Technical Manual

FUNDAMENTALS OF MINI-COMPUTER HARDWARE

015-000022-02

DATA GENERAL TECHNICAL MANUAL

FUNDAMENTALS OF MINI-COMPUTER HARDWARE

Ordering No. 015-000022

Printed in the United States of America
Rev. 02, January 1974

NOTICE

This publication was prepared by the Data General Corporation Education Department. It is intended to supplement the material presented in Data General's Fundamentals of Mini-Computer Hardware Course. PJS 2/74

Any comments concerning this publication should be forwarded to:

Education Department
Data General Corporation
Route 9
Southboro, MA. 01772

TABLE OF CONTENTS

SECTION I

NUMBER SYSTEMS

SECTION II

ARITHMETIC OPERATIONS

SECTION III

LOGIC AND LOGIC CONVENTIONS

SECTION III (Continued)

LOGIC AND LOGIC CONVENTIONS

SECTION IV

COMPUTER ORGANIZATION

SECTION V

INSTRUCTION SET - MEMORY REFERENCE

TABLE OF CONTENTS (Continued)

SECTION V (Continued)

INSTRUCTION SET - MEMORY REFERENCE

SECTION VI

ARITHMETIC LOGICAL INSTRUCTIONS (ALC)

SECTION VII

INPUT-OUTPUT (IO) INSTRUCTION

SECTION VIII

PROGRAM INTERRUPT AND DATA CHANNEL

SECTION IX

CONSOLE

SECTION X

BASIC PROGRAMMING AND PROGRAMS

LIST OF ILLUSTRATIONS

This page intentionally left blank

SECTION I

NUMBER SYSTEMS

SYSTEM BASES

Radix

To introduce the concept of number systems, look first at that number system most familiar to us, the decimal number system. The decimal number system contains ten (10) different symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), thus it is base 10, or radix 10.

In this same manner, it is possible to define a multitude of number systems by simply specifying a new base or radix. For examples, a number system with the base 5 would contain 5 symbols (0, 1, 2, 3, 4). A radix 16 number system would contain 16 symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). It is insignificant that alphabetical characters are used as the additional symbols. The significance of the symbols is their relative position in the complete set. It is the position which determines the actual value of the symbol, not the symbol itself.

It should be evident, from the following table, how counting would proceed in various number systems.

RADIX: 10	2	3	4	8	16
0	0	0	0	0	0
1	1	1	1	1	1
2	10	2	2	2	2
3	11	10	3	3	3
4	100	11	10	4	4
5	101	12	11	5	5
6	110	20	12	6	6
7	111	21	13	7	7
8	1000	22	20	10	8
9	1001	100	21	11	9
10	1010	101	22	12	A
11	1011	102	23	13	B
12	1100	110	30	14	C
13	1101	111	31	15	D
14	1110	112	32	16	E
15	1111	120	33	17	F
16	10000	121	100	20	10
17	10001	122	101	21	11
18	10010	200	102	22	12
19	10011	201	103	23	13
20	10100	202	110	24	14
21	10101	210	111	25	15
22	10110	211	112	26	16
23	10111	212	113	27	17
24	11000	220	120	30	18
25	11001	221	121	31	19
26	11010	222	122	32	1A
27	11011	1000	123	33	1B
28	11100	1001	130	34	1C
29	11101	1002	131	35	1D
30	11110	1010	132	36	1E
31	11111	1011	133	37	1F
32	100000	1012	200	40	20
33	100001	1020	201	41	21
34	100010	1021	202	42	22
35	100011	1022	203	43	23
36	100100	1100	210	44	24
37	100101	1101	211	45	25
38	100110	1102	212	46	26
39	100111	1110	213	47	27
40	101000	1111	220	50	28

The radix 10 number is more generally known as the decimal number system. Similar nomenclature is used for other number systems.

Radix	Nomenclature
10	Decimal
2	Binary
8	Octal
16	Hexadecimal

The binary number system (radix 2) contains 2 symbols (0, 1). This has special significance since most physical devices have 2 states, and each state can be represented by one of the binary symbols. For example, a switch has 2 states: OFF (0) and ON (1); a door has 2 states: CLOSED (0) and OPEN (1); numbered areas on a computer card have 2 states: NO HOLE (0) and HOLE (1); small areas on magnetic tape have 2 states: NOT MAGNETIZED (0) and MAGNETIZED (1). It is for this reason that the binary number is so widely used in computers.

In order to indicate which number system is being used, the radix is subscripted at the end of the number. For example, the decimal number 179.65 would be written as 179.65_{10}; the octal number 734.26, as 734.26_8; and the binary number 1011.01101 would be written as 1011.01101_2.

RADIX POINT

It has been shown how integer numbers (whole numbers, nonfractional numbers, etc.) are represented in various number systems. In the decimal number system, the fractional portion of a number is separated from the integer portion by a decimal point (ex: 123.24). This form of separation is used in all the number systems. In general, the point is called the radix point. In the decimal number system, it is referred to as the decimal point; in the binary number system as the binary point; and in the octal number system as the octal point.

An example of fractional numbers in different number systems is shown:

Radix	Example
2	10001101.01001101
3	10200121.21102
8	7536.0214
10	8930.6345

NUMBER SYSTEM CONVERSIONS

Binary to Decimal

Consider the number 214_{10}. This notation is shorthand for

$$\begin{aligned} 4x1 &= 4 \\ 1x10 &= 10 \\ 2x100 &= 200 \\ &\overline{214} \end{aligned}$$

In the decimal number system, this seems redundant, but it demonstrates the technique for converting a number of any radix to its decimal equivalent. In general, every number will have the following format:

$$\ldots\ldots a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3} \ldots\ldots_R$$

where a_x is the value of the digit, the subscript of a is the digit's position relative to the radix point, and R is the radix of the number. The following formula is used to convert any number to its decimal equivalent:

$$\ldots\ldots +(a_4 x R^4)+(a_3 x R^3)+a_2 x R^2)+(a_1 x R^1)+(a_0 x R^0)+$$
$$(a_{-1} x R^{-1})+(a_{-2} x R^{-2})+(a_{-3} x R^{-3})+\ldots\ldots =N_{10}$$

Example 1.1

Find N where $214.36_{10}=N_{10}$

$$\begin{aligned} 4x10^0 &= 4x1 &&= 4 \\ 1x10^1 &= 1x10 &&= 10 \\ 2x10^2 &= 2x100 &&= 200 \\ 3x10^{-1} &= 3x.1 &&= .3 \\ 6x10^{-2} &= 6x.01 &&= .06 \\ & & & \overline{214.36} \end{aligned}$$

Example 1.2

Find N where $1011.01101_2=N_{10}$

$$\begin{aligned} 1x2^0 &= 1x1 &&= 1 \\ 1x2^1 &= 1x2 &&= 2 \\ 0x2^2 &= 0 \\ 1x2^3 &= 1x8 &&= 8 \\ 0x2^{-1} &= 0 \\ 1x2^{-2} &= 1x.25 &&= .25 \\ 1x2^{-3} &= 1x.125 &&= .125 \\ 0x2^{-4} &= 0 \\ 1x2^{-5} &= 1x.03135 &&= .03125 \\ & & & \overline{11.40625} \end{aligned}$$

$$1011.01101_2 = 11.40625_{10}$$

Octal to Decimal

A similar procedure is used to convert octal numbers to decimal numbers.

Example 1.3

Find N where $1735.24_8=N_{10}$

$$\begin{aligned} 5x8^0 &= 5x1 &&= 5 \\ 3x8^1 &= 3x8 &&= 24 \\ 7x8^2 &= 7x64 &&= 448 \\ 1x8^3 &= 1x512 &&= 512 \\ 2x8^{-1} &= 2x.125 &&= .25 \\ 4x8^{-2} &= 4x.015625 &&= .0625 \\ & & & \overline{989.3125} \end{aligned}$$

$$1735.24_8 = 989.3125_{10}$$

Decimal to Binary

When converting a decimal number to a number in some other base, it is necessary to consider the integer portion (portion to the left of the decimal point) and the fractional portion (portion to the right of the decimal point) separately.

The procedure for converting the integer portion of a decimal number to some other radix R is as follows:

1. Divide the integer by R, and separate the answer into a quotient and a remainder.
2. Record the remainder.
3. Divide the quotient by R, and separate the answer into another quotient and a remainder.
4. Repeat steps 2 and 3 until a quotient of 0 is obtained.
5. Record the remainders in reverse order of their occurrence to form the converted number.

Example 1.4

Find N where $17_{10}=N_2$

Quotient	Remainder	
2 / 17		
2 / 8	1	LSB
2 / 4	0	
2 / 2	0	
2 / 1	0	
0	1	MSB

$17_{10}=10001_2$

Example 1.5

Find N where $39_{10}=N_2$

Quotient	Remainder	
2 / 39		
2 / 19	1	LSB
2 / 9	1	
2 / 4	1	
2 / 2	0	
2 / 1	0	
0	1	MSB

$39_{10}=100111_2$

The procedure for converting the fractional portion of a decimal number to some other radix R is as follows:

1. Multiply the fraction by R, and separate the answer into an integer and a fraction.
2. Record the integer.
3. Multiply the fraction by R, and separate the answer into another integer and a fraction.
4. Repeat steps 2 and 3 until a fractional answer of 0 is obtained.
5. Record the integer portion of the answers in the order of their occurrence to form the converted fraction.

Example 1.6

Find N where $.125_{10}=N_2$

Integer		Fraction
.125x2=0	+	.250
.250x2=0	+	.500
.500x2=1	+	.000

$.125_{10}=.001_2$

Example 1.7

Find N where $.6_{10}=N_2$

Integer		Fraction
.6x2=1	+	.2
.2x2=0	+	.4
.4x2=0	+	.8
.8x2=1	+	.6
.6x2=1	+	.2

$.6_{10}=.\overline{1001}_2$

The bar over a portion of a number indicates that that portion of the number is repetitive. In the above example, the group 1001 will repeat indefinitely as

.10011001100110011001.....

Example 1.8

Find N where $.425_{10}=N_2$

Integer		Fraction
.425x2=0	+	.85
.85 x2=1	+	.7
.7 x2=1	+	.4
.4 x2=0	+	.8
.8 x2=1	+	.6
.6 x2=1	+	.2
.2 x2=1	+	.4

$.425_{10}=.0\overline{1101}_2$

Example 1.9

Find N where $9.75_{10}=N_2$

2 / 9			.75x2=1 | +.5
2 / 4	+	1	.5 x2=1 | +.0
2 / 2	+	0	
2 / 1	+	0	
0	+	1	

$9.75_{10}=1001.11_2$

Decimal to Octal

The conversion of decimal numbers to octal numbers uses the same procedure.

Example 1.10

Find N where $176_{10}=N_8$

Quotient		Remainder
$8\overline{)176}$		
$8\overline{)22}$	+	0
$8\overline{)2}$	+	6 ↑
0	+	2 ↑

$176_{10}=260_8$

Example 1.11

Find N where $.3_{10}=N_8$

	Integer		Fraction
.3x8=	2 ↓	+	.4
.4x8=	3 ↓	+	.2
.2x8=	1 ↓	+	.6
.6x8=	4	+	.8
.8x8=	6	+	.4

$.3_{10} = .2\overline{31468}$

Example 1.12

Find N where $253.15_{10}=N_8$

$8\overline{)253}$			.15x8=1 ↓	+.2
$8\overline{)31}$	+	5	.2 x8=1 ↓	+.6
$8\overline{)3}$	+	7 ↑	.6 x8=4	+.8
$\overline{)0}$	+	3 ↑	.8 x8=6	+.4
			.4 x8=3	+.2

$253.15_{10} = 375.1\overline{14638}$

Binary to Octal

The octal number system has been employed, in most computer circles, as the standard number system. It is far less tedious to perform conversions between decimal and octal systems than between decimal and binary systems, and it is relatively simple to perform conversions between the octal and binary systems.

The relationship between the octal (radix 8) and binary (radix 2) number systems is the key to the conversion simplicity.

$$8=2^3$$

where the magic number is 3.

The procedure for converting binary numbers to octal numbers is as follows:

1. Starting at the binary point, proceed to the left and partition the number into groups of 3 bits (binary digits).
2. Starting at the binary point, proceed to the right and partition the number into groups of 3 bits.
3. Convert each group of 3 bits to its octal equivalent.

Example 1.13

Find N where $1011010.1011101_2=N_8$

001	011	010	.	101	110	100
1	3	2	.	5	6	4

$1011010.1011101_2=132.564_8$

NOTE: Zeroes should be added at the beginning and at the end of a number to complete a group of 3 bits.

Example 1.14

Find N where $1111000.0001111_2=N_8$

001	111	000	.	000	111	100
1	7	0	.	0	7	4

$1111000.0001111_2=170.074_8$

Octal to Binary

The procedure for converting octal numbers to their binary equivalent is equally simple. Just reverse the binary to octal conversion procedure.

To convert octal numbers to binary, perform the following:

1. Starting at the octal point, proceed to the left and convert each number to its 3 bit binary equivalent.
2. Starting at the octal point, proceed to the right and convert each number to its 3 bit binary equivalent.

Example 1.15

Find N where $173.405_8=N_2$

1	7	3	.	4	0	5
↓	↓	↓	↓	↓	↓	↓
001	111	011	.	100	000	101

$173.405_8=1111011.100000101_2$

NOTE: Leading and trailing zeroes may be dropped.

Example 1.16

Find N where $37526.1024_8=N_2$

3	7	5	2	6	.	1	0	2	4
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
011	111	101	010	110	.	001	000	010	100

$37526.1024_8 = 11111101010110.0010000101_2$

This page intentionally left blank

SECTION II

ARITHMETIC OPERATIONS

ADDITION

Binary Addition

The addition of binary numbers follows the same procedure as the more familiar addition of decimal numbers.

To add two decimal numbers, proceed as follows:

1. Add the rightmost digit of each number to obtain a sum digit and a carry digit.
2. Record the sum digit.
3. Add the next rightmost digit of each number, plus the carry digit left from the previous addition, and obtain another sum digit and carry digit.
4. Repeat steps 2 and 3, proceeding from right to left, until all the digits have been added.
5. The number constructed from the individual sum digits is the final sum.

Example 2. 1

Add the two decimal numbers 566+624.

```
                 566
                 624
6+4=10           ---     where Carry=1
                         Sum=0

                   1
                 566
                 624
1+6+2=9          ---     where Carry=0
                         Sum=9

                  01
                 566
                 624
0+5+6=11         ---     where Carry=1
                         Sum=1

                 101
                0566
                0624
1+0+0=1         ----     where Carry=0
                         Sum=1

                 101
                0566
                0624
               ----
               1190

Thus 566+625=1190
```

Binary addition follows exactly the same five steps used in decimal addition. But, it must be remembered that the binary number system has only two digits (0 and 1). The following table examines the addition of all possible operands resulting from the addition of two binary numbers:

Carry Prop.		Bit A		Bit B		Carry Gen.	Sum
0	+	0	+	0	=	0	0
0	+	0	+	1	=	0	1
0	+	1	+	0	=	0	1
0	+	1	+	1	=	1	0
1	+	0	+	0	=	0	1
1	+	0	+	1	=	1	0
1	+	1	+	0	=	1	0
1	+	1	+	1	=	1	1

Example 2. 2

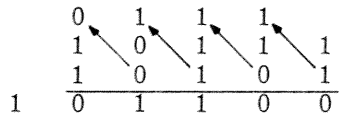
Add the two binary numbers 10111+10101.

```
                10111
                10101
1+1=10          -----   where Carry=1
                        Sum=0

                  1
                10111
                10101
1+1+0=10        -----   where Carry=1
                        Sum=0

                 11
                10111
                10101
1+1+1=11        -----   where Carry=1
                        Sum=1

                111
                10111
                10101
1+0+0=01        -----   where Carry=0
                        Sum=1

               0111
               10111
               10101
0+1+1=10       -----    where Carry=1
                        Sum=0

               10111
              010111
              010101
1+0+0=01      ------    where Carry=0
                        Sum=1

Thus 10111+10101=101100
```

The following example shows a shorthand method of keeping track of the sum and carry digits.

Example 2.3

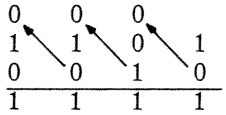
Repeat the Example 2.2 using the shorthand technique.



Thus 10111 + 10101 = 101100.

Example 2.4

Add the two binary numbers 1101 + 10.



Thus 1101 + 10 = 1111.

If the number of digits in the answer exceeds the maximum allowable number of digits, the answer is said to overflow, and the leftmost digit of the answer is called the overflow digit.

If, in Example 2.3, the maximum allowable number of digits is five, then there is an overflow, and the overflow digit is 1. In Example 2.4, there is no overflow, so the overflow digit is 0.

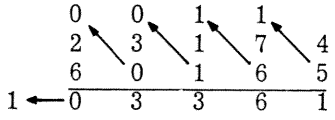
Octal Addition

The procedure for performing octal addition is similar to that used for decimal and binary addition. The fact to keep in mind is that the octal number system has 8 digits (0 thru 7), and a carry occurs when the sum exceeds 7.

In the following examples, assume the maximum allowable number of digits is 5.

Example 2.5

Add the two octal numbers 23174 + 60165.

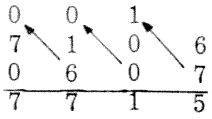


Thus 23174 + 60165 = 103361.

NOTE: In this example, an overflow occurred.

Example 2.6

Add the two octal numbers 7106 + 607.



Thus 7106 + 607 = 7715.

NOTE: In this example, no overflow occurred since the sum did not exceed the maximum allowable 5 digits.

SUBTRACTION

Complementary Arithmetic

In the previous section, the concept of "maximum allowable number of digits" was introduced. This concept is of great importance in the understanding of complementary arithmetic.

If the maximum allowable number of digits is 6, for example, then the decimal numbers

$$\begin{array}{rcccccc} & 9 & 8 & 6 & 3 & 2 & 7 \\ \text{and } 1 & 9 & 8 & 6 & 3 & 2 & 7 \end{array}$$

represent the same magnitude since the leftmost digit is an indication of overflow, and adds nothing to the value of the rightmost, maximum allowable 6 digits.

The normal counting sequence from zero is as follows:

$$\begin{array}{rcccccc} & 0 & 0 & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & 0 & 1 \\ & 0 & 0 & 0 & 0 & 0 & 2 \\ & 0 & 0 & 0 & 0 & 0 & 3 \\ & & & \vdots & & & \\ & 9 & 9 & 9 & 9 & 9 & 7 \\ & 9 & 9 & 9 & 9 & 9 & 8 \\ & 9 & 9 & 9 & 9 & 9 & 9 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

It should be evident, from the previous table, that if 1 is added to the largest number, 999999, zero is obtained and the normal counting sequence is recycled.

What happens if the counting sequence is reversed?

```
      .
      .
      .
0  0  0  0  0  3
0  0  0  0  0  2
0  0  0  0  0  1
0  0  0  0  0  0
9  9  9  9  9  9
9  9  9  9  9  8
9  9  9  9  9  7
      .
      .
      .
```

From this table, it can be seen that when 1 is subtracted from zero, the number simply cycles back through 999999, etc. But, 1 subtracted from zero is -1. To restate this, disregarding + and -, we say that, 999999 and 000001 are complements. Likewise, 000002 and 999998 are complements; 000003 and 999997 are complements; etc. Note that every number has a complement except 000000. The complement of 000000 is 000000.

To obtain the complement of a number, it is not necessary to count forwards and backwards from 000000. Simply subtract the number from (the largest possible number +1).

For the 6 digit maximum numbers used here, the largest possible number is 999999, and the largest possible number +1 is 1000000.

Example 2.7

Find the complement of 000004.

```
  1  0  0  0  0  0  0   (largest possible number +1)
- 0  0  0  0  0  4     (minus the number)
  9  9  9  9  9  6     (complement of the number)
```

Example 2.8

Find the complement of 923156.

```
  1  0  0  0  0  0  0
- 9  2  3  1  5  6     (number)
  0  7  6  8  4  4     (complement)
```

Example 2.9

```
  1  0  0  0  0  0  0
- 0  0  0  0  0  0  0
1← 0  0  0  0  0  0
```

The complementary numbers obtained in Examples 2.7, 2.8 and 2.9 are more correctly referred to as the 10's (ten's) complement of the number (999996 is the 10's complement of 000004, etc.). This further restriction of the complement is used to indicate the operand from which the number was subtracted to obtain the complement. In Examples 2.7, 2.8 and 2.9, this operand is the next power of the base (in this case 10); thus the complement obtained is the 10's complement.

It is interesting to note that the original number was subtracted from the largest possible number +1 in order to obtain the 10's complement. The same result could be obtained if the number is subtracted from the largest possible number, and 1 added to the answer.

Example 2.10

Find the 10's complement of 923156.

```
  9  9  9  9  9  9   (largest possible number)
- 9  2  3  1  5  6   (number)
  0  7  6  8  4  3
              +  1   (plus 1)
  0  7  6  8  4  4   (10's complement)
```

NOTE: 076843 is known as the 9's complement of 923156.

Therefore, an easier method of finding the 10's complement of a number is as follows:

10's complement of X = 9's complement of X, plus 1

Example 2.11

Find the 10's complement of 000000.

```
  9  9  9  9  9  9   (largest possible number)
- 0  0  0  0  0  0   (X)
  9  9  9  9  9  9   (9's complement of X)
              +  1   (plus 1)
1←0  0  0  0  0  0   (10's complement of X)
```

Now let's apply the general rules of complementation to the binary number system. In the binary number system, the complement desired is the 2's complement of the number.

In the following examples, assume that the maximum allowable number of bits (binary digits) is 7.

Example 2.12

Find the 2's complement of 0000011.

```
  1 0 0 0 0 0 0 0  (largest possible number* +1)
- 0 0 0 0 0 1 1    (X)
  1 1 1 1 1 0 1    (2's complement of X**)
```

*The largest possible number is 1111111.
**Direct binary subtraction follows the same rules as direct decimal subtraction:

0-0=0;
1-0=1;
1-1=0;
0-1=1 and borrow 1.

But, as was shown before, it is possible to subtract the number from the largest possible number and add 1 to the result.

Example 2.13

Find the 2's complement of 0000011.

```
  1 1 1 1 1 1 1  (largest possible number)
- 0 0 0 0 0 1 1  (X)
  1 1 1 1 1 0 0
            + 1  (plus 1)
  1 1 1 1 1 0 1  (2's complement of X)
```

NOTE: 1111100 is known as the 1's complement of 0000011.

Therefore, the 2's complement of a binary number may be obtained as follows:

2's complement of X = 1's complement of X, plus 1.

Example 2.14

Find the 2's complement of 1011101.

```
  1 1 1 1 1 1 1  (largest possible number)
- 1 0 1 1 1 0 1  (X)
  0 1 0 0 0 1 0  (1's complement of X)
            + 1  (plus 1)
  0 1 0 0 0 1 1  (2's complement of X)
```

Example 2.15

Find the 2's complement of 000000.

```
  1 1 1 1 1 1 1  (largest possible number)
- 0 0 0 0 0 0 0  (X)
  1 1 1 1 1 1 1  (1's complement of X)
            + 1  (plus 1)
1←0 0 0 0 0 0 0  (2's complement of X)
```

Looking closely at the 1's complements of the numbers in Examples 2.13, 2.14 and 2.15, we see that the 1's complement of the number is the number, with all the 0's changed to 1's and the 1's changed to 0's.

Example 2.16

Find the 2's complement of 1110110.

```
  1 1 1 0 1 1 0  (X)
  0 0 0 1 0 0 1  (1's complement of X)
          + 1    (plus 1)
  0 0 0 1 0 1 0  (2's complement of X)
```

Example 2.17

Find the 2's complement of 101101.

```
  1 0 1 1 0 1  (X)
  0 1 0 0 1 0  (1's complement of X)
        + 1    (plus 1)
  0 1 0 0 1 1  (2's complement of X)
```

Applying the rules of complementation to octal numbers, we see that the 8's complement of a number is the 7's complement of the number, plus 1.

Example 2.18

Find the 8's complement of 77341.

```
  7 7 7 7 7  (largest possible number)
- 7 7 3 4 1  (X)
  0 0 4 3 6  (7's complement of X)
        + 1  (plus 1)
  0 0 4 3 7  (8's complement of X)
```

Example 2.19

Find the 8's complement of 00000.

```
  7 7 7 7 7
- 0 0 0 0 0
  7 7 7 7 7  (7's complement)
        + 1
1←0 0 0 0 0  (8's complement)
```

Binary Subtraction

By employing the techniques of complementary arithmetic, it is possible to effect a subtraction using the addition process.

To perform A-B, either of two methods may be used:

1) direct subtraction of B from A; or
2) the addition of A to the complement of B.

Example 2.20

Perform 783_{10}-25_{10}

Method 1:

```
  7  8  3
-    2  5
-------------
  7  5  8   783-25=758
```

Method 2: 9 9 9-2 4=9 7 4 (9's complement of 25)
9 7 4+1 = 9 7 5 (10's complement of 25)

```
   7  8  3  (A)
+  9  7  5  (10's complement of B)
-------------
1← 7  5  8   783-25=758
```

Example 2.21

Perform 1101101_2-1011_2

NOTE: First add leading 0's to make numbers the same length.

Thus we are to perform 1101101_2-0001011_2

```
   1 1 0 1 1 0 1  (A)
   1 1 1 0 1 0 1  (2's complement of B)
-----------------
1← 1 1 0 0 0 1 0
```

1101101-1011=1100010

Example 2.22

Perform 101011_2-101011_2 (A-A)

```
   1 0 1 0 1 1  (A)
   0 1 0 1 0 1  (2's complement of A)
-----------------
1← 0 0 0 0 0 0
```

Thus, a number plus its complement always equals zero.

Octal Subtraction

Octal subtraction (A-B) may be performed by adding A to the 8's complement of B.

Example 2.23

Perform 6275_8-31_8

(Add leading 0's) 6275_8 - 0031_8

```
   6  2  7  5  (A)
+  7  7  4  7  (8's complement of B)
-------------
1← 6  2  4  4
```

6275-31=6244

Example 2.24

Perform 7000_8-76_8

```
   7  0  0  0  (A)
+  7  7  0  2  (8's complement of B)
-------------
1← 6  7  0  2
```

7000-76=6702

MULTIPLICATION
& BINARY

Binary multiplication is accomplished in a manner similar to decimal multiplication.

Example 2.25

Perform 1298_{10}x10_{10}

```
   1  2  9  8
      x  1  0
   ----------
   0  0  0  0
1  2  9  8
---------------
1  2  9  8  0
```

1298x10=12980

Example 2.26

Perform 8735_{10}x63_{10}

```
      8  7  3  5
         x  6  3
      ----------
   2  6  2  0  5
5  2  4  1  0
---------------
5  5  0  3  0  5
```

8735x63=550305

Example 2.27

Perform $10111001_2 \times 10_2$

```
  1 0 1 1 1 0 0 1
            x 1 0
  ---------------
  0 0 0 0 0 0 0 0
1 0 1 1 1 0 0 1
---------------
1 0 1 1 1 0 0 1 0
```

10111001x10=101110010

Example 2.28

Perform $101110_2 \times 1101_2$

```
      1 0 1 1 1 0
        x 1 1 0 1
      -----------
      1 0 1 1 1 0
    0 0 0 0 0 0
  1 0 1 1 1 0
1 0 1 1 1 0
-----------------
1 0 0 1 0 1 0 1 1 0
```

101110x1101=1001010110

It is interesting to note from Example 2.25 that 10 times a decimal number simply appends a 0 to the right. From Example 2.27, we see that 2 times a binary number simply appends a 0 to the right.

DIVISION-BINARY

Binary division follows the same procedures as decimal division.

Example 2.29

Perform $2288_{10} \div 13_{10}$

```
      1 7 6
   ----------
1 3 | 2 2 8 8
      1 3
      -----
        9 8
        9 1
        -----
          7 8
          7 8
          -----
            0
```

2288÷13=176

Example 2.30

Perform $101100010_2 \div 110_2$

```
          1 1 1 0 1 1
      ----------------
1 1 0 | 1 0 1 1 0 0 0 1 0
        1 1 0
        -----
        1 0 1 0
          1 1 0
          -----
        1 0 0 0
          1 1 0
          -----
          0 1 0 0 1
            1 1 0
            -----
            1 1 0
            1 1 0
```

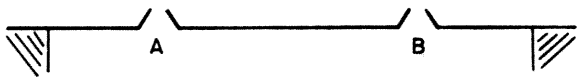
101100010 ÷ 110=111011

LOGICAL AND

In the binary number system, additional operations exist over and above addition, subtraction, multiplication, and division. These additional operations are known as logical or Boolean operations.

One such logical operation is the AND function.

Consider the drawbridge in the following figure:



DG00695

The bridge consists of 2 spans which can be opened: A and B. Obviously, the path across this bridge is continuous only if both A AND B are closed.

SPAN A	SPAN B	BRIDGE
OPEN	OPEN	OPEN
OPEN	CLOSED	OPEN
CLOSED	OPEN	OPEN
CLOSED	CLOSED	CLOSED

If the two states of each span are assigned the binary values OPEN=0 and CLOSED=1, the table can be rewritten.

A	B	A AND B A . B A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

Two binary numbers can be ANDed by simply ANDing respective bits from each other.

Example 2.31

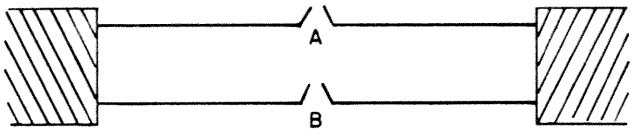
Perform 10111011 ∧ 00011011

```
1 0 1 1 1 0 1 1  (A)
0 0 0 1 1 0 1 1  (B)
0 0 0 1 1 0 1 1  (A·B)
```

NOTE: Both corresponding bits in A and B must be 1 for the resulting bit A·B to be a 1.

LOGICAL OR

Consider 2 drawbridges spanning a river as shown in the following figure:



DG00696

A path from one side of the river to the other exists if A OR B or both is closed.

SPAN A	SPAN B	PATH
OPEN	OPEN	OPEN
OPEN	CLOSED	CLOSED
CLOSED	OPEN	CLOSED
CLOSED	CLOSED	CLOSED

If we assign binary values to the states of each drawbridge, the table can be rewritten as follows:

A	B	A OR B A + B A V B
0	0	0
0	1	1
1	0	1
1	1	1

Notice that with the OR operation, if either of the corresponding bits in A or B is a 1, the resulting bit A + B is a 1.

Two binary numbers can be ORed by simply ORing respective bits from each number.

Example 2.32

Perform 10111011 ∨ 00011011

```
1 0 1 1 1 0 1 1  (A)
0 0 0 1 1 0 1 1  (B)
1 0 1 1 1 0 1 1  (A+B)
```

Example 2.33

Perform 10111011 ∨ 01000100

This is equivalent to a ∨ 1's complement of A.

```
1 0 1 1 1 0 1 1  (A)
0 1 0 0 0 1 0 0  (A1's complement of A)
1 1 1 1 1 1 1 1  (A + 1's complement of A)
```

Thus, all 1's result when a number and its 1's complement are ORed.

LOGICAL EXCLUSIVE OR

The logical OR function described in the previous section is more precisely known as the logical inclusive OR function.

The exclusive OR function can be defined as follows: The resulting bit of A ⊕ B is a 1 if either of the corresponding bits in A or B is a 1, but not if both bits in A and B are a 1.

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Example 2.34

Perform 10111011 ⊕ 00011011

```
1 0 1 1 1 0 1 1  (A)
0 0 0 1 1 0 1 1  (B)
1 0 1 0 0 0 0 0  (A⊕B)
```

SIGNED NUMBER REPRESENTATION

Sign Bit Definition

In many applications where the use of both positive and negative numbers is required, some method to indicate the sign of the number must be employed. In written text, this is done with the + and - signs. The computer, however, works with binary numbers and would not easily recognize a + or - sign. Another method must be used to indicate the sign of the number. One possibility is to define the leftmost bit of the binary number to be the sign indicator or sign bit. A one (1) in this position would indicate that the number represented by the bits to the right is negative; a zero (0) would indicate that the number is positive. Utilizing this technique of signed number representation, the sign bit is followed by the absolute value of the number. Another method of representing signed numbers employs the concept of complementary numbers, as described in Section II. B. It is this last method which will be pursued further here.

If the maximum allowable number of bits is 4, then the following numbers are possible:

0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

This set of 16 numbers is cyclic because adding 1 to 1111 brings us back to 0000.

Also, subtracting 1 from 0000, gives us 1111. If this set of numbers is said to contain only positive values, then the range of values is

0 0 0 0 thru 1 1 1 1
or 0_{10} thru 15_{10}

Suppose we divide this set in half, and define one half as representing positive values, and the other half negative values (A). Also, let's restack the set so that 0000 is at the center (B).

A		B	C (decimal)
0 0 0 0		0 1 1 1	7
0 0 0 1		0 1 1 0	6
0 0 1 0	Positive	0 1 0 1	5
0 0 1 1	Numbers	0 1 0 0	4
0 1 0 0		0 0 1 1	3
0 1 0 1		0 0 1 0	2
0 1 1 0		0 0 0 1	1
0 1 1 1		0 0 0 0	0
1 0 0 0		1 1 1 1	-1
1 0 0 1		1 1 1 0	-2
1 0 1 0	Negative	1 1 0 1	-3
1 0 1 1	Numbers	1 1 0 0	-4
1 1 0 0		1 0 1 1	-5
1 1 0 1		1 0 1 0	-6
1 1 1 0		1 0 0 1	-7
1 1 1 1		1 0 0 0	-8

Notice that all the negative numbers have a 1 in the leftmost bit position and all the positive numbers have a 0 in the leftmost bit position. Thus, if 0000 is defined as being a positive number, there is the same quantity of positive and negative values.

NOTE: If the programmer is using the leftmost bit for sign definition, care should be taken not to overflow the range of values.

Example 2.35

Perform 5+(-4)

5	0101
+ (-4)	+ 1100
1	1←0001

Example 2.36

Perform 6+(-6)

6	0110
+ (-6)	+ 1110
0	1←0000

Example 2.37

Perform 7+2

7	0111
2	+ 0010
9	1←1001

Note that in this example the desired result was not obtained because the range has been exceeded. 1001 represents -7, not +9.

Range of Signed Numbers

In the 4-bit number set of the previous section, the range of unsigned numbers is as follows:

0000 thru 1111
or 0_{10} thru 15_{10}
or 0_8 thru 17_8

The range of signed numbers, however, from table B is as follows:

1000 thru 0111
or -8_{10} thru $+7_{10}$
or -10_8 thru $+7_8$

Example 2.38

If a number set contains 16-bit numbers, the ranges are as follows:

Unsigned

0000000000000000 thru 1111111111111111
or 0_{10} thru $65,535_{10}$
or 0_8 thru 177777_8

Signed

1000000000000000 thru 0111111111111111
or $-32,768_{10}$ thru $+32,767_{10}$
or -100000_8 thru $+077777_8$

This page intentionally left blank

SECTION III

LOGIC AND LOGIC CONVENTIONS

GENERAL

1. The various circuits, logic gates, logic conventions and integrated circuits common to mini-computers are discussed in this section.

2. Integrated circuits (IC's) are defined as a number of minute circuits bonded onto a small plug in pack roughly 1/4" by 3/4" in size. IC's are also referred to as "Chips", "Bugs", "Dips" or "Packs".

3. Printed Circuit Board (PCB) - etched boards upon which are mounted numerous IC's making up an element of computer logic. A typical computer may have one to three PCB's which provide the necessary Central Processor Unit logic. (See Figure 3-1.)

4. Logic Levels - a typical mini-computer has two logic levels ground and plus three volts (+3V). Where ground might represent a zero (0) and +3V and one (1).

5. Symbols
   a. $\wedge$ or $\cdot$ represent AND. For example $A \wedge B$ or $A \cdot B$.
   b. $\vee$ or + represent OR. For example $A \vee B$ or $A+B$.

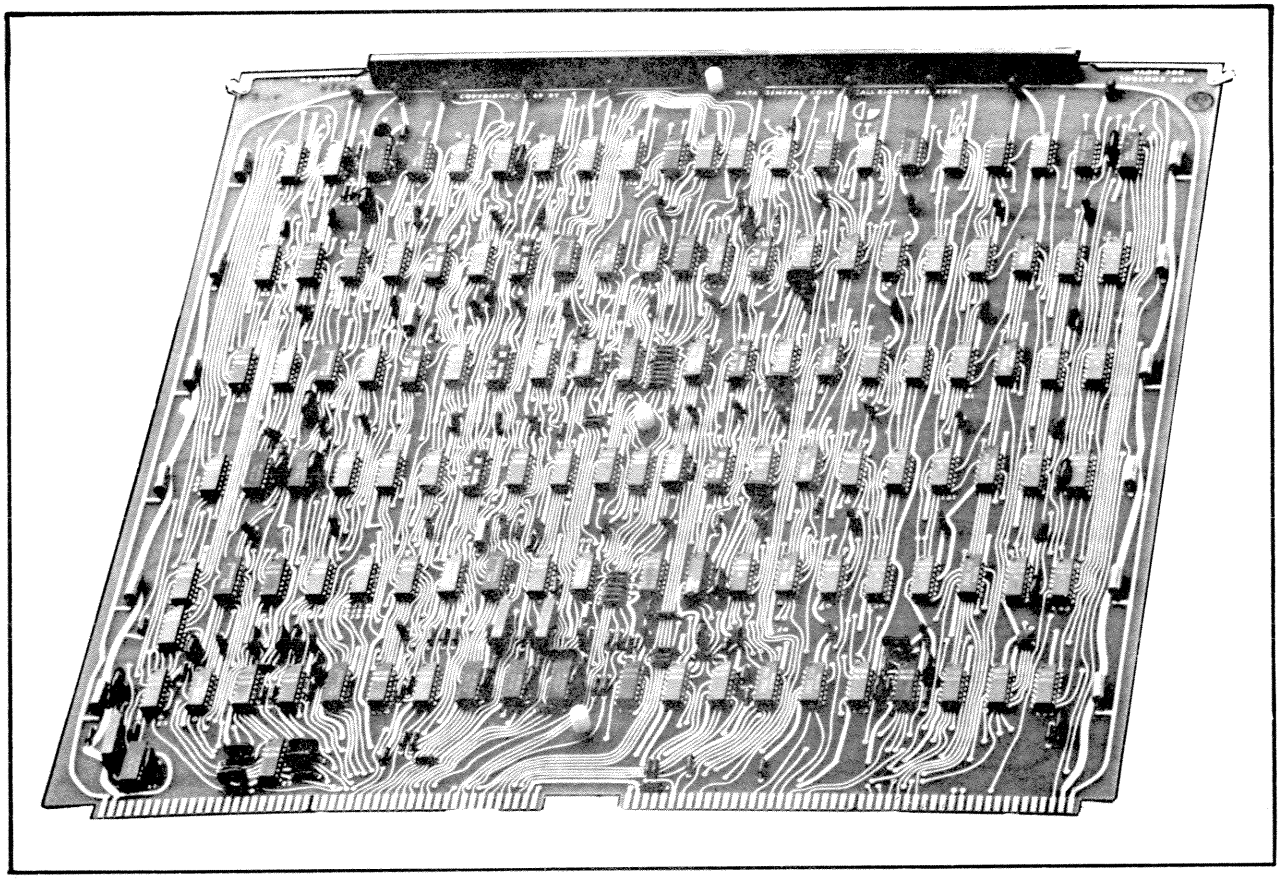
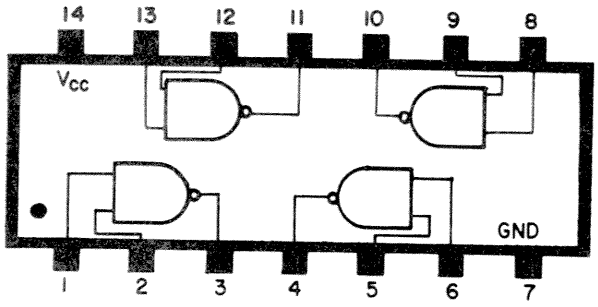


Figure 3-1 Printed Circuit Board



DG00697

TRUTH TABLE

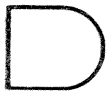
V_{IN}	V_{IN}	V_{OUT}
L	L	H
L	H	H
H	L	H
H	H	L

Figure 3-2 Logic Diagram

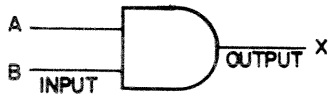
LOGIC GATES

AND Gate

a. Symbol



b. Two input - noninverting



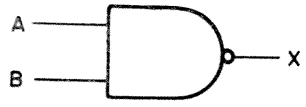
INPUT	OUTPUT
A B	X
L L	L
L H	L
H L	L
H H	H

Output is high if both inputs are high.

NOTE: A straight line into a gate represents a high condition. A straight line with a small circle will indicate a low condition.o—

Definition: Will give a desired output when ALL input conditions are satisfied.

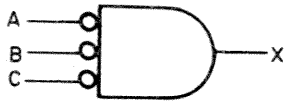
c. Two input - inverting (NAND Gate)



INPUT	OUTPUT
A B	X
L L	H
L H	H
H L	H
H H	L

Output is low if both inputs are high.

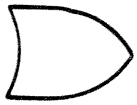
d. Three input - inverting (NEGATIVE NAND)



INPUT	OUTPUT
A B C	X
L L L	H
L L H	L
L H L	L
L H H	L
H L L	L
H L H	L
H H L	L
H H H	L

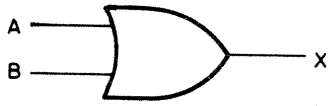
OR Gate

a. Symbol



Definition: Will give a desired output when ANY input condition is satisfied.

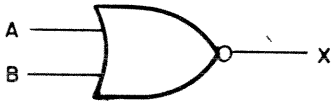
b. Two input - noninverting (OR)



INPUT	OUTPUT
A B	X
L L	L
L H	H
H L	H
H H	H

Output is high if either or both inputs are high.

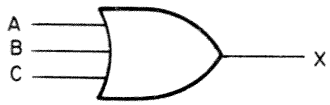
c. Two input - inverting (NOR Gate)



INPUT	OUTPUT
A B	X
L L	H
L H	L
H L	L
H H	L

Output is low if either or both inputs are high.

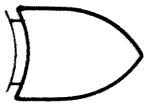
d. Three input - noninverting



INPUT	OUTPUT
A B C	X
L L L	L
L L H	H
L H L	H
L H H	H
H L L	H
H L H	H
H H L	H
H H H	H

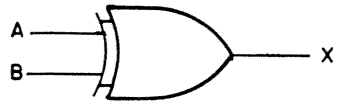
XOR Gate

a. Symbol



XOR TRUTH TABLE

b. XOR (Exclusive OR)

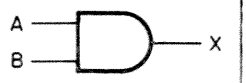
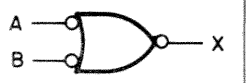
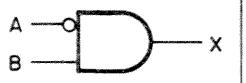
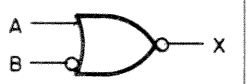
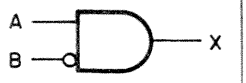
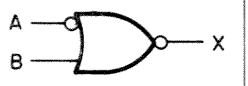
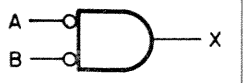
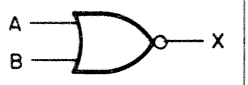
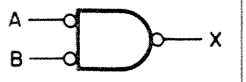
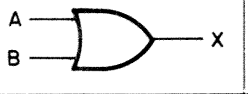
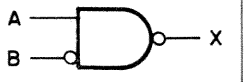
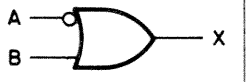
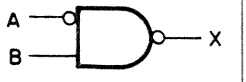
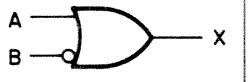
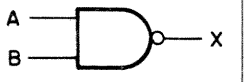
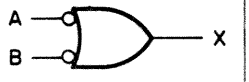


INPUT	OUTPUT
A B	X
L L	L
L H	H
H L	H
H H	L

Output is high with one input Low and the other High.

Definition: Will give a desired output only when both input signals are different.

Basic Logic Diagrams - The following illustrates the applications and functions of two variables and equivalents.

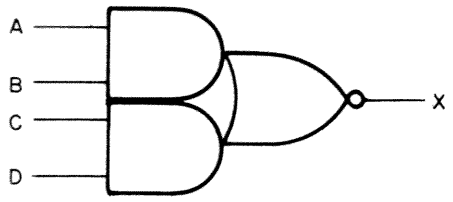
GATES AND	GATES OR	A	B	X
		H H L L	H L H L	H L L L
		H H L L	H L H L	L L H L
		H H L L	H L H L	L H L L
		H H L L	H L H L	L L L H
		H H L L	H L H L	H H H L
		H H L L	H L H L	H L H H
		H H L L	H L H L	L H L H
		H H L L	H L H L	L H H H
		L L H H	L H L H	L H H L

DG00698

Figure 3-3 Logic Chart

Multiple Input Gates

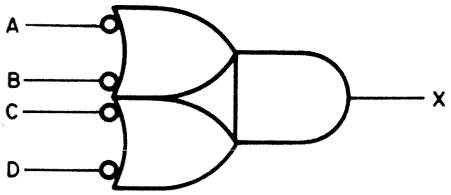
a. MULTIPLE INPUT AND/OR - INVERTING



Output is low if both A and B are High or both C and D are High.

INPUT	OUTPUT	INPUT	OUTPUT
A B C D	X	A B C D	X
L L L L	H	H L L L	H
L L L H	H	H L L H	H
L L H L	H	H L H L	H
L L H H	L	H L H H	L
L H L L	H	H H L L	L
L H L H	H	H H L H	L
L H H L	H	H H H L	L
L H H H	L	H H H H	L

b. MULTIPLE INPUT AND/OR - INVERTING



Output is High if A or B is low and C or D is low.

INPUT	OUTPUT	INPUT	OUTPUT
A B C D	X	A B C D	X
L L L L	H	H L L L	H
L L L H	H	H L L H	H
L L H L	H	H L H L	H
L L H H	L	H L H H	L
L H L L	H	H H L L	L
L H L H	H	H H L H	L
L H H L	H	H H H L	L
L H H H	L	H H H H	L

Flip-Flops

a. A flip-flop is a device that has two stable states. One state is called the "SET" state, and the other state is called the "CLEAR" or "RESET" state. A flip-flop can be in only one of its stable states at a time. So it will be either set or cleared.

b. In Figure 3-4 we have a symbol for a "SET-RESET" flip-flop. By applying a logic high to the "S" or "SET" input the flip-flop will become "SET". To clear or reset the flip-flop we must apply a logic high signal to the "R" or reset input.

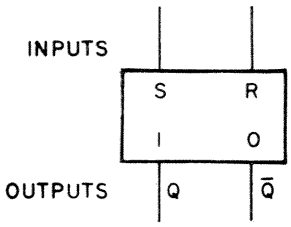


Figure 3-4 Flip-flop

c. The outputs from the flip-flop are called the "1" or "0" outputs. Sometimes these are referred to Q and $\overline{Q}$ respectively.

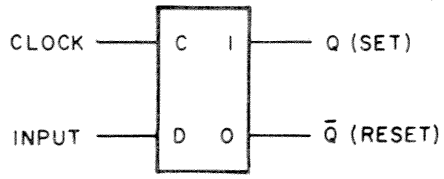
d. The levels on the output lines depends on the state of the flip-flop. If the flip-flop is "SET" the "1" output would be a logic high and the "0" output would be a logic low. If the flip-flop were reset, the "1" output would be a logic low and the "0" output would be a logic high.

STATE	"1"	"0"
SET	HIGH	LOW
RESET	LOW	HIGH

e. Since the "1" output is always high when the flip-flop is set, this output is sometimes referred to as the "SET OUTPUT". The same applies for the clear output.

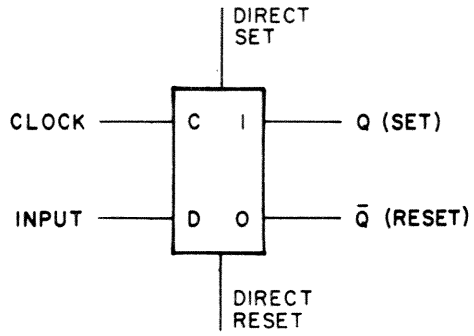
f. "D" type flip-flop:

1. A "D" type flip-flop will set or reset upon state of "D" input and a clock pulse.



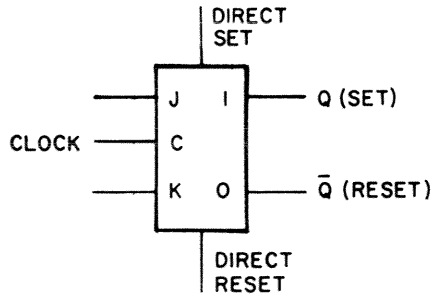
INPUT	OUTPUTS		CLOCK PULSE
D	Q	$\bar{Q}$	
L	L	H	⎍
H	H	L	⎍

2. A "D" type flip-flop can also have direct set and reset inputs which override the "D" input.



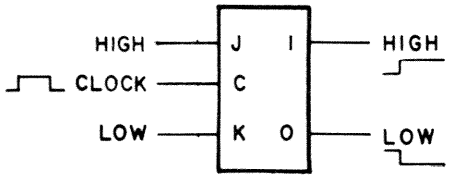
g. J-K type flip-flop:

1. A J-K flip-flop has an input configuration different than other flip-flops, which allows it to be used many different ways.

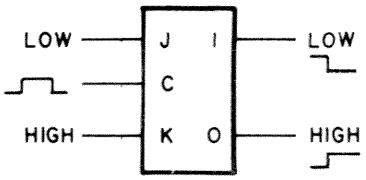


2. Rules:

a. Set-a J-K flip-flop can be set one of two ways. It can be directly set or set by applying a logic high on the J input, a logic low on the K input and a clock pulse.



b. Reset-A J-K flip-flop can be reset one of two ways. It can be directly reset or reset by applying a logic low to the J input, a logic high to the K input and a clock pulse.

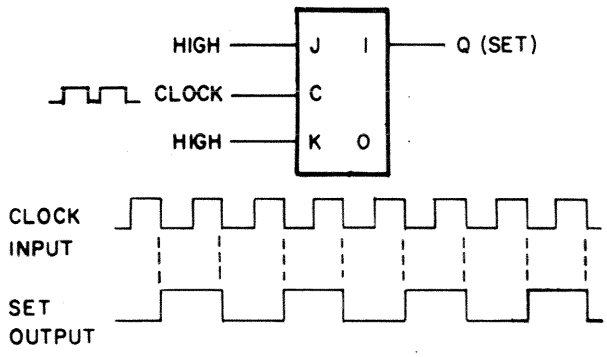


h. Flip-flop Applications:

1. A flip-flop,because of its two stable states,can be utilized to store 2 levels of information. For example, if a flip-flop is set it is equal to "Yes" if reset it is equal to "No".

2. Since in the binary system we only deal in 1's and 0's a flip-flop would be ideal to represent binary numbers. If a flip-flop is set we say it's storing a one "1". If cleared we say that it's storing a "0".

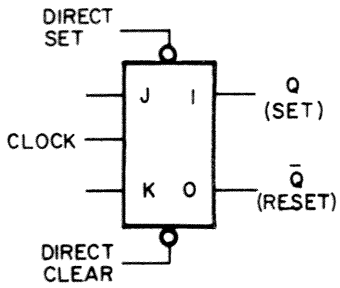
3. If we take a J-K flip-flop and connect both J and K inputs to a high level we have what is called a toggle flip-flop.



4. The clock input will alternately set and reset the flip-flop, so that the output signal from the set side looks as shown. Note that it takes 2 complete clock cycles to produce 1 cycle from the set output.

5. By applying a signal to the clock input and obtaining an output from either the set or reset output, the signal developed will be exactly one-half the frequency of the clock input. This is a divide by 2 operation which is an inherited characteristic of a flip-flop.

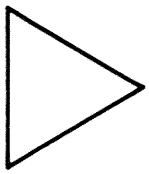
6. J-K



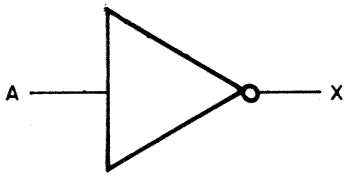
INPUT	OUTPUT		CLOCK
J-K	Q	$\bar{Q}$	
L-L	NO CHANGE		⌐¬
L-H	L	H	⌐¬
H-L	H	L	⌐¬
H-H	TOGGLE		⌐¬

Inverters

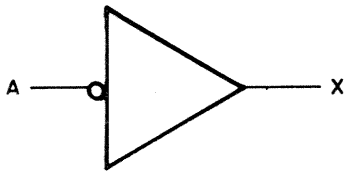
a. Symbol



b. Positive to Negative

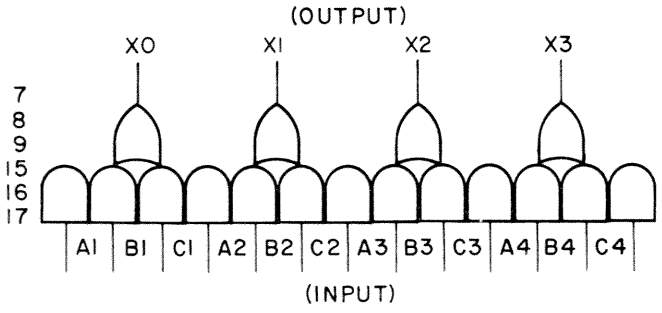


c. Negative to Positive



Multiplexer

a. One of three inputs selected and presented as input to another element, usually an Adder.



DG00736

b. Pin Identification

7 - 8 - 9		Enables
15		Complement
16 - 17		Select Input
L	L	No Select
L	H	A Input (4 bits)
H	L	B Input (4 bits)
H	H	C Input (4 bits)

c. Rules

1) With 3 enables High and no select the 4 outputs will be ground.

2) With 1 enable Low and no select the 4 outputs will be High.

3) With 1 enable Low and select the 4 outputs will be High.

4) With 3 enables High, select and complement Low, the 4 outputs will follow the 4 selected inputs.

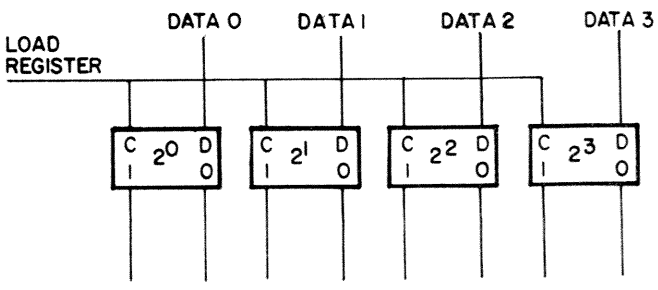
5) With 3 enables High, select and complement High, the 4 outputs will be the one's complement of the 4 selected inputs.

Registers

a. Definition:
A device that can receive information upon command, hold that information without modification and transfer it upon request.

b. Register Elements:
1. Most Registers are made up of flip-flops. The number of flip-flops in a particular register is dependent upon the amount of data handled.

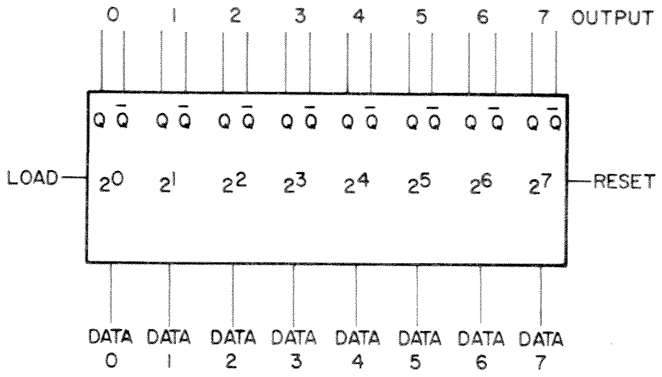
c. Register Configuration:
1. Registers could be shown in their components parts as in Figure 3-5.



DG00699

Figure 3-5 Basic Register

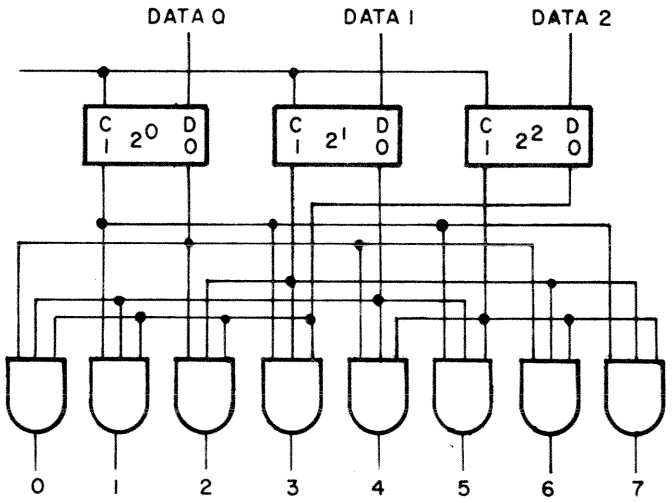
2. Sometimes when many flip-flops make up a register it is shown as in Figure 3-6.



DG00735 Figure 3-6 Register Containing 8 Flip-flops

Here the register has 8 flip-flops, all are loaded with the load input, and reset with the reset input. The outputs are from the Q and $\overline{Q}$ outputs of each flip-flop.

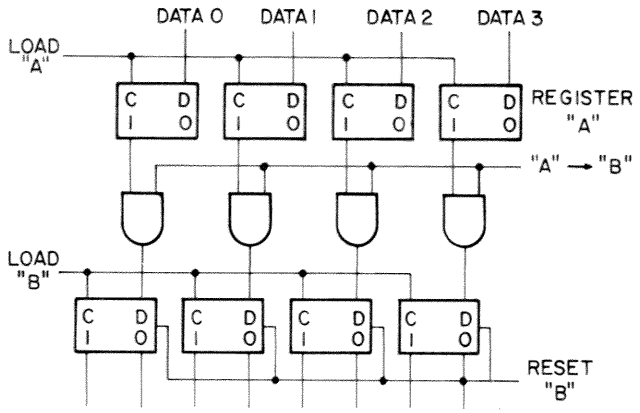
3. A register's contents may be decoded by using "AND"gates connected to the register's outputs (As shown in Figure 3-7.)



DG00700 Figure 3-7 Decoded Register

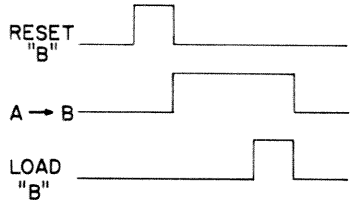
A three bit register is connected to 8 AND gates. The AND gates decode the contents of the register. Only one AND gate can be enabled at a time.

d. Transfers: (See Figure 3-8.)
To be able to take the contents of one register and load it into another.
1. Parallel - transfers all data bits simultaneously.



DG00733 Figure 3-8 Parallel Transfer

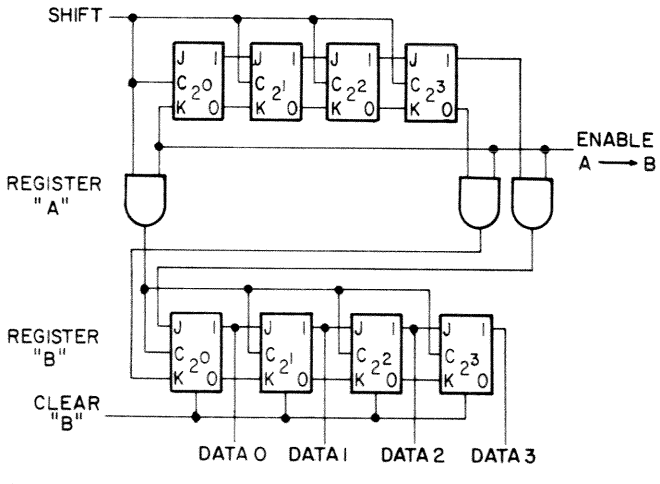
To do the parallel transfer correctly the following signals are enabled.



DG00734

At the end of the load "B" pulse whatever data was in register "A" is now in register "B". Note that all bit positions were loaded at the same time.

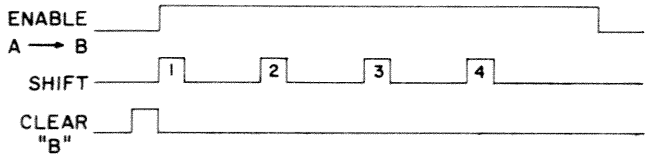
2. Serial Transfer: (See Figure 3-9.) Moves the contents of one register to another register - one bit at a time.



DG00731

Figure 3-9 Serial Transfer

a. Below are the required signals needed to perform the shift, and the contents of each register after every shift.

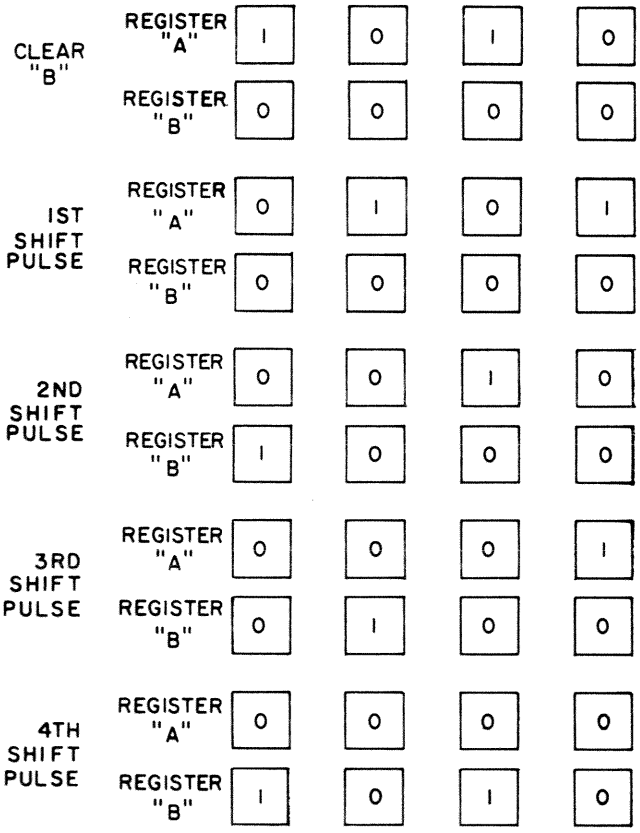


DG00732

b. The first thing done is to clear the "B" register.

c. Then the enable "A→ B" is present (High).

d. If the "A" register had the number 1010 in it, the transfer would look like this:



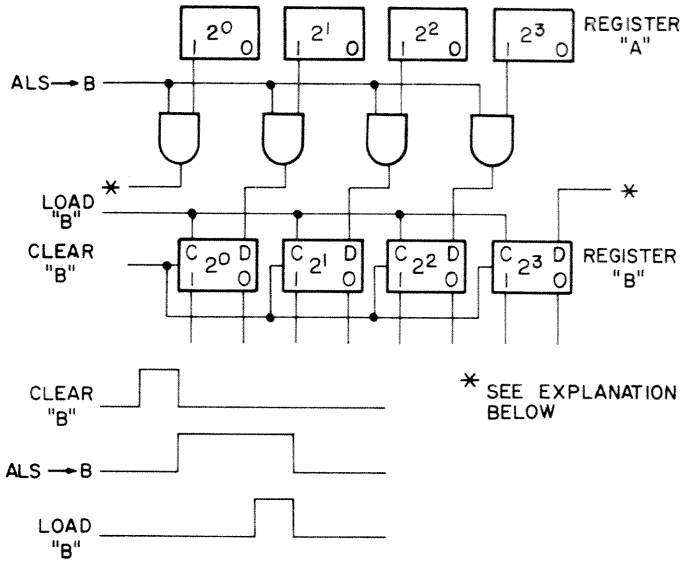
DG00701

After the fourth shift pulse the transfer is complete.

e. When doing a serial transfer the number of clock pulses (shift pulses) must equal the total number of flip-flops in the register, not enough, or too many shift pulses will result in the loss of data being transferred.

f. Shift Transfers: (Fig. 3-10)

1. Left shift transfer - the contents of one register is shifted left one place and loaded into another register.



DG00730

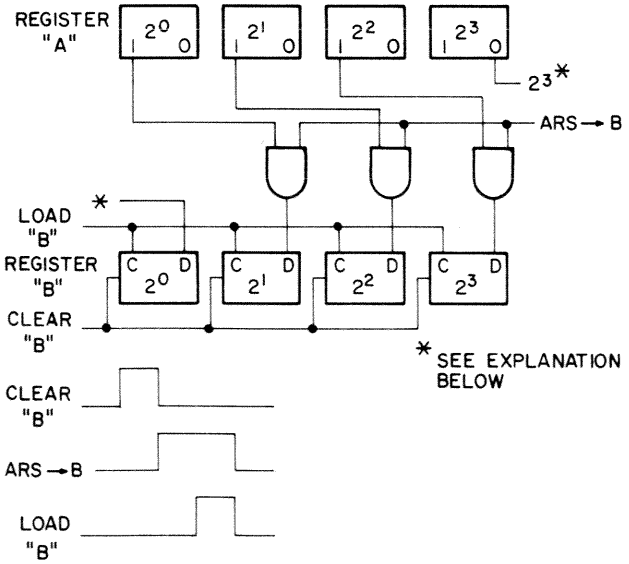
Figure 3-10 Left Shift Transfer

2. Note that in the left shift:
   2^3 in "A" is applied to 2^2 in "B".
   2^2 in "A" is applied to 2^1 in "B".
   2^1 in "A" is applied to 2^0 in "B".
   2^0 in "A" is treated in a special way.

3. In some computers bit 2^0 in "A" is lost during a left shift. In other machines bit 2^0 is applied to a flip-flop which indicates the state of what 2^0 was. And still in other machines bit 2^0 is applied to bit 2^3 in "B", so that the bit is not lost. This is called end around shifting.

4. If "A" were equal to 0011_2 and then left shifted transferred into "B". "B" would be equal to 0110_2. Notice that now "B" is twice as much as what the value was in "A". A left shift of one place actually multiplies the contents of a register by 2.

5. RIGHT SHIFT:(Fig. 3-11)
   a) The contents of one register is right shifted one place and loaded into another register.



DG00729 Figure 3-11 Right Shift Transfer

b) In Figure 3-11, a right shift transfer is accomplished by taking flip-flop 2^0 in "A", and applying it to flip-flop 2^1 in "B". 2^1 in "A" is sent to 2^2 in "B", and 2^2 in "A" is sent to 2^3 in "B". Flip-flop 2^3 in "A" is handled in a special way.

c) If "A" initially contained 0110_2, a right shift one to "B" would give us 0011_2 in "B". Looking at the value of the data in "A" and "B", we see that the number in "B" is half of the number in "A". Doing a right shift causes the contents of a register to be divided by 2.

d) Flip-flop 2^3 in "A" is handled one of three ways.

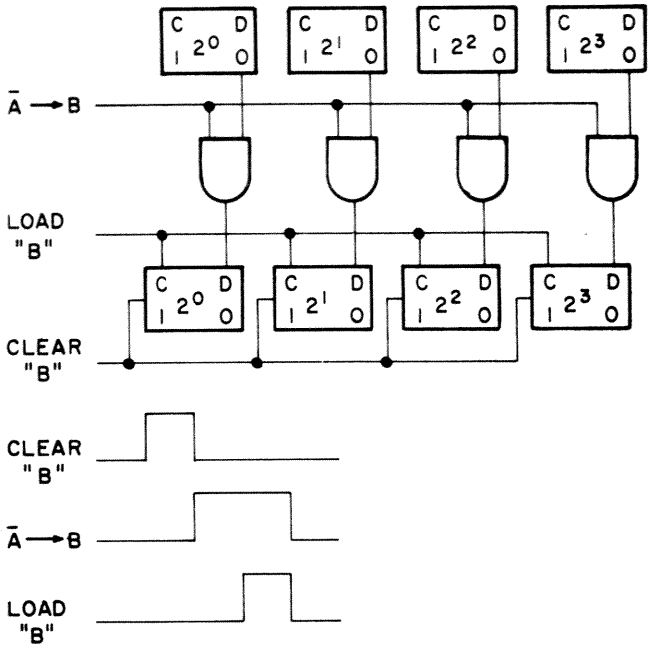
   1. It is lost - not applied anywhere.

   2. It is applied to a special flip-flop.

   3. It is applied to flip-flop 2^0 in "B", so that the bit value in 2^3 in "A" is not lost.

6. COMPLEMENT TRANSFER

a) To obtain the 1's complement of a number contained in a register, all we have to do is transfer the 0's output of a register into another register.

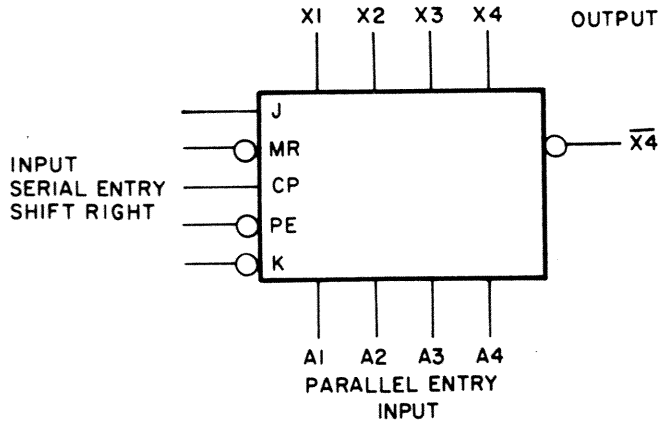


DG00739 Figure 3-12 Complement Transfer

b) If register "A" had the number 1010_2 and a complement transfer to "B" was enabled, then "B" would have 0101_2 in it after the transfer.

Four Bit Shift Register

a. Type A (Fairchild 9300)



DG00702

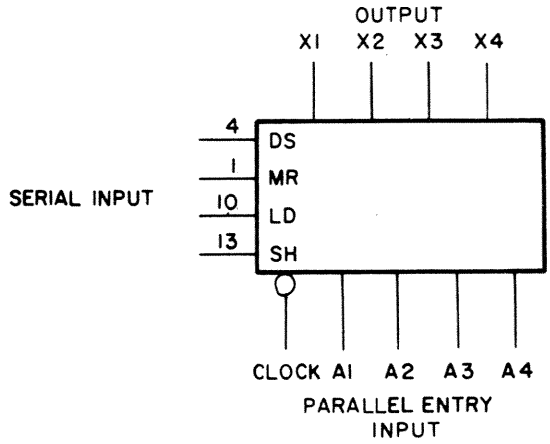
1) Terms
MR - Master Reset
CP - Clock Pulse
PE - High (Load serially shift right)
PE - Low (Load parallel entry)
J, $\overline{K}$ - Serial entry

2) Operation

a) Load serially shift right on the clock pulse - J, $\overline{K}$ is the serial input to most significant bit, while the 4 bits shift right and the least significant bit is lost.

b) Parallel entry on the clock pulse - the 4 inputs are loaded simultaneously in parallel.

b. Type B (Signetics 8271)



DG00703

1) Terms
DS - Data Input (Serial Data)
MR - Master Reset
LD - Load (Parallel)
SH - Shift (Right and load serially)

Truth Table

LD	SH	Effect when clocked
L	L	No change
L	H	Load serially and shift right
H	L	Load parallel entry
H	H	Load serially and shift right

NOTE: Shift overrides load.

2) Operation
   a) With both LD, SH Low on the clock, there is no change (in effect Hold as is)

   b) Load Serially and shift right - DS is the serial input. On the clock pulse DS is loaded into the most significant bit, all 4 bits shift right and the least significant bit is lost.

   c) Parallel entry - on the clock pulse the 4 inputs are loaded simultaneously in parallel.

This page intentionally left blank

SECTION IV

COMPUTER ORGANIZATION

DEFINITIONS

1. CPU - Central Processor Unit. Frequently referred to as the "mainframe". Basically, it is the computer less Peripherals.

2. BIT - A unit of information. Contraction of binary digit. The smallest unit of information in a Binary system of notation. It represents a choice between two possible states, usually "ONE" and "ZERO".

3. BYTE - A group of bits (usually eight) forming a character.

4. WORD - A fixed number of bits treated as a unit and capable of being stored in one memory location. A word can be interpreted by the computer as:
   a. An Instruction word
   b. A Memory Address word
   c. A Data word

5. WORD LENGTH - A fixed number of bits normally determining the size of the machine, i. e., 12 bit machine, 16 bit machine, 24 bit machine, etc. Figure 4-1 illustrates a typical 16 bit word. Note that bits are numbered from left to right.

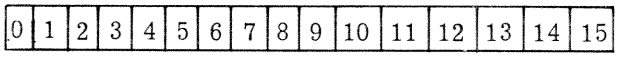


Figure 4-1 A 16 bit Word

6. MSB - Most Significant Bit. Bit 0 in a great number of mini-computers is the most significant bit, while in a few, bit 15 is considered the most significant bit.

7. SIGN BIT - In a computer where bit 0 is the MSB, this bit is also regarded as the sign bit. In a Signed Machine bit 0 is actually considered the sign bit by the hardware. In an Unsigned Machine, the Program itself regards bit 0 the sign while to the Hardware, it is as any other bit.

8. PROGRAM - A group of sequential computer instructions, i. e., a series of step directions to the computer.

9. HARDWARE - Physical components of a computer such as Logic chips, chassis, printed circuit boards, etc.

10. SOFTWARE - Computer System Programs, including tapes, listings and documentation.

11. SERIAL MACHINE - Each bit of a word is processed individually. The advantage here is less logic required, therefore, cheaper. However, processing each bit separately is time-consuming, resulting in a slower machine.

12. PARALLEL MACHINE - Each bit of a word processed simultaneously i. e., in parallel, requires more logic and is more expensive than the serial machine. It is, however, much faster.

13. NIBBLE MACHINE - Unique to Data General machine. A compromise between Serial and Parallel machines. A Nibble is four bits, therefore a Nibble machine processes four bits simultaneously. Requires less logic than parallel, is less expensive and still nearly approaches the speed of a parallel machine.

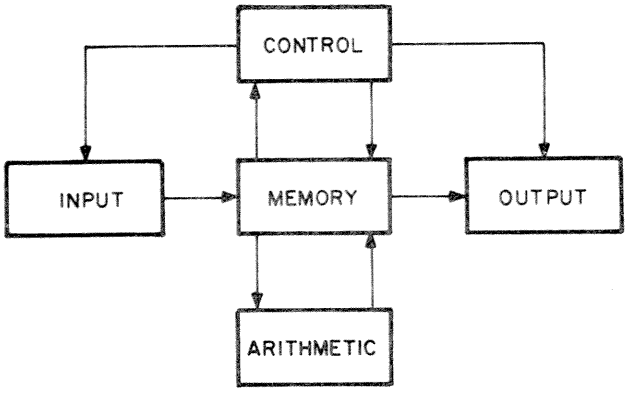
14. PERIPHERAL - A device external to the "main frame" providing outside communication with the CPU. Referred to as IO (Input/Output) devices.

    A partial list of devices and direction of communication:

    TELETYPE®* - Input/Output device
    PAPER TAPE READER - Input device
    PAPER TAPE PUNCH - Output device
    DISK - Input/Output device
    DRUM - Input/Output device
    MAGNETIC TAPE - Input/Output device
    CASSETTES - Input/Output device
    CRT DISPLAY - Input/Output device
    LINE PRINTER - Output device
    CARD READER -Input device
    CARD PUNCH - Output device

* Teletype® is a registered trademark of Teletype Corporation, Skokie, Illinois. All references to teletypes in this manual shall apply to this mark.

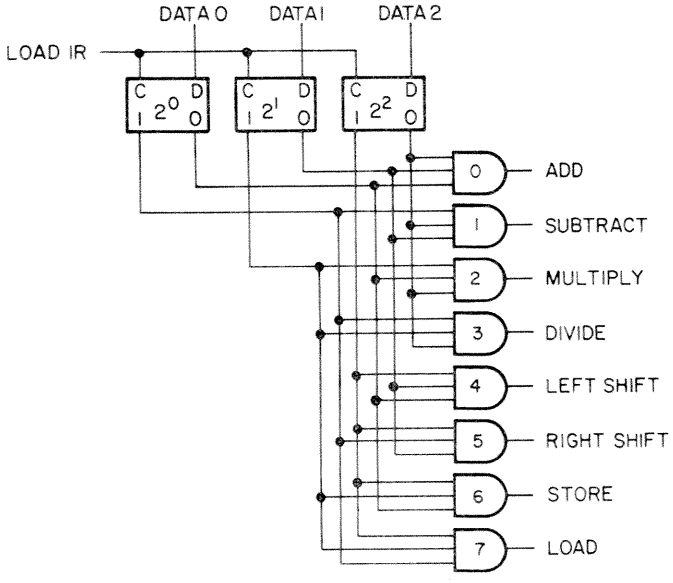
1. Major Areas:
   A basic computer is comprised of 5 sections. Each section performs a specific job.



DG00704

   a) Control Section - At all times governs and times the actions of the computer. It is made up of 3 areas.

      1) Instruction Register and Instruction Decodes.
      A computer must be told what to do. The commands that tell it what to do are called instructions. The instructions are nothing more than a group of binary bits which represent what is desired of the machine. The binary information is placed into a register and the output of the register is decoded. The decoded outputs are signals which are sent to other sections of the machine and cause the necessary actions to take place in order to perform the instruction.



DG00740

Figure 4-2 Instruction Register Decoding

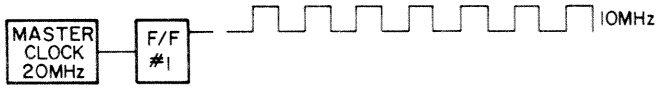
For example:
Figure 4-2 shows an instruction register which has three flip-flops. The outputs are decoded into one of 8 possible instructions.

The instruction and codes for Figure 4-2 are shown below:

000 = Add	001 = Left Shift
100 = Subtract	101 = Right Shift
010 = Multiply	011 = Store
110 = Divide	111 = Load

These decoded outputs are sometimes called command enables.

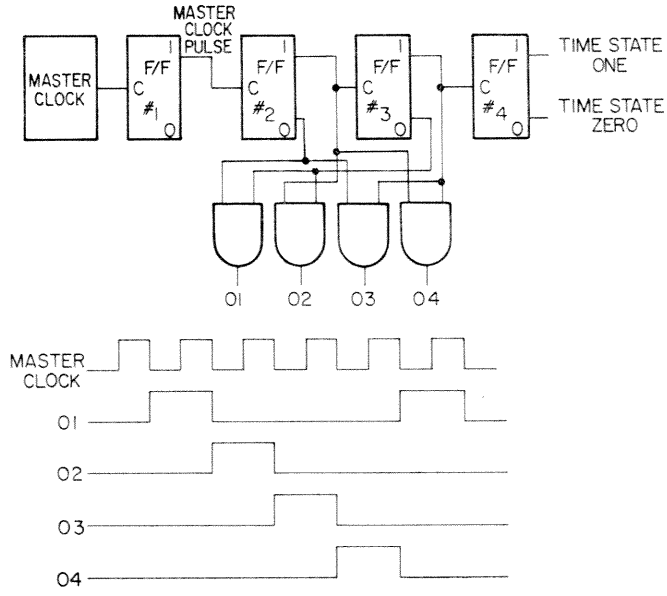
2. Timing
   Everything that occurs within a computer must occur in a logical sequence. The timing circuits develop the timing signals which, along with the instruction decodes, make things happen within the computer in a nice orderly fashion. A main part of the timing section is a circuit called "Master Clock". This is a very accurate oscillator, whose output signal will develop all other timing signals within the computer.



DG00727

Figure 4-3 Master Clock

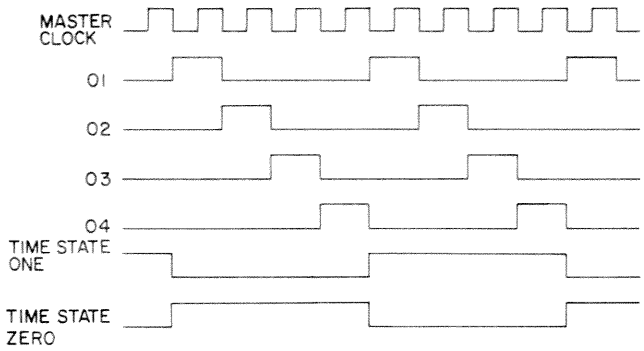
In Figure 4-3 the Master Clock oscillator feeds flip-flop #1, whose output is a 10MHz square wave. One clock signal is not enough to perform all the tasks within the computer. Therefore, the Master Clock is used to produce 4 more timing signals.



DG00741

Figure 4-4 Master Clock, Phase Timing, and Time State Timing

Notice how the clock phase's are produced over and over again by the master clock to keep track of our clock phases, flip-flop #4 is used to develop a signal called "Time State One" or "Time State Zero". This provides a means of specifying which group of phase signals are being used. Below is a diagram of the complete set of timing signals developed.



DG00742

These timing signals would be used to perform a function within the machine along with the IR decodes. For example:

a) At time state zero and phase 2 numbers could be added together.

b) At time state one, phase 1 data could be parallel transferred from one register to another.

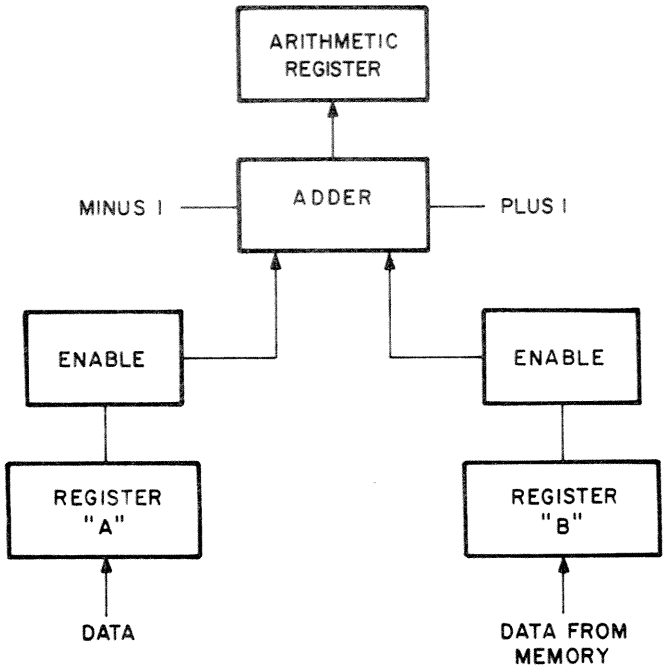
3. Program Counter
Controls program sequence. Within the memory of a computer is a program which will cause the computer to perform a function. The program could be used to calculate payroll or analyze sales for a company or keep track of inventories. The program is made up of individual instructions, in logical order, to perform the desired function.

If we had a 10 instruction program it would be stored in memory as follows:

ADDRESS	1.	Inst. #1
ADDRESS	2.	Inst. #2
ADDRESS	3.	Inst. #3
ADDRESS	4.	Inst. #4
ADDRESS	5.	Inst. #5
ADDRESS	6.	Inst. #6
ADDRESS	7.	Inst. #7
ADDRESS	8.	Inst. #8
ADDRESS	9.	Inst. #9
ADDRESS	10.	Inst. #10

To perform the program, the program counter would start at address one and then be incremented to address two and so on to address ten. The program counter controls the sequence in which instructions are executed. The program counter is a register that contains the address of the instruction being executed. This address is then incremented by one to obtain the next instruction.

Arithmetic Section- performs all of the arithmetic and logical operations. The basic arithmetic section has 2 input registers, which hold the numbers used, an adder circuit which can only add, and a register to hold the output of the adder (answer).



DG00705

1. In adding two numbers together, one number would be in the "A" register and the other number would be in the "B" register. Upon command from the control section the two numbers are added in the adder and the resultant sum placed in the arithmetic register.

2. To perform subtraction, the minuend is placed in the "A" register and the subtrahend is placed in "B". Then the number in the "A" register is sent to the adder, the complement (1's) of the number in "B" is sent to the adder. Here we are performing subtraction by the process of addition using complementary arithmetic. Since we're using 1's complement a signal "Plus One" is enabled so that the correct answer is placed in the arithmetic register.

3. To add one to a number, the number would be placed in the "A" register. Then the "A" register would be enabled to the adder. The "B" register would be disabled so the adder sees 0's from "B". The plus one line is enabled and the number plus one is placed into the arithmetic register.

4. To subtract one from a number, the same procedure for addition is followed except the minus one line is brought up instead of plus one. So that the arithmetic register holds the number minus one.

5. A logical AND produces a one in a bit position if the contents of both registers are 1's in that position.

   Example: And 10110="A" Register
   01111="B" Register
   00110= Arithmetic Register

6. A logical OR produces a one in a bit position if the contents of either register has a one in that bit position.

   Example: "A" Register = 01011
   "B" Register = 10100
   Logical "OR" 11111

   Contents of Arithmetic Register is equal to 11111.

I/O SECTION
INPUT/OUTPUT

1. Allows communications between the computer and an external device, normally referred to as a peripheral device.

2. A special set of instructions is necessary to enable the computer to transfer data to and from peripheral devices. These instructions are called I/O instructions.

3. Two of the Basic I/O instructions are "DATA IN" and'DATA OUT".

4. "DATA OUT" enables the computer to talk to another device. This is a transfer of data from the computer to the peripheral device.

5. "DATA IN" allows transfer of data from a peripheral device to the computer.

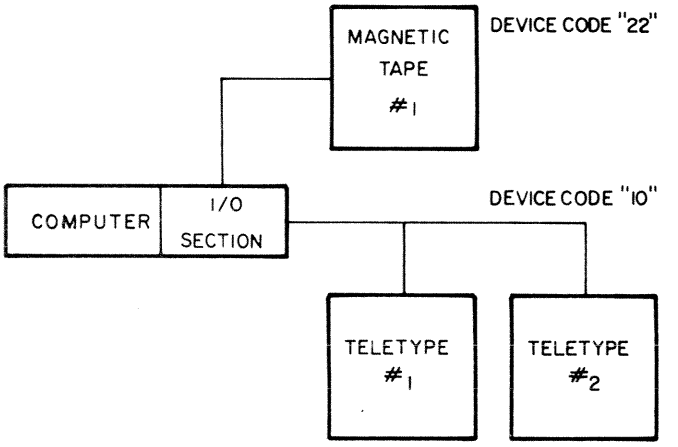
6. Since there are usually many types of peripheral equipment connected to a computer, there must be some way in which to specify which device the computer will talk to. As part of the instruction word, are "Device-Codes". Each peripheral device would have its own code. Teletypes could have a device code, say of "10". A magnetic tape could have a device code, say of "22". So if we did a data out with a device code of 22 the computer would only be talking with the magnetic tape unit and no one else.

7. Control signals allow the computer to control a peripheral device. The computer could be able to turn a device OFF or ON, move paper or know the status of a piece of peripheral equipment.

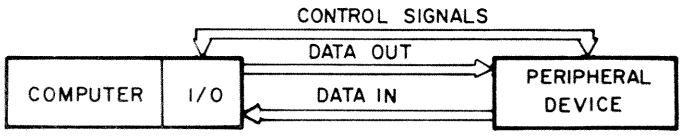
8. A DATA OUT to a teletype without knowing if the teletype is turned on, could result in losing all outputed data if the machine was turned off.

9. With control signals, the first step is to turn the teletype on and wait for a signal from the teletype that says its on. Then, output the data. When through outputting, another control signal could be sent to turn the teletype off.

10. The Data that comes from the computer to a peripheral device, or from a peripheral device to the computer must travel over special data lines. These data lines are separate from the control signals. The number of data lines depends on the type of peripheral device. One device may have 8 data lines while another will have 16 data lines.



DG00706



DG00707

MAJOR REGISTERS

General- A device which can receive information upon command, hold that information without modification and transfer it upon request.

a. Definition - consists of any number of bits up to the word length of the machine. Normally comprised of flip-flops, one per bit.

b. Functions
   1) Decode an instruction.
   2) Store or retrieve data from core memory.
   3) Govern program sequence.
   4) Provide an address for core memory.
   5) Modify data.

Types

a. Switch register - electro-mechanical switch register. The 16 bit console data switches fall into this type.

b. Indicator Register - visual console indication consisting of lamps and lamp drivers.

c. Memory - Ferrite core register (location or address) in memory.

d. CPU Registers - hardware registers consisting of flip-flops.

Typical Registers - Figure 4-5 illustrates a typical 3 bit register. All registers, regardless of size, operate basically in the same manner. On a clock pulse (herein labelled LOAD) each flip-flop either sets/resets according to its D input. The output of the register could represent a count, a Time state or an instruction code, etc. Various areas of the computer logic sampling this register would react accordingly.

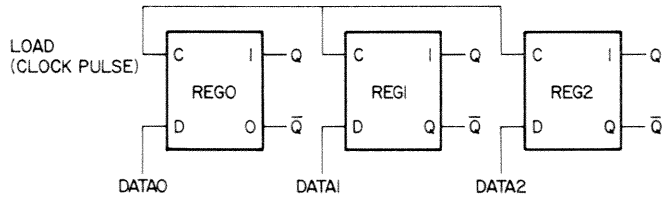


Figure 4-5 3 Bit Register

CPU Hardware Registers

a. PROGRAM COUNTER (PC) - Controls program sequence (order in which instructions are executed). The PC is normally incremented at the beginning of each instruction and therefore is actually pointing to the next instruction in the program sequence.

b. MEMORY ADDRESS (MA) - Supplies an address to core memory. This address represents the location in core which is currently being referenced. The MA is normally loaded from PC and therefore, receives an incremented address at the end of each instruction. The NOVA computer family of computers have a 15-bit MA capable of addressing 32K of core memory.

c. INSTRUCTION REGISTER (IR) - Contains the instruction word for the instruction currently being executed. The IR receives its data from the location in core memory as specified by MA. CPU logic decodes the word in IR and causes various events to occur in the execution of the instruction.

d. MEMORY BUFFER (MB) - Data Read from core memory is received by the MB. Data written into core memory is obtained from the MB. As its name implies, the MB acts as a buffer between memory and the CPU.

e. MEM REGISTER (MEM) - Actually a memory bus (not necessarily a flip-flop register) receiving its data from the MB and communicating with the CPU and in some cases, the Data Bus.

f. ACCUMULATOR (AC) - Involved in conjunction with the Adder in all Arithmetic and Logical computations performed by the computer.

g. CARRY (CRY) - a one bit register useful in detecting overflow. Overflow can be described as follows:

> Adding two 16 bit numbers (assuming a 16 bit machine) which results in a sum of 17 bits obviously creates a problem and results in the setting of Carry. The program monitoring the Carry bit in this case would detect overflow, rescale the numbers and repeat the addition.

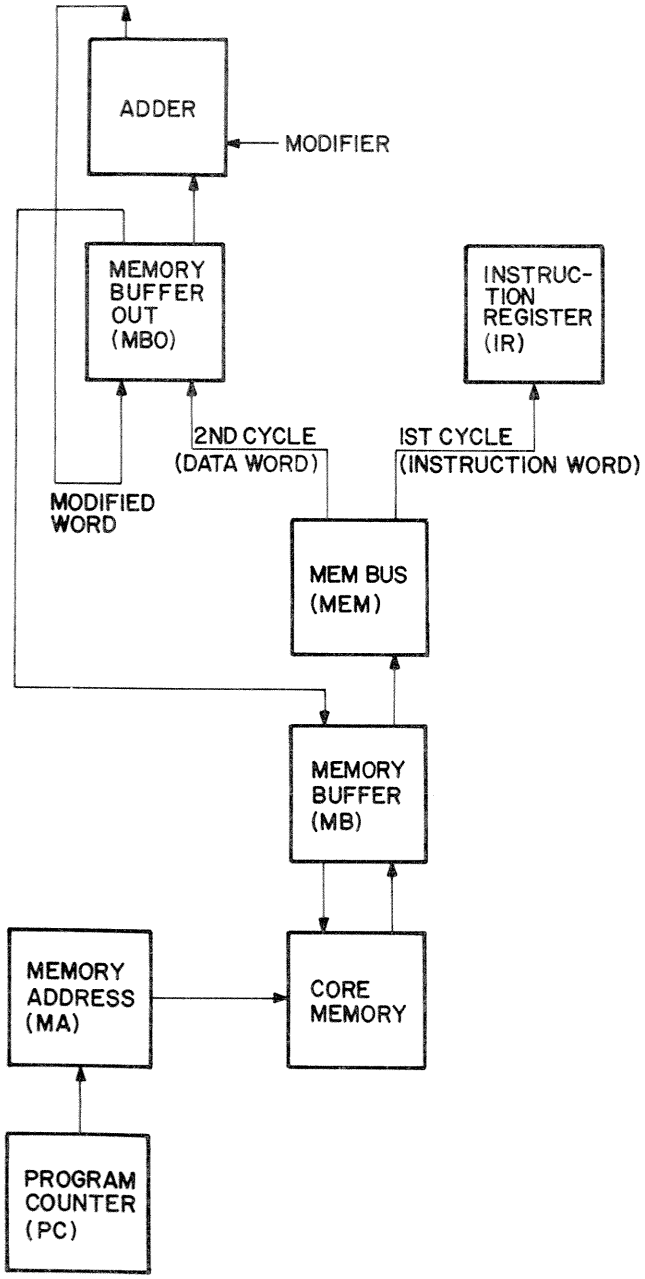
h. ARITHMETIC REGISTER (AR) - involved in arithmetic computations, and in some machines, similar to an accumulator. In the Nova family of computers it is a temporary register, retaining the result of a computation as performed by the Adder.

i. ACCUMULATOR BUFFER (ACB) - Temporary storage buffer receiving a result from the Adder and normally transferring this result to an Accumulator at the appropriate time.

j. MEMORY BUFFER OUT (MBO) - A CPU register that communicates with memory and the Data Bus.

k. MULTIPLIER QUOTIENT (MQ) - a register involved in hardware Multiply/Divide.

l. DATA BUS - 16 Data lines (assuming a 16 bit machine) that provide communication between the CPU and IO (Input/Output) devices. The bus is a Trans receiver (Bi-directional).



DG00743

Figure 4-6 Register Interrelation

Register Interrelation (Refer to Figure 4-6)

a. Figure 4-6 illustrates the path of data flow necessary to modify a data word. The particular instruction would be a 2 cycle instruction (covered in greater detail further on in this manual).

1) Cycle one - instruction word is retrieved from memory (address specified by MA) goes through the MB to MEM BUS and thence to the IR.

2) Cycle two - MA now contains address as specified by the instruction word. This is the address of the data word to be modified. The word is retrieved from memory through the MB to MEM BUS and to the MBO. From the MBO to the Adder where the word is modified and returned to MBO. From MBO the modified word is loaded into the MB and written into memory.

MAJOR STATES

General

a. Major States are in effect machine cycles.

b. When running the machine goes from state to state in logical order as required to execute the running program.

c. The machine can only be in one state at a given time.

d. The order of states is governed by a logical element called the Major States Device. This device and associated logic is continuously looking ahead, in effect, determining the future state.

States (Cycles)

a. KEY - a manual cycle entered as the result of a Key action originating at the console. Most console Keys result in the machine entering this cycle. The Key cycle is a non-memory, i.e., memory is not referenced.

b. KEYM (KEY MEMORY) - following the Key cycle this cycle is entered by those keys where referencing memory is a necessity. The KEYM cycle differs from the Key cycle in that it is a memory cycle storing or retrieving a word from memory. Starting a program results initially in a Key cycle, and once running this cycle as well as KEYM should not occur again until the program is halted and restarted.

c. FETCH - does as its name implies, fetches an instruction word from memory. The word retrieved from memory in this cycle can only be an instruction word.

d. DEFER - Indirect Addressing. To go indirectly through one address to reach another. This cycle is also involved in Auto-Indexing and Auto-Decrementing. These subjects are treated in greater detail in the next section of this manual.

e. EXECUTE - Two cycle instructions must follow the Fetch cycle with Execute. The instruction word is retrieved from core in Fetch and the instruction itself is executed in the Execute cycle. Two cycle instructions must reference memory twice to properly execute the instruction. This will be further discussed in later sections, however, memory can only be referenced once per cycle - hence an instruction requiring two memory references is a two cycle instruction.

f. PROGRAM INTERRUPT (PI) - to be discussed under later section.

g. DATA CHANNEL (DCH) - to be discussed under later sections.

TIME STATES

1. Individual Major States are further sub-divided into Time States.

2. The device generating individual Time States is referred to as the Time State Generator (TSG).

3. Time States are in turn sub-divided into Processor Time States.

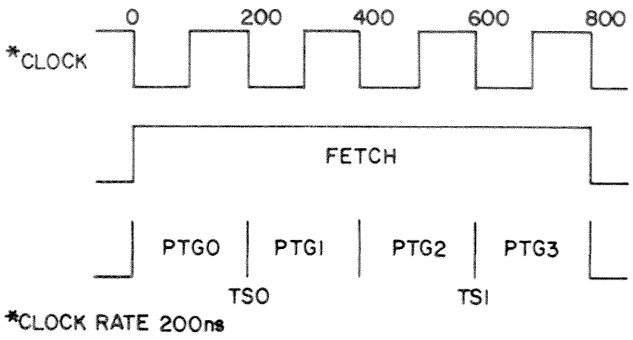
4. The device that sub-divides Time States is called the Processor Time Generator (PTG).

5. Figure 6 illustrates the basic timing of a typical mini-computer.

   a. Fetch cycle consists of Time State Zero (TS0) and Time State one (TS1).

   b. TS0 in turn is divided into Processor Time Zero (PTG0) and one (PTG1).

   c. TS1 in turn is divided into Processor Time two (PTG2) and three (PTG3).



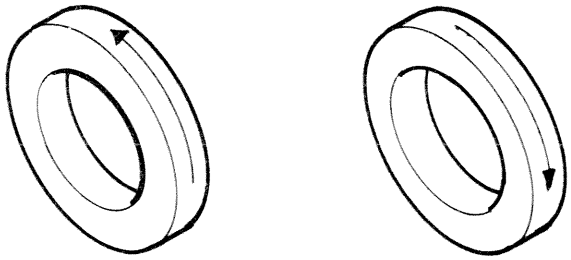
*CLOCK RATE 200ns

DG00719

MEMORY

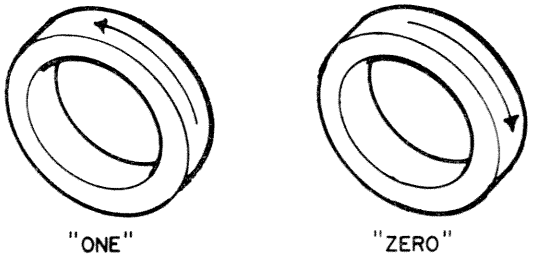
Magnetic Core Theory

Most computer memories are built around a simple structure called a core. A core is a donut shaped object made of ferrous material. The material make up of the core allows it to become magnetized and to hold its magnatism. Because of the cores shape, the magnetic field of the core will be in one of two directions as shown.



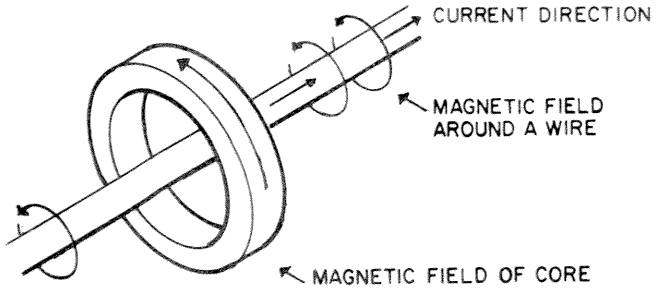
DG00718

Since a core will be used to store binary information, one magnetic field direction would indicate a "Binary One" and the opposite field direction would indicate a "Binary Zero".



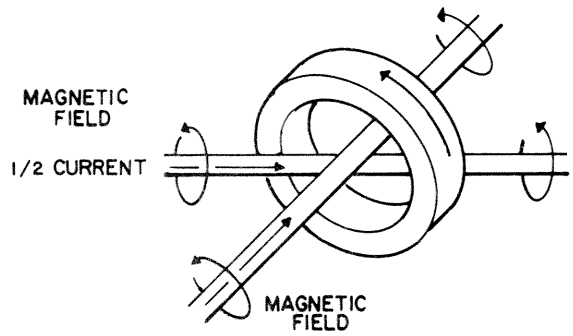
DG00716

The way we magnetize a core is by passing current through a wire. The magnetic field around the wire will magnetize the core.



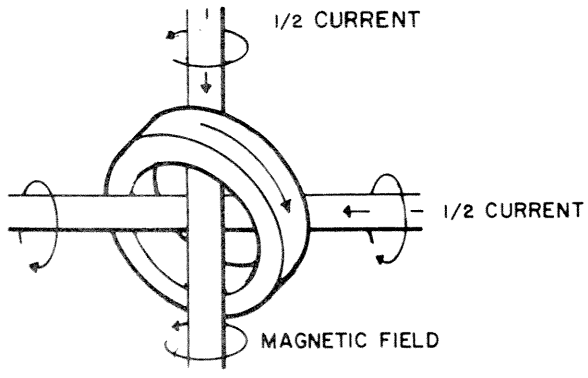
DG00717

A single wire is not appropriate for computer memories. Instead, two wires are used. Each wire will carry exactly half of the current necessary to magnetize the core. At the intersection of the two wires within the core, the magnetic fields of the wires add together and are enough to magnetize the core.



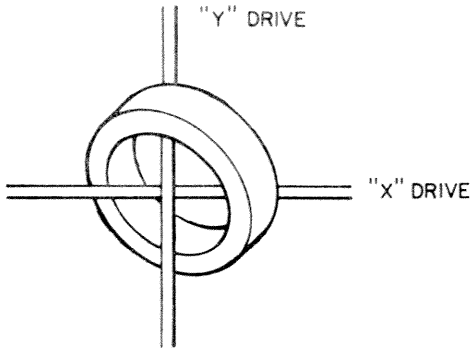
DG00715

To magnetize the core in the opposite direction we pass current through the wires in the opposite direction.



DG00713

The wires that are used to magnetize the core are usually called the "X" and "Y" drive lines. The "X" line is usually the horizontal wire and the "Y" drive is usually the vertical wire.



DG00714

Using this ability to magnetize a core in one of two directions, information can be stored and retrieved. This is the basis of a computer memory.

Definitions

a. MEMORY CYCLE - any major state wherein the CPU references memory. Consists of two separate identities, i.e., Read followed by Write, each occurring in separate time states.

b. READ - the first half of the memory cycle wherein a word is "Read" out of the specified memory location.

c. STROBE - with Read is part of the Read portion of the memory cycle.

d. WRITE - the second half of the memory cycle where the word is rewritten into the specified memory location. The rewritten word might be the same word as Read out or could be a modified word.

e. INHIBIT - A part of the write half of the memory cycle.

f. DESTRUCTIVE READ OUT - Reading a word out of the specified location in memory results in the zeroing of that location, consequently the word is destroyed. The write half of the cycle restores the word or writes in a new modified word. Destructive Read Out is not necessarily a desirable feature and memories were not designed to operate in this manner. Unfortunately, it is the "nature of the beast" and must be tolerated.

g. SENSE AMPLIFIERS - involved in the read portion of the cycle. Sense whether a given bit of the word is a "one" or a "zero".

h. FERRITE CORES - referred to as donuts, small circular magnetic cores capable of being magnetized in either of two states where magnetizing in one state represents a "zero" and in the opposite state a "one". Core diameters more or less determine the speed of the memory. Cores currently in use are 18mil in diameter.

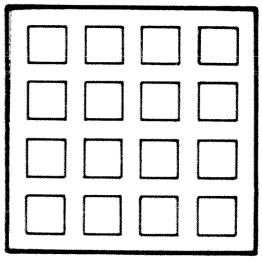
i. MEMORY ADDRESS (MA) - as mentioned previously, its content determines which location in core is to be referenced.

j. MEMORY BUFFER (MB) - communicates with core on both the Read and Write portions of the Memory Cycle. Receives the word from core memory when Reading, while the word written in core memory comes from the MB.

k. MEMORY SIZE - rated by the number of locations (addresses) within the memory. For instance, a 4K memory actually has 4096 decimal locations (7777 octal), while an 8K memory has 8192 decimal locations (17777 octal). Memories are currently available in 2K, 4K, 8K and 16K versions.

l. PLANE - a plane contains 4096 (decimal) cores representing for instance, all the bit zeroes of 4096 locations. It stands to follow that a 16 bit machine would necessarily require 16 planes, one for each bit.

m. MAT - a group of planes, one place for each bit of the machine word as illustrated in Figure 4-7.

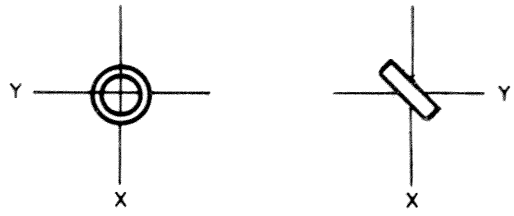


DG00711

Figure 4-7 Core Mat with 16 Planes

Description

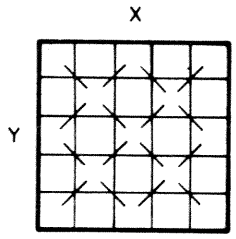
As stated previously, a plane contains 4096 decimal cores. Each core on the plane is mounted at the intersection of two wires called X and Y lines. A plane can be visualized as a fine wire screen or mesh (similar to a window screen) where a core is mounted at the intersection of each horizontal (and vertical) wire of the screen. (See Figure 4-8.)



DG00712

Figure 4-8 Core Mountings

A plane contains 64 x 64 = 4096 cores, and is actually a 64 x 64 matrix. Figure 4-9 illustrates a plane utilizing a 4 x 4 matrix for the purpose of explanation. The slanted lines represent the cores. Note that the cores are mounted in opposition to each other, this is to reduce the effect of noise. As stated previously, the actual plane would be a 64 x 64 matrix with 4096 cores.



DG00710

Figure 4-9 Core Plane

A 4K memory has 4096 locations. It is the function of the Memory Address Register (MA) to specify one of 4096 decimal locations when referencing memory. Figure 4-10 illustrates address selection.

MA	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	STACK SELECT			Y SELECT						X SELECT					

Figure 4-10 Address Selection

NOTE: that MA bits 1, 2, 3 select a particular 4K memory (Stack Select) this is illustrated in the following table:

MA 1 2 3	4 - 15	
0 0 0	XXXX	4K
0 0 1	XXXX	8K
0 1 0	XXXX	12K
0 1 1	XXXX	16K
1 0 0	XXXX	20K
1 0 1	XXXX	24K
1 1 0	XXXX	28K
1 1 1	XXXX	32K

Stack Select in the NOVA Line computers is accomplished by utilizing jumpers, each 4K memory board having its own unique jumper code reflecting MA bits 1, 2, 3. MA bits 1-3 address up to 8 x 4K memories for a total of 32K. Y select bits 4-9 have a range from 0 - 64 decimal or 0 - 77 octal. The same being true of X select bits 10-15. The uppermost location in a 4K memory is 7777 octal, MA bits 4-15 being all ones, therefore Y select = 77 and X select = 77 or 7777.

It should be noted at this point that Y select will in effect select one of 64 (77 octal) Y switches while X select selects one of 64 (77 octal) X switches. Figure 4-11 illustrates switch selection.

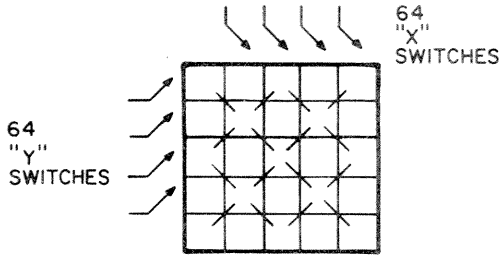


Figure 4-11 X and Y Switches

Read

Figure 4-12 illustrates Read portion of the memory cycle. a 1/2X Read current is passed through the selected X switch while a 1/2 Y Read current is passed through the selected Y switch. Each core (64) on the selected X and Y lines see a 1/2 current which is not sufficient to change the state of a core. It will be noted that the selected core sees a full Read current.

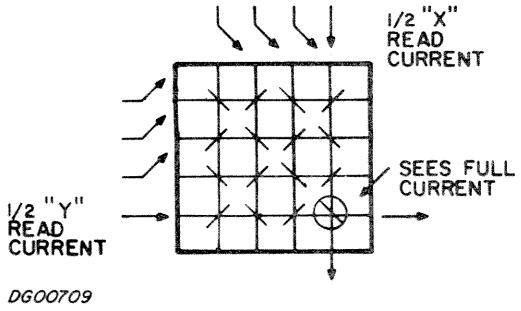
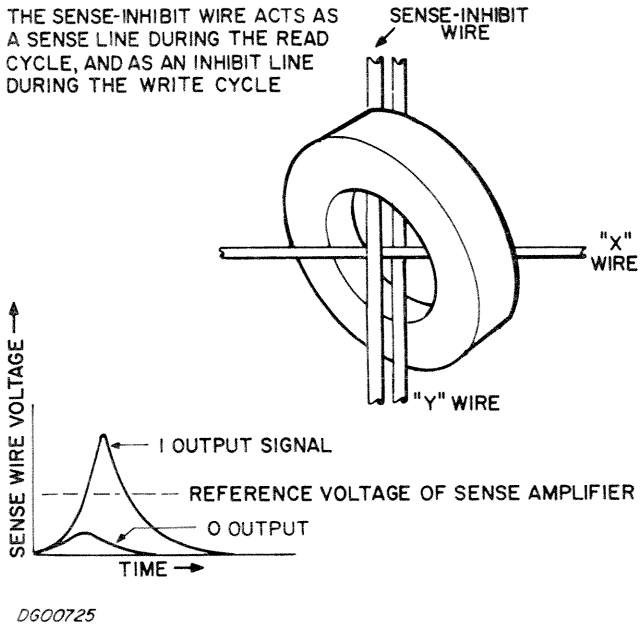


Figure 4-12 Read Currents

Sense Line

Referring to Figure 4-12, the selected core on the plane sees a full Read current, and if the core is magnetized in the one state it will change to the opposite state (zero). If the selected core is initially magnetized in the zero state, the direction of the Read current will not change its state, it remains a zero.

Figure 4-13 illustrates the addition of a third wire to the core plane. This third wire when Reading acts as a Sense Line. It is the function of this third wire to sense a change from a one to a zero. The Sense Line terminates in a Sense Amplifier which amplifies the weak signal on the Sense Line. The output of individual sense amplifiers Anded with strobe is the input to individual Memory Buffer Bits (MB). It should be noted that a 16 bit machine has 16 planes, consequently 16 Sense Lines and 16 Sense Amplifiers. Note also, at the conclusion of Read, each bit on a "one" has gone to zero, i. e., Destructive Read Out.



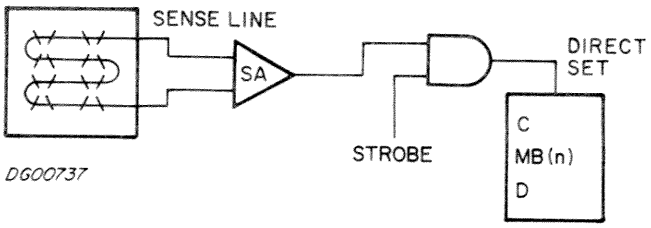


Figure 4-13 Sense Line and Amplifier

Write

Passing a 1/2 X Write current and a 1/2 Y Write current (in the opposite direction of Read) through the selected X and Y lines results in Writing in memory. Keep in mind that all bits in the selected location are now zero as a result of Reading. Reversing the direction of current would magnetize all bits in the selected location to the "one" state.

Inhibit Line - the third wire which while Reading in the Sense Line becomes on Write the Inhibit Line. The bits that are to remain a zero are prevented from switching to a one by the Inhibit Line. (See Figure 4-14.)

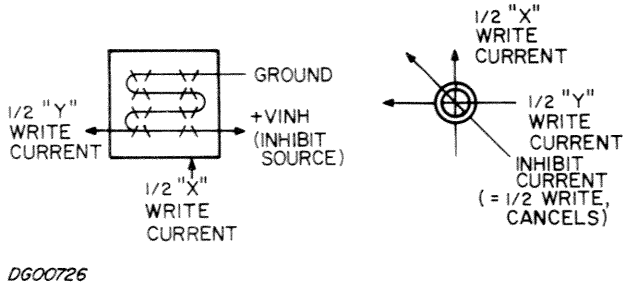
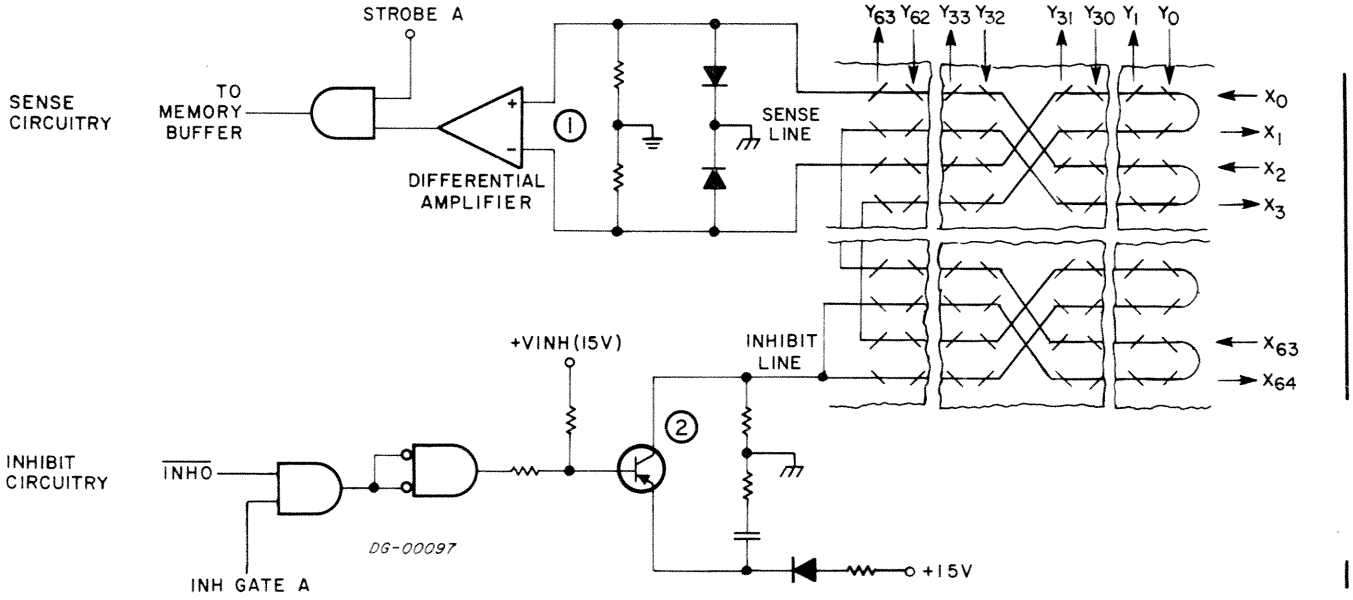


Figure 4-14 Write and Inhibit Currents



Simplified Schematic of the Core Memory's Sense and Inhibit Circuitry

This page intentionally left blank

SECTION V

INSTRUCTION SET - MEMORY REFERENCE

GENERAL

A group of six (6) instructions which reference a location in memory.

Instructions of this group will:

a. Move a data word from the CPU to a location in memory.

b. Move a data word from a location in memory to the CPU.

c. Modify the contents of a location in memory.

d. Alter the flow of the program, i.e., change the sequence of instruction execution.

INSTRUCTION WORD AND ADDRESSING

Decoding

The instruction word is fetched from the specified location in memory and is loaded into the Instruction Register (IR).

The six instructions decode as follows:

IR bits	0	1	2	3	4	
	0	0	0	0	0	JMP
	0	0	0	0	1	JSR
	0	0	0	1	0	ISZ
	0	0	0	1	1	DSZ
	0	0	1	A	C	LDA
	0	1	0	A	C	STA

If the first three bits are zero it will be noted that bits 3 and 4 decode one of four instructions. These instructions do not require an accumulator (AC) and are referred to as non AC instructions. With the first three bits different from zero, the instruction is either LDA or STA either of which require an accumulator (AC). It should be noted here that the Nova family of computers each have four (4) Accumulators (AC0, AC1, AC2, AC3). Bits 3 and 4 decode one of the 4 Accumulators for the LDA or STA instructions.

Cycles

The instructions JMP and JSR require one memory cycle for execution and are therefore referred to as Single cycle instructions.

Instructions ISZ, DSZ, LDA and STA require two memory cycles for execution and are therefore two cycle instructions.

Addressing

The content of IR bits 6-15 determine the location in core specified by the instruction.

Bits 6 and 7 decode one of four address modes.

Bits 8-15 contain either an address or a displacement, depending on the specified address mode.

Figure 5-1 illustrates the above.

IR 6-7 (Mode) | 8 - 9 - 10 - 11 - 12 - 13 - 14 - 15

Modes

6	7	
0	0	Page Zero
0	1	Relative
1	0	Base AC2
1	1	Base AC3

Figure 5-1 IR Bits 6-15 Breakdown

Page Zero Mode

1) Consists of memory locations 0-377 octal.
2) Bits 8-15 in this mode contain an address. It will be noted that with eight address bits available (8-15) the maximum address is 377 octal.
3) Page zero contains addresses that can be reached directly from any area of core.

Relative Mode

1) An address calculated relative to the current location.
2) Bits 8-15 in this mode contains a displacement vice an address as in the page zero mode.
3) To provide for both positive and negative displacements bit 8 is the sign of the displacement.
4) With bit 8 a sign bit, seven bits are available for displacement, thus the range is +177 and -200. With a negative displacement, bits 8-15 represent the 2's complement of the displacement.
5) Examples:

500/Inst 50, 1
Current location = 500
Displacement = +50
Calculated Address 550

NOTE: The 1 indicates Relative Mode (01)

500/Inst-50, 1
Current location = 500
Displacement -50
Calculated Address 430 (octal)

Base Mode

1) An address calculated relative to AC2 or AC3.
2) Bits 8-15 in Base Mode contains a positive or negative displacement with bit 8 the sign of the displacement.
3) AC2 and AC3 are Index Accumulators.
4) Example:

500/Inst 100, 2
Assume AC2 = 1000
Displacement = +100
Calculated Address 1100

NOTE: The 2 indicates Base Mode AC2 (10)

500/Inst -70, 3
Assume AC3 = 670
Displacement = -70
Calculated Address 600

NOTE: The 3 indicates Base Mode AC3 (11)

5) In Base Mode the displacement range is the same as in Relative Mode, i.e., +177, -200. The displacement is added to or subtracted from the contents AC2 or AC3.

MEMORY REFERENCE INSTRUCTIONS

JMP - Alters the normal program sequence, i.e., transfers program control from one area of the program to another. A single cycle instruction.

Examples:
500/JMP 300

a) Address Mode - Page Zero
b) Action - transfer control from location 500 to location 300.
c) Decoding

IR 0-1-2-3-4|5|6-7|8-9-10-11-12-13-14-15
0 0 0 0 0|0|0 0|1 1 0 0 0 0 0 0

JMP PG ZERO ADDRESS = 300

500/JMP 70, 1

a) Address Mode - Relative
b) Action - Transfer control from location 500 to location 570
c) Decoding

IR 0-1-2-3-4|5|6-7|8-9-10-11-12-13-14-15
0 0 0 0 0|0|0 1|0 0 1 1 1 0 0 0

JMP REL DSPL = 70

500/JMP 50, 2
AC2=700

a) Address Mode - Base AC2
b) Action - Transfer control from location 500 to location 750
c) Decoding

IR 0-1-2-3-4|5|6-7|8-9-10-11-12-13-14-15
0 0 0 0 0|0|1 0|0 0 1 0 1 0 0 0

JMP BASE AC2 DSPL = 50

NOTE: The address modes of the following instructions are varied to illustrate various modes.

JSR - Jump to a subroutine (Jump and save the return). Transfer program control to a subroutine and save the return address. The return address is stored in AC3.

Subroutine - a group of instructions utilized many times in a program. Written as one subroutine and called by the program via a JSR. Figure 5-2 illustrates the principle behind subroutines.

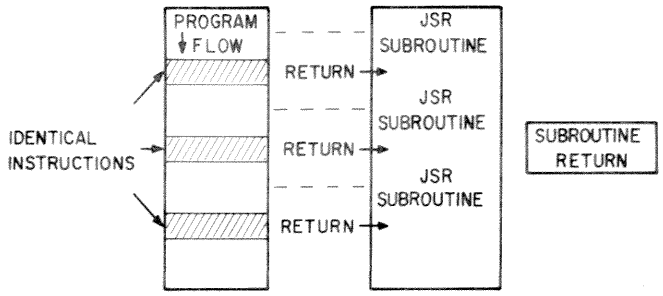


Figure 5-2 Subroutine Principles

If the 3 identical areas in Figure 5-2, contained 4 instructions each for a total of 12 instructions, it can be seen that a saving results by writing a subroutine consisting of 5 instructions (1 additional instruction for the return) plus 3 JSR's for a total of 8. The JSR is a single cycle instruction.

Example:

500/JSR 50, 1

a) Address Mode - Relative

b) Action-Jump to a subroutine at location 550.

c) Decoding

```
IR 0-1-2-3-4-|5|6-7-| 8-9-10-11-12-13-14-15
   0 0 0 0 1 |0|0 1 | 0 0  1  0  1  0  0  0
```

JSR REL

d) Octal - 004450

ISZ - Increment and skip if zero. Modifies a memory word. Increments the contents of the specified memory location. Skip the next instruction occurs if as a result of the increment the contents of the addressed location goes to zero. Normally used to increment a counter. ISZ is a two cycle instruction.

Examples:

500/ISZ -50, 1

a) Address Mode - Relative

b) Action - Increment the contents of memory location 430, skip if zero.

c) Decoding

```
IR 0-1-2-3-4|5|6-7 | 8-9-10-11-12-13-14-15
   0 0 0 1 0|0|0 1 | 1 1  0  1  1  0  0  0
```

ISZ REL (DSPL = -50)

d) Octal - 010730

DSZ - Decrement and skip if zero. Basically the same as ISZ except decrement. DSZ is a two cycle instruction.

Examples:

500/DSZ 200

a) Address Mode - Page Zero

b) Action - Decrement the contents of memory location 200, skip if zero

c) Decoding

```
IR 0-1-2-3-4|5|6-7 | 8-9-10-11-12-13-14-15
   0 0 0 1 1|0|0 0 | 1 0  0  0  0  0  0  0
```

DSZ PG ZERO (ADDRESS = 200)

d) Octal - 014200

LDA - Load Accumulator from the specified memory location. Transfers a data word from memory to the CPU. LDA is a two cycle instruction.

Examples:

500/LDA 1, 50, 2 (AC2 = 700)

a) Address Mode - Base AC2

b) Action - Load AC1 with the contents of memory location 750.

c) Decoding

```
IR 0-1-2|3-4|5|6-7 | 8-9-10-11-12-13-14-15
   0 0 1|0 1|0|1 0 | 0 0  1  0  1  0  0  0
```

LDA AC1 BASE AC2

d) Octal - 025050

STA - Store the contents of the specified Accumulator into the designated memory location. Transfer a data word from the CPU to Memory. STA is a two cycle instruction.

Examples:

500/STA 2, 377

a) Address Mode - Page Zero

b) Action - store the contents of AC2 into memory location 377.

c) Decoding

```
IR 0-1-2|3-4|5|6-7 | 8-9-10-11-12-13-14-15
   0 1 0|1 0|0|0 0 | 1 1  1  1  1  1  1  1
```

STA AC2 PG ZERO ADDRESS 377

d) Octal 050377

EXECUTION

Single cycle instructions are executed in a Fetch cycle.

Two cycle instructions require a Fetch cycle followed by Execute.

FETCH	EXECUTE
JMP	ISZ
JSR	DSZ
ISZ	LDA
DSZ	STA
LDA	
STA	

INDIRECT ADDRESSING

General

a) Memory Reference instructions can be indirectly addressed.

b) An indirect instruction is also referred to as a Deferred instruction.

c) Basically Indirect implies going through one location in memory to reach another.

d) Recalling address modes, Page zero mode contains addresses 0-377 with any location on the page directly addressable. Indirect addressing might be utilized to reach addresses that can not be reached directly.

e) In the previous discussion regarding Instruction descriptions it will be noted that IR bit 5 was a zero in all cases. IR bit 5 in a Memory Reference Instruction is the Defer or Indirect bit.

f) The Assembler convention for Defer is the "AT" sign @.

Examples:

In location 500 we have an instruction that must reference memory location 750. This it cannot do directly, however by going indirect through say location 300 (which must contain 750) we could reach 750.

```
500/LDA 2, @ 300
300/750
```

a) Address Mode - Page Zero, Indirect.

b) Action - Load AC2 indirect through 300. This instruction would load AC2 with the contents of memory location 750.

c) The contents of memory location 300 (750) is in this case treated as the effective address of instruction.

d) Decoding

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

LDA AC2 PG ZERO ADDRESS 300

NOTE: Bit 5 the defer/Indirect bit is set

```
500/STA 2, @ 150
150/1777
```

a) Address Mode - Page Zero, indirect.

b) Action - Store the contents of AC2 indirect through 150. This instruction would store the contents of AC2 in memory location 1777.

```
500/ISZ @ 50, 1
550/3000
```

a) Address Mode - Relative Indirect

b) Action - ISZ indirect through location 550. This instruction would increment the contents of memory location 3000. Note that the calculated address is 550.

Defer Cycle

Single cycle instructions indirectly addressed become two cycle instructions Fetch followed by Defer.

Two cycle instructions indirectly addressed become three cycle instructions, Fetch, Defer and Execute.

Multi-Level Defer

One indirect followed by another.

Bit 0 of the addressed location is the multi-level Defer bit.

Example:

```
500/JMP @ 300
300/000700
700/next instruction
```

a) One level of Defer. Bit 0 of location 300 is a zero.

b) The next instruction executed would come from location 700.

500/JMP @ 300
300/100700
700/1000
1000/next instruction

a) Two levels of Defer. Bit 0 of location 300 is a one. This would add one more Defer cycle and cause the CPU to go indirect through location 700.

b) The next instruction executed would come from location 1000.

AUTO INDEX

Auto Increment

Memory locations 20-27 when addressed indirectly are auto increment registers.

Example:

JMP @ 20
20/1777

a) The contents of location 20 is incremented to 2000, and becomes the effective address of the JMP.

b) 2000 is written into location 20.

LDA 2, @ 23
23/725

a) The contents of location 23 is incremented to 726, and becomes the effective address of the LDA.

b) 726 is written into location 23.

Auto Decrement

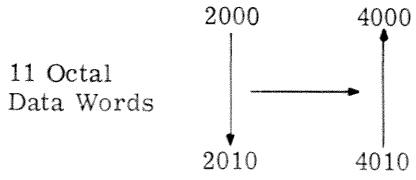
Memory locations 30-37 when addressed indirectly are auto decrement registers.

Basically the same as auto increment except decrement.

Generally useful for incrementing or decrementing addresses.

Programming Example:

a) It is desired to move 11 octal data words in core from one area to another in reverse order.



b) Routine

```
LDA  2, @ 20    20/ 1777
STA  2, @ 30    30/ 4011
ISZ  200        200/ -11
JMP  - 3, 1
```

c) Explanation

1) Note that Location 20 (Auto Increment) contains a number one less than that desired while Location 30 has a number one greater than that desired. Location 200 has a count of -11 representing the number of words to be moved.

2) The contents of Location 2000 is loaded (LDA) into AC2 (1777+1). The contents of AC2 is stored (STA) in Location 4010 (4010-1). The ISZ increments Location 200 but does not yet skip (not 0). The JMP -3, 1 (1-3, Relative mode) returns to the LDA instruction getting the next data word. The Program would remain in this loop until ISZ zeroes Location 200 at which time the JMP would be skipped and the program would be released from the Loop.

This page intentionally left blank

SECTION VI

ARITHMETIC LOGICAL INSTRUCTIONS (ALC)

GENERAL

1. A group of instructions that perform Arithmetical or Logical operations.
2. Instructions in this class do not reference memory.
3. ALC Instructions operate on data already in the Accumulators.
4. All operations are performed between two Accumulators.
   a) Example:
      ADD ACS, ACD
   b) The operation is ADD
   c) ACS - Source Accumulator (ACS0, ACS1, ACS2, ACS3)
   d) ACD - Destination Accumulator (ACD0, ACD1, ACD2, ACD3)
5. The result goes to the Destination Accumulator with the Source undisturbed.
6. ACS and ACD may be the same Accumulator.
7. ALC's are single cycle instructions, executed within a Fetch.

INSTRUCTION WORD AND DECODING

1. IR bit 0 set (1) signifies to the machine that the instruction just retrieved from core is an ALC.
2. IR bits 1-2 decode one of four Source Accumulators (ACS).
3. IR bits 3-4 decode one of four Destination Accumulators (ACD).
4. There are 8 ALC instructions. These are decoded by IR bits 5-6-7.
5. Figure 6-1 illustrates the above.

IR	0	1 - 2	3 - 4	5 - 6 - 7
	1			
	ALC	ACS	ACD	OPERATION

Figure 6-1 ALC Decoding

6. Instructions

5	6	7	
0	0	0	COM
0	0	1	NEG
0	1	0	MOV
0	1	1	INC
1	0	0	ADC
1	0	1	SUB
1	1	0	ADD
1	1	1	AND

7. Description

Bits					
5	6	7			
0	0	0	ACS, ACD	COM	;compute the 1's complement ;of the number in ACS, and ;put the result into ACD.
0	0	1	ACS, ACD	NEG	;compute the 2's complement ;(negative) of the number in ;ACS, and put the result into ;ACD.
0	1	0	ACS, ACD	MOV	;copy (move) the number in ;ACS into ACD.
0	1	1	ACS, ACD	INC	;add one (increment) to the ;number in ACS and put the ;result into ACD.
1	0	0	ACS, ACD	ADC	;add the 1's complement of ;the number in ACS to the ;number in ACD and put the ;answer into ACD.
1	0	1	ACS, ACD	SUB	;subtract the number in ACS ;from the number in ACD and ;put the answer into ACD. ;Subtraction is performed by ;taking the 1's complement ;of the number in ACS adding ;this to the number in ACD, ;then adding 1 to the result. ;(2's complement subtraction).
1	1	0	ACS, ACD	ADD	;add the number in ACS to ;the number in ACD and put ;the answer into ACD.
1	1	1	ACS, ACD	AND	;perform a logical AND oper- ;ation between the number in ;ACS and the number in ACD ;and put the result into ACD.

SECONDARY OPERATIONS

1. Load and NO Load

a) IR bit 12 of an ALC is the Load/No Load bit.

1) IR 12 a "zero"-Load the result of an ALC into the Destination Accumulator.

2) IR 12 a "one" - Do Not Load the result of an ALC into the Destination Accumulator.

3) The sharp sign (#) is the Assembler convention for No Load.

4) The ability to Load or Not Load provides the Programmer the means to compare two data words without disturbing either number by not loading.

Examples:

1) ADD 3, 2

a) Add the contents of ACS3 to the contents of ACD2, put the sum in ACD2.

b) Instruction word

IR	0	1 - 2	3 - 4	5 - 6 - 7	8 - 9	10 - 11	12
	1	1 1	1 0	1 1 0	0 0	0 0	0
	ALC	ACS3	ACD2	ADD			LOAD

2) ADD # 3, 2

a) Add the contents of ACS3 to the contents of ACD2, do not load the sum into ACD2.

b) Instruction word

IR	0	1 - 2	3 - 4	5 - 6 - 7	8 - 9	10 - 11	12
	1	1 1	1 0	1 1 0	0 0	0 0	1 NO
	ALC	ACS3	ACD2	ADD			LOAD

Load Modes

a) IR bits 8-9 determine the manner in which the result of an ALC is loaded into the Destination Accumulator.

b) Modes

IR 8	9	
0	0	Load Direct
0	1	Load Shifted Left (L)
1	0	Load Shifted Right (R)
1	1	Load Swapped (S)

c) Load Direct

1) IR bits 8-9=00

2) The result of an ALC is loaded into the Destination Accumulator (ACD) directly (as is, no modification).

3) Example - MOV 2, 3

a) Move the contents of ACS2 directly into ACD3 (no modification).

b) Instruction word

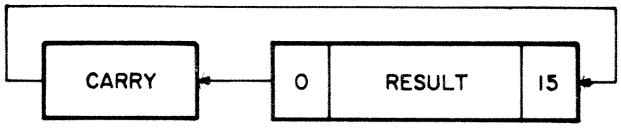
IR	0	1 - 2	3 - 4	5 - 6 - 7	8 - 9	10 - 11	12
	1	1 0	1 1	0 1 0	0 0	0 0	0
	ALC	ACS2	ACD3	MOV	DIR		LOAD

Load Shifted Left

1) IR bits 8-9=01

2) The result of an ALC is loaded into the Destination Accumulator (ACD) shifted Left one.

3) Shift Left



a) On a shift operation the result and the Carry are shifted together (17 bit shift). The Carry is a one bit extension of the result and indicates overflow. Carry is discussed in greater detail further in this section.

Example:

SUBL 0, 1

a) Subtract the contents of ACS0 from ACD1, put the difference shifted Left (note the "L" of the instruction word) into ACD1.

b) Instruction word

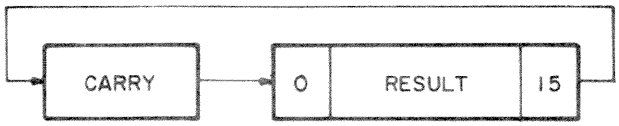
IR	0	1 - 2	3 - 4	5 - 6 - 7	8 - 9	10 - 11	12
	1	0 0	0 1	1 0 1	0 1	0 1	0
	ALC	ACS0	ACD1	SUB	LEFT		LOAD

Load Shifted Right

1) IR bits 8-9=10

2) The result of an ALC is loaded into the Destination Accumulator (ACD) Shifted Right one.

3) Shift Right



Example:

INCR 1,3

a) Increment the contents of ACS1 and load into ACD3 Shifted Right. (Note the "R" of the instruction word).

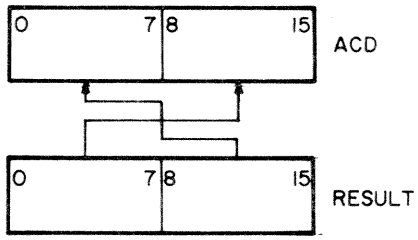
b) Instruction word

```
IR   0 1 - 2  3 - 4 5 - 6 - 7   8 - 9 10 - 11 12
     1 0   1  1   1 0   1   1   1   0  0    0  0
   ALC ACS1 ACD3   INC    RIGHT          LOAD
```

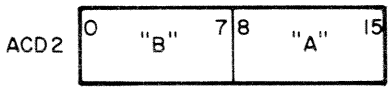
Load Swap

1) IR bits 8-9=11

2) The result of an ALC is loaded into the Destination Accumulator (ACD) with the right and left Bytes of the result swapped.



Load Swap is useful when outputting Data to an IO device. For example, ACD2 might contain 2 characters A and B as shown:



a) We would output character "A" to the device and with a MOVS 2,2 exchange the contents of ACD2 and output character "B".

Carry and Carry Base

a) A one bit extension of an ALC result informs the program of an overflow. In a 16 bit machine adding two numbers whose sum is greater than 16 bits would result in overflow. The capacity of the registers (16 bits) has been exceeded. CPU logic simply detects overflow, sets a Flip-flop called Carry, and frankly, could care less. It is the function of the Program to concern itself with overflow and Carry.

b) IR bits 10-11 allow the programmer to establish a base for Carry as part of an ALC operation.

c) Base

IR 10-11			MNEMONIC
0	0	AS IS	-
0	1	ZERO	Z
1	0	SET	O
1	1	COMPLEMENT	C

d) Overflow complements the Carry Flip-flop.

e) Examples and description:

Bits 10-11					
0	0	AS IS	1,2	-	;the base value of the ;Carry bit is whatever ;the value of the Carry ;bit happens to be at the ;time this instruction ;is encountered. An ;overflow causes this ;base value to be ;completed.
0	1	ZERO	1,2	Z	;the base value of the ;Carry bit is forced to ;a zero. An overflow ;causes the Carry bit ;to become 1.
1	0	ONE	1,2	O	;the base value of the ;Carry bit is forced to ;a 1. An overflow ;causes the Carry bit ;to become zero.
1	1	COMPLEMENT	1,2	C	;the base value of the ;Carry bit is the com- ;plement of whatever ;the value of the Carry ;bit happens to be at ;the time this instruc- ;tion is encountered. ;An overflow causes ;this base value to be ;completed.

ALC Skips

a) IR bits 13-14-15 of an ALC instruction decode of eight ALC Skips.

b) It is possible to Skip on the state of Carry, the result of an operation, either or both.

c) Skips

13	14	15	
0	0	0	Never Skip
0	0	1	SKP
0	1	0	SZC
0	1	1	SNC
1	0	0	SZR
1	0	1	SNR
1	1	0	SEZ
1	1	1	SBN

d) Description

Bits 13	14	15	If the test mnemonic is	then
0	0	0	(nothing)	no test is made and the testing phase of the instruction is ignored.
0	0	1	SKP	(unconditional SKip) no test is made and the testing phase of the instruction is ignored. However, the next instruction in the program sequence is skipped.
0	1	0	SZC	(Skip on Zero Carry) a test is made on the Carry bit resulting from the operation. If this new Carry bit is zero, the next instruction in the program sequence is skipped.
0	1	1	SNC	(Skip on Non-zero Carry) a test is made on the Carry bit resulting from the operation. If this new Carry bit is nonzero, the next instruction in the program sequence is skipped.
1	0	0	SZR	(Skip on Zero Result) a test is made on the 16-bit result from the operation. If this 16-bit result is zero, the next instruction in the program sequence is skipped.
1	0	1	SNR	(Skip on Non-zero Result) a test is made on the 16-bit result from the operation. If this 16-bit result is nonzero, the next instruction in the program sequence is skipped.
1	1	0	SEZ	(Skip if Either or both are Zero) a test is made on the 16-bit result and the new Carry bit. If either or both are zero, the next instruction in the program sequence is skipped.
1	1	1	SBN	(Skip if Both are Non-zero) a test is made on the 16-bit result and the new Carry bit. If both are nonzero, the next instruction in the program sequence is skipped.

ALC INSTRUCTION WORD

1. As seen, all 16 bits of an ALC instruction have meaning to the CPU logic during the execution of the Instruction.

2. Examples of ALC Instruction words:

   a) SUBCL #2, 1, SZR

      1. The instruction is Subtract (ACS2, ACD1).
      2. Carry Base is its complement (C).
      3. Load Shifted Left (L).
      4. Don't Load (#).
      5. Skip a zero result (SZR).

```
IR  0   1-2  3-4  5-6-7  8-9   10-11  12   13-14-15
    1   1 0  0 1  1 0 1  0 1   1  1   1    1  0  0
  ALC ACS2 ACD1  SUB   LEFT  COMP LOAD   SZR
```

      6. Octal 146574

   b)ADCOR 3, 2, SNC

      1) The instruction is add complemented (ACS3, ACD2).
      2) Carry Base is SET (0).
      3) Load Shifted Right (R).
      4) Load.
      5) Skip on a non-zero Carry.

ALC ADDER CONCEPT

1. Figure 6-2 illustrates a typical mini-computer Adder and associated registers.

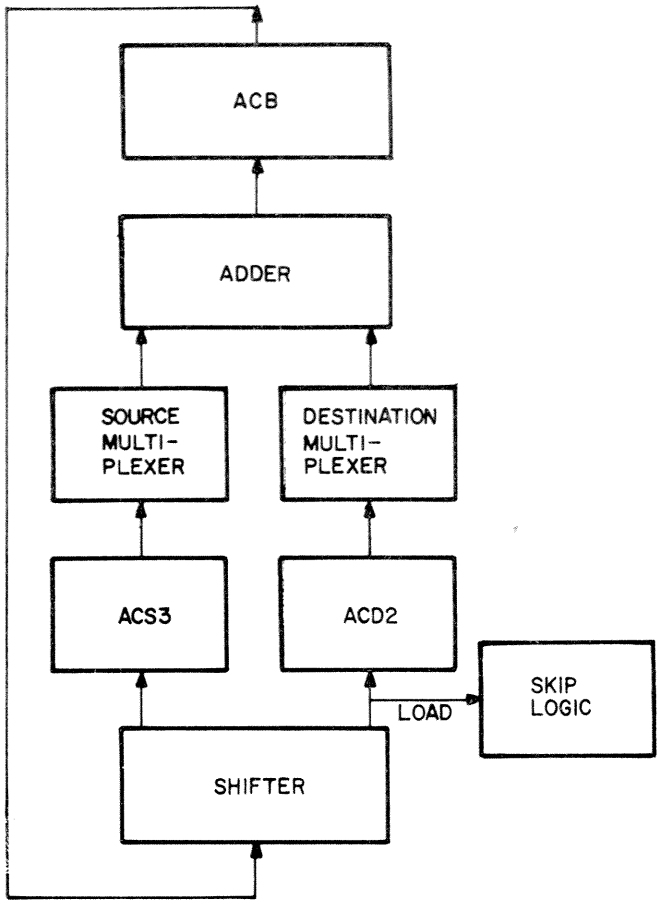
   a) The Adder receives two inputs simultaneously from two multiplexers. The Source and Destination multiplexers are communicating with the Accumulators.

   b) The Adder performs the addition and its output representing the sum is loaded into ACB.

   c) ACB (Accumulator Buffer) is a temporary Buffer, holding the sum while a decision is made as to the method of loading and whether to load or not.

   d) The ACB communicates with the Shifter Element. It is here that the load direct, Right or Swapped is accomplished.

   e) The ALC Skip Logic looks at the output of the Shifter and determines whether a Skip condition has been or not been met.



DG00720

Figure 6-2 Adder Concept (ADD 3, 2)

2. In summation, it can be seen that an ALC instruction is powerful and versatile, performing up to 5 functions in one instruction word.

This page intentionally left blank

SECTION VII

INPUT-OUTPUT (IO) INSTRUCTION

GENERAL

1. The capability of a computer would be limited if it had access only to its Memory and Arithmetic Unit.
2. Input-Output (IO) instructions allow the computer to communicate with the "outside world". Data can be transferred from the CPU to a device and from a device to the CPU. Thus, the name Input-Output, for the CPU "Outputs" data to a device while a device "Inputs" data to the CPU. It should be noted here that references to In or Out are always with respect to the CPU.
3. Data is transferred from or to Accumulators on communications between the CPU and devices.

INSTRUCTIONS

Instruction Word

IR 0	1-2	3-4	5-6-7	8-9	10-11-12-13-14-15
0	1 1	X X	X X X	X X	X X X X X X
IO		AC	INST		DEVICE CODE

a) IR bits 0 - 1 - 2 = 011 specifies IO.

b) IR bits 3 - 4 decode one of four accumulators.

c) IR bits 5 - 6 - 7 decode one of eight instructions.

d) IR bits 8 - 9 to be discussed further in this section.

e) IR bits 10 - 15 are Device Code bits.

Instruction Set

IR 5-6-7	Instruction
0 0 0	NIO
0 0 1	DIA
0 1 0	DOA
0 1 1	DIB
1 0 0	DOB
1 0 1	DIC
1 1 0	DOC
1 1 1	SKP

a) All instructions less NIO and SKP require an Accumulator as part of the instruction word.

b) The CPU has the ability to communicate with three 16 bit Registers. Register A, B or C.

c) The "I" in the instruction Mnemonic indicates incoming while the "O" outgoing.

IO Pulses

a) Decode by IR bits 8 - 9

IR 8 - 9	Pulse	MNE
0 0	NONE	--
0 1	START	S
1 0	CLEAR	C
1 1	IO PLS	P

b) IR bits 8 - 9 decode a pulse on any instruction other than SKP. The bits determine a Skip condition on the SKP instruction.

c) Pulse Description

1) START (S) directed to a device effectively puts the device in motion, i.e., Start a device.
2) CLEAR (C) directed to a device puts the device into an IDLE or NULL state. Normally issued to a device at the completion of an operation
3) IOPLS (P) normally not seen by standard devices. Provided for the benefit of those designing special interfaces.

Device Codes

a) General

1) Each device tied to the mainframe sees an IO Instruction as issued by the CPU. The device whose code is in IR bits 10 - 15 responds.
2) The six bits (10 - 15) allotted to device selection allow for a maximum of 64 decimal codes (77 octal).

b) Typical devices, codes and Mnemonics (MNE)

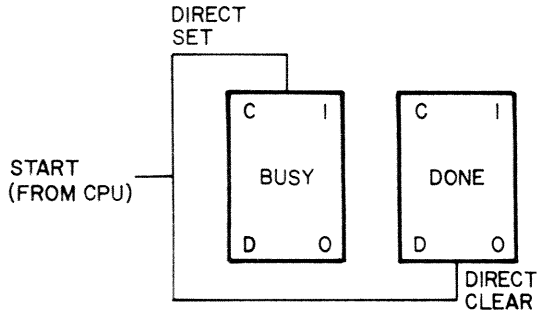
Device	Code	MNE
High Speed Paper Tape Reader	12	PTR
High Speed Paper Tape Punch	13	PTP
Teletype (Input)	10	TTI
Teletype (Output)	11	TTO
DISK	20	DSK
Magnetic Tape	22	MTA

Typical Device Flags

a) Each device has (among others) two flags, the BUSY and DONE. These are normally D Type Flip-Flops.

b) A START pulse directed to a device will SET the BUSY flag and CLEAR the DONE. (See Figure 7-1.)

c) The setting of BUSY normally is all that is required to put a device into motion.



DG00721

Figure 7-1 Done and Busy Flip-flops

INSTRUCTION DESCRIPTION

NIO

a) No Input-Output - i.e., does not transfer data.

b) Basically a command or control instruction. Utilized for instance to put a device into operation.

c) This command without the Mnemonic S, C or P is in itself a NOP (NO Operation). No pulse is Generated, therefore the specified device does not respond.

d) Examples:

1) NIOS PTR (device code 12).

Instruction Word

IR 0-1-2	3-4	5-6-7	8-9	10-11-12-13-14-15
0 1 1	0 0	0 0 0	0 1	0 0 1 0 1 0
IO	(NOAC)	NIO	STRT	DEVICE CODE 12

1. Action - A start pulse is directed to the High Speed Paper Tape Reader (PTR). Busy is SET and the reader will fetch one character (one line) from the paper tape.

2. NIOC PTR

Instruction Word

IR 0-1-2	3-4	5-6-7	8-9	10-11-12-13-14-15
0 1 1	0 0	0 0 0	1 0	0 0 1 0 1 0
IO	(NOAC)	NIO	CLR	DEVICE CODE 12

1) Action - A clear pulse is directed to the High Speed Paper Tape Reader. The effect of this pulse is to clear both Reader flags, Busy and Done. The reader is now in an idle/null state.

3) NIOP - Not normally seen by standard devices, principally for special device interfaces.

DIA

a) Data In Register A.

b) An input instruction.

c) Transfers a data word from a device buffer into the Specified Accumulator.

d) Example:

1) DIA 3, PTR (Device Code 12)

Instruction Word

IR 0-1-2	3-4	5-6-7	8-9	10-11-12-13-14-15
0 1 1	1 1	0 0 1	0 0	0 0 1 0 1 0
IO	AC3	DIA	NONE	DEVICE CODE 12

1) Action - Transfer a data word from PRT character Buffer. Put the word into Accumulator three (AC3).

2) DIA 2, TTI (Device Code 10)

Instruction Word

IR 0-1-2	3-4	5-6-7	8-9	10-11-12-13-14-15
0 1 1	0 0	0 0 1	0 0	0 0 1 0 1 0
IO	AC2	DIA	NONE	DEVICE CODE 10

1) Action - transfer a character in TTI's buffer to Accumulator two (AC2).

2) DOA

a) Data Out Register A.

b) An Output instruction.

c) Transfer a data word from the specified AC to the device Buffer.

d) Example:

1) DOA 0, PTP (Device Code 13)

Instruction Word

IR	0-1-2	3-4	5-6-7	8-9	10-11-12-13-14-15
	0 1 1	0 0	0 1 0	0 0	0 0 1 0 1 1
	IO	AC0	DOA	NONE	DEVICE CODE 13

1) Action - Transfer a character from AC0 to the High Speed Paper Tape Punch (PTP) Buffer.

DIB

a) Data In Register B

b) An Input Instruction

c) Transfer a data word from device Register B to the Specified Accumulator.

d) Same as the DIA except Register B.

e) At this point, the following should be mentioned. The smaller devices such PTR, PTP and Teletype normally communicate via Register A. These devices transmit a character of 8 bits. They in effect utilized one-half of Register A. The larger devices such as Disks, Magnetic Tapes, etc., due to their complexity, might utilize all three Registers A, B as well as C. This will be further discussed in the chapter dealing with Data Channel.

DOB

a) Data Out Register B.

b) An Output instruction.

c) Transfer a data word from the designated Accumulator to device Buffer B.

DIC

a) Data IN Register C.

b) An Input instruction.

c) Transfer a data word from device Buffer C to the designated Accumulator.

DOC

a) Data Out Register C.

b) An Output instruction.

c) Transfer data word from the designated Accumulator to device Buffer B.

SKP

a) IO Skip instruction.

b) Does not pass data, therefore does not require an Accumulator.

c) IR Bits 8 - 9 have an entirely different meaning for this instruction than the preceding seven.

d) Decoding and Mnemonics.

IR 8 - 9	MNE
0 0	BN
0 1	BZ
1 0	DN
1 1	DZ

1) B and D refer to the two device flags, Busy and Done.

2) We have the ability to skip another state of Busy, not equal to zero (N) or equal to zero (Z). The same is true of the Done flag.

e) Examples:

1) SKPBN PTR.

Instruction Word

IR	0-1-2	3-4	5-6-7	8-9	10-11-12-13-14-15
	0 1 1	0 0	1 1 1	0 0	0 0 1 0 1 0
	IO	NONE	SKP	BN	DEVICE CODE 12

1) Action - Skip the following instruction if the Reader's Busy flag is set. Skip if the Reader is Busy.

2) SKPBZ PTR

Instruction Word

IR	0-1-2	3-4	5-6-7	8-9	10-11-12-13-14-15
	0 1 1	0 0	1 1 1	0 1	0 0 1 0 1 0
	IO	NONE	SKP	BZ	DEVICE CODE 12

1) Action - Skip the following instruction if the Reader's Busy flag is zero (Clear). Skip if the Reader is not Busy (Done).

3) SKPDN PTR

Instruction Word

IR	0-1-2	3-4	5-6-7	8-9	10-11-12-13-14-15
	0 1 1	0 0	1 1 1	1 0	0 0 1 0 1 0
	IO	NONE	SKP	DN	DEVICE CODE 12

1) Action - Skip the following instruction if the Reader's Done flag is Set. Skip if the Reader is Done.

2) SKPDZ PTR

Instruction Word

IR	0-1-2	3-4	5-6-7	8-9	10-11-12-13-14-15
	0 1 1	0 0	1 1 1	1 1	0 0 1 0 1 0
	IO	NONE	SKP	DZ	DEVICE CODE 12

1) Action - Skip the following instruction if the Reader's Done flag is zero (BUSY).

The above Skip instructions can be directed to any device, not necessarily restricted to the Reader as illustrated.

PROGRAMMING EXAMPLES

1. General

a) The CPU and various devices communicate over an IO BUS.

b) IO BUS Configuration.

1) Control Lines - a number of Lines that govern the actions of devices. Start and Clear as generated by the CPU are examples of control lines.

2) Data Bus - Bi-directional, capable of transferring data in either direction. The Bus in a 16 bit machine would include 16 data lines (one per bit). The four Accumulators (in the CPU) communicate with the Bus on the CPU end while device registers A, B or C are tied to the Bus on the device end.

2. Program Control of Paper Tape Reader.

a) The following illustrates the reading of one character from the Paper Tape.

```
NIOS   PTR
SKPDN  PTR
JMP.  -1
DIAS 2, PTR
```

b) Explanation - NIOS PTR

1) A START pulse is directed to the Reader, setting its BUSY, clearing its DONE. This puts the Reader in motion and tape moves.

2) The character Read from Tape is loaded into the Reader's Buffer. (See Figure 7-2.)

3) Upon completion, tape stops, and the State of the two flags is Reversed, i.e., Set Done, Clear Busy. The character remains in the Reader's Buffer.

```
SKPDN PTR
Jmp. -1
```

1) These two instructions allow the program to detect completion on the part of the Reader.

2) The above is referred to as a "Listen Loop". In effect, listen for the flag. The JMP. -1, says in effect, jump current location minus one. (Jump back to the SKPBN).

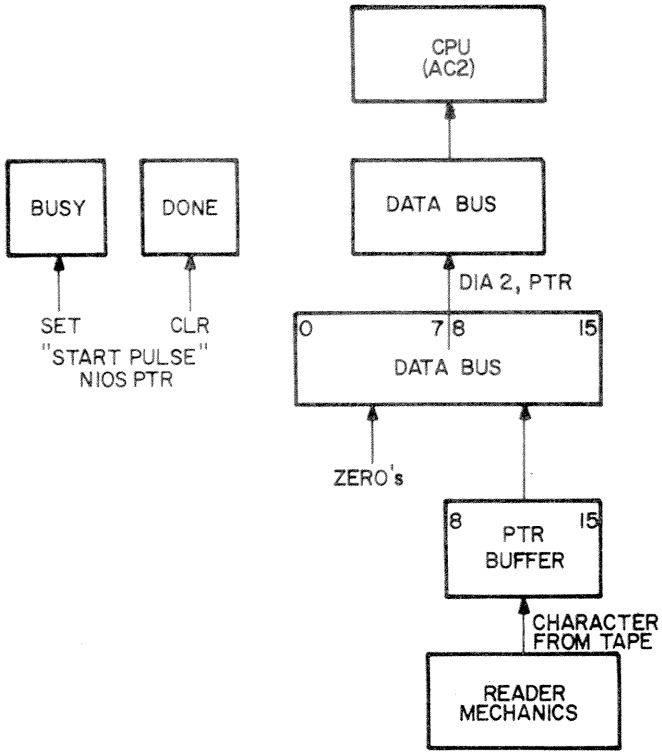
3) The program Loops between the two instruction, until such time as the Reader sets its Done flag. Setting of Done releases the program from the Loop.

4) When the speeds of the CPU and Reader are considered, the reason for the ""Listen Loop" becomes obvious. The CPU's speed is measured in nanoseconds, while that of the Reader in milliseconds. The CPU must necessarily wait on the device, this it does by looping on the flag.

DIAS 2, PTR

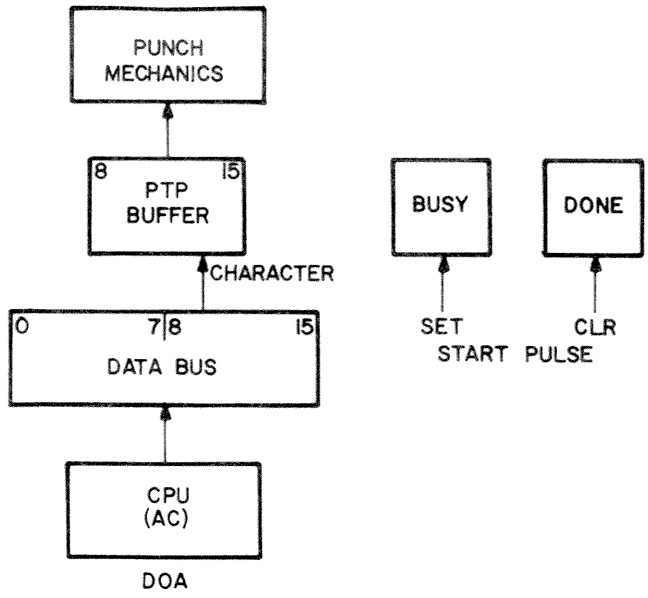
1) Data In A (DIA) transfers the character in the Reader's Buffer over the data bus to AC2.

2) The Start Pulse again puts the Reader in motion, the Reader now reads the second line of tape (character).



DG00722

Figure 7-2 Paper Tape Reader I/O Controls



DG00723

Figure 7-3 Paper Tape Punch I/O Control

Program Control of Paper Tape Punch

a) The following illustrates the punching of a character on Paper Tape.

```
DOAS 3,  PTP
SKPDN    PTP
JMP.  -1
```

b) Explanation

1) DOAS 3, PTP

The DOA (Data Out A) transfers a character in AC3 over the data bus to the Punch Buffer. The Start Pulse activates the Punch, punching the character in the Buffer.

2) SKPDN PTP

Same as for Paper Tape Reader, detect Punch completion prior to sending the next character. (See Figure 7-3.)

CPU IO INSTRUCTIONS

1. General

a) The Central Processor can be treated as a device.

b) Device Code 77 has been assigned to the Processor.

c) An IO instruction directed to a device behaves in a predictable manner while the same instruction directed to the CPU (Device Code 77) will have an entirely different effect.

d) CPU Mnemonics - basically for the benefit of the Programmer, having meaning to a program called the Assemblies and discussed in a later section.

2. Instructions/Mnemonics

a) READS

1) Equivalent to DIB X, CPU

2) Action-Read the 16 console data switches into ACX.

3) The DIB to a device would being back a 16 bit data word from Register B. To the CPU, the same instruction beings back the 16 data switches.

b) INTA - discussed under Program Interrupt.

c) MSKO - discussed under Program Interrupt.

d) IORST - IC Reset - clear all IO Busy and Done flags, etc.

e) HALT - Programmed Halt. Machine Halts.

f) INTEN - discussed under Program Interrupt.

g) INTDS - discussed under Program Interrupt.

h) Figure 7-4 - illustrates the relationship between a given mnemonic and its equivalent IO instruction.

5 - 6 - 7	DEVICE	CPU
0 0 0	NIO	----
0 0 1	DIA	READS
0 1 0	DOA	----
0 1 1	DIB	INTA
1 0 0	DOB	MSKO
1 0 1	DIC	IORST
1 1 0	DOC	HALT
1 1 1	SKP	----

8-9	DEVICE AND SKP	DEVICE - SKP	CPU AND SKP	CPU AND SKP
0 0	NOP	BN	--	ION (1)
0 1 (S)	START	BZ	INTEN	ION (0)
1 0 (C)	CLEAR	DN	INTDS	PWR LOW (1)
1 1 (P)	IOPLS	DZ	--	PWR LOW (0)

Figure 7-4 CPU I/O Instruction Decode

SECTION VIII

PROGRAM INTERRUPT AND DATA CHANNEL

PROGRAM INTERRUPT

General

An Interrupt System permits a Program to Start one or more devices, allowing the Program to continue without waiting for the device(s) to finish.

The following routine illustrates the advantage of Program Interrupt.

```
DOAS 2, PTP
SKPDN  PTP
JMP. -1
```

1. The DOAS transfers a character from AC2 over the Data Bus to the Punch Buffer, while the Start Start pulse initiates the Punch.
2. The remaining instructions form a "Listen Loop", the Program looping for roughly 16 milliseconds waiting for the Punch to finish.
3. It can be seen that the CPU, capable of executing an instruction in nanoseconds, is wasting a great deal of computing time waiting for the device. The Interrupt eliminates this need to wait on the device.

The completion of a task by a device signals an Interrupt. The Program is automatically interrupted, leaves the main sequence of instructions, goes to a subroutine, handles the device and returns to the main Program.

The Program must turn on the Interrupt System (Enable) before an Interrupt can occur.

Data Channel has priority over Program Interrupt.

An Interrupt cannot "break" the current instruction, i. e., the instruction has to be completed before an Interrupt Request can be honored.

Once an Interrupt occurs, the hardware turns OFF the Interrupt System.

CPU Mnemonics

a) INTEN

1) Interrupt Enable.
2) Equivalent to NIOS CPU.
3) Turns "ON" the Interrupt System.

b) INTDS

1) Interrupt disable
2) Equivalent to NIOC CPU
3) Turns "OFF" the Interrupt System.

Basic Interrupt Sequence

```
Location  Content     Comment
   300    INTEN      ;Turns on Interrupt
   301    NIOS PTR   ;Start the Reader
   302    INST       ;Continue Main Program
                     ;without waiting for Reader.
   303    INST       ;Reader will Interrupt when
    ↓                ;Done.
   700    INST
   701    INST DONE  ;Reader sets Done
   702               ;Interrupt occurs
                     ;PC is stored in location
     0    702        ;Zero.  Pointing to 702
     1    2000       ;Address of Subroutine

  2000    INST       ;Get character
          INST       ;from Reader, store in
          INST       ;memory and Start
          INST       ;Reader to get next
          INST       ;character.
          INTEN      ;Turn Interrupt back "ON"
          JMP @ 0    ;Return to main Program.
                     ;Next instruction is in
                     ;location 702.
```

Interrupt Sequence Description

1) INTEN - Turns on the Interrupt (Enable).
2) NIOS PTR - Starts Reader.
3) Program continues.
4) Assume executing the instruction in Location 701 when Reader sets Done. This instruction must be completed before the Reader Interrupt Request can be honored.
5) The Program Counter (PC) is pointing to address 702. This is the return Address. The instruction in location 702 will be the first instruction executed upon returning to the main routine.
6) The PC is stored in Location 0, while Location 1 contains the address of the subroutine.

7) The hardware forces a JMP @ 1 which transfers Program control to the Subroutine.

8) When returning from the subroutine, the Interrupt is turned on and return is accomplished.

Polling Technique

a) In the foregoing description of the basic interrupt sequence, it was assumed that only one device was in use.

b) When operating one device, using the Program Interrupt, when an interrupt does occur, we assumed that it was caused by the one device.

c) Consider the case of two or more devices.

1) When an interrupt occurs, program control is transferred to a Subroutine.

2) Within this subroutine it is the function of the Program itself to determine which device made the request. This can be done by Sampling Flags (Pol the flags).

3) The following assumes four devices working:

```
SKPDZ PTR       ;Test Reader
JMP PTRSBR      ;Exit Reader Subroutine
SKPDZ PTP       ;Test Punch
JMP PTPSBR      ;Exit Punch Subroutine
SKPDZ TTI       ;Test TTI
JMP TTISBR      ;Exit TTI Subroutine
SKPDZ TTO       ;Test TTO
JMP TTOSBR      ;Exit TTO Subroutine
```

a) If a device Done Flag is zero, it did not request the interrupt.

b) Note that we have assigned a priority to the devices. This was accomplished by the sequence of flags examined. Device speed is the criteria here, i. e., PTR is faster than PTP who in turn is faster than TTI and TTO.

b) INTA

1) Interrupt acknowledge.

2) A CPU Mnemonic equivalent to a DIB X, CPU.

3) The IO instruction DIB to a device would bring back a data word.

4) A DIB 2, CPU brings back a device code. This is the code of the highest priority device asking for an interrupt. The six bit device code is loaded into Accumulator bits 10 - 15. The Program looking at these bits identifies the device.

5) INTA does exactly what the Polling technique did, Identify the device making the interrupt request. Note that it is a more sophisticated method, eliminating quite a few instructions.

DATA CHANNEL

General

1) Devices tied to the Interrupt are devices that normally transfer a character at a time.

2) Devices tied to the channel transfer an entire block of data. A Disk for instance, transfers an entire sector while a Magnetic Tape would transfer an entire record.

3) Data Channels are said to have direct access to memory and transfer data at a high rate of speed.

4) The Data Channel steals a Data Channel cycle from the CPU. The running program is not aware that a block transfer is occurring. Basically when the device is ready to transfer a word, it momentarily interrupts the Program by stealing a Data Channel cycle and passes the word from or to memory.

Data Channel Description

1) The CPU communicates with a device through its controller (Interface).

2) The controller in turn communicates with up to 8 devices. (Units 0 - 7). These would be identical devices, i. e., 8 DISKS per controller or 8 Tape Transports per controller.

Figure 8-1 illustrates the path of communication between the CPU and a controller.

a) RQENB (Request Enable) - a clock pulse sent out periodically by the CPU, it looks for channel requests.

b) DCHR - Data Channel Request. The device controller wants the channel.

c) DCHA - Data Channel acknowledge.

1) The controller has the channel.

2) The controller must specify a memory location.

3) The controller must specify the mode, i. e., direction of transfer. Derived from 2 mode bits DCHM0 and DCHM1.

d) DCHO - Data Channel Out. Derived by the CPU from the mode bits. Basically the CPU indicates to the controller that the CPU has retrieved a word from the specified memory address, that the word is on the Data Bus and tells the controller to get the word.

e) DCHI - Data Channel In. Derived by the CPU from the mode bits. Basically the CPU tells the controller to put a data word on the Data Bus and that the CPU will get the word and write it into the specified memory address.

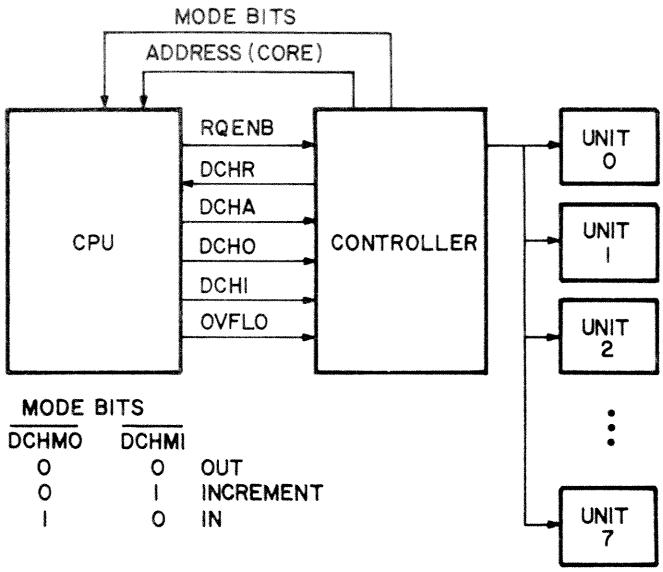


Figure 8-1 Data Channel Signal Transfer

Modes

1) OUT - From the CPU to the controller (Device). To a DISK or Magnetic Tape this would be a write command.

2) INC - Increment mode. This mode provides for the monitoring of events, i. e., the closing of Relay contacts. It keeps a running count in a specified memory location. OVFLO (See Figure 8-1) Overflow can only occur in the increment mode. If we continuously increment a given location in memory overflow will eventually occur. This is detected by the CPU and the device controller notified.

3) IN - From the controller (Device) to the CPU. To a DISK or Magnetic Tape this would be a Read Command.

The number of words to be transferred is determined by the controller.

Devices tied to the channel are also tied to the Program Interrupt. The Data Channel is utilized to pass the data while the Interrupt notifies the program that the entire transfer (Block) is complete.

SECTION IX

CONSOLE

GENERAL

1) Contains switches and keys for controlling the operation of the computer. There are keys capable of starting, resetting and examining the various facets of the system.
2) Indicator registers provide the operator with a visual indication of the computer status.
3) The console provides for manual control of the computer.

REGISTERS

Address

a) 15 bit register.
b) Displays the current memory address.

Data

a) Displays a data word where the Data word is:
    1) Contents of a memory location.
    2) Contents of an Accumulator.
    3) An instruction word.
    4) An Address.

Operation Indicators

a) RUN - Indicates the state of the machine, when LIT a program is running, when OFF the computer is halted.
b) ION - The Program Interrupt System is ON (Enabled).
c) CARRY - An overflow has occurred during an Arithmetic computation.
d) FETCH - The next major state of the machine will be Fetch.
e) DEFER - The next major state of the machine will be Defer.
f) EXECUTE - The next major state of the machine will be Execute.

SWITCHES AND KEYS

Switches

a) 16 Console Data Switches.
b) In certain key functions the contents of the switches is an Address.
c) In other key functions the contents of the switches is a data word.

Keys

a) Accumulator.
    1) Four Keys, one for each accumulator.
    2) Examine - allow the operator to examine the contents of AC0, AC1, AC2 or AC3.
    3) Deposit.

Allow the operator to deposit data into AC0, AC1, AC2 or AC3.

The data to be deposited is in the 17 Console Data Switches.

Accumulator Examine or Deposit Keys are ineffective with the computer running.

b) Reset.
    1) Will HALT the computer at the end of the current instruction.
    2) Reset in addition to causing the computer to Halt, also clears out all IO (Input-Output) flags.
c) Stop - HALTS computer at end of current instruction.
d) START.
    1) Starts a Program at the location specified by the contents of the Console Data Switches.
    2) One of two keys capable of loading an initial address.
    3) Start is ineffective with the computer running.
e) Continue - continue a program from the point at which it was Halted. The Halt could have been initiated by a programmed Halt (CPU MNE HALT) or Key Stop. In either case pressing Continue allows the program to continue. Continue is ineffective with the computer running.

f) Deposit

1) Allows the operator to deposit data into any core location.

2) The data to be deposited is in the Console Data Switches.

3) Deposit cannot load an address, therefore, the following procedure must be adhered to.

Assume that data word 17775 is to be deposited into location 500.

a) Put 500 into Console Data Switches.

b) Press Examine (Examine loads the address 500 into PC, MA).

c) Put data word 17775 into Console Data Switches.

d) Press Deposit - loads 17775 into memory location 500.

Deposit is ineffective with the computer running.

g) Deposit Next

1) Allows the operator to deposit data into sequential core locations.

2) The data to be deposited is in the Console Data Switches.

3) Deposit next cannot load an address, therefore, the following must be adhered to:

Assume that the indicated data is to be deposited into the following sequential locations:

700	17050
701	23450
702	17550
703	07756

a) Put address 700 into Console Data Switches.

b) Press Key Examine (Loads Address).

c) Put 17050 into Console Data Switches.

d) Press Deposit - loads 17050 into Location 700.

e) Put 23450 into Console Data Switches.

f) Press Key Deposit Next - this loads 23450 into Location 701.

g) Put 17550 into Console Data Switches.

h) Press Key Deposit Next - this loads 17550 into Location 702.

i) Put 7756 into Console Data Switches.

j) Press Key Deposit Next - loads 7756 into Location 703.

k) Loading procedure is completed.

Deposit Next is ineffective with the computer running.

h) Examine

1) Allows the operator to examine the contents of any memory location.

2) The Address of the location to be examined is placed in the Console Data Switches.

3) The contents of a location is displayed in the Data Register.

4) Examine is ineffective with the computer running.

i) Examine Next

1) Allows the operator to examine sequential locations in memory.

2) Procedure.

Assume we wish to examine locations 500 - 502.

a) Place address 500 in Console Data Switches.

b) Press Examine - Contents of 500 is displayed in Data Register.

c) Press Examine Next - Contents of 501 is displayed in Data Register.

d) Press Examine Next - Contents of 502 is displayed in Data Register.

Examine Next is ineffective with the computer running.

j) Memory Step

1) Allows the operator to examine the execution of an instruction cycle/cycle.

2) Sometimes called Single Cycle or Single Step.

3) The computer executed one cycle and halts for each Key action.

4) Example

LDA 3, @ 200

a) A three cycle instruction.

b) Pressing Memory Step puts the machine through a Fetch Cycle and Halt. The Defer indicator is on indicating that the next cycle is Defer.

c) Pressing Memory Step now puts the machine through a Defer cycle and Halt. The Execute indicator is on indicating that the next cycle is Execute.

Much can be determined about the logic state of the machine with this key.

Careful observation of Console Indicators while utilizing Memory Step can be an aid in Hardware Debugging.

Memory Step is ineffective with the computer running.

Instruction Step - basically the same as Memory Step, however, the machine executes the entire instruction prior to halting.

1) Program Load - an option that reads the Bootstrap Loader (discussed next section) from Read only Memory (ROM) into core. Eliminates the need for loading the Bootstrap Loader via the Console Switches.

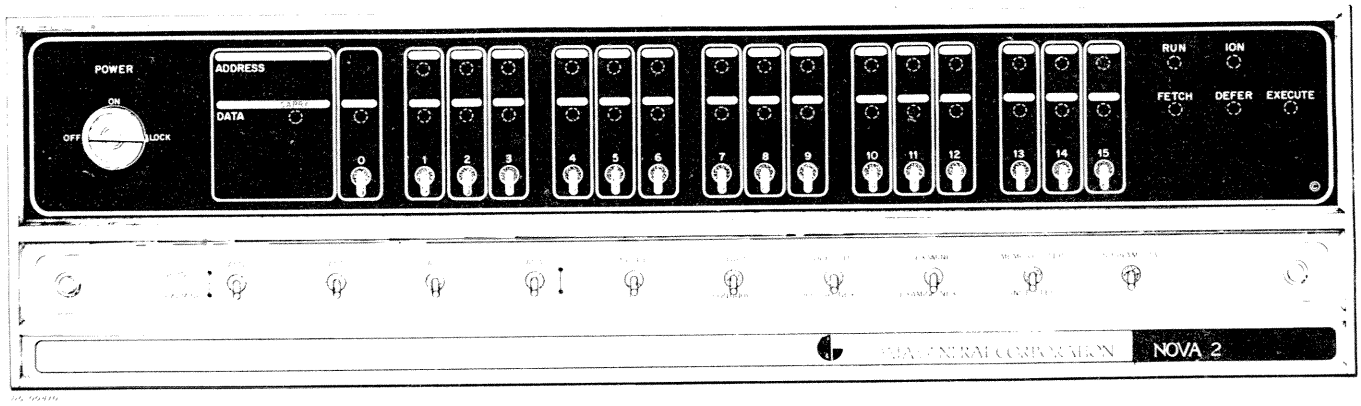
k) Power Lock and Key

1) Three positions.

a) OFF

b) ON - Power on and with a Program running, only Keys Stop and Reset are operative. I. E., the Program can be stopped. The Console Data Switches can also be read by the program (CPU MNE READS).

c) LOCK - Power ON and with a Program running, all Keys are frozen, i. e., the program cannot be halted. Console Data Switches can be Read by the Program. In the Lock position and Program not running, Keys Start and Continue are operative to Start the Program, however, once started these Keys are ineffective.



Typical Mini-Computer Console

This page intentionally left blank

SECTION X

BASIC PROGRAMMING AND PROGRAMS

GENERAL

1. Assembler Language

    a) Programs are written in assembler language.

    b) The following illustrates

```
LDA 2, 300
LDA 3, 301
ADD 2, 3
STA 3, 302
```

2. Machine Language

    a) Assembler language has absolutely no meaning to the computer, it being capable of recognizing only ones and zeroes.

    b) A program written in assembler language to be of use to the computer must be translated into machine language of 1's and 0's.

3. ASSEMBLER - a program that translates from assembler language to machine language. The assembler takes a Source Tape (Assembler Language) and from this tape creates an Object Tape (Machine language).

LOADING PROGRAMS

Before a program can be executed it must be brought into memory. This requires that a loading program already reside in memory. If the memory is empty, one can use the automatic loading switches on the SUPERNOVA computer, NOVA 800 or NOVA 1200, but with the NOVA computer or with a NOVA 800 or 1200 without the program load option, one must use the data switches to deposit a bootstrap loader, which is ordinarily used only to being in a more extensive binary loader. This latter program is then used to read the object tapes of all other programs. The binary loader usually resides in high core where it is not disturbed by any of the standard software. But if an undebugged user routine inadvertently destroys the binary loader, it can be restored by first reloading the bootstrap manually.

Below are two versions of the standard bootstrap loader, one for the teletype reader, the other for the high speed reader. This program loads data relatively to its own position in memory. Although the bootstrap can be placed anywhere, the usual procedure is to place it in high core, beginning at the seventeenth (twenty-first octal) location from the top, so that the binary loader also resides in high core. The program is shown here for placement at the top of a 4K memory.

The bootstrap loader reads a tape in a special format in which each word is divided into four 4 bit characters. Each character occupies channels 1-4 (the right half) of a line on the tape. The first character of a word, containing bits 0-3, is indicated by a 1 in channel five. The tape can begin with any number of blank lines. The first two words are STA 1, 1, 1 and JMP . -4, which are stored in the final two loader locations as indicated in the listing. The third, fifth, ... words are STA instructions that address AC1, the fourth, sixth,.... words are data. The bootstrap executes each odd-numbered word to store the succeeding data word in the location specified by the STA instruction. The final odd-numbered word is a HALT, which stops the processor.

In the following listings the first two columns at the left give each memory location and its contents for a 4K memory. The remaining columns are a standard program listing. To load the program simply use the switches to place the octal numbers in the locations specified. For a memory of any other size, load the bootstrap beginning at a location whose address is 20_8 less than the largest address.

```
;BOOTSTRAP LOADER, TELETYPE VERSION

07757   126440   GET:    SUBO    1,1        ;Clear AC1, Carry
07760   063610           SKPDN   TTI
07761   000777           JMP     .-1        ;Wait for Done
07762   060510           DIAS    0,TTI      ;Read into AC0 and restart reader

07763   127100           ADDL    1,1        ;Shift AC1 left 4 places
07764   127100           ADDL    1,1
07765   107003           ADD     0,1,SNC    ;Add in new word
07766   000772           JMP     GET+1      ;Full word not assembled yet
07767   001400           JMP     0,3        ;Got full word, exit

07770   060110   BSTRP:  NIOS    TTI        ;Enter here, start reader
07771   004766           JSR     GET        ;Get a word
07772   044402           STA     1,.+2      ;Store it to execute it
07773   004764           JSR     GET        ;Get another word
                         ...                ;This will contain an STA (first STA 1,.+1)
                         ...                ;This will contain JMP .4

;BOOTSTRAP LOADER, HIGH SPEED READER VERSION

07757   126440   GET:    SUBO    1,1        ;Clear AC1, Carry
07760   063612           SKPDN   PTR
07761   000777           JMP     .1         ;Wait for Done
07762   060512           DIAS    0,PTR      ' Read into AC0 and restart reader

07763   127100           ADDL    1,1        ;Shift AC1 left 4 places
07764   127100           ADDL    1,1
07765   107003           ADD     0,1,SNC    ;Add in new word
07766   000772           JMP     GET+1      ;Full word not assembled yet
07767   001400           JMP     0,3        ;Got full word, exit

07770   060112   BSTRP:  NIOS    PTR        ;Enter here, start reader
07771   004766           JSR     GET        ;Get a word
07772   044402           STA     1,.+2      ;Store it to execute it
07773   004764           JSR     GET        ;Get another word
                         ...                ;This will contain an STA (first STA 1,.+1)
                         ...                ;This will contain JMP .-4
```

To use the bootstrap to load the binary loader or any other program in the special format, follow these steps:

1. Put the special format tape in the reader and turn it on.
2. Press RESET.
3. For a 4K system set the data switches to 00770 (7 less than the largest address).
4. Press START.
5. The bootstrap loader begins at location 7770.

Binary Loader

A standard loader for loading program tapes in the type of object tape format generated by the assembler (refer to the assembler manual) is available in several forms. Program tape number 091-000004 (writeup 093-000003) is the binary loader for use with the manually loaded bootstrap given at the beginning of this section: 091-000036 (writeup 093-000051) is the binary loader prefaced by the sizing and loading program for use with the NOVA 800 and 1200 program load; 081-000001 (writeup 093-000003) is the binary loader prefaced by both the equivalent SUPERNOVA computer bootstrap and the sizing program. Following an automatic load, the operator can read an object tape on the same device simply by pressing CONTINUE. To load an object tape in any other circumstances, follow this procedure.

1. Put the object tape in the paper tape reader or teletype.
2. Set the data switches to x7777.
3. If you are using the paper tape reader, turn on data switch 0; otherwise turn it off.
4. Press START.

If a starting address is given on the object tape, control will be transferred to that location when loading is complete. Otherwise, the loader will halt with the address lights displaying x7740, and the user must start the program from the console.

The binary loader computes a checksum over every data block and start block read. If a checksum failure occurs over a block, the loader halts with x7726 displayed in the address lights. Reposition the tape to the beginning of the last block read and press CONTINUE. If the checksum failure again occurs, the object tape is probably in error.

Generate a new tape before attempting to load the program again.

Diagnostic Programs

The NOVA computer Diagnostics are individual programs which together test all logical operations of the Computer system. Individually the programs test various logic areas of the Computer and IO. The majority of the diagnostic routines are capable of diagnosing malfunctions down to the logic level. The diagnostics provide a means of measuring the performance of the system on a repeatable basis. Copies of the diagnostic tapes as well as individual program documentation are part of the software package delivered with the NOVA computer. Individual program documentation provides information as to operating procedures, error interpretation, console switch settings and logical areas tested. Certain diagnostics are normally part of the daily and weekly preventive maintenance routines.

NOVA DIAGNOSTIC PROGRAMS

Program	Description
Address Test	Routine to test the memory address section logic.
Checkerboard III	Worst case memory noise test. Program verifies proper operation and sense amps, inhibit drivers, and memory currents.
Nova Logic Test	Gate by gate test of CPU Logic (less IO).
Nova Instruction Timer	Routine to test CPU clock logic, prints instruction times of basic Nova instruction set.
Exerciser	Reliability test - tests CPU logic, TTY reader, punch, high speed paper tape reader, paper tape punch and real time clock. Halts on error.
Arithmetic Test	Exercises the arithmetic and logical instructions of the Nova Computers.

Sample Diagnostic Loop

1. Deposit data word in AC2.
2. Deposit program in core.
3. Start - Program halts - Load address in console switches and continue.

```
LOC
0000    063077    DOC 0, CPU      ;Halt Inst.
0001    060477    DIA 1, CPU      ;Reads Switches
0002    044011    STA 1, 11       ;Store Addr
0003    052011    STA 2, @ 11     ;Data to Addr
0004    000001    JMP .-3         ;Loop
```

Note: The address can be varied by changing the contents of the Console switches. The above routine will store the contents of AC2 (Data word) into the address in AC1. It is useful in monitoring Read/Write currents and the Inhibit current.

This page intentionally left blank

APPENDIX A

GLOSSARY OF TERMS AND DEFINITIONS

ACCESS TIME

Time required to obtain information from storage (read-time) or to put information away in storage (write-time).

ADDRESS

(1) An identification, as represented by a name, label, or number, for a register, location in storage, or any other data source or network.

(2) Loosely, any part of an instruction that specifies the location of an operand for the instruction.

ALGOL

International Algebraic Language.

ALPHANUMERIC

Pertaining to a character set that contains both letters and numerals, and usually other characters.

ANALOG COMPUTER

Device using voltages, forces, fluid volume or other continuously variable physical quantities to represent numbers in calculations.

ASCII

American Standard Cord for Information Interchange.

AUTOMATIC SEND-RECEIVE SET (ASR)

A combination of teletype writer (TTY) transmitter and receiver from either the keyboard or paper tape.

BATCH-PROCESSING

A technique by which items to be processed must be coded and collected into groups prior to processing.

BINARY CODED DECIMAL

Pertaining to a decimal notation in which the individual decimal digits are each represented by a group of binary digits, e.g., in the 8-4-2-1 binary coded decimal notation, the number twenty-three is represented as 0010 0011 whereas in binary notation, twenty-three is represented as 10111.

BINARY COMPUTER

Device using on-off switches (electro-mechanical relays, vacuum tube or transistor circuits, magnetic rings, etc.) to represent numbers (in binary number system) for calculations.

BINARY NUMBER

A number, usually consisting of more than one figure, representing a sum, in which the individual quantity represented by each figure is based on a radix of two. The figures used are 0 and 1.

BIT

A unit of information content. Contraction of "binary digit" a bit is the smallest unit of information in a binary system of notation. It is the choice between two possible states, usually designated one and zero.

BLOCK DIAGRAM

A diagram of a system, computer in which selected portions are represented by annotated boxed and interconnecting lines.

BREAK

Hardware - controlled action which occurs between instructions or between cycles of an instruction that does not affect the normal sequence of instruction execution.

BUFFER

A storage device used to compensate for a difference in rate of flow of data or time of occurrence of events, when transmitting data from one device to another. Either a hardware or software storage area for date.

BYTE

A group of bits (usually six to eight) forming a character.

CATHODE RAY TUBE (CRT)

A display device similar to a T. V.

CHANNEL

A path for electrical transmission between two or more points. Also called a circuit, facility, line link or path.

GLOSSARY OF TERMS AND DEFINITIONS

CHARACTER

A digit, letter or other symbol, usually requiring six to eight bits of storage.

COBOL (COMMON BUSINESS ORIENTED LANGUAGE)

A business data processing language.

COMMAND

Loosely, an instruction in machine language.

COMPILE

To prepare a machine language program from a computer program written in high level programming language.

COMPILER

A program that compiles.

COMPUTER, STORED PROGRAM

A digital computer that, under control of its own instructions, can synthesize, alter and store instructions as though they were data and can subsequently execute these new instructions.

COMPUTER WORD

A sequence of bits treated as a unit and capable of being stored in one computer location.

DATA

Information recorded systematically.

DATA SET

A unit of data storage and retrieval in an operating system, consisting of a collection of data in some prescribed arrangement and described by control information (label) that the system has access to.

DEBUG

To isolate and remove the mistakes from a routine or malfunction from a computer.

DOWN-TIME

Time when a computer is not available for operation, usually because of a failure in the equipment.

DUPLEX

In communications, pertaining to a simultaneous two-way and independent transmission in both directions (cometimes referred to as "full duplex"). Contrast with half-duplex.

FIELD

A specified area used for a particular category of data, e.g., a group of card column used to represent a wage rate or a set of bit locations in a computer word used to express the address of the operand.

FORTRAN (FORmula TRANslating system)

One of several specific procedure oriented languages.

HALF-DUPLEX

Pertaining to an alternate one-way-at-a-time, independent transmission (sometimes referred to as single). Contrast with duplex.

HARDWARE

The physical equipment or devices forming a computer and peripheral equipment.

HIGH ORDER POSITION

The left most position in a number of word.

HOLLERITH CARD

A punched card.

INDEXING

Process of establishing memory addresses by adding the value in an address field of an instruction to a value stored in a specified index register.

INTERRUPT

Caused by an action external to the running program which changes the contents of the program counter, thereby changing the sequence of instruction execution.

JOB

An externally specified unit of work (translation or execution of a program) for the computing system from the standpoint of operating system control.

JOB QUEUE

A line-up of jobs to be processed under operating system control.

JUSTIFIED

To adjust exactly as to spacing, to align a set of characters to RIGHT or LEFT margins.

LOAD

To put data into a register or storage or to put a magnetic tape drive or to put cards into a card reader or punch.

LOW-ORDER POSITION

The right most position in a number or word.

LOOPING

A sequence of instructions that are repeated until a terminal condition prevails.

MACHINE LANGUAGE

A language designed for interpretation and use by a machine without translation.

MACRO INSTRUCTION

An instruction that is replaced in a routine by a predetermined sequence of machine instructions.

MAGNETIC CORE STORAGE

A storage device in which binary data is represented by the direction of magnetization in each unit of an array of magnetic material.

MAGNETIC DISC

A flat circular plate with a magnetic surface on which data can be stored by selective magnetization of portions of the flat surface.

MAGNETIC DRUM

A right circular cylinder with a magnetic surface on which data can be stored by selective magnetization of portions of the curved surface.

MAGNETIC TAPE

A tape with a magnetic surface on which data can be stored by selective polarization of portions of the surface.

MAIN FRAME

The central processor of the computer system. This is synonymous with (CPU) and central processing unit. All that portion of a computer exclusive of the input, output, peripheral and in some instances, storage units.

MASKING

Process of setting or inhibiting internal program controls to allow or prevent transfers which may occur upon actions by the CPU.

MEDIUM

The material, or configuration thereof, on which data is recorded, e.g., paper tape, cards, magnetic tape.

MICROSECOND

.000001 second; used to describe computer operation speed. One millonth of a second; 10^{-6}.

MNEMONIC CODE

A code used in assembly languages to call to mind some operation. For example,SUB might stand for subtract or CRA might mean clear the accumulator.

MODULE

The input to, or output from, a single execution of an assembler, a compiler, or a loader; hence, a program unit that is discrete and identifiable with respect to compiling, combining with other units (linking), and loading.

MULTIPLEXING

The division of a transmission facility into two or more channels.

MULTIPROCESSING

Technique for interleaving unrelated routines and programs so that they run almost concurrently on one central processor.

NANOSECOND

A billionth of a second, 10^{-9} seconds.

NOISE

Loosely, any disturbance tending to interfere with the normal operation of a device or system.

OBJECT PROGRAM

The program which is the output of an automatic coding system. Often the object program is a machine language program ready for execution, but it may well be in an intermediate language.

OCTAL NUMBER

A number of one or more figures, representing a sum in which the quantity represented by each Figure is based on a radix of eight. The figures used are 0, 1, 2, 3, 4, 5, 6, and 7.

ON-LINE

Connected directly to the central computer (e.g., an electric typewriter in direct communication with computer processor).

OPERAND

That which is operated upon. An operand is usually identified by an address part of an instruction.

OPERATING SYSTEM

An integrated collection of service routines for supervising the sequencing of programs by a computer. Operating systems may perform debugging, input-output, accounting, compilation, and storage assignment tasks. Synonymous with monitor system and executive system.

OPERATION CODE

A code that represents specific operations.

OVERLAPPING

The act of reading information into or writing it from a portion of memory while computing operations continue in other portions; input/output processor and central processor share memory.

OVERLAY

A load module which is to be placed in main storage locations occupied by another module; several modules may occupy the same storage area at different times.

PERIPHERAL EQUIPMENT

The auxiliary machines which may be placed under the control of the central computer. Examples of this are card readers, card punches, magnetic tape feeds and high-speed printers.

POLLING

A centrally controlled method of calling a number of points to permit them to transmit information.

PROGRAM

A plan for solving a problem. A group of logical instructions to a computer to solve a problem.

PSEUDO INSTRUCTION

A group of characters having the same general form as a computer instruction, but never executed by the computer as an actual instruction but a command to an assembler.

REAL-TIME PROCESSING

The processing of information or data in a sufficiently rapid manner so that the results of the processing are available in time to influence the process being monitored.

RESPONSE TIME

The amount of time elapsed between generation of an inquiry at a data communications terminal and receipt of a response at that same terminal.

ROUTINE

A subdivision of a program consisting of two or more instructions that are functionally related.

SOFTWARE

The totality of programs and routines used to extend the capabilities of computers, such as compilers, assemblers, routines, and subroutines.

GLOSSARY OF TERMS AND DEFINITIONS

SOURCE PROGRAM

A program written in a language designed for ease of expression of a class of problems or procedures, by humans. A generator assembler or compiler routine is used to perform the mechanics of translating the source program into an object program in machine language.

SUBROUTINE

A portion of a routine that causes a computer to carry out a well-defined mathematical or logical operation.

TABLE

A collection of data, each item being uniquely identified either by some label or by its relative position.

TIMESHARE

To use a device for two or more interleaved purposes.

This page intentionally left blank

APPENDIX B

ABBREVIATIONS

Listed below are the most commonly used abbreviations of registers, key operations, components, instruction and signal names.

ACD	Destination Accumulator
ACDP	Accumulator Deposit
ACEX	Accumulator Examine
AC0	Accumulator 0
AC1	Accumulator 1
AC2	Accumulator 2
AC3	Accumulator 3
ACS	Source Accumulator
ACTG0, ACTG1	Accumulator Timing Generator Outputs 0 & 1
ALC	Arithmetic Logic Class (instruction)
ALU	Arithmetic Logic Unit
AND	AND (logic instruction)
AR	Arithmetic Register
AUT INC + DEC	Autoincrement or Autodecrement
CARRY	Carry (arithmetic function)
CLK	Clock
CLR	Clear
CON DATA	Console Data
CON INST	Console Instruction
CON RQ	Console Request
CONT	Continue Switch at Console
CPU	Central Processor Unit
CPU CLK	Central Processor Unit Clock
CPU INST	Central Processor Unit Instruction
CRY	Carry
D	Defer
DATIA	Data In A (I/O instruction)
DATIB	Data In B (I/O instruction)
DATIC	Data In C (I/O instruction)
DATOA	Data Out A (I/O instruction)
DATOB	Data Our B (I/O instruction)
DATOC	Data Out C (I/O instruction)
DATA0 thru DATA15	I/O Data bus signals, 16 bits wide
DCH	Data Channels
DCHA	Data Channel Acknowledge
DCHI	Data Channel In
DCH INC	Data Channels Increment
DCHM (0 or 1)	Data Channel Mode (0 or 1) Code type of Data Channel Cycle requested by Device
DCHO	Data Channel Out
DCHP IN	Data Channel Priority In
DCHP OUT	Data Channel Priority Out
DCHR	Data Channel Request
DEFER	Defer (instruction execution state)
DIV	Divide (instruction)
D MULT	Destination Multiplexer
DP	Deposit (Console function)
DPN	Deposit Next (Console function)
D SET	Defer Set
DSZ	Decrement and Skip if Zero (instruction)
DS0-DS5	Device Select lines 0 thru 5
E	Execute
EFA	Effective Address
EXEC	Execute
EX	Examine (Console function)
EXN	Examine Next
F	Fetch
FAST DCH	Fast (High Speed) Data Channels
FETCH	Fetch (State Accessing next instruction from Memory)
HALT	Halt (Machine State)
INC PC	Increment Program Counter

ABBREVIATIONS

INH GATE A	Inhibit Gate A (Memory)
INH GATE B	Inhibit Gate B (Memory)
INH0 thru INH15	Inhibit (Memory Buffer) Register outputs 0 thru 15
INHIBIT	Inhibit (Memory Writing function)
INH TRANS	Inhibit Transmission
INTA	Interrupt Acknowledge
INTP IN	Interrupt Priority In (to Device)
INTP OUT	Interrupt Priority Out (from Device)
INTR	Interrupt (Bus Signal from Device)
INT RQ	Interrupt Request
IO OR I/O	Input/Output
ION	Interrupt On
IORST	Input/Output Reset
IO SKIP	Input/Output Skip (instruction)
IO STUTTER	Cycle extend for IO operation
IO UNPROTECTED	Indicates IR contains IO instruction
IR0 thru IR15	Instruction Register outputs 0 thru 15
ISTP	Instruction Step (Console switch)
ISZ	Increment and Skip if Zero (instruction)
JMP	Jump (instruction)
JSR	Jump to Subroutine (instruction)
KEY	Operational Cycle manually implemented at the Console
KEYM	Key Memory (access cycle)
LDA	Load Accumulator (instruction)
LOAD AC	Load Accumulator
LOAD ACB	Load Accumulator Buffer (Shifter)
LOAD AR	Load Arithmetic Register
LOAD CRY	Load Carry
LOAD IR	Load Instruction Register
LOAD MBO	Load Memory Bus Outputs (CPU Interface Register)
LOAD PC	Load Program Counter
MA LOAD	Memory Address Load
MA1 thru MA15	Memory Address Register Outputs 1 thru 15
MB	Memory Buffer
MB CLEAR	Memory Buffer Clear
MB LOAD	Memory Buffer Load
MBC8 thru MBC15	Memory Buffer Computer Outputs 8 thru 15
MBO0 thru MBO15	Memory Buffer (bus) Outputs 0 thru 15
MD1 thru MD15	Memory (address) Data (input lines) 1 thru 15
MEM CLK	Memory Clock
MEM OK	Memory OK (Power Supply Monitor signal)
MEM OUT	Memory (bus) Out
MEM0 thru MEM15	Memory Bus lines 0 thru 15
MQ0 thru MQ15	Multiplier Quotient Register Outputs 0 thru 15
MSKO	Mask Out (instruction)
MSTP	Memory Step (Console switch)
MTG0 thru MTG3	Memory Timing Generator (signals) 0 thru 3
MULT0 thru MULT3	Multiplexer Output (signals) 0 thru 3
NON ACD INST	Non Destination Accumulator Instruction
OVFLO	Signal to Device that memory location being incremented via Data Channels has Overflowed
PC	Program Counter
PC CLK	Program Counter Clock
PC TO MEM	Program Counter to Memory
PEND	Pending, e.g., INT PEND
PI	Program Interrupt
PI SET	Program Interrupt Set
PL	Program Load

ABBREVIATIONS

PLUS ONE	Plus One (to the Adder)
PRESET	Preset (Computer initializing signal)
PTG0 thru PTG3	Processor Timing (pulses) 0 thru 3
PTG0 thru PTG5	Processor Timing Generator (signals) 0 thru 5
PWR LOW	Power Low (Power Monitor output signal)
PWR FAIL	Power Fail
READ CY	(Memory) Read Cycle
READ 1	Read 1 (Memory Timing signal, CPU-1)
READ 2	Read 2 (Memory Timing signal, CPU-1)
READ 1B	Read 1B (Memory Timing signal, Memory)
READ 2B	Read 2B (Memory Timing signal, Memory)
RESTART	RESTART (power Monitor output signal)
RESTART ENABLE	Signal that permits RST and STOP Console Key functions
RINH0 thru RINH15	(Collector) Resistor, Inhibit Driver
ROM ENABLE	PL Read Only Memory Enable
RQENB	Request Enable
RST	Restart (Console switch)
RUN	Primary operational requirement for program execution
SARD	Select Address
S BUFFER	Source Buffer
SELB	Selected Busy (Bus signal)
SELD	Selected Done (Bus signal)
SELECT	Decoded (Memory) Select signal
SET ION	Set Interrupt On
SHIFT ACB	Shift Accumulator Buffer
SHL	Shift Left
SHR	Shift Right
SL0 thru SL15	Sense Lines (Memory Stack 0 thru 15
S MULT	Source Multiplexer
SNS0 thru SNS15	Sense Amplifier Outputs 0 thru 15
STA	Store Accumulator (instruction)
STOP	(Processor) Stop
STOP INH	(Processor) STOP INHIBIT
STRB A	Strobe A (Memory Stack)
STRB B	Strobe B (Memory Stack)
STRB C	Strobe C (Memory Stack)
STRB D	Strobe D (Memory Stack)
STROBE	Strobe (signal, CPU-1)
STOP RQ	(Processor) Stop Request
STRT	Start (Console switch)
SWP	Swap (bytes)
TS0	Time State 0
TS0 thru TS3	Time State 0 thru 3
TS3	Time State 3
TTI	Teletype In (Teletype Keyboard/Reader Buffer)
TTO	Teletype Out (Teletype Teleprinter/Punch (Buffer)
WRITE	Control function, Memory Cycle Timing, CPU-1
WRITE AC	Write Accumulator (logically associated with AC Write signal)
WRITE MEM	Write Memory (enable X and Y Memory drivers)
XOR	Exclusive OR (Boolean)
XRS	X (plane) Read Source (Memory Stack)
XWS	X (plane) Write Source (Memory Stack)
YRS	Y (plane) Read Source (Memory Stack)
YWS	Y (plane) Write Source (Memory Stack)
32 VNR	+32 Volts, Not Regulated
±SL0 thru ±SL15	Memory Stack Bipolar sense inputs to Sense Amplifiers
+VINH	+ (Memory) Inhibit Voltage

ABBREVIATIONS

$+V_{LAMP}$	+Lamp Voltage (Console indicators)
+VMEM	+Voltage Memory
+5 OK	+5Volt (power) Operating properly

APPENDIX C

REFERENCES

1. Digital Computer Systems Principles
   Second Edition - 1973
   Herbert Hellerman State University of N. Y.
   at Binghamton - McGraw-Hill

2. Digital Computer Fundamentals
   Third Edition - 1972
   Thomas C. Bartee, Harvard University
   McGraw-Hill

3. Digital Systems Fundamentals
   1972
   John Motil, California State University
   Northridge

4. Standard Dictionary of Computers and
   Information Processing
   Martin H. Welk, Chief, Data Management Div.
   U. S. Army Research Office
   Mayden Book Company, Inc.

5. Pulse, digital and Switching Waveforms
   Millman & Taub
   McGraw-Hill

6. How to Use the NOVA Computers
   Data General Corporation
   Southboro, Mass.

This page intentionally left blank

CUT ALONG DOTTED LINE

READERS COMMENT FORM

DOCUMENT TITLE: .....

Your comments, accompanied by answers to the following questions, help us improve the quality and usefulness of our publications. If your answer to a question is "no" or requires qualification, please explain.

How did you use this publication?

( ) As an introduction to the subject.
( ) As an aid for advanced knowledge.
( ) For information about operating procedures.
( ) To instruct in a class.
( ) As a student in a class.
( ) As a reference manual.
( ) Other.....

Did you find the material:

- Useful..... YES ( ) NO ( )
- Complete..... YES ( ) NO ( )
- Accurate..... YES ( ) NO ( )
- Well organized..... YES ( ) NO ( )
- Well written..... YES ( ) NO ( )
- Well illustrated..... YES ( ) NO ( )
- Well indexed..... YES ( ) NO ( )
- Easy to read..... YES ( ) NO ( )
- Easy to understand..... YES ( ) NO ( )

We would appreciate any other comments; please label each comment as an addition, deletion, change, or error and reference page numbers where applicable

COMMENTS

PAGE	COL	PARA	LINE	FROM	TO

From:

NAME..... TITLE.....
FIRM..... DIV. .....
ADDRESS.....
CITY..... STATE..... ZIP.....
TELEPHONE..... DATE.....

Data General Corporation
ENGINEERING PUBLICATIONS
COMMENT FORM
DG-00935

FOLD DOWN FIRST FOLD DOWN

FIRST CLASS
PERMIT NO. 26
SOUTHBORO
MASS. 01772

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

Postage will be paid by:

DataGeneral
Southboro, Massachusetts 01772

ATTENTION: Engineering Publications

FOLD UP SECOND FOLD UP

STAPLE