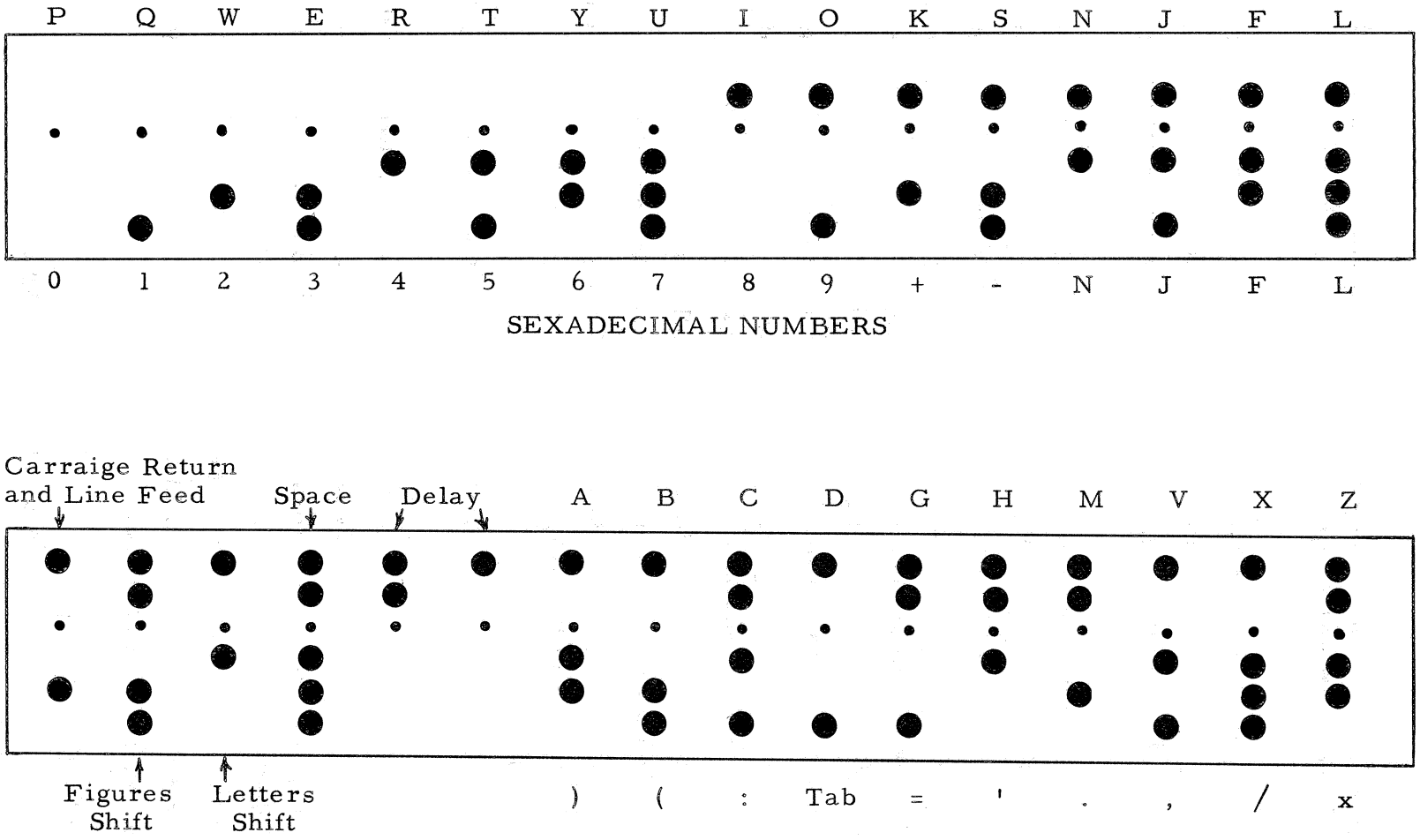
In normal operation, the computer will accept only characters (frames) which do not have the fifth hole punched. Only under control of 91 orders will "5th hole" characters be accepted.

For MISTIC purposes, it is useful to designate the holes on the tape as the "1", "2", "4", "8" and "5th" holes, reading from bottom to top, corresponding to 2^0, 2^1, 2^2 and 2^3 in the binary system, plus a "5th" hole. It is easy to "read" these numbers from tape since no numbers have the fifth hole punched, and one can look at the holes that are punched and add 1, 2, 4 and 8 accordingly. For example, note the code for the numbers in Figure 12. 1.

In addition to the numbers, the printer format control characters are frequently used, and these are grouped separately in Figure 12. 1. These characters then are the ones most commonly used; after some experience, they will be memorized. The other characters, letters and miscellaneous symbols are used much less often; if needed, they can be interpreted from Figure 12. 1.

Examination of Figure 12. 1 will show two symbols listed for each code character, except the printer format control characters. These correspond to the upper and lower case symbols on a typewriter. The symbol that will be printed is determined by whether the "letters shift" character or the "numbers shift" character was most recently "read" by the printer. Thus, the same piece of tape will print out either group of symbols depending upon which "shift" character precedes the tape. The computer, however, interprets all characters as though they were numbers. For example, if none of the holes on the tape are punched, the computer will interpret a zero; but the printer will print a 0 if preceded by a letters shift character. Similarly, if the "2" and "8" holes are punched, the printer will supply a + if a figures shift preceded the character, but the computer will interpret this as a K, or sexadecimal 10.

In using the computer, it is usually necessary to supply only numbers (characters without the "5th" hole punched) to the machine since it fails to recognize any other characters anyway, but it is convenient for certain man-imposed purposes to print out the results in a particular format. In making up tapes, then, characters are put on the tape which control the printer and which



Complete Tape Code
Figure 12.1

have no effect on the computer. These are known as the printer format control characters.

With this introduction, it is now possible to explain the use of the equipment available in the Tape Preparation Room and some of the mechanics of tape preparation.

The paper tape used for most of the work in the Computer Laboratory is an oil-impregnated paper. Consequently some care should be exercised in using the tape to keep it away from clothes and important papers which might be ruined by oil spots.

A second consequence of the oil is that it will pick up dirt very easily. This leads to two difficulties: (a) some holes on the tape may be at least partially covered, and (b) the dirt will be carried to a reader and deposited. The first difficulty is of consequence any time a photoelectric reader is being used, since even partial blocking of a hole may cause the reader to misinterpret this bit. The second difficulty affects any reader because dirt clogs up the reading holes of the photoelectric reader and some of the moving parts of the mechanical readers.

There are some general procedures in tape handling that should be followed, then, in order to insure error-free operation of the equipment. Tapes should be handled carefully with particular attention to the following items.

1) Keep tapes clean. Baskets have been provided and these should be used to keep the tape off the floor. Only one error in the tape caused by dirt is necessary to make radical changes in the program.

2) Be sure tapes are "free" when going through a reader so that they will not be torn. Any torn tapes should be reperforated and a good copy obtained to use on the computer.

3) Some corrections can be made on a tape by using a hand punch to punch holes which have been omitted. This is particularly valuable for adding the "2" hole when changing "20" and "24" orders to "22" and "26" after a program has been code checked. However, care should be taken that the holes are "clean"--that is, no fuzzy edges should appear where holes are punched.

4) A tape splicer is available in the Tape Preparation Room and may be used for editing and combining tapes.

12.4. Printing Format Control

Teletypewriters are used for printing on paper the symbols represented by the punched tape. The printers are set up so that they will print about 80 characters per line. Print-out format is controlled by tape characters also. Format control characters are space, carriage return and line feed, figures

(numbers) shift, letters shift, and two different delay characters. The printer performs the above functions when the proper tape code comes along regardless of whether a figures shift or a letters shift has preceded it. In addition, a tabulator feature is available, but the tab character must be preceded by a figures shift character.

Most of the above controls are self-explanatory. The space character simply moves the printer along one space without printing a character. Carriage return and line feed are accomplished by a single character, and this simply means returning the carriage to begin printing at the left-hand margin and feeding the paper up one line. Letters shift and figures shift characters cause the printer to print the upper and lower case symbols respectively, similar to an ordinary typewriter. Unlike the typewriter, however, the particular shift character used retains control until the opposite one comes along. So, for most usage, it is sufficient to have a figures shift at the beginning of the tape only. The two delay characters perform the same function; that is, they cause the printer to take a one-count rest. In particular, one delay character is usually needed following a carriage return-line feed character or tabulator character in order to give the carriage mechanism time to function before the next character to be printed comes along. Without it, some overprinting may occur on the left-hand side of the page, a character may be printed as the carriage returns, or columns will fail to line up properly. The tab feature causes the printer to move to the next tab stop. These stops are set at 10, 25, 40 and 55 characters from the left-hand side.

Several pieces of equipment are available for the various operations required in getting information into and out of the computer. The functions of these units are described below along with operating instructions.

12.5. Teletype Perforator

The first operation after writing a program is to translate it to punched tape. The keyboard perforators are for this purpose. With the unit plugged into a wall receptacle it is ready to use. An ON-OFF switch is located on the left side, and the unit is ON when the switch is in the "up" position. A roll of blank tape feeds through a punch block and out the left side of the perforator. As each character is punched, the tape advances one character space which allows this character to be seen just emerging from the punch block. If, at any time, a mistake is made in punching a character, the tape may be backspaced by using the lever located just above the ON-OFF switch. A space character (all five holes) is now punched, and then the correct character is punched. A fast feed (or repeat) button is located to the right of the keyboard. If a key is held down along with this button, the

character corresponding to that key will be repeated until the repeat button is released. Care should be taken to push the repeat button after the selected key has been held down and to release it before releasing the selected key. Otherwise, the perforator may fail to punch correctly.

When starting to punch a tape, it is usually well to punch a series of delay characters first to provide a leader at least six inches long. This will leave some room on the tape on which to write so that it may be labeled, and it also leaves some "handling" room. Then a carriage return-line feed character and an appropriate shift (usually figures shift) character should be punched. These will insure that the printer is correctly set up when the tape is printed out. After any other carriage return a delay character should be punched to insure time for the carriage to completely return. When the tape has been completed, four or five inches of 5th hole characters (delays or spaces will do) should be punched out for a tail.

12.6. Teletype Reperforator

The teletype reperforator is a standard piece of teletype equipment. It is used in conjunction with a transmitter-distributor (teletype language) or reader. These two form a unit which reads a tape and punches an exact duplicate. The reperforator will reproduce at the rate of six characters per second. Two major operations are thus made possible: (a) combining two or more tapes into one continuous tape (for ease in handling), and (b) editing.

The units are served by a common power supply, and this can be turned on by a switch located under the table near the left side. This switch should be turned ON first, and the reader ON-OFF switch (located on the front of the reader) should be OFF. If the tape is long enough to be rolled up, drop the coil of tape on a reel so that it will feed from the reel to the reader. On the reader is a hinged gate which latches to hold the tape down against the reading fingers. This can be lifted by raising the protruding lever at the front of the gate. The tape should now be placed under the tight-tape stop (the movable arm at the right side of the reader). Make sure the sprocket holes are lined up and laying down correctly over the sprocket wheel, hold the tape securely in this position, and close the "gate". Then turn the reader ON.

For editing purposes it is necessary to skip and insert other characters as the tape is read. The reader may be stopped by using the reader ON-OFF switch or by lifting the tight-tape stop. The tape can then be shifted or another tape inserted as required. The holes in the tape are sensed by five "fingers" which can be seen on the reader. The character which is directly over these fingers will be the next one read and punched by the reperforator. A tail of space characters can be punched

on the tape when it has been completed by pushing down on the lever which is available through a hole in the lid of the reperforator.

12.7. Printer

Once information has been punched on tape, it is desirable to find some painless way of reading this--ergo, printer. There are three printers available, one of which is portable and can be used for output directly from the computer. Otherwise, the printers along with a TD (transmitter-distributor), or reader, constitute a unit which will print out the contents of a tape. As with the teletype reperforator, a common power supply operates both the reader and the printer. The switch for this is located under the table and this should be switched ON first. The reader should be turned OFF, the tape placed on a reel and fed under the tight-tape stop onto the reader, taking care that the sprocket holes correctly fit down over the sprocket wheel. The reader can then be turned ON and the tape will be printed out.

Two buttons on the front of the printer will feed the paper and return the carriage independently of the reader. The printers also have an automatic carriage return and line feed which will operate when the carriage reaches the right-hand limit. Printing is at the rate of 10 characters per second.

12.8. Tape Comparer

When reproducing the tape, it is essential to know that the copy is exactly like the original. For short tapes, visual comparison, accomplished by holding the tapes in a superimposed position and examining against a light background, is perhaps the fastest and easiest. For long tapes, the comparer is better. This unit has two high-speed photoelectric readers which are connected to comparison circuits. The tapes to be compared are placed in the two readers and the comparer then checks one against the other. As long as the tapes agree, the comparer operates at 300 characters per second. When an error is detected, the unit stops and the "compared" characters are displayed on the control panel. The tape can then be corrected or marked appropriately and the comparison continued. The readers can be set for one of three modes of operation: (a) skip no characters, (b) skip all 5th hole characters, and (c) skip all space characters.

This unit has power supplied to it always to avoid a delay while waiting for it to warm up. After pushing the SET button and selecting the mode of operation, the tapes to be compared are placed in the appropriate readers and started. Another switch is available to permit advancement of the tape one character at a time when such an operation is desirable.

12.9. High-speed Reperforator

The high-speed reperforator performs the same function as the teletype reperforator but is much faster and consequently of particular value in preparing long tapes. The unit consists of two photoelectric readers and a punch. Operation at full speed is 60 characters per second. Since there are two readers, it is possible to handle two different tapes, alternately selecting the tape which is to be copied. The neon lights on the control panel indicate the last character read from tape, whether punched or bypassed. Four switches and a push button are used to control the reperforator. Their use is outlined below.

STOP-RUN Switch. This is a two-position switch which determines whether the unit is stopped or running. It should be in the STOP position before changing the position of any other switch except the START-OBEY-IGNORE switch.

FRONT-REAR-CRLF Switch. This is a three-position switch, and with it the following operations are enabled.

a) Read the tape in the front reader.

b) Read the tape in the rear reader.

c) Punch carriage return-line feed characters without operating the readers.

MODE CONTROL Switch. This four-position switch works in conjunction with the START-OBEY-IGNORE switch to determine the way to read a tape, as follows:

a) SINGLE CHARACTER PUNCH--Punch one character and stop.

b) STOP PUNCH ON CRLF--Punch up to and including the next carriage return-line feed character, and stop.

c) SINGLE BYPASS--Skip one character and stop.

d) STOP BYPASSING ON CRLF--Skip all characters up to and including next carriage return-line feed character, then stop.

START-OBEY-IGNORE Switch. This is a three-position switch, with the START position being a momentary contact; that is, the switch will return itself to the OBEY position. If the switch is moved to START after a stop, the instruction of the Mode Control switch will be obeyed once and another stop will occur. If it is set to IGNORE, it will cause all of the stops specified by the Mode Control switch to be ignored.

CHAPTER 13

CALCULATION OF RUNNING TIME

There are several good reasons for knowing how long it will take MISTIC to perform a particular computation. First of all, when using a computer which costs up to a few hundred dollars per hour to operate, it is necessary that it be scheduled efficiently and this requires knowledge of the time required for each problem. Secondly, there must be some idea of how long to let the machine run before expecting an answer, since mistakes either in the program or in the machine could allow the computer to "loop" indefinitely. Again, when using the card reader-punch, it is necessary to limit computation time between card rows in order to prevent a hang-up.

13.1. Estimating Time

There are several ways to estimate the required time. All library routines indicate the amount of time required so that the estimate for a program using library routines exclusively can be determined by these formulas.

When running the same program several times with only changes in data, the time required for one run serves as a good estimate for further runs. Of course, this does not solve the time problem for the first run, but it will prove to be a very good estimate for succeeding runs.

Another way to estimate time is to note the time required for code checking and use this as a guide to determine time requirements for production runs.

Finally, a simple formula for estimating running time is given by

$$T = \frac{N_o}{16} + \frac{N_s}{40} + \frac{N_m + N_d}{1.5} \text{ milliseconds}$$

where N_o = number of orders obeyed
N_m = number of multiplications obeyed
N_d = number of divisions obeyed
N_s = number of shifts obeyed

This formula allows for running time only, so input and output time must be added to this. Input and output time will be determined by the routines used, and it can be calculated from formulas given by these routines, or a rough estimate can be made using the times indicated in Table 13.1.

Some additional information regarding time for the card reader-punch is necessary. The reader operates at a maximum speed of 200 cards per minute and

the punch at 100 cards per minute. The maximum speeds may be reduced considerably by particular input and output routines, and reference must be made to these routines for more specific information.

When programming for card operations, there are some particular time conditions which must be met. Once a card cycle has been initiated, the entire card must be processed. It requires twelve 94 orders to read a card or twelve 96 orders to punch a card, one for each row. Between cards, any amount of computation time is permissible, but between rows the computation time is limited to 8.7 milliseconds when reading and 17.4 milliseconds when punching. If inter-row computation is not completed in this time, the card row will have passed the read or punch station. To prevent errors when this might happen without the user's knowledge, MISTIC will hang up on the 94 or 96 order which should have operated on the bypassed row. A light on the card reader console indicates this condition.

ORDER TYPE	EXECUTION TIME		
0n, 1n	24n	microseconds	
2, 3, 4, 5, J	55	microseconds	
3, not executed	18	microseconds	
6	1,200	microseconds	
7	1,000	microseconds	
80n, 81n, 91n (n is a multiple of 4)	1,000n	microseconds	
82n	4,200n	microseconds	by punch
	25,000n	microseconds	by printer
92	17,000	microseconds	per character by punch
	100,000	microseconds	per character by printer
K, S, F, L	100	microseconds	
N	30	microseconds	

Table 13.1

13.2. Calculating Time

There are times when it is desirable to have a better estimate than is given by the methods of the previous section as, for example, when making up the formulas for a library program. Table 13.1 gives the approximate amount of time necessary for each of the orders.

In most cases, it will probably be found that an estimate is not very close to the actual time; a factor of 2:1 may even indicate a "good" estimate. Even the use of Table 13.1 will not permit a perfect estimate, since some of the times listed are only approximate. For example, a multiplication may take as little as 900 microseconds, the exact time depending upon the multiplier. However, it is in the best interests of the programmer to have an estimate as accurate as possible, and the methods outlined will lead to usable results.

CHAPTER 14

THE USE OF CARDS AND CARD EQUIPMENT FOR MISTIC

The cards acceptable to the MISTIC for input and output are what are known as IBM 80 column cards. The means of inputting and outputting the cards is an IBM 528 Accumulating Reproducer. The use of card input and output does not preclude the use of paper tape input or output, and in fact, both methods of transmission are frequently used in certain problems.

14.1. The Cards

An eighty column card is illustrated in Figure 14.1.

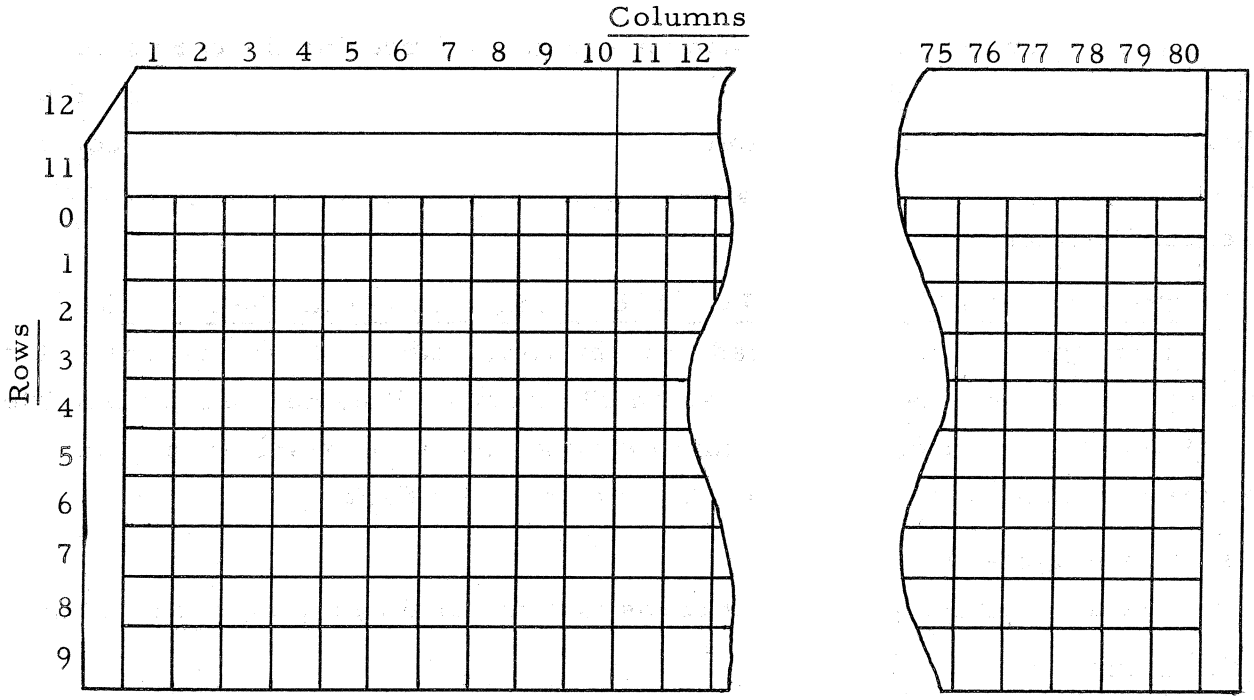


Figure 14.1

The digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 are indicated by a perforation in the corresponding row. For example, if the first three columns of a card should denote the number 407, then in column 1, row 4 should be perforated; in column 2, row 0 should be perforated, and in column 3, row 7 should be perforated. For the uses of the 11 and 12 rows, one should consult the specific library programs.

One can observe that an eighty column card could represent an eighty digit number. Since there is seldom need for such a number, the card is divided into allotted sets of columns, called fields. Thus, if one wished to use ten digit numbers, eight such numbers would fit on a card, and the card would then be said

to have eight 10-column fields. Of course, the length of the various fields on a card need not be constant. Thus, on one card there could be fields of length 2, 7, 9, 8, 4 and 3, for example. Such a card is said to possess variable field length.

A set of cards, all bearing data related to the same problem, is called a deck. Two possibilities concerning field structure throughout a deck prevail.

1.) It is possible that the exact field structure of one card is the same as the field structure of all the other cards in the deck. Such a situation is referred to as a deck with fixed format.

2.) It is possible that the field structure of some cards in the deck is not the same as the field structure of other cards in the deck. This situation is called a deck with variable format.

14.2. Card Preparation

The Computer Laboratory does not in general provide the card stock for individual users. The only piece of card preparation equipment maintained by the Laboratory is a hand punch, suitable for small scale use. In order to acquire card stock and to prepare cards, the user may contact the Tabulating Department, Research Division.

14.3. The Operation of the IBM Type 528 in Conjunction with the MISTIC

For input, each card is read, row by row, with the 12 row normally being read first. Each row is translated as an 80 binary digit number and transmitted to the AQ register. For output, 80 binary digit numbers are transferred from the AQ register to a card, a row at a time, 12 row normally first.

14.4. The Plugboard

The reading and punching of cards is controlled by a plugboard and the card instructions.

The plugboard is a device which may be wired in various patterns, so as to determine which bit position of AQ will correspond to a particular column on the card. The pertinent features of a plugboard are shown in Figure 14.2. Each row has holes in it, which are not drawn in. Thus, there are 40 holes in the PUNCH DIRECT rows, 8 holes in the short PUNCH rows, and 20 in the longer PUNCH rows.

The holes in the PUNCH rows, and in the READ rows, refer to the bit positions of A and Q; thus, the first hole in the PUNCH row refers to a_0, the next to a_1, the fortieth to a_{39}, the forty-first to q_0, and so on similarly for READ. The holes in PUNCH DIRECT refer to the columns of a card to be punched. Thus, the first hole in the PUNCH DIRECT refers to column 1 of the card, the second hole to column 2, etc. By connecting a PUNCH hole (by means of a wire) to a PUNCH DIRECT hole, one establishes that a given bit position of AQ is transferred to a

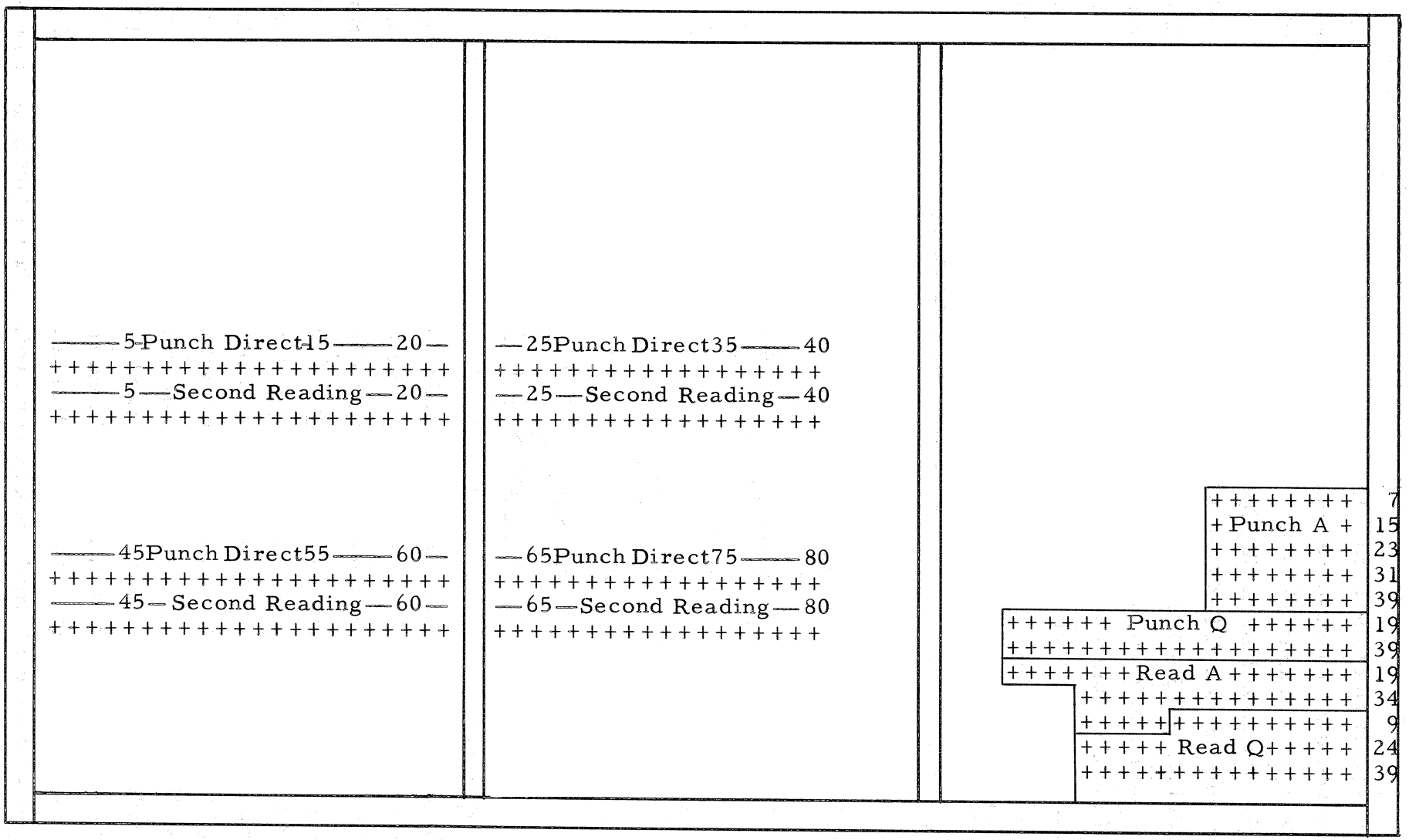

5 Punch Direct 15 20
5 Second Reading 20
25 Punch Direct 35 40
25 Second Reading 40
45 Punch Direct 55 60
45 Second Reading 60
65 Punch Direct 75 80
65 Second Reading 80
Punch A
Punch Q
Read A
Read Q
7
15
23
31
39
19
39
19
34
9
24
39

Figure 14.2

given column position of the card. Reading is handled in the same way.

For example, suppose we wish to read a card so that column 1 goes to a_7, column 2 to a_8 and column 13 to a_0. We would then wire SECOND READING and READ so that the first hole of SECOND READING is connected to the eighth hole in READ, the second hole of SECOND READING is connected to the ninth hole of READ, and the 13th hole of SECOND READING is connected to the first hole of READ.

If a user intends to use card equipment, it is well for him to consult with a member of the Computer Laboratory staff concerning his card format. It may be that his format can be adapted to a certain fixed format in use in the Laboratory, thus bypassing the problems of obtaining and wiring a second plugboard. The fixed format plugboard makes the following assignments:

Bit Position	to	Card Column
a_0		1
a_1		2
a_2		3
a_3		4
$a_4 - a_{39}$		9 - 44
q_0		5
q_1		6
q_2		7
q_3		8
$q_4 - q_{39}$		45 - 80

Table 14.1

14.5. Card Orders

The card input order is 94 with an address of zero. The output order is 96 with an address of zero. There are separate card feed hoppers for input and output. There is a card read station, capable of reading one row, and a card punch station, capable of punching one row. When a 94F is issued, if no card is in the read station, one card advances to the read station from the hopper, and the first row (normally the 12 row) is read; the card continues to advance. If another 94F order is given before the 11 row enters the read station, the 11 row will be read. Otherwise, the computer will stop and the program will have to be corrected. Thus, in order to read the contents of one card, twelve 94F orders must be given. Similar statements apply to output and 96F orders.

The cards are input to the read station at the rate of 200 cpm (cards per minute), which means 300 milliseconds per card. After a 94F order has been executed and if the card has not been entirely read, the next 94F must be executed

within about 8.7 milliseconds to avoid the computer stop mentioned above. When a card has been entirely read, the next 94F order can be given at any time with no hang-up occurring. However, to achieve the maximum read-in of 200 cards per minute, the next 94F order after the completion of a card read must occur within 17.4 milliseconds. Otherwise the next card can not advance until 300 milliseconds have elapsed since the previous card was read. This last option is the one ordinarily used, so that the read-in rate generally is only 100 cpm.

The 96F order obeys the same rules as above, except that all times for 96F orders are twice those of the 94F orders.

14.6. Card Programs in the Library

At present there are programs (N1M, N3M) for inputting variable format decks and fixed format decks. There is an output program (P3M) for fixed format decks. There is a program (NP1M) for converting card data to tape for use with K2 and K9. Many other versions of input and output can be constructed around these programs. Finally, K2 and K9 have been converted to accept direct card input.

INDEX