This watermark does not appear in the registered version - http://www.clicktoconvert.com

UNIT I

NUMBER SYSTEM AND BINARY CODES

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1.0 AIM AND OBJECTIVES

This unit being the first unit, it introduces you to the world of computers. At the end of the unit you will be able to know how:

1. An explanation of positional notation is given and the idea of the base, or radix, of a number system is presented.
2. The binary number system is explained as well as how to add, subtract, multiply and divide in this system. The techniques for converting from binary to decimal and decimal to binary are given.
3. Negative numbers are represented in computers by using a sign bit, and this concept is explained. Negative numbers are often represented by using a complemented form rather than a signed magnitude form. The two major complemented forms, true complement and radix minus one are described.
4. The representation of decimal numbers using bi-stable devices can be accomplished with a binary coded decimal (BCD) system and several of these are explained.
5. The octal and hexadecimal number systems are widely used in computer literature and manufacturer's manuals. These number systems are explained along with conversion techniques to and from decimal and binary.

1.1 INTRODUCTION

As a mathematician, Laplace could well appreciate the decimal number system. He was fully aware of the centuries of mental effort and sheer good luck, which had gone into the development of the number system we use, and he was in a position to appreciate its advantages.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Our present number system provides modern mathematicians and scientists with a great advantage over those of previous civilizations and is an important factor in our rapid advancement.

Since hands are the most convenient tools nature has provided, human beings have always tended to use them in counting. So the decimal number system followed naturally from this usage.

As even simpler system, the binary number system has proved the most natural and efficient system for computer use, however, and this chapter develops this number system along with other systems used by computer technology.

1.2 NUMBER SYSTEM

A number system of base (also called radix) r is a system, which have r distinct symbols for r digits. A number is represented by a string of these symbolic digits. To determine the quantity that the number represents, we multiply the number by an integer power of r depending on the place it is located and then find the sum of weighted digits.

1.2.1 DECMAL NUMBER SYSTEM

Decimal system is the most commonly used number system. Our present system of numbers has 10 separate symbols namely 0,1,2,3,4,5,6,7,8 and 9, which are called Arabic numerals. We would be forced to stop at 9 or to invent more symbols if it were not for the use of positional notation.

- ❖ The digit of a number system is a symbol, which represents an integral quantity.
- ❖ The base or radix of a number system is defined as the number of different digits, which can occur in each position in the number system. The decimal system has a base or radix of 10.

An example of earlier types of notation can be found in Roman Numerals, which are essential additive: III=I+I+I, XXV=X+X+V. The only importance of position in Roman numerals lies in whether a symbol precedes or follows another symbol (IV= 4, while VI=6).

In the beginning it was so difficult for the mathematicians to calculate the roman numerals and to perform arithmetic operations but now it is a great beauty that it is enough to learn only the 10 basic numerals and the positional notational system in order to count as to any desired figure. After memorizing the addition and multiplication tables and learning a few simple rules, we can perform all arithmetic operations.

The actual meaning of 168 can be seen more clearly if we notice that it is spoken as “one hundred and sixty eight”. Basically, the number is a contraction of 1*100+6*10+8. The important point is that the value of each digit is determined by its position. Written numbers are always contracted, however, and only the basic 10 numerals are used, regardless of the size of the integer written. The general rule for representing numbers in the decimal system by using positional notation is as follows:

This watermark does not appear in the registered version - http://www.clicktoconvert.com

$a_{n-1}\ 10^{n-1} + a_{n-2}\ 10^{n-2} + \ldots\ldots + a_0$ is expressed as $a_{n-1,}\ a_{n-2}\ \ldots\ a_0$
where n is the number of digits to the left of the decimal point.

1.2.2 BISTABLE DEVICES

The basic elements in early computer are relays and switches. The operation of a switch or relay can be seen to be essentially bi stable, or binary in nature; that is, the switch is either on (1) or off (0). The principal circuit elements in modern computers are transistors. Because of the large number of electronic parts used in computers, it is highly desirable to utilize them in such a manner that slight changes in their characteristics will not affect their performance. The best way of accomplishing this is to use circuits, which are basically bi stable (have two possible states).

1.2.3 BINARY NUMBER SYSTEM

Digital computers use the binary number system, which has only two symbols: 0 and 1. The numbers in binary system are represented as combinations of these two symbols. The decimal system uses power of 10 and binary system uses powers of 2.

The binary digit is also referred to as Bit (the acronym for Binary Digit). A string of 4 bits is called a nibble and a string of 8 bits is called a byte. A byte is the basic unit of data in computers. The number 125 actually means $1*\ 10^2 + 2*\ 10^1 + 5*10^0$. In binary system, the same number (125) is represented as 1111101 meaning

$$1*2^6 + 1*\ 2^5 + 1*\ 2^4 + 1*2^3 + 1*\ 2^2 + 0*\ 2^1 + 1*2^0$$

The table 1.1 lists the first 20 binary numbers.

Decimal	**Binary**	**Decimal**	**Binary**
1	1	11	1011
2	10	12	1100
3	11	13	1101
4	100	14	1110
5	101	15	1111
6	110	16	10000
7	111	17	10001
8	1000	18	10010
9	1001	19	10011
10	1010	20	10100

Table 1.1 First 20 Binary Numbers

This watermark does not appear in the registered version - http://www.clicktoconvert.com

To express the value of a binary number, therefore, a $_{n-1}$ 2^{n-1} + a $_{n-2}$ 2^{n-2} + ……. + a $_0$ is expressed as a $_{n-1,}$ a $_{n-2}$ ….. a $_0$ where a_i is either 1 or 0 and n is the number of digits to the left of the binary (radix) point.

1.2.4 OCTAL NUMBER SYSTEM

The octal number system has a base, or radix as 8: eight different symbols are used to represent numbers. These are commonly 0,1,2,3,4,5,6,7. We show the first 20 octal numbers and their decimal equivalents in the table 1.2.

DECIMAL	OCTAL	DECIMAL	OCTAL
0	0	11	13
1	1	12	14
2	2	13	15
3	3	14	16
4	4	15	17
5	5	16	20
6	6	17	21
7	7	18	22
8	10	19	23
9	11	20	24
10	12	21	25

Table 1.2 First 20 Octal Numbers

To convert an octal number to a decimal number, we use the same sort of polynomial as was used in the binary case, except that we now have a radix of 8 instead of 2. Therefore 1213 in octal is

$$=1*8^3+2*8^2+1*8^1+3*8^0$$
$$=512 + 128 + 8 + 3 \quad =651$$

in decimal. Also, 1.123 in octal is $1*8^0+1*8^{-1}+2*8^{-2}+3*8^{-3}$ =1.83/512 in decimal.

1.2.5 HEXADECIMAL NUMBER SYSTEM

When the machine is handling numbers in binary but in groups of four digits, it is convenient to have a code for representing each of these sets of four digits. Since 16 possible different numbers can be represented, the digits 0 through 9 will not suffice. So the letters A, B, C, D, E and F are also used. Hexadecimal numbers are strings of these digits. The numbers in decimal, binary and hexadecimal is shown in the table 1.3.

BINARY	HEXADECIMAL	DECIMAL
0000	0	0
0001	1	1
0010	2	2

This watermark does not appear in the registered version - http://www.clicktoconvert.com

0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Table 1.3 First 16 Hexadecimal Numbers

The base or radix of a number system is defined as the number of different digits, which can occur in each position in the number system. The decimal number system has a base, or radix of 10. Thus the system has 10 different digits (0,1,2, …,9), any one of which may be used in each position in a number. History records the use of several other number systems.

Self Check Exercise 1

Say True or False

1. A byte is equal to 8 bits and can represent a character internally. True/False
2. A program is a sequence os instructions designed for achieving a task/goal. True/False
3. One MB is equal to 1024 KB True/False

1.3 CONVERSIONS

1.3.1. DECIMAL NUMBER TO BINARY NUMBER

To convert a decimal number into binary number it requires successive division by 2 writing down each quotient and its remainder. The remainders are taken in the reverse order, which is the binary equivalent of the decimal number. For example, let it is required to convert the decimal number 25 to its binary equivalent.

```
2 | 25
2 | 12    -1
2 |  6    -0
2 |  3    -0
     1    -1
```

The binary equivalent for 25_{10} $=11001_2$

This watermark does not appear in the registered version - http://www.clicktoconvert.com

To convert decimal fractions into equivalent binary fractions repeatedly double the decimal fraction. The number (0 or 1) that appears on the left is written separately. The bits that are written in this manner are read from top to bottom with a decimal point on the left. For example if the given number is 0.0625, conversion is done in the following manner.

0.625	*2 =	0	.1250
0.125	*2 =	0	.25
0.25	*2 =	0	.5
0.5	*2 =	1	.0

Bits on the left:0
0
0
1

The Multiplication cannot be continued further, as the fractional part in the previous step has already become zero. Therefore, $0.0625_{10} = .0001_2$

1.3.2. DECIMAL NUMBER TO OCTAL NUMBER

Conversion from decimal to octal can be performed by repeatedly dividing the decimal number by 8 and using each remainder as a digit in the octal number being formed. For instance, to convert decimal number 200 to an octal representation, we divide as follows.

$$\begin{array}{r|l}
8 & 200 \\ \hline
8 & 25 \ \ - \ 0 \\ \hline
 & \ \ 3 \ \ \ - \ 1
\end{array}$$

Therefore $(200)_{10} = (310)_8$

1.3.3. DECIMAL NUMBER TO HEXADECIMAL NUMBER

One way to convert decimal to Hexadecimal is the hex dabbles. The idea is as divide successively by 16, writing down the remainders. Here is a sample of how it is done. To convert decimal 2429 to hexadecimal,

$$\begin{array}{r|l}
16 & 2429 \\ \hline
16 & 154 \ - \ 15 \ \text{———} \ \ F \\ \hline
 & \ \ \ 9 \ \ - \ 10 \ \text{———} \ \ A
\end{array}$$

Therefore $(2429)_{10} = (9AF)_{16}$

1.3.4. BINARY NUMBER TO DECIMAL NUMBER

For converting the value of Binary numbers to decimal equivalent we have to find its quantity, which is found by multiplying a digit by its place value. The following example illustrates the conversion of binary numbers to decimal system.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

$101 = 1*2^{3-1} + 0*2^{3-2} + 1*2^{3-3}$
$= 1*2^{2} + 0*2^{1} + 1*2^{0}$
$= 4 + 0 + 1$
$= 5$

$1001 = 1*2^{4-1} + 0*2^{4-2} + 0*2^{4-3} + 1*2^{4-4}$
$= 1*2^{3} + 0*2^{2} + 0*2^{1} + 1*2^{0}$
$= 8+1$
$= 9$

$11.011 = 1*2^{2-1} + 1*2^{2-2} + 0*2^{2-3} + 1*2^{2-4} + 1*2^{2-5}$
$= 1*2^{1} + 1*2^{0} + 0*2^{-1} + 1*2^{-2} + 1*2^{-3}$
$= 2+1+1/4+1/8$
$= 3\ 3/8$

1.3.5. BINARY NUMBER TO OCTAL NUMBER

There is a simple trick for converting a binary number to an octal number. Simply group the binary digits into groups of 3, starting at the octal point, and read each set of three binary digits according to the following table 1.4.

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

Table 1.4 First 8 Octal Numbers

Let us convert the binary number 011101 into octal. First, we break binary number into 3 digits (011 101). Then converting each group of three binary digits, we get 35 in octal. Therefore 011101 binary = 35 in octal.

$011101 = 011\ 101$
$= 3\quad 5$
$= (3\ 5)_8$

1.3.6. BINARY NUMBER TO HEXADECIMAL NUMBER

To convert binary to hexadecimal, we simply break a binary number into groups of four digits and convert each group of four digits according to the preceding code. Here are some examples:

Example:1

$(10111011)_2 = 1011\ 1011$
$= B\quad B$

This watermark does not appear in the registered version - http://www.clicktoconvert.com

$= (B\ \ B)_{16}$

Example:2

$(10010101)_2 = 1001\ 0101$

$= \quad 9 \quad\ 5$

$= (9\ \ 5)_{16}$

1.3.7. OCTAL NUMBER TO DECIMAL NUMBER

To convert an octal number to a decimal number, we use the same sort of polynomial as was used in the binary case, except that we now have a radix of 8 instead of 2. Therefore 1213 in octal is

$=1*8^3+2*8^2+1*8^1+3*8^0$

$=512 + 128 + \ 8 \ + \ 3 \quad =651$

in decimal. Also, 1.123 in octal is $1*8^0+1*8^{-1}+2*8^{-2}+3*8^{-3}$ =1 83/512 in decimal

1.3.8. OCTAL NUMBER TO BINARY NUMBER

The conversion from octal number to binary number is easily accomplished. Each octal bit is converted to its three digit binary equivalent.

Example: 1

$(\ 2 \quad 6 \quad 1 \quad 5 \quad 3\ .\ 7 \quad 4 \quad 0 \quad 6)_8$

$(010\ 110\ 001\ 101\ 011.\ 111\ 100\ 000\ 110)_2$

1.3.9. OCTAL NUMBER TO HEXADECIMAL NUMBER

The method of converting octal to hexadecimal number is to convert the given octal number to binary number and then arrange the binary digits into groups of 4 starting at the binary point.

Example :1

Convert octal number 714.06 to hexadecimal.

$(714.06)_8 =(111\ 001\ 100.000\ 110)_2$

$=0001\ 1100\ 1100\ .\ 0001\ 1000$

$=1\ C\ C\ .\ 1\ 8$

The hexadecimal equivalent of $(714.06)_8$ is $(1CC.18)_{16}$

1.3.10. HEXADECIMAL NUMBER TO BINARY NUMBER

To convert a hexadecimal number to a binary number, convert each hexadecimal digit to its 4-bit equivalent using the code. For instance, here's how 9AF converts to binary.

$(\ \ 3 \quad 0 \quad 6 \quad .\ \ D)_{16}$

$(\ 0011\ 0000\ 0110\ .\ 1101)_2$

9 A F

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1001 1010 1111

As another example, C5E2

```
C       5       E       2
|       |       |       |
1100    0101    1100    0010
```

1.3.11. HEXADECIMAL NUMBER TO OCTAL NUMBER

The conversion of Hexadecimal number to octal number involves two steps. First the method suggests to go from hexadecimal to binary numbers and second from binary to octal numbers Convert the hexadecimal into binary by writing 4 bits binary value for each bit in hexadecimal number and then arrange the binary digits into groups of three starting at the binary point.

Example: convert Hexadecimal (1E.C) to octal conversion

$(1E.C)_{16} = (0001\ 1110.1100)_2$

$= (011\ 110.110)$

$=36.6$

The octal equivalent of $(1E.C)_{16}$ is $(36.6)_8$

1.3.12. HEXADECIMAL NUMBER TO DECIMAL NUMBER

The conversion of Hexadecimal to decimal is straightforward but time consuming. In Hexadecimal number system each digit position corresponds to a power of 16. The weights of the digit positions in a hexadecimal number are as follows: For instance, BB represents

$BB=B*16^1 +B*16^0$

$=11*16+11*1$

$=176+11$

$=187$

Self – Check Exercise 2

1. Convert the following binary numbers to decimal.

(a) 1100.1101
(b) 10101010

…………………………………………………………………………………………………
…………………………………………………………………………………………………
…………………………………………………………………………………………………

2. Convert the following decimal numbers to binary.

(a) 23
(b) 49.25
(c) 892

…………………………………………………………………………………………………
…………………………………………………………………………………………………
…………………………………………………………………………………………………

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1.4 BINARY ADDITION AND SUBTRACTION

Binary addition is performed in the same manner as decimal addition. The complete table for binary addition is as follows:

0+0=0
0+1=1
1+0=1
1+1=0 plus a carry over of 1

'Carry over' are performed in the same manner as in decimal arithmetic. Since 1 is the largest digit in the binary system, any sum greater than 1 requires that a digit be carried.

Examples:

Decimal	Binary	Decimal	Binary
5	101	3 ¼	11.01
6	110	5 ¾	101.11
11	1011	9	1001.00

Subtraction is the inverse operation of addition. To subtract, it is necessary to establish a procedure for subtracting a larger from a smaller digit. The only case in which this occurs with binary numbers is when 1 is subtracted from 0. It is necessary to borrow 1 from the next column to the left. This is the binary subtraction table.

0-0=0
1-0=1
0-1=1 with a borrow of 1
1-1=0

Examples:

Decimal	Binary	Decimal	Binary
9	1001	16	10000
-5	-101	-3	-11
4	100	13	1111

1.5 BINARY MULTIPLICATION AND DIVISION

The table for binary multiplication is very short, with only four entries instead of the many for normal decimal multiplication

0*0=0
0*1=0
1*0=0
1*1=1

The following examples of binary multiplication illustrate the simplicity of each operation. It is only necessary to copy the multiplicand if the digit in the multiplier is 1 and to copy all 0's if the digit in the multiplier is a 0.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

```
    111            10.1
    101  *         10.1  *
  -------        -------
    111             101
   000             000
  111             101
  -------        -------
  100011          110.01
  -------        -------
```

The complete table for binary division is as follows:

0/1=0

1/1=1

Examples:

```
            111
   110   | 101010
         |  110
          ------
            1001
             110
          ------
              110
              110
           ------
                0
           ------
```

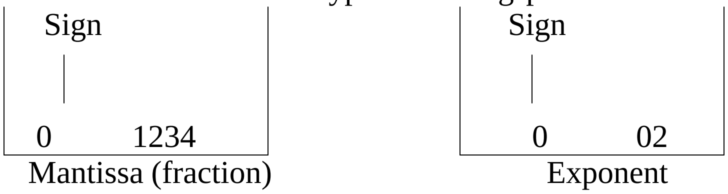
```
             111.1
   110   | 101101
         |  110
          ------
           1010
            110
          ------
            1001
             110
          ------
             0110
              110
          ------
                0
          ------
```

1.6 FLOATING POINT REPRESENTATION

Floating point numbers consists of two parts. The first part of the number is a signed fixed-point number, which is termed as mantissa, and the second part specifies the decimal or binary point position and is termed as an Exponent. The mantissa can be an integer or a fraction.

Example:

A decimal +12.34 in a typical floating-point notation is 12.34=0.1234 * 10^2



This watermark does not appear in the registered version - http://www.clicktoconvert.com

$12.34= 1234 * 10^{-2}$

Sign
0
1234
Sign
1
02

This number in any of the above form (if represented in BCD) requires 17 bits for mantissa (1 for sign and 4 each decimal digit as BCD) and 9 bits for exponent (1 for sign and 4 for each decimal digit as BCD). Exponent indicates the correct decimal location. In the first case where exponent is +2, indicates that actual position of the decimal point is 2 places to the right of the assumed position, while exponent –2 indicates that the assumed position of the point is 2 places towards the left of assumed position. The assumption of the position of the point is normally the same in a computer resulting in a consistent computational environment.

Floating-point numbers are often represented in normalized form. A floating-point number whose mantissa does not contain zero as the most significant digit of the number is considered to be in a normalized form. For example, a BCD mantissa +370 which is 0 0011 0111 000 is in normalized form because these leading zero's are nor part of a 0 digit. On the other hand a binary number 0 01100 is not in a normalized form. The normalized form of this number will be

0 1100

Arithmetic operations involved with floating point numbers are more complex in nature, takes longer time for execution and require complex hardware. Yet the floating point representation is a must as it is useful in scientific calculations. Real numbers are normally represented as floating point numbers.

1.7 COMPLEMENTS

Complements are quite often used to represent negative numbers in digital computers for simplifying the subtraction operation and logical manipulation. For instance, the number N_2 has to be subtracted from N_1 i.e, N_1 - N_2 then without using subtraction the complement form of negative number is formed and then added. It can be pointed out that since the subtraction of number N_2 from N_1 is same as the addition of N_1 and complement of N_2 (i.e) $N_1 + (-N_{2)}.$ There are two types of complements for each base –1 system.

2. The r's Complement
3. The (r-1)'s Complement

When the value of the base is substituted the two types receive the names 2's and 1's complement for binary number or 10's and 9's complement for decimal numbers. The r's complement is sometimes called as "True Complement" and the (r-1)'s complement as "Radix minus one's complement".

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1.7.1 The (r-1)’s complement

Given a positive number N in base r with an integer part of N digits and a fraction part of m digits, then (r-1)’s complements can be defined as r^n - r^m - N. The (r-1)’s complement in decimal system is 9’s complement and 1’s complement in case of binary. Some numerical examples of 9’s complement is as follows:

Example: 1 The 9’s complement of 52510 is
$(10^5 -1) – 52510$
=99999-52510
=47489

The 9’s complement of 0.3266 is
$=(1-10^{-4}) – 0.3266$
=0.9999 – 0.3266
=0.6733

The 9’s complement of 25.638 is
$=(10^2 – 10^{-3}) – 25.638$
=99.999-25.638
=74.361

Example:2

The 1’s complement of 101100 is
$=(2^6 – 1)-101100$
=(1000000 – 1) – 101100
=111111-101100
=010011

The 1’s complement of $(0.0110)_2$ is
$= (1-2^{-4})_{10} – 0.0110$
=1-0.0001 –0.0110
=0.1111-0.0110
=0.1001

The 1’s complement of 10.101 is
$=(2^2 – 2^{-3})_{10} – 10.101_B$
$=(100-0.001)_B - 10.101_B$
=11.1111-10.101
=1.010

1.7.2 The r’s complement

Given a positive number N in base r with an integer part of n digits the r’s complement of N is defined as r^n- N for N 0 and 0 for N=0. The r’s complement in decimal system is 10’s complement and 2’s complement in case of binary system.

Example: 1

The 10’s complement of $(52510)_{10}$ is
$=10^5 – 52510$
=47490

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The 10's complement of 0.3266_{10} is
=1-0.3266
=0.6734
The 10's complement of 25.638_{10} is
$=10^2 - 25.638$
=74.362
The 2's complement of 101100 is
$=(2^6) - 101100_B$
$=(1000000 - 101100)_B$
=010011
The 2's complement of (0.0110) is
$=(1-0.0110)_B$
= 0.1010
The 2's complement of 10.101 is
$=(2^2)_D - 10.101$
$=(100 - 10.101)_D$
=01.011

The r's complement can be obtained from the (r-1)'s complement after the addition of r to the least significant digit.

From the examples, one can notice that:

- ❖ 9's complement of a decimal number is formed simply by subtracting every digit by 9.
- ❖ The 1's complement of a binary number is even simpler to form, the 1's are changed to 0's and 0's are changed to 1's.

For example, one can notice that

- ❖ 2's complement of 101101 is obtained from the 1's complement 010010 by adding 1 to give 010011.
- ❖ 10's complement can be formed by forming 9's complement and then adding a 1 to the least significant digit.

Self – Check Exercise 3

1. Find the 1's and 2's complement of the following fixed point numbers
(a) 10100010
(b) 00000000
(c) 11001100

…………………………………………………………………………………………………
…………………………………………………………………………………………………

2. Add the following numbers in 8 – bit register using signed 2's complement notation.
(a) +50 and –5
(b) +45 and –65
(c) +75 and +85
(d) –75 and –85
Also indicate the overflow condition if any.

…………………………………………………………………………………………………
…………………………………………………………………………………………………

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1.8 BINARY CODED DECIMAL NUMBER REPRESENTATION

In BCD number system a group of binary bit is used to represent each of 10 decimal digits. For instances, an obvious and natural code is a simple weighted binary code as shown in table 1.5.

BINARY CODE	DECIMAL DIGIT
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

Table 1.5 Binary Coded Decimal

This is known as a binary coded decimal 8421 code or simply BCD. Notice that 4 binary bits are required for each decimal digit and each bit is assigned a weight; for instance the rightmost bit has a weight of 1, and the leftmost bit in each code group has a weight of 8. By adding the weights of the positions in which 1's appear, the decimal digit represented by a code group may be derived. This is somewhat uneconomical since 24=16, and thus the 4 bits could actually represent 15 different values. For the decimal number 1246 to be represented, 16 bits are required: 0001 0010 0100 0110

Examples:

Convert decimal 4019 to BCD

4	0	1	9
0100	0000	0001	1001

The BCD equivalent of $(4019)_{10\text{ is}}$ 0100 0000 0001 1001

Convert BCD number 0001 1001 0000 0111 to decimal

0001	1001	0000	0111
1	9	0	7

The decimal equivalent of BCD Number 0001 1001 0000 0111 is 1907. BCD numbers are useful wherever decimal information is transferred into a computer. The pocket calculator is one of the best examples for the application of BCD numbers. Other examples of BCD system are electronic counters, digital voltmeter and digital clocks.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1.9 EXCESS – 3 CODE:

The Excess-3 code is a decimal code that has been used in older computers. This is an unweighted code. Its binary code assignment is obtained from the corresponding BCD equivalent binary number after the addition of binary 3 (0011).

DECIMAL	BCD	EXCESS 3 CODE
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100
10	0001 0000	0100 0011
11	0001 0001	0100 0100

Table 1.6 Excess – 3 code

1.10 GRAY CODE

Digital systems can process data in discrete form only. Many physical systems supply continuous output data. The data must be converted into digital form before they can be used by a digital computer. Continuous, or analog information is converted into digital form by means of an analog to digital converter. The reflected binary or gray code is shown in the table is sometimes used for the converted digital data. The advantage of the gray code over straight binary numbers is that the gray code changes by only one bit as it sequences from one number to the next. In other words, the change from any number to the next in sequence is recognized by a change of only one bit from 0 to 1 data is represented by the continuous change of a shaft position. The shaft is partitioned into segments with each segment assigned a number. If adjacent segments are made to correspond to adjacent Gray code numbers, ambiguity is reduced when the shaft position is in the line that separates any two segments.

Gray code counters are sometimes used to provide the timing sequences that control the operations in a digital system. A gray code counter is a counter whose flip-flop go through a sequence of states. Gray code counters remove the ambiguity during the change from one state of the counter to the next because only one bit can change during the state transition.

Gray Code	Decimal Equivalent	Gray Code	Decimal Equivalent
0000	0	1100	8
0001	1	1101	9
0011	2	1111	10
0010	3	1110	11

This watermark does not appear in the registered version - http://www.clicktoconvert.com

0110	4	1010	12
0111	5	1011	13
0101	6	1001	14
0100	7	1000	15

Table 1.7 : 4 Bit Gray Code

1.11 ARITHMETIC CIRCUITS

Arithmetic circuit such as binary adders, parallel binary adder and BCD adder are explained with circuit diagram.

1.11.1 HALF ADDER

A basic module used in binary arithmetic elements is the half-adder. The function of the half-adder is to add two binary digits, producing a sum according to the binary addition rules shown in the table 1.8.

INPUT	**SUM OF BITS**
0+0	0
0+1	1
1+0	1
1+1	0 With a carry of 1

Table 1.8 Addition Table

The following Figure.1.1 shows a design for a half-adder, two inputs are designated as X and Y and two outputs, designated as S and C. The half-adder perform binary addition operation for two binary inputs as shown in table 1.9. This is arithmetic addition, not logical or Boolean algebra addition.

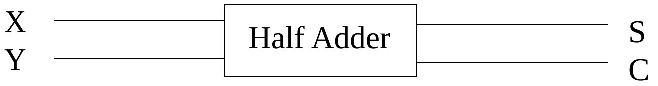


Figure. 1.1 Half Adder – Block Diagram

Input		Output	
X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 1.9 Truth Table

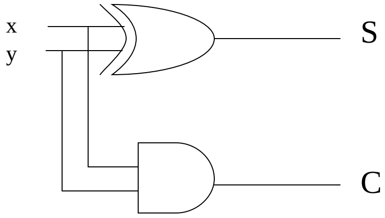


Figure. 1.2 Logic Diagram

This watermark does not appear in the registered version - http://www.clicktoconvert.com

In the half-adder diagram there are two inputs to the half-adder and two outputs. If either of the inputs is a 1 but not both, then the output on the S line will be a 1. If both inputs are 1s, the output on the C line will be a 1. For all other states, there be a 0 output on the carry line. These relationships may be written in Boolean form as follows.

$$S = XY' + X'Y$$
$$C = XY$$

1.11.2 FULL ADDER

The adder circuit is capable of adding the content of two registers. It must include provision for handling carries as well as an addend and augends bits. So there must be three inputs to each stage of a multi digit adder, except the stage for the least significant bits. One for each input from the numbers being added, one for any carry that might have been generated or propagated by the previous stage.

There are three inputs to the full-adder X and Y inputs from the respective digits of the registers to be added, the C_i input, which is for any carry generated by the previous stage. The two outputs are S, which is the output value for that stage of the addition, and C_0, which produces the carry to be added into the next stage. The Boolean expressions for the input output relationships for each of the two outputs are as follows:

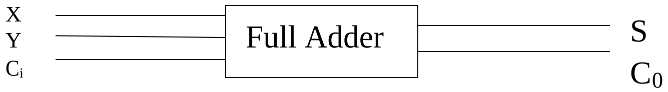


Figure 1.3 Full adder circuit – Block Diagram

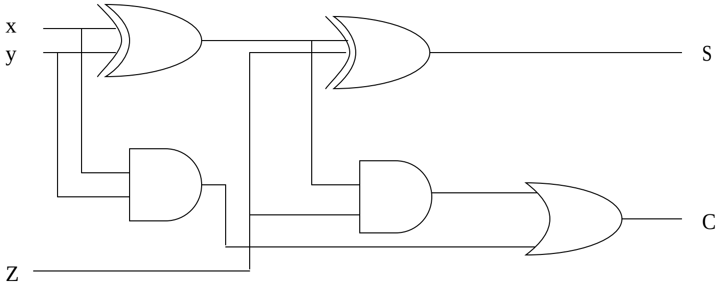


Figure. 1.4 Logic Diagram

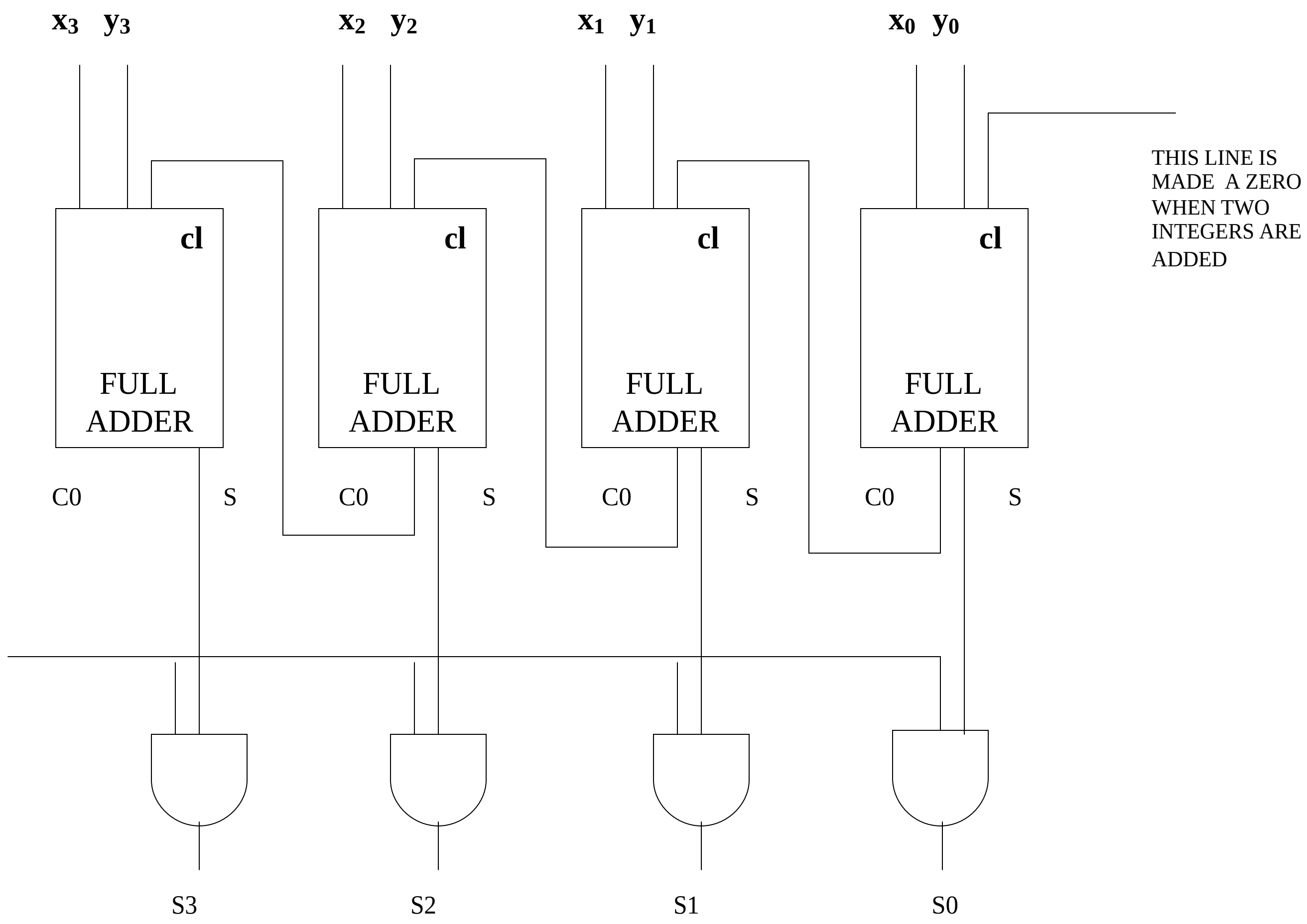
INPUT			OUTPUT	
X	**Y**	**CI**	**S**	**C0**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1
1	1	1	1	1

Table 1.10 Truth Table

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1.11.3 PARALLEL BINARY ADDER

The purpose of this adder is to add two 4-bit binary integers. The adder inputs are named X_0 through X_3 and the augend bits are represented by Y_0 through Y_3. A 4-bit parallel binary adder is illustrated in the figure 1.5.



Consider the additi Figure. 1.5 Parallel Binary Adder

0111 whe

0011 where Y_3=0, Y_2=0, Y_1=1 and Y_0=1

Sum = 1010

The sum should therefore be S_3=1, S_2=0, S_1=1 and S_0=0. The operation of the adder may be checked as follows. Since X_0 and Y_0 are the least significant digits, they cannot receive a carry from a previous stage. In the problem above X_0 and Y_0 are both 1s, their sum therefore 0 and a carry is generated and added into the full-adder for bits X_1 and Y_1. Bits X_1 and Y_1 are also both 1s, as is the carry input to this stage.

Therefore the sum output line S_1 carries a 1 and the carry line to the next stage also carries a 1. Since X_2 is a 1, Y_2 is a 0 and the carry input is 1. The sum output line S_2 will carry a 0, and the carry to the next stage will be a 1. Both inputs X_3 and Y_3 are equal to 0, and the carry input line to this adder stage is equal to 1. Therefore, the sum output line S_3 will represent a 1 and the carry output line designated as "overflow" will have a 0 output.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1.11.4 BINARY CODED DECIMAL ADDER

Arithmetic units which perform operations on numbers stored in BCD form must have the ability to add 4-bit representations of decimal digits. To do this BCD adder is used. A block diagram symbol for an adder is shown in the figure 1.6.

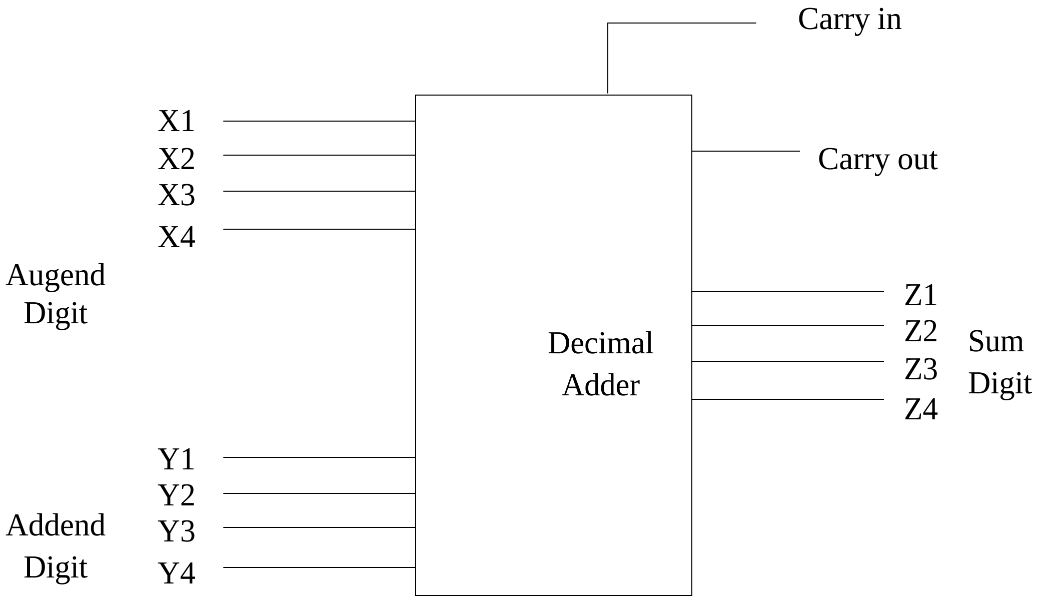


Figure 1.6 BCD Adder

The adder has an augends digit input consisting of four lines, an addend digit input for four lines, a carry-in and a carryout, and a sum digit with four output lines. The augend digit, addend digit and sum digit with four-output line. The augend digit, addend digit and sum digit are each represented 8, 4, 2, 1 BCD form. The purpose of the BCD adder in figure 1.6 is to add the augend and addend digits and the carry-in and produce a sum digit and carry out.

There are eight inputs to the BCD adder; four X_i, or augend, inputs and four Y_i (or) addend digits. Each input will represent a 0 or a 1, during a given addition. If 3 (0011) is to be added to 2 (0010), then X_8=0, X_4=0,X_2=1 and X_1=1; Y_8=0,Y_4=0,Y_2=1 and Y_1=0.

A further difficulty arises when a carry is generated. If 7_{10} (0111) is added to 6_{10} (0110), a carry will be generated, but the output from the base – 16 adder will be 1101. This 1101 does not represent any decimal digit in the 8,4,2,1 system and must be connected. The method used to correct this is to add 6_{10} (0110) to the sum from the base – 16 address whenever a carry is generated. This addition is performed by adding 1's to the weight 4 and weight 2 position output lines from the base – 16 adder when a carry is generated.

The adder performs base – 16 addition and corrects the sum, if it is greater than a1 by adding 6. Several Examples of this are shown below.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

(i)8+7=15 1000+0111
(ii) 9+5=14 1001
$\underline{0101}$
1110
+$\underline{0110}$
Carry Generated $\underline{10100}$ = 14

1.11.5 HALF SUBTRACTOR

A half subtractor subtracts a bit from another. The subtraction table (or truth table) of a half subtractor is shown below. The half subtractor has two input bits A and B two output bits, a difference DIFF = (A-B) and a Borrow.

INPUTS		BORROW	DIFF
A	B		
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

DIFFERENCE= $\overline{A}B+AB$
BORROW= $\overline{A}B$

Table 1.11 Truth Table

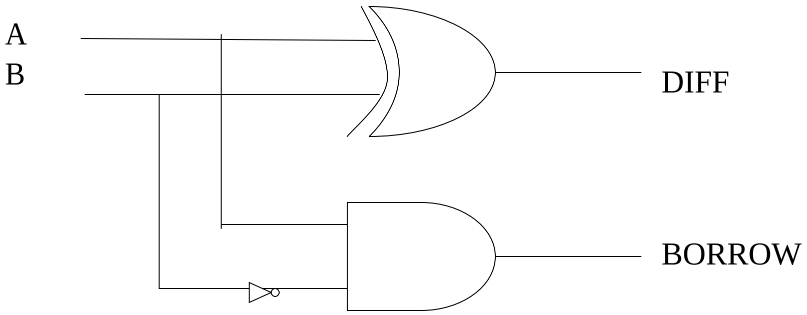


Figure 1.7 Half Subtractor

From the truth table shown in the table 1.11, it can be seen that DIFF = A-B and borrow which has been implemented on the logic circuit for half subtractor.

1.11.6 FULL SUBTRACTOR

A full subtractor subtracts with three bits (A-B-C). The third bit C is the borrow from previous stage. The truth table of a full subtractor is given in table 1.12.

From the truth table of the full subtractor it can be seen that DIFF = (A B C) and borrow = A’B + BC + CA’. This logic has been implemented.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

INPUTS			BORROW	DIFF
A	B	C		
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
1	0	0	0	1
0	1	1	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

⊕ ⊕

Table 1.1.2 Truth Table

1.11.7 PARALLEL BINARY SUBTRACTOR

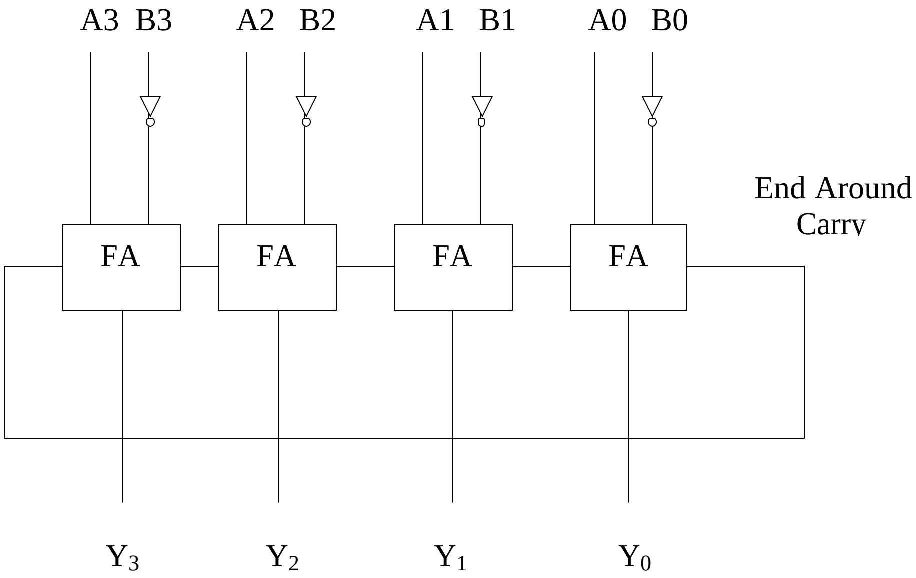


Figure 1.8 Parallel Binary Subtractor

When a binary number is to be subtracted from another using 1's complement method, the following circuit can be used. The number to be subtracted is first complemented using inverters. The complemented number is then added to the minuend using full adders. The carry resulting from the addition is added to the least significant bit as shown in Figure 1.8.

PARALLEL BINARY ADDER - SUBTRACTOR

In implementing a combined adder and subtractor circuit, the primary problem of complementing a number has to be taken care, that is, to form the complement of the number to be subtracted. For the 1's complement system, if the storage register is composed of flip flops, the 1's complement can be formed by simply connecting the complement of each input to the

This watermark does not appear in the registered version - http://www.clicktoconvert.com

adder. The 1 which must be added to the least significant position to form a 2's complement may be added when the two numbers are added by connecting a 1 at the CARRY input of the adder for the least significant bits.

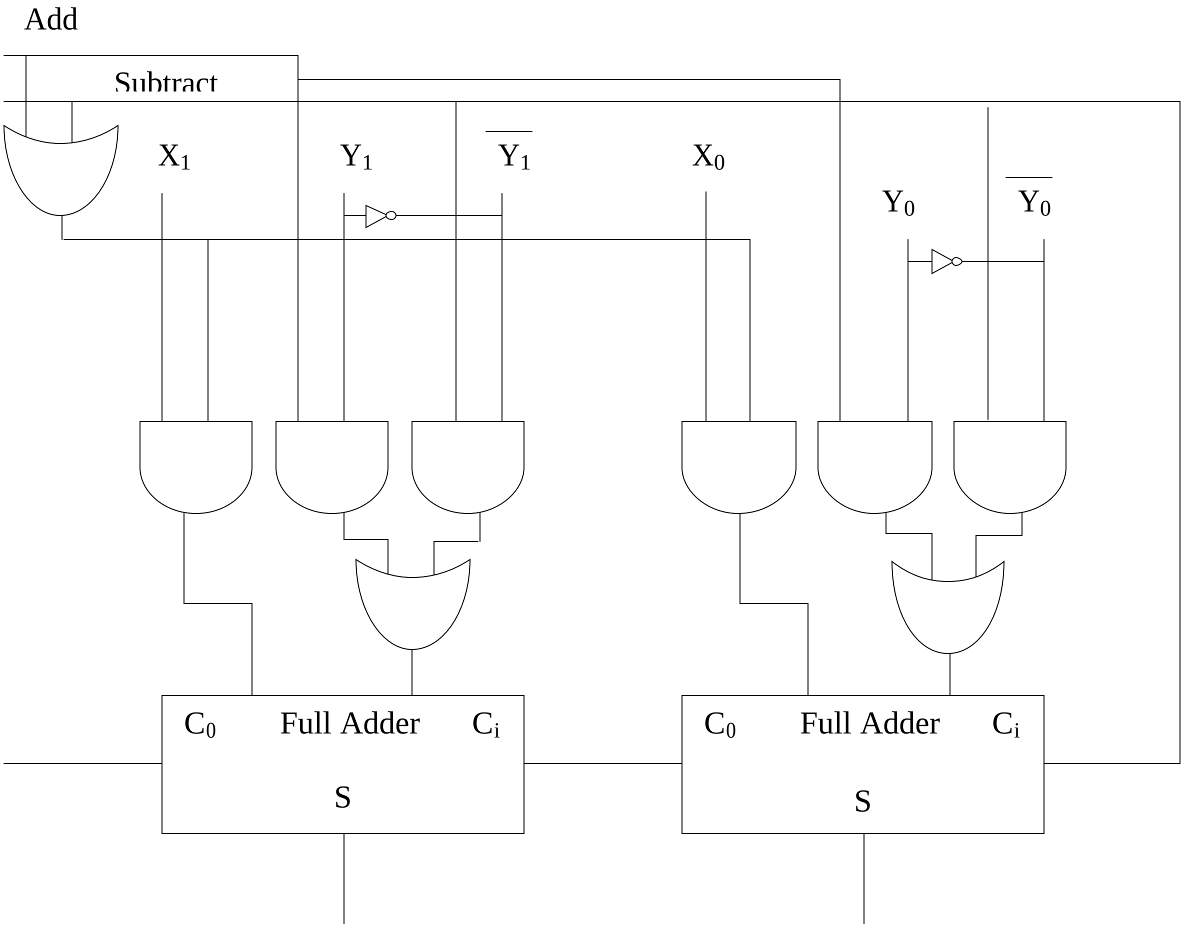


Figure. 1.8.(a) Parallel Arithmetic Element

A complete logical circuit capable of adding or subtracting two signed 2's complement number is shown below.

To add: Add line is made 1.
To Subtract: Subtract line is made 1.

One number is represented by X_1 X_0 and the other number is represented by Y_1Y_0. There are two control signals. ADD and SUBTRACT. If the add control line is made a 1 then the sum of the number X and Y will appear as S_1 S0. If the subtract line is made a 1, then the difference between X and Y will appear S_1 $S_{0.}$

The AND to OR gate network connected to each input selects either Y or Y. An ADD causes Y_1 to enter the appropriate full adder, while a subtract causes Y_1 to enter the full adder, to either add or subtract each x input is connected to the appropriate full adder, and a 1 is added by connecting the SUBTRACT signal to the C_i input of the full adder for the lowest order bits X_0 and Y_0.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The configuration in the figure 1.8.1 is the most frequently used for addition and subtraction because it provides a simple, direct means for either adding or subtracting positive numbers or negative numbers.

Self Check Exercise 4

1. Can a full adders be constructed using two half adders?
……………………………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

1.12 Digital Logic

Since Boolean functions are expressed in terms of AND, OR & NOT operations, it is easier to implement a Boolean function with these types of gates. The possibility of constructing gates for other logic operations is of practical interest. Factors to be weighted when considering the constructing of other types of logic gates are

(1) The feasibility and economy of producing the gate with physical components,
(2) The possibility of extending the gate to more than two inputs,
(3) The basic properties of the binary operator, such as commutativity and associativity, and
(4) The ability of the gate to implement Boolean functions alone or in conjunction with other gates.

Boolean Function	**Operator Symbol**	**Name**	**Comments**
F_0=0		Null	Binary constant 0
F_1=xy	x.y	AND	x and y
F_2=xy’	x/y	Inhibition	x but not y
F_3=x		Transfer	x
F_4=x’y	y/x	Inhibition	y but not x
F_5=y		Transfer	Y
F_6=xy’+x’y	x ⊕ y	Exclusive – OR	x or y,but not both
F_7=x+y	x+y	OR	X or y
F_8=(x+y)’	x ˘ y	NOR	Not – OR
F_9=xy+x’y’	(x⊕ y)’	Equivalence	x equals y
F_{10}=y’	y’	Complement	Not y
F_{11}=x+y’	x ∨ y’	Implication	If y, then x
F_{12}=x’	x ∧ y’	Complement	Not x
F_{13}=x’+y	(x ∧ y)’	Implication	If x, then y
F_{14}=(xy)’	x ^ y	NAND	Not - AND
F_{15}=1	All ones	Identity	Binary Constant 1

Table 1.13 Boolean Expression for the 16 Functions of two variables

This watermark does not appear in the registered version - http://www.clicktoconvert.com

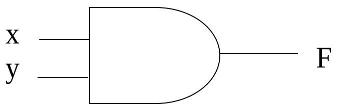
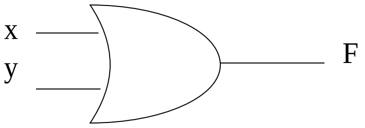
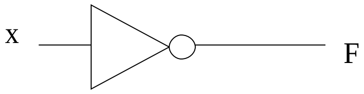
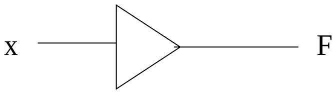
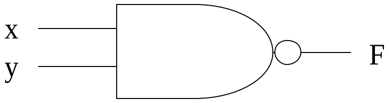
Of the 16 functions defined in the table 1.13, two are equal to a constant and four are repeated twice. There are only ten functions left to be considered as candidates for logic gates. Two- inhibition and implication-are not commutative or associative and thus are impractical to use as other logic gates. The other eight: complement, transfer, AND, OR, NAND, NOR, exclusive-OR, and equivalence, are used as standard gates in digital design.

Each gate has one or two binary input variables designated by x and y and one binary output variable designated by F. The AND, OR, and inverter circuits were defined. The inverter circuit inverts the logic sense of a binary variable. It produces the NOT, or complement, function. The small circle in the output of the graphic symbol of the inverter (referred to as a *bubble*) designates the logic component. The triangle symbol by itself designates a buffer circuit. A buffer produces the transfer function, but does not produce a logic operation, since the binary value of the output is equal to the binary value of the input. This circuit is used for power amplification of the signal and is equivalent to two inverters connected in cascade.

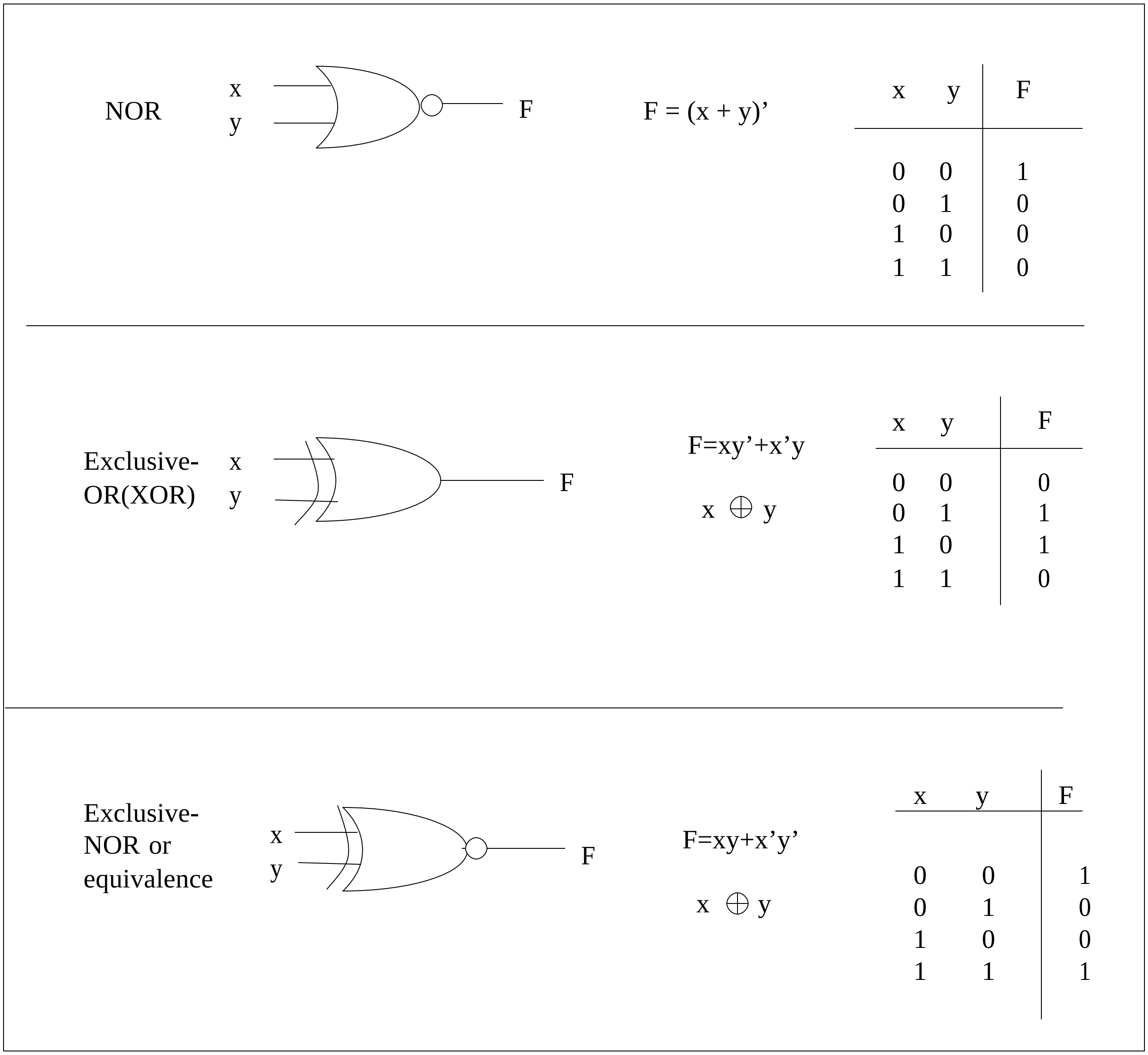
The NAND function is the complement of the AND function, as indicated by a graphic symbol that consists of an AND graphic symbol followed by a small circle. The NOR function is the complement of OR function and uses the OR graphic symbol followed by a small circle. The NAND and NOR gates are extensively used as standard logic gates and are in fact far more popular than the AND and OR gates. This is because NAND and NOR gates are easily constructed with transistor circuits and because digital circuits can be easily implemented with them.

The exclusive-OR gate has a graphical symbol similar to that of the OR gate, except for the additional curved line on the input side. The equivalence, or exclusive-NOR, gate is the complement of the exclusive-OR, as indicated by the small circle on the output side of the graphic symbol.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Name	Graphic Symbol	Algebraic Function	Truth Table
AND		F = xy	x y F 0 0 0 0 1 0 1 0 0 1 1 1
OR		F=x+y	x y F 0 0 0 0 1 1 1 0 1 1 1 1
Inverter		F = x’	x F 0 1 1 0
Buffer		F = x	x F 0 0 1 1
NAND		F= (xy)’	x y F 0 0 1 0 1 1 1 0 1 1 1 0

This watermark does not appear in the registered version - http://www.clicktoconvert.com



x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

Figure 1.9 Digital Logic Gates

EXTENSION TO MULTIPLE INPUTS

The gates shown in Figure. 1.9 – except the inverter and buffer – can be extended to have more than two inputs. A gate can have extended to have multiple inputs if the binary operation it represents is commutative and associative. The AND and OR operations, defined in Boolean algebra, possess these two properties. For the OR function, we have

$$x + y = y + x \text{ (commutative)}$$

and

$$(x + y) + z = x + (y + z)= x + y + z \text{ (associative)}$$

which indicates that the gate inputs can be interchanged and that the OR function can be extended to three or more variables.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The NAND and NOR functions are commutative, and their gates can be extended to have more than two inputs, provided that the definition of the operation is slightly modified. The difficulty is that the NAND and NOR operators are not associative [i.e., (x ˘ y) ˘ z x ˘ (y ˘ z)], as shown in Figure 1.10 and the following equations:

(x ˘ y) ˘ z = [(x + y)’ + z]’ = (x + y)z’ = xz’ + yz’
x ˘ (y ˘ z) = [x + (y + z)’]’ = x’ (y + z) = x’y + x’z

To overcome this difficulty, we define multiple NOR (or NAND) gate as a complemented OR (or AND) gate. Thus, by definition, we have

x ˘ y ˘ z = (x + y + z)’
x ^ y ^ z = (xyz)’

The graphic symbols for the 3-input gates are shown in Figure. 1.11. In writing cascaded NOR and NAND operations, one must use the correct parenthesis to signify the proper sequence of the gates. To demonstrate this, consider the circuit of Figure. 1.11 (c)**.** The Boolean function for the circuit must be written as

F = [(ABC)’(DE)’]’ = ABC + DE

The second expression is obtained from DeMorgan’s Theorem. It also shows that an expression in sum of products form can be implemented with NAND gates.

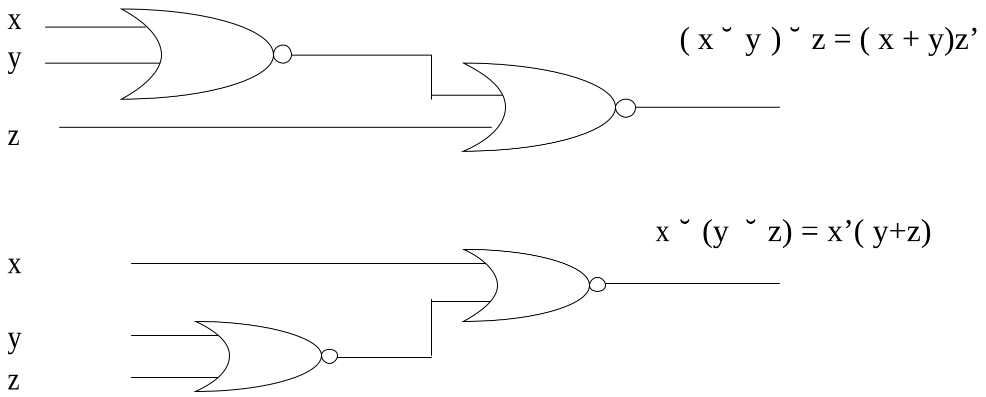
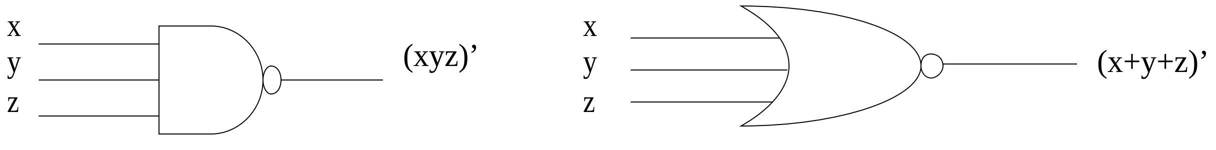
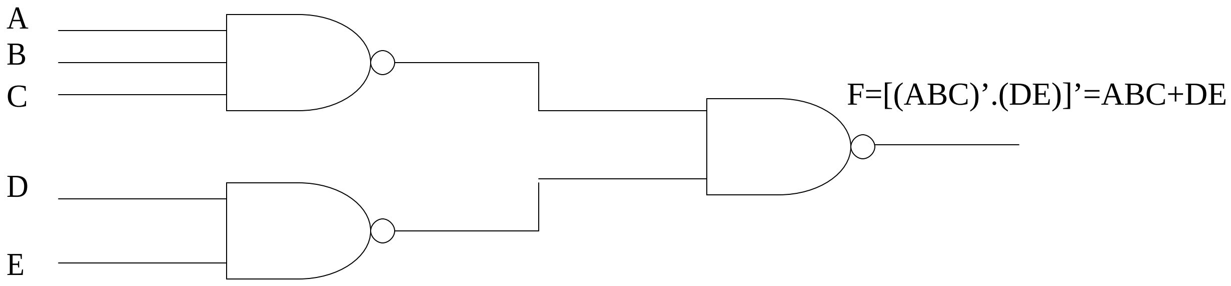


Figure. 1.10 Demonstrating the non associativity of the NOR operator (x ˘ y) ˘ z x ˘ (y ˘ z)



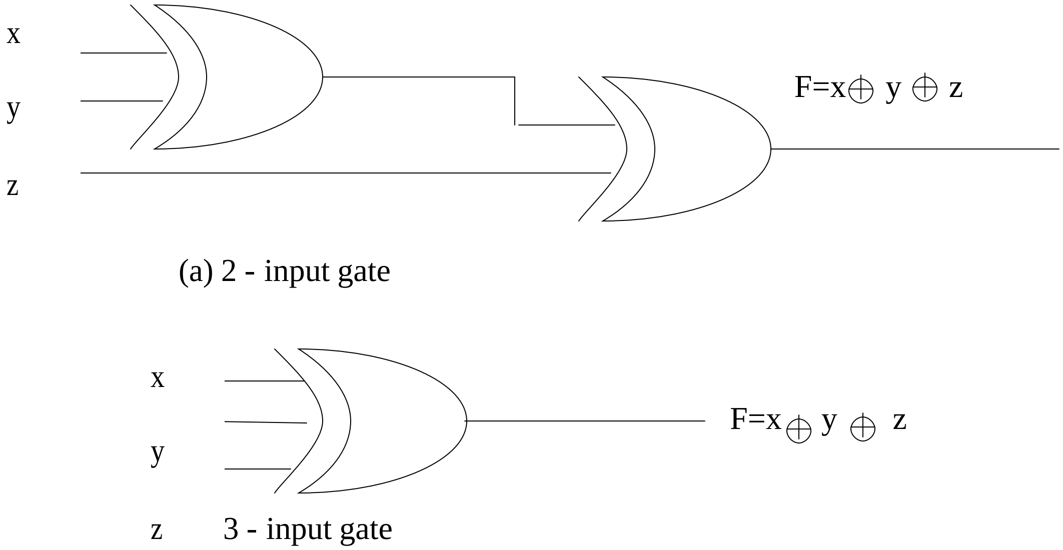
a) 3 - input NAND gate (b) 3 - input NOR gate

This watermark does not appear in the registered version - http://www.clicktoconvert.com



(c) Cascaded NAND gate

Figure 1.11 Multiple Input and Cascaded NOR and NAND gates



(a) 2 - input gate

3 - input gate

Figure. 1.12 3 – input Exclusive – OR gate

x	**y**	**z**	**F**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Table 1.14 Truth table

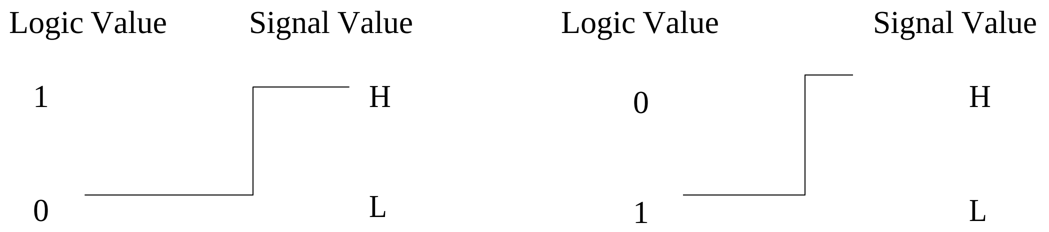
The exclusive-OR and equivalence gates are both commutative and associative and extended to more than two inputs. However, multiple-input exclusive-OR gates are uncommon from the hardware standpoint. In fact, even a 2-input function is usually constructed with other

This watermark does not appear in the registered version - http://www.clicktoconvert.com

types of gates. Moreover, the definition of the function must be modified when extended to more than two variables. The exclusive-OR is an odd function, i.e., it is equal to 1 if the input variables have an odd number of 1's. The construction of a 3-input exclusive-OR function is shown in Figure. 1.12**.** It is normally implemented by cascading 2-input gates, as shown in (a). Graphically, it can be represented with a single 3-input gate, as shown in (b). The truth table in table 1.14 clearly indicates that the output F is equal to 1 only if one input is equal to 1 or if all three inputs are equal to 1, i.e., when the total number of 1's in the input variable is odd.

POSITIVE AND NEGATIVE LOGIC

The binary signal at the inputs and outputs of any gate has one of the two values, except during transition. One signal value represents logic-1 and the other logic-0. Since two signal values are assigned to two logic values, there exists two different assignments of signal level to logic value, as shown in Figure. 1.13. The higher signal level is designated by H and the lower signal level is designated by L. Choosing the high-level H to represent logic-1 defines a positive logic system. Choosing the low-level L to represent logic-1 defines a negative logic system. The terms positive and negative are somewhat misleading since both signals may be positive or both may be negative. It is not the actual sign values that determine the type of logic, but rather the assignment of the logic values to the relative amplitudes of the two signal levels.



(a) Positive Logic

(b) Negative Logic

Figure.1.13 Signal Assignment and Logic Polarity

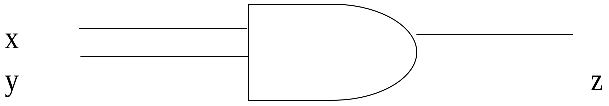
x	**y**	**F**
L	L	L
L	H	L
H	L	L
H	H	H

(a) Truth table with H and L

x
y
Digital gate
z

(b) Gate Block Diagram

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

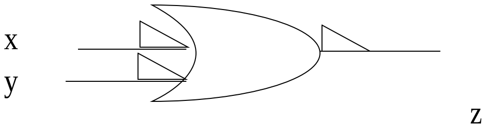


This watermark does not appear in the registered version - http://www.clicktoconvert.com

(c) Truth table for positive logic

x	y	F
1	1	1
1	0	1
0	1	1
0	0	0

(d) Positive logic AND gat



(e) Truth table for Negative logic

(f) Negative logic OR gate

Figure. 1.14 Demonstration of positive and negative logic

Hardware digital gates are defined in terms of signal values such as H and L. It is up to the user to decide on a positive or negative logic polarity. Consider, for example, the electronic gate shown in Figure. 1.14. The truth table for this gate is listed in Figure. 1.14 (a). It specifies the physical behavior of the gate when H is 3 volts and L is 0 volts. The truth table of Figure. 1.14 (c) assumes positive logic assignment, with H=1 and L=0. This truth table is the same as the one for the AND operation. The graphical symbol for a positive logic AND gate is shown in Figure. 1.14 (d).

Now consider the negative logic assignment for the same physical gate with L=1 and H=0. The result is truth table of Figure 1.14 (e). This table represents the OR operation even though the entries are reversed. The graphical symbol for the negative logic OR gate is shown in Figure. 1.14 (f)**.** The small triangles in the inputs and output designate a polarity indicator. The presence of this polarity indicator along a terminal signifies that negative logic is assumed for the signal. Thus, the same physical gate or as negative logic OR gate.

The conversion from positive logic to negative logic, and vice versa, is essentially an operation that changes 1's to 0's and 0's to 1's in both the inputs and the output of a gate. Since this operation produces the dual of a function, the change of all terminals from one polarity to the other results in taking the dual of the function. The result of this conversion is that all AND operations are converted to OR operations (or graphic symbols) and vice versa. In addition, on must not forget to include the polarity-indicator triangle in the graphic symbols when negative logic is assumed.

1.12.1 The Basic Gates

Logical Addition (OR Gate)

If A and B are the input logic variable and Y is the output Variable, the truth table for a two input OR gate is given in Figure. 1.15.

(a) Truth table

A	**B**	**Y**
0	0	0
0	1	1
1	0	1
1	1	1

(b) Graphical Symbol

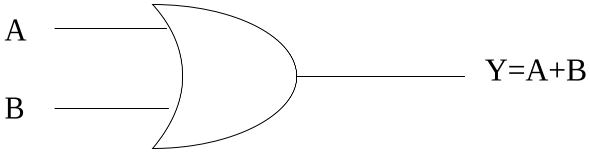


Figure. 1.15 OR gate

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The OR gate produces the inclusive – OR function, that is, the output is 1. If input A or input B or both inputs are 1, otherwise the output is a 0.In other words, in an n input OR gate, if any input is a 1, the output is a 1. For an OR gate the logical relationship of inputs and output can be written as,

Y=A+B+C+D+E+……

Logical Multiplication (AND Gate)

The AND gate has one or more inputs and a single output. The output of an AND gate is equal to the multiplication of its inputs. The truth table for a two input AND gate is shown in Figure. 1.16. The output is 1 if both A and B are 1 and 0 otherwise. In general, in an n input AND gate, only if all the inputs are at logic 1, the output will be 1.

The Input – Output relationships can be written as

Y=A.B.C.D.E.F…..
Or Y=ABCDEF….

(a) Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

(b) Graphic Symbol

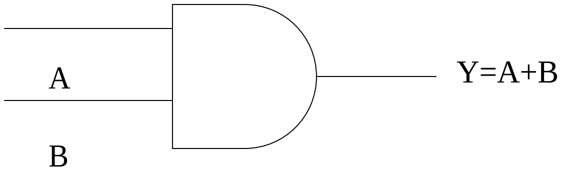


Figure. 1.16 AND Gate

Logical Inversion: Compliment (NOT gate)

The complement function is nothing but inversion, 0 is changed to 1 and 1 to 0. The inverter circuit is also referred to as a NOT gate and it has a single input and single output. The truth table for a NOT gate is shown in table 1.17.

(a) Truth Table

A	Y
0	1
1	0

(b) Graphic Symbol

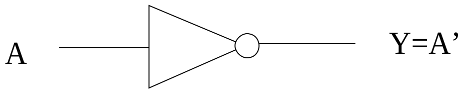


Figure. 1.17 NOT Gate

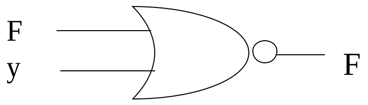
The NOT logic can be written as, Y=A’ or Y=A. The small circle in the output of the graphic symbol of an inverter designates a logic complement.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1.12.2 NOR Gate

A NOR gate is shown in Figure 1.18. Figure shows the NOR gate block diagram symbol with inputs A, B, C and the output ABC. This shows the NOR gate's output will be a 1 only when all three inputs are 0's. If any input represents a 1, then the output of a NOR gate will be a 0. The operation of the gate can be analyzed using the equivalent block diagram circuit as shown in the figure 1.18 which has an equivalent circuit showing an OR gate and an inverter. The inputs A, B, C are OR ed by the OR gate, giving A+B+C, which is complemented by the inverter, yielding (A+B+C)=ABC. The truth table for a NOR gate is shown in table.

A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	0



(a) Truth Table

(b) Graphic Symbol

Figure. 1.18 NOR Gate

1.12.3 NAND Gate

A NAND gate is shown in figure 1.19. The inputs A, B and C and the output from the gate is written A+B+C. The output will be a 1 if A is a 0 or B is a 0 or C is a 0, and the output will be a 0 only if A, B and C are all 1's. The operation of the gate can be analyzed using the equivalent block diagram circuit show in the figure1.19 which has an AND gate followed by an inverter. If the inputs are A, B, and C then the output of the AND gate will be A.B.C and the complement of this is (A.B.C) =A+B+C as shown in the figure 1.19. The truth table for a NAND gate is shown in table.

(a) Truth Table

(b) Graphical Symbol

A	**B**	**Y**
0	0	0
0	1	1
1	0	1
1	1	0

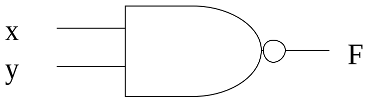


Figure. 1.19 NAND Gate

1.12.4 XOR Gate

The exclusive – OR (XOR), denoted by $\oplus$, is a logical operation that performs the following Boolean operation:

$$x \oplus y=xy'+x'y$$

This watermark does not appear in the registered version - http://www.clicktoconvert.com

It is equal to 1 if only x is equal to 1 or if only y is equal to 1, but not when both are equal to 1, The truth table for a XOR gate is shown in the Figure. 1.20.

(a) Truth Table

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

(b) Graphical Symbol

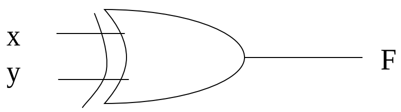


Figure. 1.20 NOR Gate

1.13 LET US SUM UP

This completes our discussion on the introductory concepts of Computer Architecture. The information given on various topics such as data representation, Arithmetic Circuits etc. although is exhaustive yet can be supplemented with additional readings. In fact, a course in an area of computer science must be supplemented by further readings to keep your knowledge up to date, as the computer world is changing with leaps and bounds. In addition to further readings the student is advised to study several Indian Journals on computers to enhance his knowledge.

1.14 LESSON – END ACTIVITIES

1. Correct the following into octal and Hex Decimal (i) $(25)_{10}$ (ii) $(110101)_2$ (iii) $(125.73)_{10}$
2. Discuss in detail about Binary adders.
3. Discuss about Half adder and Full adders.

11.15 POINTS FOR DISCUSSION

1. Try to differentiate the full adder and full subtractor.
2. How can construct the full subtractor.. Using Half subtractor. Explain.
3. Discuss about digital logic circuits.

1.16 CHECK YOUR PROGRESS: MODEL ANSWERS

Self-Check Exercise 1

1.

(a) True
(b) True
(c) True

Self – Check Exercise 2

1.

(a) 12.8125

This watermark does not appear in the registered version - http://www.clicktoconvert.com

(b) 170

2.

(a) 10111
(b) 110001.01
(c) 1101111100

Self- Check Exercise 3

1.

Numbers	10100010	00000000	11001100
1's Complement	01011101	11111111	00110011
2's Complement	01011110	00000000	00110100

2.

(a) 0 0101101 +45
(b) 1 1101100 -25
(c) Overflow
(d) Underflow

Self – Check Exercise 4

1 Yes

1.17 REFERENCES

1. Computer System Architecture, M. Morris Mano, Pearson Education Publication Third Edition.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

UNIT II

COMBINATIONAL LOGIC CIRCUITS & SEQUENTIAL CIRCUITS

2.0 AIMS AND OBJECTIVES

At the end of this unit you will be able to describe:

- What are flip-flops and gates?
- Combinational and sequential circuits and their applications thereof.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

- Some of the useful circuits of a computer system such as multiplexers, decoders etc.
- How a very basic mathematical operation; the addition is performed by a computer.

2.1 INTRODUCTION

In this unit you will be exposed to some of the basic components, which forms the most essential part of a computer. You will come across the terms like computer bus, binary adders, and logic gates, flip flop, combinational and sequential circuits. These circuits are the backbone for any computer system and knowing them will be quite essential.

2.2 BOOLEAN ALGEBRA

Basic concepts of Boolean algebra

Digital computers use the binary number system with bits 0 and 1. The human logic tends to be binary in a number of situations (i.e., true or false, yes or no).

Modern digital computers are designed and maintained, and their operation is analyzed, by using techniques and symbology from a field of mathematics called modern Algebra.

Algebraists have studied for over a hundred years that mathematical systems are called Boolean algebra. The name Boolean algebra honors a fascinating English Mathematician, George Boole, who in 1854 published a classic book. An investigation of the laws of thought, on which are founded the mathematical theories of Logic and Probabilities. Boole's stated intention was to perform a mathematical analysis of logic.

Starting with his investigation of the laws of thought, Boole constructed a "Logical algebra".

The three basic functions in Boolean algebra are

- Logical addition
- Logical Multiplication and
- Complementation

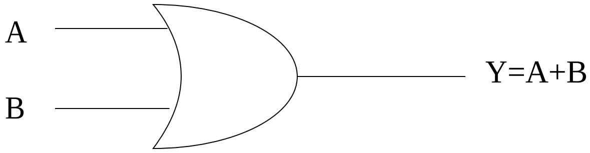
The addition of two logic variables are referred to as OR and addition symbol '+' is used for this operation. Multiplication of two logic variables is referred to as AND, representing this operation with the symbol '.' And complementation of a logic variable is NOT represented by the symbols '~' or''''. In the equation Z=X.Y+X', if X, Y and X' are logic variables, the first term is the logical multiplication of x and y, second term is the complementation of X and the '+' sign in between the two terms represent the logical addition.

1. Logical Addition (OR Gate)

If A and B are the input logic variable and Y is the output Variable, the truth table for a two input OR gate is given in Figure. 2.1.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

A	**B**	**Y**
0	0	0
0	1	1
1	0	1
1	1	1



(a) Truth table for two input OR gate (b)Graphical Symbol of two input OR gate

Figure. 2.1 OR Gate

The OR gate produces the inclusive – OR function, that is, the output is 1, if input A or input B or both inputs are 1, otherwise the output is a 0.In other words, in an n input OR gate, if any input is a 1, the output is a 1. For an OR gate the logical relationship of inputs and output can be written as,

Y=A+B+C+D+E+……

2. Logical Multiplication (AND Gate)

The AND gate has one or more inputs and a single output. The output of an AND gate is equal to the multiplication of its inputs. The truth table for a two input AND gate is shown in Table. The output is 1 if both A and B are 1 and 0 otherwise. In general, in an n input AND gate, only if all the inputs are 1, the output will be 1.

The Input – Output relationships can be written as
Y=A.B.C.D.E.F…..
Or Y=ABCDEF….

(a) Truth Table (b) Graphic Symbol

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

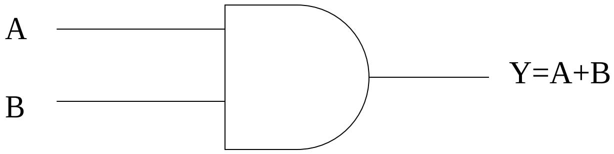


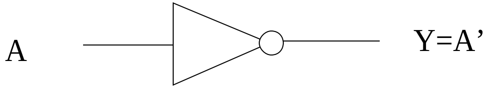
Figure. 2.2 AND Gate

3. Logical Inversion: Compliment (NOT gate)

The complement function is nothing but inversion, 0 is changed to 1 and 1 to 0. The inverter circuit is also referred to as a NOT gate and it has a single input and single output. The truth table for a NOT gate is shown in Figure. 2.3.

(a) Truth Table (b) Graphic Symbol

A	Y
0	1
1	0



This watermark does not appear in the registered version - http://www.clicktoconvert.com

Figure. 2.3 NOT Gate

The NOT logic can be written as, Y=A’ or $Y=\overline{A}$. The small circle in the output of the graphic symbol of an inverter designates a logic complement.

Demorgan’s Theorems

(i) (A+B)’=A’.B’
(ii) (A.B)’=A’+B’

Demorgan’s first theorem states that the complement of a sum equals the product of the complements. Demorgan’s second theorem states that the complement of a product equals the sum of the complement.

Demorgan’s theorems can be extended to any number of variables. The two relationships can be written as,

(i) (A+B+C+D+E+…)’=A’.B’.C’.D’.E’…..
(ii) (ABCD…)’=A’+B’+C’+D’…..

These equalities can be easily proved as shown in truth tables.

Table2.1 Demorgan’s Theorem:1

A	**B**	**A+B**	**(A+B)’**
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

A	**B**	**A’**	**B’**	**A’.B’**
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Table2.2 Demorgan’s Theorem:2

A	**B**	**AB**	**(AB)’**
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

The last column in the left hand side tables is equivalent to the corresponding columns in the right hand side. The complement of any Boolean expression or part of any expression can be found as follows:

1. The + symbols in the given expression should be changed to dots (.) and dot symbols should be changed to +;
2. Each variable in the given expression should be complemented.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

For example, to find the complements of the following expression, Y=A.B.C+A.B'.C Changing the + symbols to dots and vice versa we get, (A+B+C).(A+B'+C); Applying the second rule the expression changes to (A'+B'+C')-(A'+B+C)
Hence Y'=(A'+B'+C').(A'+B+C').

Demorgan's theorem's expresses a basic duality. For example.
(A+B)+C=A+(B+C) is the dual of (AB)C=A(BC)

2.2.2 KARNAUGH MAP

Karnaugh map method for simplifying expressions:

We have examined the derivation of a Boolean algebra expression for a given function by using a table of combinations to list desired function values. To derive a sum of products expression for the function, a set of product terms was listed, and those terms for which the function was to have a value 1 were selected and logically added to form the desired expressions.

The table of combinations provides a nice, natural way to list all values of Boolean function. There are several other ways to represent or list function values, and the use of certain minds of maps, which we will examine, also permits minimization of the expression formed in a nice graphic way.

For example, if the function has only two variables, the number of squares will be 2n, (i.e,)2^2=4. Similarly, for a three input variable function the number of squares in the karnaugh map will be 2^3=8 and for four variable functions, 2^4=16 square will be present.

	A'	A
B'	A'B'	AB'
B	A'B	AB

	B'C'	B'C	BC	BC'
A'	A'B'C	A'B'C	A'BC	A'BC'
A	AB'C'	AB'C	ABC	ABC'

	C'D'	C'D	CD	CD'
A'B'	A'B'C'D'	AB'C'D	A'B'CD	A'B'CD'
A'B	A'BC'D'	A'BC'D	A'BCD	A'BCD'
AB	ABC'D'	ABC'D	ABCD	ABCD'
AB'	AB'C'D'	AB'C'D	AB'CD	AB'CD'

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Table Two input variable function

A	B	Product	Output
0	0	A’B’	0
0	1	A’B	1
1	0	AB’	1
1	1	AB	0

Karnaugh map for 2 input variable function

	0	1
0	0	1
1	1	0

The position of A and B can be interchanged

Table Three input variable function

A	B	C	Product	Output
0	0	0	A’B’C’	0
0	0	1	A’B’C	1
0	1	0	A’BC’	1
0	1	1	A’BC	0
1	0	0	AB’C’	1
1	0	1	AB’C	0
1	1	0	ABC’	0
1	1	1	ABC	1

Karnaugh map for 3 input variable functions

	00	01	11	10
0	0	1	0	1
1	1	0	1	0

As there are 2^n different squares available, each combination of input variable is represented by the square.

Table four input variable function

A	B	C	D	Product	Output
0	0	0	0	A’B’C’D’	0
0	0	0	1	A’B’C’D	1
0	0	1	0	A’B’CD’	1
0	0	1	1	A’B’CD	0
0	1	0	0	A’BC’D’	1
0	1	0	1	A’BC’D’	0
0	1	1	0	A’BCD’	0
0	1	1	1	A’BCD	1

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1	0	0	0	AB’C’D’	1
1	0	0	1	AB’C’D	0
1	0	1	0	AB’CD’	0
1	0	1	1	AB’CD	1
1	1	0	0	ABC’D’	0
1	1	0	1	ABC’D	1
1	1	1	0	ABCD’	1
1	1	1	1	ABCD	0

. Karnaugh map for 4 input variable functions

	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

The input variables are split to represent all the combination of the inputs. For example, in the two variable function, the two variables are treated as x and y. Coordinates and the map is drawn. In the variable function variable A is treated as x and B and C are treated as Y and so on.

In a karnaugh map the adjacent squares differ in bit combination in only one position in the fourth row entries for the same column one, in a four variable function are ABC’D’, AB’C’D’.

For a two variable function F (A,B) there can be four minterms in all A B, AB, AB. For a three variables F (A, B, C) the possible minterms are ABC, ABC, ABC, ABC, ĀBC, ABC (Eight minterms all). In all general when there are n variables. 2^n minterms are possible. In a Boolean function, if all the terms are minterm, then it is said to be in canonical sp form.

Example:

Y= ABC + ABC + ABC is a function in canonical form.

Product of sum canonical form:

Maxterm: If a sum term (or term) of a Boolean function of n variables, contain all the n variables, either complemented or uncomplemented, then it is called a maxterm or a standard sum.

For a two variable function F(A, B) there can be four maxterms in all the (A+B), (A+B), (A+B), (A+B). For a three variable function F (A, B, C) there can be eight maxterm and so on….

This watermark does not appear in the registered version - http://www.clicktoconvert.com

2.2.3 Canonical Form:

Canonical forms for Boolean functions are sum of products (SOP) form and basic forms. Canonical forms of Boolean expressions go into two level logic networks. That is the longest path through which a signal must pass through from input to output is through two gates. This characteristic makes the Canonical forms most desirable.

2.2.4 CONSTRUCTION AND PROPERTIES

Since there are a finite number of Boolean functions of n input variables, yet an infinite number of possible logic expressions you can construct with those n input values, clearly there are an infinite number of logic expressions that are equivalent (i.e., they produce the same result given Boolean function using a canonical, or standardized, form. For any given boolean function there exists a unique canonical form. This eliminates some confusion when dealing with Boolean functions .

Actually, there are several different canonical forms. We will discuss only two here and employ only the first of the two. The first is the so-called **sum of minterms** and the second is the **product of maxterms.** Using the duality principle, it isvery easy toconverty between these two.

2.2.5 PRIME IMPLICANTS

When choosing adjacent squares in a map, we must ensure that all the minterms of the function are covered when combining the squares. At the same time, it is necessary to minimize the number of terms in the expression and avoid any redundant terms whose minterms are already covered by other terms. Sometime there may be two or more expression that satisfy the simplification criteria. The procedure for combining the squares in the map may be made more systematic if we understand the meaning of the referred to as prime implicants and essential prime implicants. A prime implicants is a product term obtained by the combining the maximum possible number of the adjacent squares in the map. If a minterm in a square is covered by only One prime implicant, that prime implicant is said to be essential.

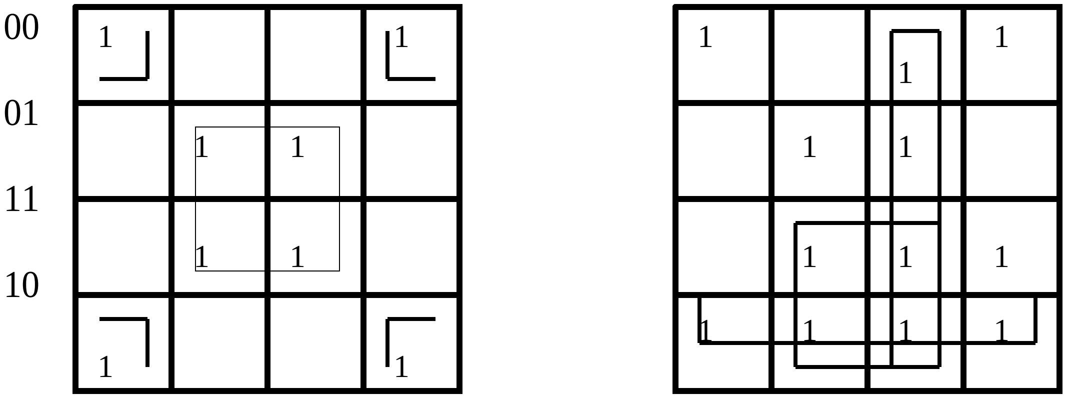
The prime implicant of a function can be obtained from the map by combining all possible maximum numer of squares. This means that a single 1 on a map represents a prime implicant if is not adjacent to any other 1's. Two adjacent 1's form a prime implicant, provided that they are not within a group of four adjacent squares. Four adjacent 1's form a prime implicant are found by looking at each squares, and so on. The essential prime implicants are found by looking at each square marked with a 1 and checking the number of prime implicant that covers the minterm.

Consider the following four-variable Boolean functions:

$$F(A, B, C, D) = \Sigma (0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$$

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The minterm of the function are marked with 1's in the map of Fig. 3-11. Part (a) of the figure shows two essential prime implicants. One term is essential because there is a only one way to include minterm *mo* with in four adjacent squares. These four squares define the term B'D' similarly, there is only one way that minterm *m5* can be combined with four adjacent squares



(a) Essential prime implicant BD and B'D' (b) Prime Implicants CD, B'C AD, and AB'

figure 3-11

2.2.6. SIMPLIFICATION USING-PRIME IMPLICANTS

And this gives the second term BD. The two essential prime implicant cover eight minterms. The remaining three minterms, *m3, m9,* and *m11* must be considered next.

Figure 3-11(b) shows all possible ways the three minterms can be covered with prime implicants. Minterms *m3* can be covered with either prime implicant CD or B'C. Minterm *m9* can be covered with either AD and AB'. Minterm *m11* any one of the four prime implicants. The simplified expression is obtained from the logical sum of the two essentialare four possible ways that the function can be expressed with four product terms of the literals each:

$$
\begin{aligned}
F &= BD + B'D' + CD + AD \\
&= BD + B'D' + CD + AB' \\
&= BD + B'D' + B'C + AD \\
&= BD + B'D' + B'C + AB'
\end{aligned}
$$

The previous example has demonstrated that the identification of the prime implicants in the map helps in determining the alternatives that are available for obtaining a simplified expression.

The procedure for finding the simplified expression from the map requires that we first determine all the essential prime implicants. The simplified expression is obtained from the logical sum of all the essential prime implicants plus other prime implicants that may be needed

This watermark does not appear in the registered version - http://www.clicktoconvert.com

to cover any remaining minterms not covered any remaining minterms not covered by the essential prime implicants. Occasionally, there may be more than one way of combining squares and each combining squares and each combination may produce an equally simplified expression.

2.2.6 DON’T CARE CONDITIONS

The logical sum of the minterms associated with a Boolean function specifies the condition under which the function is equal to 1.The function is equal to 0 for the rest of the minterms. This assumes that all the combinations of the values for the variables of the function are valid. In practice, there are some applications where the function is not specified for certain combinations of the variables. As an example, the four-bit binary code for the decimal digits has six combinations that are not used and consequently are considered as unspecified. Functions that have unspecified outputs for some input combinations are incompletely specified functions. In most applications, we simply don’t care what value is assumed by the function for the unspecified minterms. For this reason, it is customary to call the unspecified minterms of a function don’t-care condition. These don’-care condition can be used on a map to provide further simplification of the Boolean expression.

It should be realized that a don’t-care minterm is a combination of variables whose logical value is not specified. It cannot be marked with a 1 in the map because it would require that the function always be a 1 for such combination. Likewise, putting

A 0 on the square requires the function to be a 0. To distinguish the don’t-care condition from 1’s and 0’s, an X is used. Thus, an X inside a square in the map indicates that we don’t care whether the value of 0 or 1 is assigned to F for the particular minterm.

When choosing adjacent squares to simplify the function in a map, the don’t-care minterm may be assumed to be either 0 or 1. When simplifying the function, we can choose to include each don’t-care minterm with either the 1’s or 0’s, depending on each combinations given the simplest expression.

EXAMPLE 3-9

Simply the Boolean function

$$F(w,x,y,z) = \quad (1,3,7,11,15)$$

Which has the don’t-care conditions

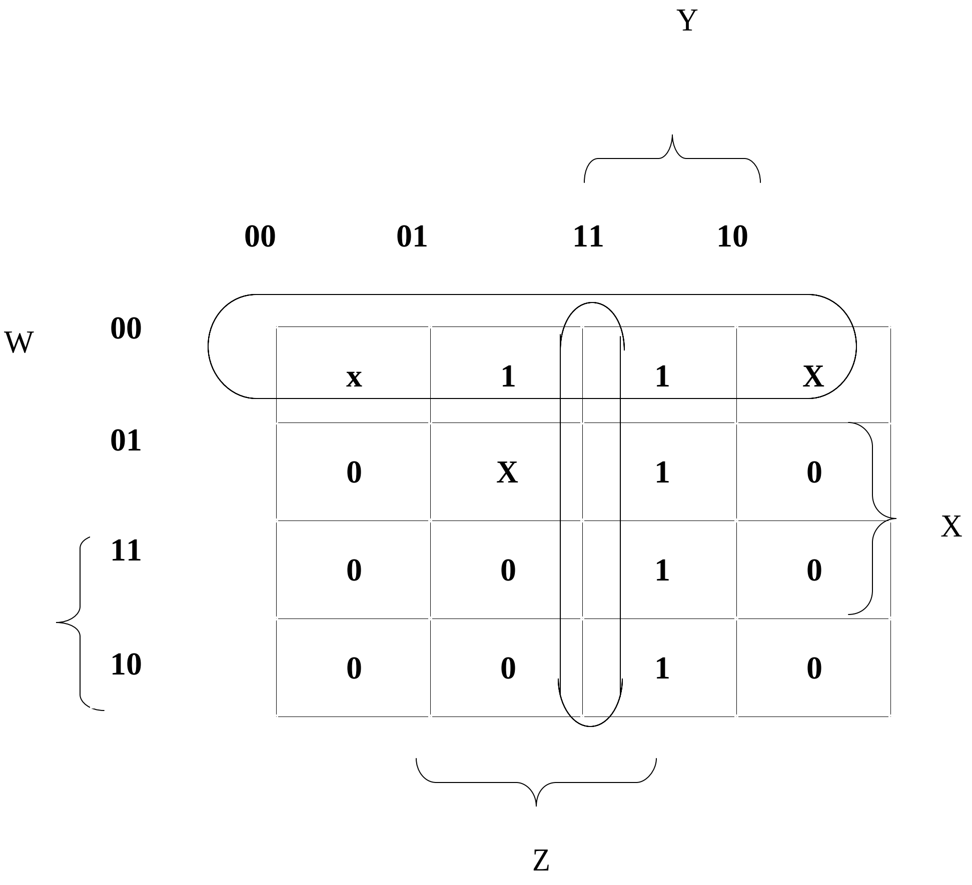
$$d\,(w,x,y,z) = \quad (0,2,5)$$

The minterms of the variable combination that make the function equal to 1. The minterms of d are the don’t-care minterms that may be assigned either 0 or 1. The map simplification is shown in Figure.2.4. The minterms of F are marked b 1’s, those of d are marked by X’s , and the remaining squares are filed with 0’s . To get simplified expression in sum of

This watermark does not appear in the registered version - http://www.clicktoconvert.com

products, we must include all five 1’s in the map, but we may or may not include any of the X’s depending on the way the function is simplified. The term yz covers the four minterms in the third column. The remaining minterm m_1 can be combined with minterm m_3 to give the three-literal term w’x’z’. However, by including one or two adjacent X’s we can combine four adjacent squares to give a two-literal term. In part(a) of the diagram , don’t-care minterms 0 and 2 are included with the 1’s , which results in the simplified function

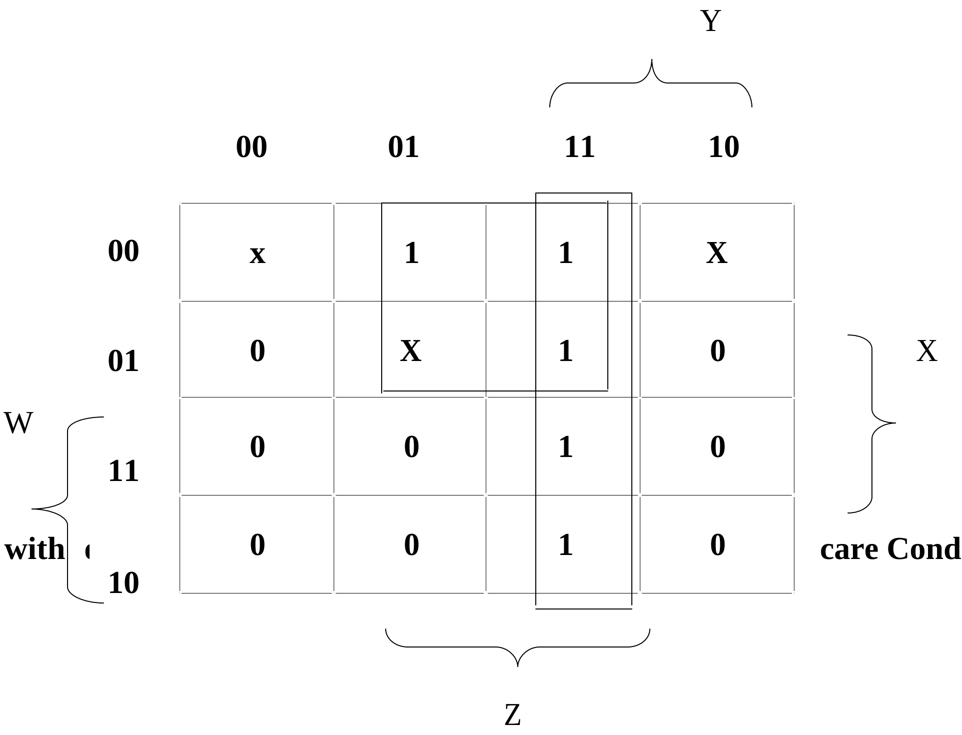
$$F= yz + w'x'$$



	00	01	11	10
00	x	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

Figure 2.4 Karnaugh Map

This watermark does not appear in the registered version - http://www.clicktoconvert.com



	00	01	11	10
00	x	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

Example with (care Conditions

In part (b), don't-care minterm 5 is included with the 1's and the simplified function now is

$$F = yz + w'z$$

Either one of the preceding two expressions satisfies the conditions stated for this example.

The previous example has shown that the don't-care minterms in the map are initially marked with X's and are considered as being either 0 or 1. The choice between 0 and 1 is made de-pending on the way the incompletely specified function is simplified. Once the choice is made, the simplified function obtained will consist of a sum of minterms that includes those minterms that were initially unspecified and have been choosen to be included with the 1's. Consider the two simplified expressions obtained in example

$$F(w,x,y,z) = yz+w'x' = \quad (0,1,2,3,7,11,15\ )$$

$$F(w,x,y,z)= yz+w'z = \quad (1,3,5,7,11,15)$$

Both expressions include minterms 1, 3,7,11 and 15 that make the function F equal to 1. The don't-care minterms 0, 2 and 5 are treated differently in each expression. The first expression includes minterms 0 and 2 with 1's and leaves minterms 5 with the 0's. The second expression represents two functions that are algebraically unequal. Both cover the specified minterms of the function, but each covers different don't-care minterms. As far as the incompletely specified function is concerned, either expression is acceptable because the only difference is in the value of F for the don't-care minterms.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

It is also possible to obtain a simplified product of sums expression for the function of Figure.3-17. In this case, the only way to combine the 0’s is to include don’t-care minterms 0 and 2 with the 0’s to give a simplified complemented function:

$$F' = z'+wy'$$

Taking the complement of F1 gives the simplified expression in product of sums:

$$F(w,x,y,z)=z(w'+y)= \quad (1,3,5,7,11,15)$$

For this case, we include minterms 0 and 2 with 0’s and minterms 5 with the 1’s.

2.2.7 SUM OF PRODUCTS

Minterm: If a product term (AND) of a function of ‘n’ variables, contain all the variables, in complemented form or uncomplemented form, then it is called a “minterm” or “standard product”.

Sum of product method:

The four possible ways to AND two input signals that are in complemented and uncomplemented from. These outputs are called fundamental products. The following table lists each fundamental product next to the input conditions producing a high input for instance, AB is high when A and B are low. AB is high when A is high and so on.

A	B	FUNDAMENTAL PRODUCT
0	0	A’ B’
0	1	A’ B
1	0	A B’
1	1	A B

The idea of fundamental products applies to three or more input variables. For example, assume three input variables: A, B, C and their complements. There are eight ways to AND three input variables and their complements resulting in fundamental products of ABC, ABC, ABC, ABC, ABC, ABC, ABC, and ABC.

The following table summaries the fundamental products by listing each one next to the input condition that results in a high output. For instance, when, A=1, B=0 and C=0, the fundamental products results in an output of Y= ABC = 1.0.0=1

Fundamental products of three inputs

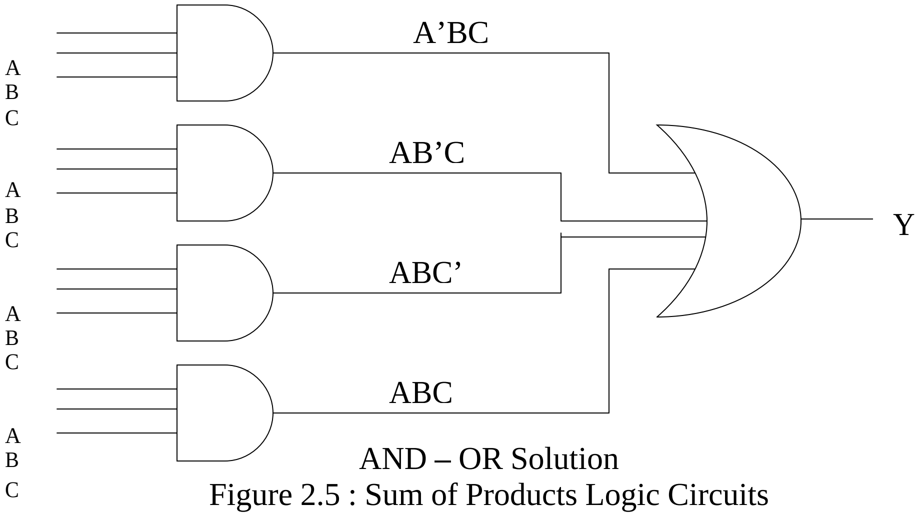
A	B	C	FUNDAMENTAL PRODUCT
0	0	0	A’B’C’
0	0	1	A’B’C
0	1	0	A’BC’

This watermark does not appear in the registered version - http://www.clicktoconvert.com

0	1	1	A’BC
1	0	0	AB’C’
1	0	1	AB’C
1	1	0	ABC’
1	1	1	ABC

Logic circuit:

After you have a sum of products equation, you can derive the corresponding logic circuit by drawing and AND-OR network, or what amounts to the same thing, a NAND-NAND network.



AND – OR Solution

Figure 2.5 : Sum of Products Logic Circuits

2.2.8 PRODUCTS OF SUMS

In a Boolean function, if all the terms are maxterm, then it is said to be in canonical product of the sum form.

Y= (A+B+C) (A+B+C)(A+B+C) is a canonical product of sums function.

Products of sums method:

With the sum of products method and design starts with a truth table that summarizes the desired input- output conditions. The next step is to convert the truth table into an equivalent sum of products equation. The final step is to draw the AND –OR network or its NAND-NAND equivalent.

The product of sums method is similar. Given a truth table you identify the fund mental sums needed for a logic design. Then by ANDing these sums equation corresponding to the truth table.

Wit the sum of products method, the fundamental product produce an output 1 for the corresponding input condition. But with the product of sum method, the fundamental sum products an output 0 per the corresponding input condition.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Converting the truth table to an equation

A	B	C	Y
0	0	0	0 A+B+C
0	0	1	1
0	1	0	1
0	1	1	0 A+B'+C'
1	0	0	1
1	0	1	1
1	1	0	0 A'+B'+C
1	1	1	1

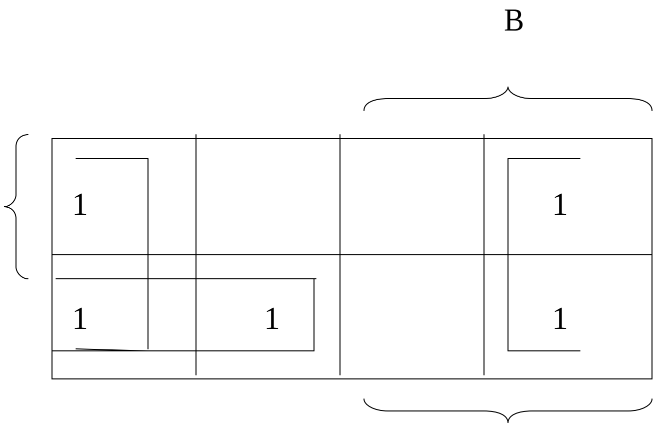
Here consider the following truth table into equation you want to get the product of sum equation. Here A=0, B= 0, C= 0 then sum for three inputs is A+B+C

There are four squares marked with 1's. One for each minterm that produces 1 for the function. These squares belong to minterms 3, 4, 6 and 7. Two adjacent squares are combined in the third column. This column belongs to both B and C and produces the term BC. The remaining two squares with 1's in the two comers of the second row are adjacent and belong to row. A and the two columns of C', so they produce the term AC'. The simplified algebraic expression for the function is the OR of the two terms.

F= BC+ AC'

The second example simplifies the following Boolean function.

F (A, B, C) = (0, 2, 4, 5, 6)

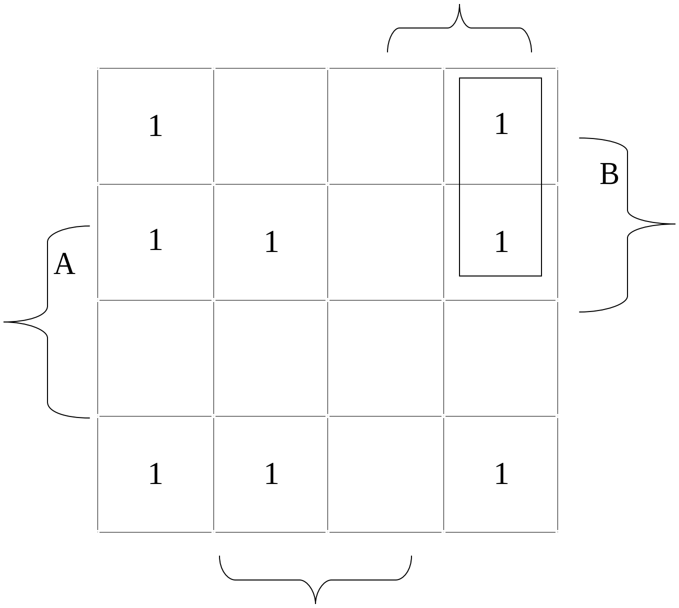


1			1
1	1		1

The five minterms are marked with 1's in the corresponding squares of the three variable maps shown in the above figure. The four squares in the first and fourth column are adjacent and represent the term C'. The remaining square marked with 1 belongs to minterm 5 and can be combined with the square of minterm 4 to produce the term A. The simplified function is F = C' + AB'

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The third example needs a four variable map.



1			1
1	1		1
1	1		1

Map for F (A, B, C, D) = (0, 1, 2, 6, 8, 9, 10)

The area in the map covered by these four variable function consists of the squares marked with 1's in the four corners that, when taken as a group, gives the term B'D'. This is possible because these four squares are adjacent when the map is considered with top and bottom or left and right edges touching. The two 1's on the left of the top row are combined with the two 1's on the left of the bottom row to give the term B'C'. The remaining 1 in the square of minterm 6 is combined with minterm 2 to give the term A' CD'. The simplified function is

F =B'D'+ B'C'+ A' CD'

2.2.9 SIMPLIFICATION

The product denotes the AND terms and the sum denotes the OR of these terms. It is convenient to obtain the algebraic expression for the function in a product – of – sums form. With a minor modification, a product of sums form can be obtained from a map.

The procedure for obtaining a product of sums expression follows from the basic properties of Boolean algebra. The 1's in the map represent the minterms that produce 1 for the function. The squares not marked by 1 represent the minterms that produce 0 for the function. If we mark the empty squares, we obtain the complement of the function'. Taking the complement of F' produces an expression for F in product of sums form. The best way to show this is by example.

We wish to simplify the following Boolean function in both sum of products form and product of sums form.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

F(A,B,C,D)= (0,1,2,5,8,9,10)

	C'D'	C'D	CD	CD'
A'B'	1	1	0	1
A'B	0	1	0	0
AB	0	0	0	0
AB'	1	1	0	1

The 1's marked in the above map represent the minterms that produce a 1 for the function. The square marked with 0's represent minterms not included in F and therefore denote the complement of F. Combining the square with 1's gives the simplified function in sum of products form:

F=B'D'+B'C'+A'C'D

If the square marked with 0's are combined as shown in the diagram, we obtain the simplified complemented function.

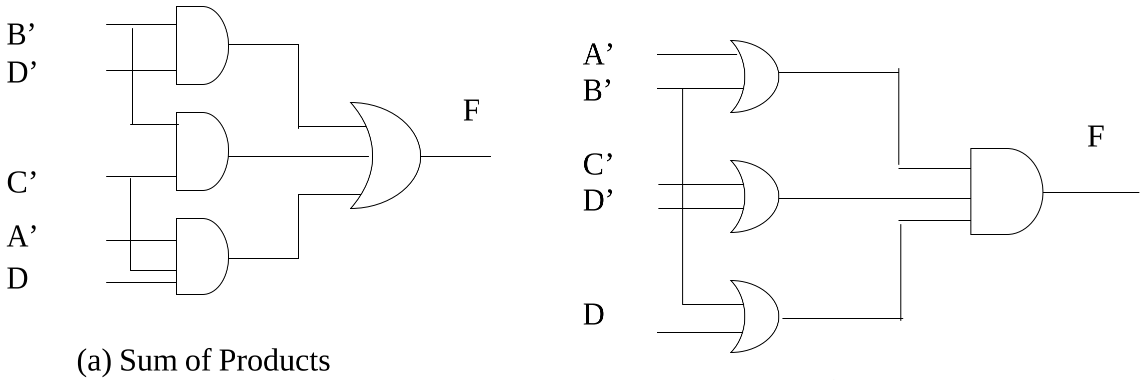
F'=AB+CD+BD'

Taking the complement of F', we obtain the simplified function in product of sums form:

F=(A'+B')(C'+D')(B'+D')

The logic diagrams of the two simplified expressions are shown in the figure 2.6.

Logic diagrams with AND, OR gates.



(a) Sum of Products
F=B'D'+B'C'+A'C'D

(b) Products of Sums
F=(A'+B')(C'+D')(B'+D)

Figure 2.6 Logic diagrams with AND, OR gates.

The sum of products expression is implemented in figure 2.6 (a) with a group of AND gates, one for each AND term. The outputs of the AND gates are connected to the inputs of a single OR gate. The product of sums expression is implemented in figure 2.6 (b) with a group of OR gates, one for each OR term. The outputs of the OR gates are connected to the inputs of a single AND gate. In each case it is assumed that the input variables are directly available in their

This watermark does not appear in the registered version - http://www.clicktoconvert.com

complement, so inverters are not included. The general form by which any Boolean function is implemented when expressed in one of the standard forms. AND gates are connected to a single OR gate when in sum of products form. OR gates are connected to a single AND gate when in product of sums form.

Self Check Exercise: 1

1. Draw a Karnaugh Map for five variables.

……………………………………………………………………………………………..
……………………………………………………………………………………………..
……………………………………………………………………………………………..

2. Map the function having four variables in Karnaugh's map. The function is F(A,B,C,D) = Σ (2,6,10,14).

……………………………………………………………………………………………..
……………………………………………………………………………………………..
……………………………………………………………………………………………..

3. Find the optimal logic expression for the above function. Draw the resultant logic diagram.

……………………………………………………………………………………………..
……………………………………………………………………………………………..
……………………………………………………………………………………………..

4. Can a full adder be constructed by using two half adders?

………………………………………………………………………………………………
………………………………………………………………………………………………
……………………………………………………………………………………..………..

2.3 SEQUENTIAL CIRCUITS

2.3.1 Flip Flop

A flip flop is a bi stable device, that is, it can remain in one of the two stable states which are designated as "0" and "1" states. It is the fundamental logic circuit used for storing information in digital systems. Different types of shift registers and counters are designed only using flip flops, which can be built using NOR gates or NAND gates. A flip flop has two outputs, one of which is the complement of the other. They are called Normal and complement outputs.

2.3.2 RS Flip Flop

The RS Flip flop can be implemented in many ways. One such implementation is shown in Figure 2.7 . There are two inputs to the RS Flip Flop. These lines are used to control the output of the flip- flop. The working of the flip-flop is as follows:

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Case: 1

When S=0 and R=0. Now both the NAND gates A and B output logic 1. Hence the outputs of C and D depend only on the feedback inputs (secondary inputs). The primary inputs, that is outputs of gates A and B are don't cares now. In other words the entire circuit behaves as a latch. Thus the circuit will "hold on" to its previous output. This state is called "HOLD" state.

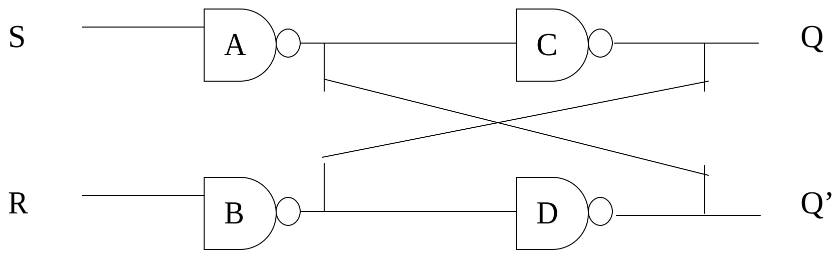


Figure 2.7 RS Flip Flop

Figure 2.7 shows an RS Flip Flop constructed with four NAND gates A, B, C and D. It has two inputs S and R and two outputs Q and Q'. (The state of any flip-flop is known by the state of a output only).

Case: 2

When S=0 and R=1. Now gate A will output 1 and B will output as 0. The output of D that is Q' will be 1, making both inputs to gate c as 0. Hence Q will be in a 1 state. This state of the flip flop is known as "RESET" state.

Case: 3

When s=1 and R=1. This input condition is prohibited. This is because when both S and R are equal to 1, gates A and B will output 0', which will force both Q and Q' to 1. This is against the principle of operation of a flip flop. Moreover, if the inputs are now changed, the next state of the flip flop is unpredictable. The next state actually depends on which gate is faster to change its present state. This prohibited state is also called "RACE" condition.

The above stated input output relations are represented below on a table. This table is called truth table.

INPUTS		**OUTPUT**	**MODE**
S	**R**	**Q(N+1)**	
0	0	Q(N)	HOLD
0	1	0	RESET
1	0	1	SET
1	1	*	Prohibited

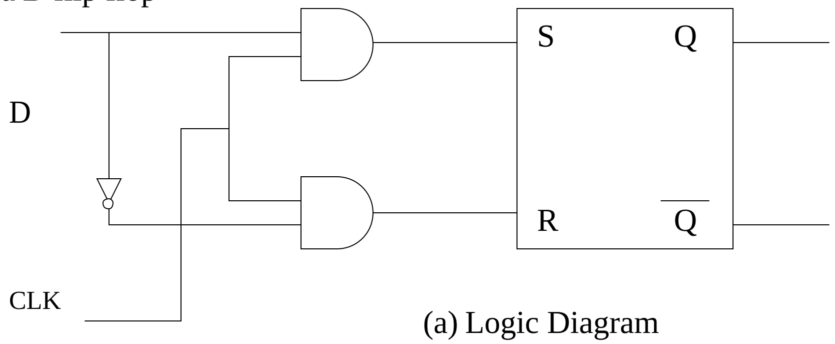
This watermark does not appear in the registered version - http://www.clicktoconvert.com

Case 4:

When S=1 and R=0. This is just the reverse of case 2. On similar arguments we can see that now Q=1 and Q’=0. This state of the flip-flop is known as “SET” state.

2.3.3 D flip flop

In the RS Flip flop the condition R=1 and S=1 is forbidden. This state can be avoided by connecting an inverter between S and R inputs. The flip flop with this modified connection is called a D flip flop



(a) Logic Diagram

CLK	D	Q	D
O	X	Q	Q
1	0	0	1
1	1	1	1

(b) Characteristics Table

Figure 2.8: D Flip Flop

When the clock is 0 and the D input does not affect the output. So when the clock is 0, D is treated as don’t care. The value of D is prevented from reading the output until a clock pulse occurs. When the clock is high, both the AND gates are enabled and the value of D appears at Q. When the clock goes low and last value is retained by Q. This Flip-flop is also called as a delay Flip-flop or data flip-flop.

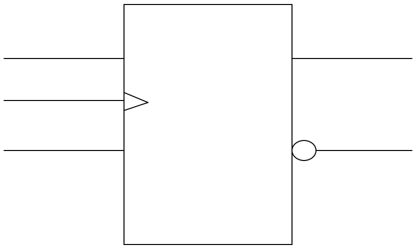
2.3.4 JK Flip-flop

A flip-flop is a refinement of the SR flip-flop in that the in determinate condition of the SR type is defined in the JK type. Inputs J and K behave like inputs S and R to set and clear the flip-flop, respectively. When inputs J and K are both equal to 1, a clock transition switches the outputs of the flip-flop to their complement state.

The graphic symbol and characteristic table of the JK flip-flop are shown in Figure. 2.9. The J input is equivalent to the S (set) input of the SR flip-flop, and the K input is equivalent to the R (clear) input. Instead of the indeterminate condition, the Jk flip-flop has a complement condition Q (t+1)= Q’ (t) when both J and K are equal to 1.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

(a) Graphic symbol



(b) Characteristic table

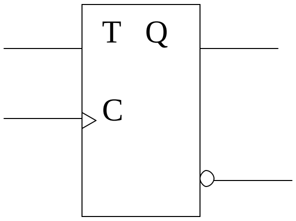
J	K	Q(t+1)	
0	0	Q(r)	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	Q'(t)	Complement

Figure 2.9 JK Flip flop

2.3.5 T Flip Flop

Another type of flip flop is the toggle flip flop. This flip flop is obtained from a JK flip flop when inputs J and K are connected to provide a single input designated by t. The T flip flop therefore has only two conditions. When T=0 (J=K=0) a clock transition does not change the state of the flip flop. When T=1(J=K=1) a clock transition complements the state of the flip flop. These conditions can be expressed by a characteristics equation.

$$Q(t+1)=Q(t)\oplus T$$



T	Q(t+1)	
0	Q(t)	No Change
1	Q'(t)	Complement

(a) Graphic symbol

(b) Truth Table

Figure. 2.10 T flip flop

This watermark does not appear in the registered version - http://www.clicktoconvert.com

2.4 MULTIPLEXER

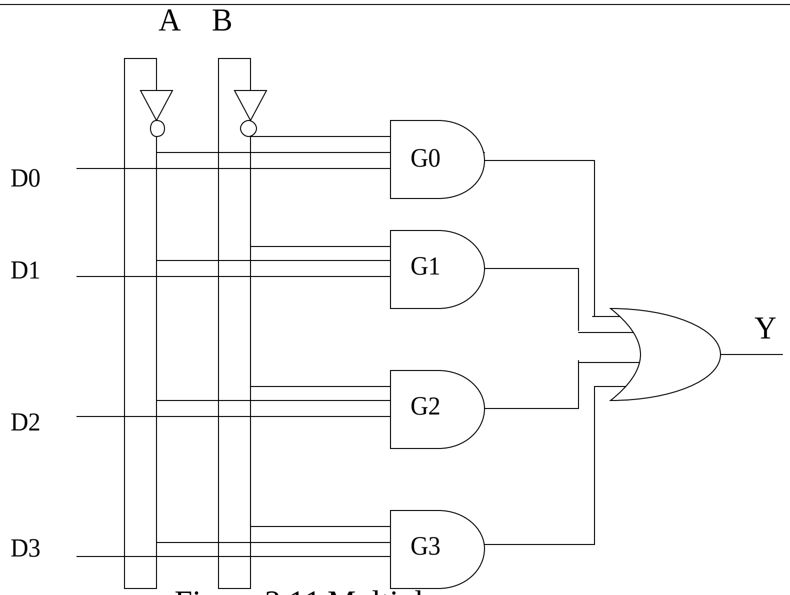


Figure 2.11 Multiplexer

A multiplexer is a combinational logic circuit, which has many inputs and only one output. (Multiplexer means “Many to one”). By applying a suitable control input, any data input can be sent to the output. Figure 2.11 shows a four data input multiplexer. D_3, D_2, D_1 and D_0 are data inputs. A and B are control inputs. Y is the output of the multiplexer. (Note that the multiplexer can be obtained by modifying a decoder circuit).

When control input AB=00, Gate=0 is enabled and hence $Y=D_0$. Similarly if AB=10, Gate 2 is enabled and hence $Y=D_2$ and so on.

2.5 DEMULTIPLEXER

A demultiplexer performs a function opposite to that of a multiplexer. It has one data input and several output lines. Based on the value of the control input, one of the output lines will become active and will output the data input across it. Figure 2.12 shows a 1 to 4 demultiplexer. D is the data input Y_3, Y_2, Y_1 and Y_0. ie. Y_0=D. Similarly when the control input AB=10, Y_2=D.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

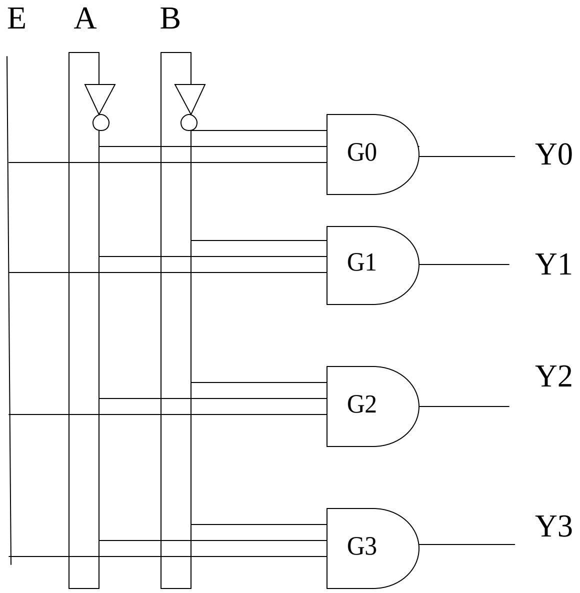
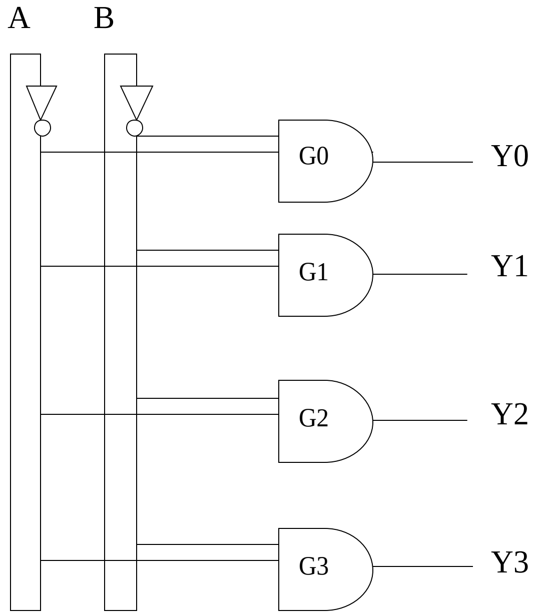


Figure 2.12 Demultiplexer

2.6 DECODER

A decoder has several output lines and control input lines. Based on the value of the control input (or select input), one of the output lines will become active. If there are four control input lines, then the decoder can have up to a maximum of sixteen (2 power 4) output lines. The decoder is generally used to select one among the many devices. It is widely used as address decoder in a computer system. (The outputs of a decoder can become the select inputs to the memory locations. In such case, the control inputs become the address of the memory location). Figure 2.13 shows a simple decoder with two control inputs and four outputs.



This watermark does not appear in the registered version - http://www.clicktoconvert.com

Figure 2.13 Decoder

When the control input AB=00, the AND gate G_0 is enabled and its output Y_0 is high. Suppose AB=10, then G_2 is enabled and hence Y_2 will be high.

2.7 ENCODER

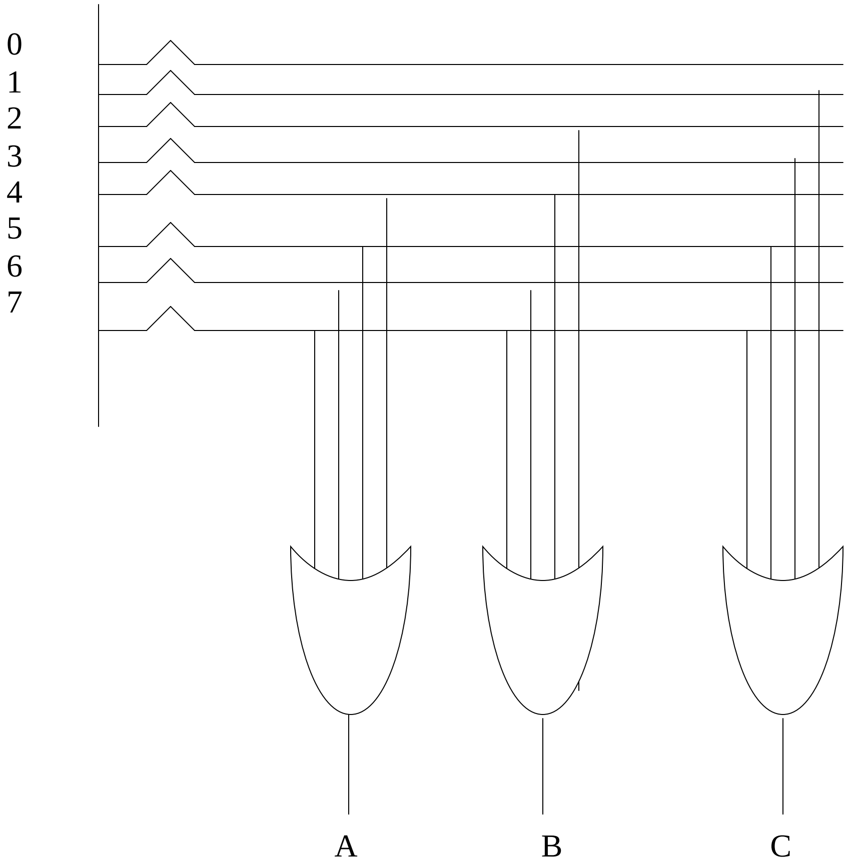


Figure 2.14 Encoder

An encoder is a digital circuit that generates the binary code corresponding to the input number. An octal-to-Binary encoder takes an octal input (in some symbolic form) and generates its binary equivalent as output. Similarly, a Decimal-to-Binary encoder takes a decimal input and generates an equivalent binary output. Figure 2.14 shows an Octal-to-Binary encoder using OR gates. (A Decimal-to-Binary encoder has ten input lines and uses four OR gates at the output).

2.8 COUNTERS

A counter is a sequential circuits that counts the number of incoming clock pulses. It consists of an array of flip-flops. In the following sections, negative edge triggered JK flip-flops are used for discussions on the working of the various types of counters or basically there are two types of counters. “Parallel counters (or) Synchronous counters” and “Ripple counters (or) Asynchronous counters”.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Asynchronous Binary Up Counter

Let us consider a three bit counter for simplicity. It must have a count sequence 000 001…111, as shown in the table below. Q_0, Q_1, Q_2 are outputs of the flip-flops are cleared with a PC RESET input such that $Q_2Q_1Q_0$=000. Thereafter with every clock, the output increments as shown.

CK	Q_2	Q_1	Q_0
RESET	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0
9	0	0	1
10	0	1	0

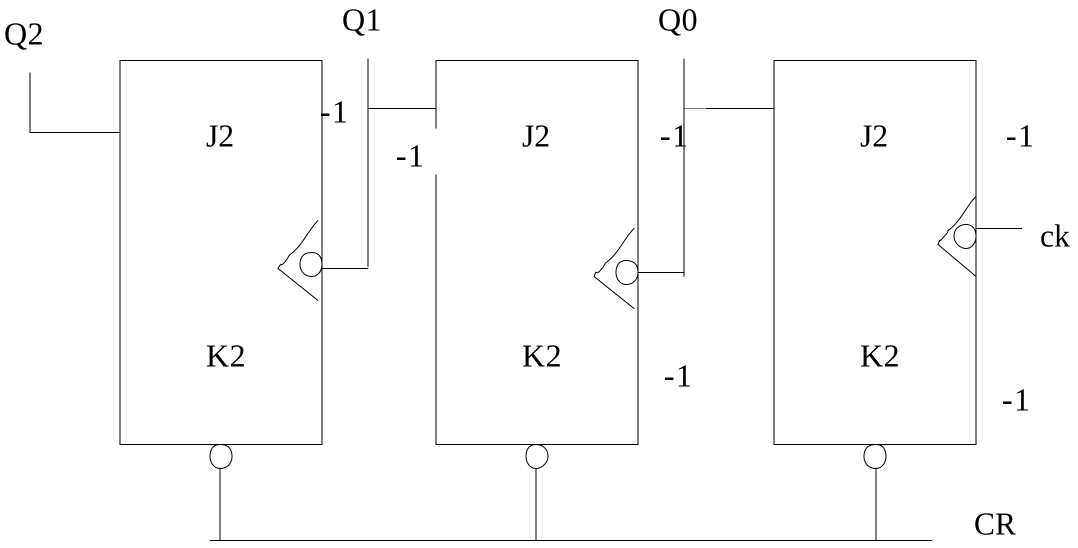


Figure 2.15 Asynchronous Binary Up Counter

After every eight clocks the count sequence repeats itself. From the count sequences table given above it can be seen that Q_0 toggles with every clock. Hence the CK pulse is directly

This watermark does not appear in the registered version - http://www.clicktoconvert.com

applied to the first flip-flop. But Q (i) toggles whenever Q (i-1) makes a negative transition that is when Q(i-1) changes from 1 to 0. Hence the clocks for subsequent flip-flops are obtained from the Q (i) output of the previous flip-flop. Note that in an asynchronous counter the J and K inputs are always kept high so that the flip-flop toggles when a clock arrives.

Asynchronous Binary down Counter

A binary down counter counts with every clock and hence it is initially SET with DCSET input.

CK	Q_2	Q_1	Q_0
SET	1	1	1
1	1	1	0
2	1	0	1
3	1	0	0
4	0	1	1
5	0	1	0
6	0	0	1
7	0	0	0
8	1	1	1
9	1	1	0
10	1	0	1

And so on.

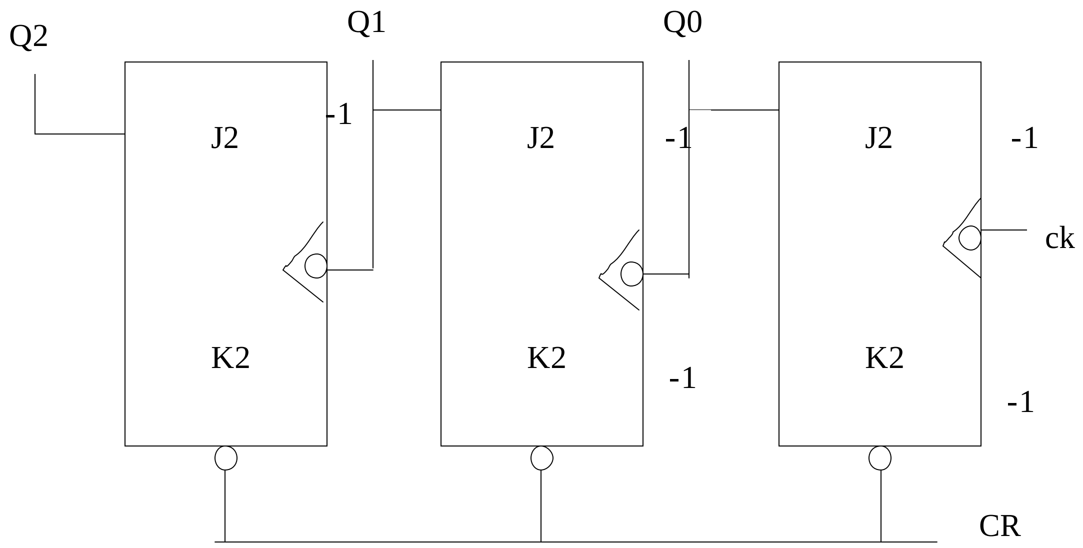


Figure 2.16 Asynchronous Binary down Counter

The counter state table is given below. It can be seen from the count sequence table that Q0 toggles with every clock. Hence the first flip-flop is clocked directly. For the other flip-flops Q (i) toggles only when Q (i-1) makes a positive transition. That is when Q (i-1) makes a negative transition. Hence the clocks are derived from the complementary outputs for these flip-flops.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

For convenience, a three bit natural binary counter is treated in the above discussions. The same logic can be extended to 4-bit (or) 5-bit counter or an n-bit counter.

Synchronous Binary Counter

In a synchronous counter, the flip-flops are all clocked simultaneously. Hence the J and K inputs are made high only when necessary. (Recall that in asynchronous counters, J and K inputs are always high, that is the flip-flops are in toggle mode and when a clock arrives the flip-flop toggles).

CK	**Q2**	**Q1**	**Q0**
RESET	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0
9	0	0	1
10	0	1	0

And so on.

A careful study of the count sequence table will reveal that a flip-flop toggles only when the outputs of the previous flip-flops are high. ie. Q(i) toggles only when
Q(i-1) AND Q(i-2) AND Q(i-3) … AND Q(0)=1.

This fact is made use of in realizing a synchronous counter.

Self Check Exercise : 2

1. What are sequential Circuits? How are they different from combinational circuit?
…………………………………………………………………………………………………
…………………………………………………………………………………………………
…………………………………………………………………………………………………..
.
2. Can Ripple counter be constructed from a shift register?
…………………………………………………………………………………………………
…………………………………………………………………………………………………
…………………………………………………………………………………………………..

This watermark does not appear in the registered version - http://www.clicktoconvert.com

2. 9 LETS US SUM UP

This unit provides you the information regarding the basis of a computer system. The key elements for the design is the combinational and sequential circuits. With this developing scenario in the forefront and the expectations of Ultra Large Scale Integration(ULSI) in view, it is not far off when design of logical circuit will be confined to single microchip components.

2.10 LESSON – END ACTIVITIES

(1) Construct a sequential circuits for J.K. and D flip flops.
(2) Discuss about product of SUM and SUM of the product.

2.11 POINTS FOR DISCUSSIONS

(1) Study about the combinational circuit and sequential circuits.
(2) Analysis the usage of sequential circuits on the basis of digital computer.

2.12 CHECK YOUR PROGRESS: MODEL ANSWERS

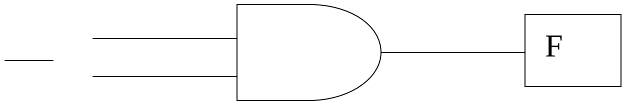
Self-Check Exercise: 1

1.

AB \ CDE								
00								
01								
11								
10								

2. The logic Expression is: $F=C\overline{D}$

3.



4. Yes

Self Check Exercise: 2

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1. The logic circuits present output depends on the past inputs. These circuits store and remember information. The sequential circuits unlike combinational circuits are time dependent. Normally the current output of a sequential circuit depends on the state of the circuit and on the current input to the circuit. It is a connection of flip flops and gates.

2. Yes.

2.13 REFERENCES

1. Digital Principles and Applications, Albert Paul malvino, Donald P Leach, McGrawHill Publishing Company

This watermark does not appear in the registered version - http://www.clicktoconvert.com

UNIT III

MICROPROCESSOR

3.0 AIMS AND OBJECTIVES

At the end of this unit, you should be able to:

- Describe a Microprocessor
- Identify the instruction set and the addressing modes of 8085
- Identify the peripherals and interfaces.

3.1 INTRODUCTION

A microprocessor is a multipurpose, programmable logic device that reads binary instructions from a storage device called memory accepts binary data as input and processes data according to those instructions and provides results as output. A typical programmable machine

This watermark does not appear in the registered version - http://www.clicktoconvert.com

can be represented with three components: microprocessor, memory and I/O as shown in figure3.1

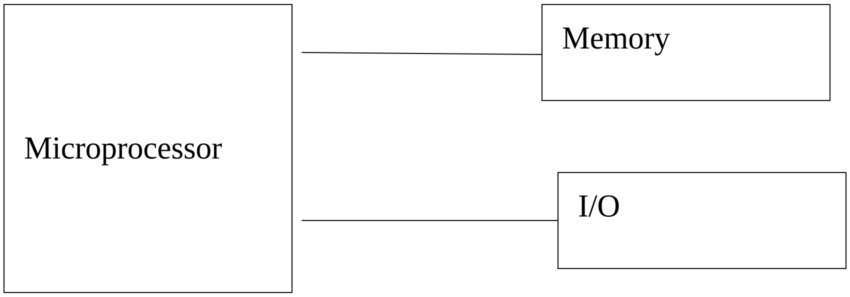


FIGURE 3.1 A Programmable devices

Microprocessor, memory and I/O Components are work together or interact with each other to perform a given task; thus they compromise a system. The physical components of this system are called hardware.

A set of instructions written for the microprocessor to perform a task is called a program. A group pf programs is called software. The machine (system) represented in figure can be programmed to turn traffic lights on and off, compute mathematical functions or keep track of a guidance system.

The microprocessor applications are classified primarily in two categories:

1. Reprogrammable systems
2. Embedded systems

Reprogrammable Systems

In reprogrammable systems, such as microcomputers, the microprocessor is used for computing and data processing.

These systems include general – purpose microprocessors capable of handling large data, mass storage devices and peripherals.

Embedded Systems

In embedded systems, the microprocessor is a part of a final product and is available for reprogramming to the end user. A copying machine is a typical example of an embedded system.

BINARY DIGITS

The microprocessor operates in binary digits 0 and 1, also known as bits. Bit is an abbreviation for term binary digit. These digits are represented in terms of electrical voltages in the machine

This watermark does not appear in the registered version - http://www.clicktoconvert.com

0 represents low voltage level and
1 represents high voltage level.

A MICROPROCESSOR AS A PROGRAMMABLE DEVICE

The microprocessor is programmable that can be instructed to perform given tasks within its capacity. The engineers designing a microprocessor determine a set of tasks the microprocessor should perform and design the necessary logic circuits and provide the user with a list of instructions the processor will understand.

For example, an instruction for adding two numbers may look like a group of eight binary digits such as 1000 0000.These instructions are simply a pattern of 0s and 1s.The user (programmer) selects instructions from the list and determines the sequence of execution for a given task. These instructions are entered or stored in storage, called memory, which can be read by the microprocessor.

MEMORY

Memory is like the pages of a notebook with space for a fixed number of binary numbers on each line. However, these pages are generally made of semiconductor material. Typically, each line is an 8-bit register that can store 8 binary bits, and several of these registers are arranged in a sequence called memory. These registers are always grouped together in powers of two.

INPUT/OUTPUT

The user can enter instructions and data into memory through devices such as keyboard or simple switches. These devices are called input devices. The microprocessor reads the instruction from the memory and processes the data according to those instructions. The results can be displayed by a device such as seven segment LEDS or printed by a printer. These devices are called output devices.

MICROPROCESSOR AS A CPU

We can also view the microprocessor as a primary component of a computer. The CPU is the primary and central player in communicating with devices such as memory, input and output. The timing of the communicating process is controlled by the group of circuits called the control unit. The CPU contains various registers to store data. The arithmetic logic unit (ALU) to perform arithmetic and logical operations, instruction decoders, counters and control lines. The CPU reads instruction from memory and performs the task specified. It communicates with input/output devices either to accept or to send data. With the advent of the integrated circuit technology it became possible to built the CPU on a single chip; this came to be known as microprocessor.

ORGANIZATION OF A MICROPROCESSOR-BASED SYSTEM

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Fig 3.2. Shows a simplified but formal structure of a microprocessor-based system or a product. Since a microcomputer is one among many microprocessor-based systems, it will have the same structure as show in fig 3.2. It includes three components: microprocessor. I/O (input/output) and memory (read/write memory and read-only memory). These components are organized around a common communication path called a bus. The entire group of components is also referred to as a system or a microcomputer system, and the components themselves are referred to as sub-systems. At the outset, it is necessary to differentiate between the terms in popular literature. The microprocessor is one component of the microcomputer. On the other hand, the microcomputer is a complete computer similar to any other computer, except that **CPU** function of the microcomputer is performed by the microprocessor. Similarly, the term peripheral is used for input/output devices. The various components of a microprocessor-based product or microcomputer are shown in fig 3.1 and their functions are described in this section.

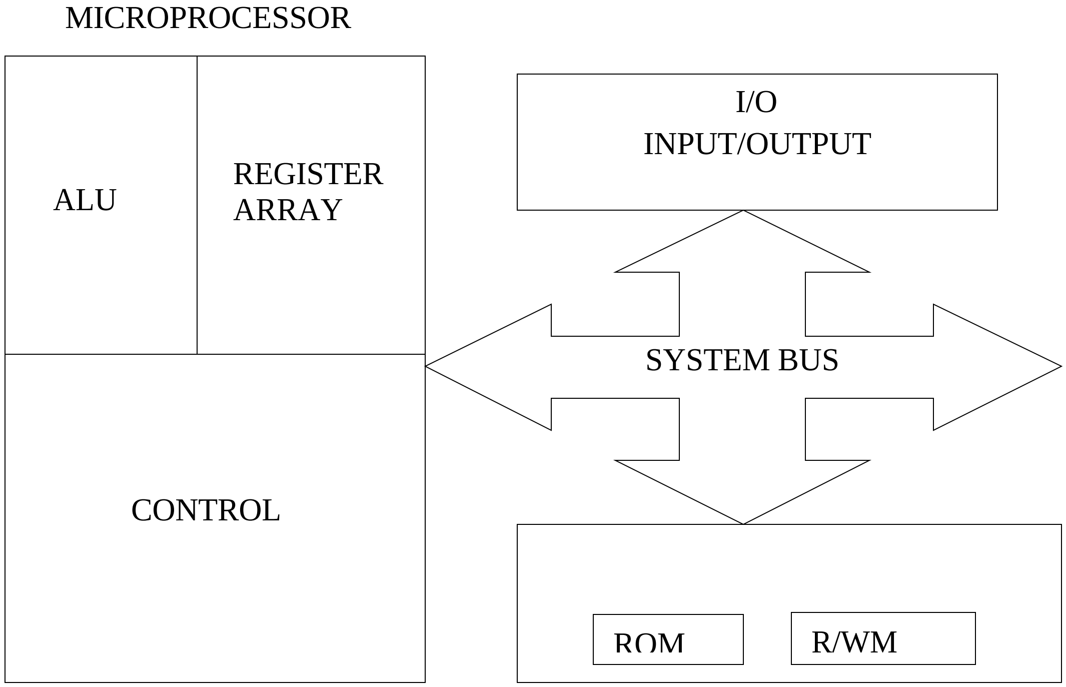


Figure 3.2 Microprocessor-Based Systems with Bus Architecture

MICROPROCESSOR

The microprocessor is a semiconductor device consisting of electronic logic circuits manufactured by using either a large-scale integration (LSI) or very-large-scale integration (VCLSI) technique. The microprocessor is capable of performing various computing functions and making decisions to change the sequence of program execution, in large computers, a CPU implemented on one or more circuit boards performs these computing functions. The microprocessor is in many ways similar to the CPU. The microprocessor can be divided into three segments for the sake clarity, as shown in fig 3.2.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

- Arithmetic/logic unit (ALU)
- Register array
- Control unit.

Arithmetic/Logic Unit

This is the area of the microprocessor where various computing functions are performed on data. The ALU unit performs arithmetic operations as addition and subtraction, and such logic operations as AND, OR and exclusive OR. Results are stored either in register or in memory.

Register Array

This area of the microprocessor consists of various registers. These registers are primarily used to store data temporarily during his execution of a program. Some of the registers are accessible to the user through instructions.

Control Unit

The control unit provides the necessary timing and control signals to all the operations in the microcomputers. It controls the flow of data between the microprocessor and memory and peripherals.

Now the question is: what is the relationship among the programmer's instruction (binary pattern of 0s and 1sa), the ALU, and the control unit? This can be explained with the example of a full adder circuit. A full adder circuit can be designed with registers, logic gates, and a clock. The clock initiates the adding operation. Similarly, the bit pattern of an instruction initiates a sequence of clock signals, activates the appropriate logic circuits in the ALU, and performs the task. This is called microprogramming, which is done in the design stage of the microprocessor. The bit pattern required to initiate theses micro program operations are given to the programmer in the form of an instruction set of the microprocessor. The programmer selects appropriated bit patterns from the set for a given task and enters them sequentially in memory through input devices. When the CPU reads these bit patterns one at a time, it initiates appropriate micro programs through the control unit, and performs the task specified in the instructions.

MEMORY

Memory stores such binary information as instructions and data, and provides that information to the microprocessor whenever necessary. To execute programs the microprocessor reads instructions and data from memory and performs the computing operations in its ALU section. Results are either transferred to the output section for displayer stored in memory for later use. The memory block shown in fig 1.3 has two sections

- Read only memory (ROM)
- Red/Write memory (R/WM), popularly known as Random- Access memory (RAM).

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The ROM is used to store programs that do not need alterations. The monitor program of a single board microcomputer is generally stored in the ROM. This program interprets the information entered through a keyboard and provides equivalent binary digits of the microprocessor. Programs stored in the ROM can only be read; they cannot be altered.

The Read/Write memory (R/WM) is also known as user memory. It is used to store user programs and data. In signal board microcomputers, the monitor program monitors the Hex keys and stores those instructions and data in the R/W memory. The information stored in this memory can be easily read and altered.

I/O (INPUT / OUTPUT)

The third component of a microprocessor – based system is I/O (input/output); it communicates with the outside world. I/O includes two types of devices; input and output; these I/O devices are also known as peripherals.

The input devices such as a keyboard, switches and an analog- to – digital (A/ D) converter transfer binary information (data instructions) from the outside world to the microprocessor. Typically, a microcomputer used in college laboratories includes either hexadecimal keyboard or an ASCII keyboard as an input device. The hexadecimal keyboard has 16 data keys (0 to 9 and A to F) and some additional function keys to perform such operations as storing data and executing programs. The ASCII keyboard is similar to a typewriter keyboard, and it is used to enter programs in an English like language. Although the ASCII keyboard is found in most microcomputers, single board microcomputers generally have Hex keyboards, and microprocessor – based products such as microwave oven have decimal keyboards.

The output devices transfer data from the microprocessor to the outside world. They include devices such as light emitting diodes (LED's), a cathode ray tube (CRT) or video screen, a printer, X-Y plotter, a magnetic tape, and digital – to analog (D/A) converter. Typically, single board microcomputers and microprocessor – based products include LED's, seven – segment LED's, and alphanumeric LED displays as output devices. Microcomputers (PCs) are generally equipped with the output devices such as video screen and a printer.

SYSTEM BUS

The System Bus is a communication path between the microprocessor and peripherals; it is nothing but a group of wires to carry bits. In fact, there are several buses in the system. All peripherals share the same bus; however, the microprocessor communicates with only one peripheral at a time. The timing is provided by the control unit of the microprocessor.

3.2 MICROPROCESSOR ARCHITECTURE

The microprocessor is a programmable logic device, designed with registers, flip-flops and timing elements. The microprocessor has a set of instructions, designed internally to manipulate data and communicate with peripherals. This process of data manipulation and communication is determined by the logic design of the microprocessor.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The microprocessor can be programmed to perform functions on given data by selecting necessary instructions from its set. These instructions are given to the micro-processor by writing them into its memory. Writing instructions and data is done through an input device such as a keyboard. The microprocessor reads or transfers one instruction at a time, matches it with its instruction set, and performs the data manipulation indicated by the instruction. The result can be stored in memory or sent to such output devices as LEDs or a CRT terminal. In addition, the microprocessor can respond to external signals. It can be interrupted, reset, or asked to wait to synchronize with lower peripherals. All the various functions performed by the microprocessor can be classified in three general categories;

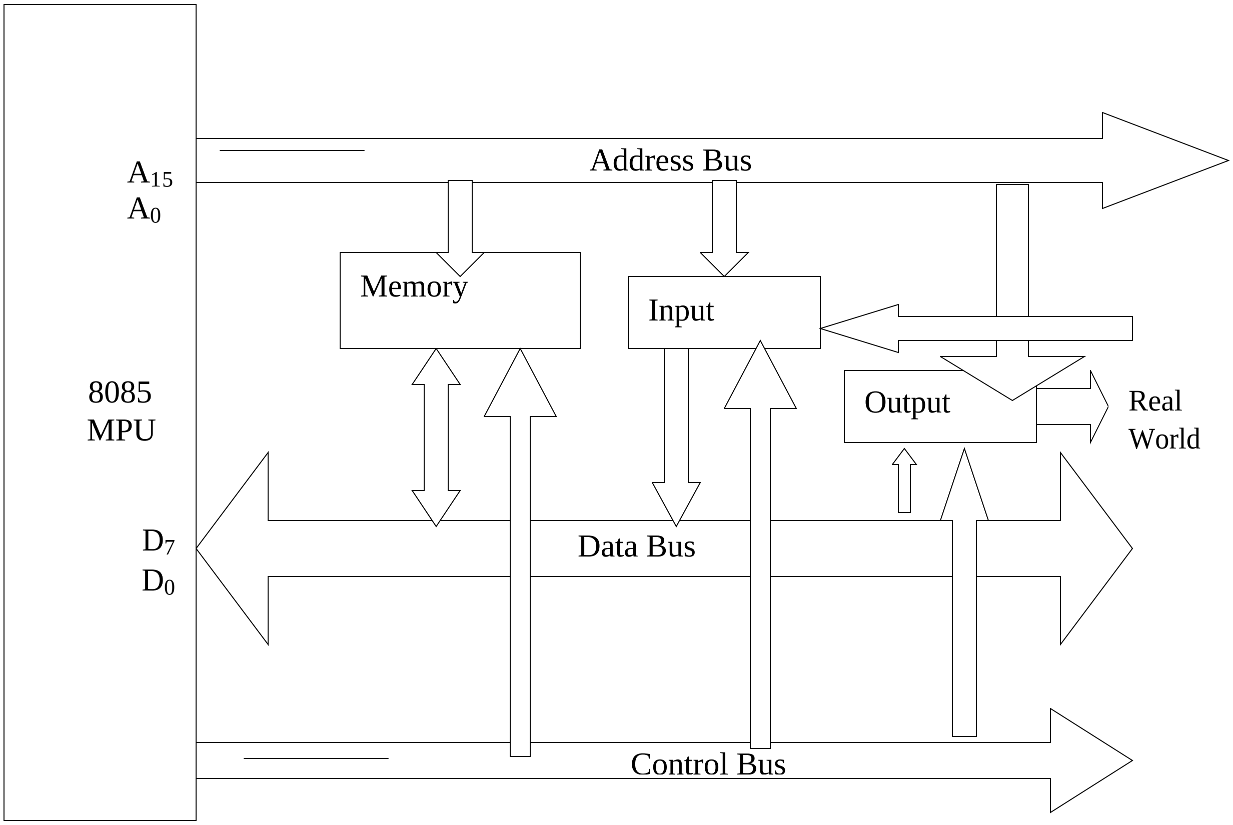
- Microprocessor- initiated operations
- Internal data operations
- Peripherals (or external initiated) operations

To perform these functions, the microprocessor requires a group of logic circuits and a set of signals called control signals. However, early processors did not have the necessary circuitry on one chip; the complete units were made up of more than one chip. Therefore, the term micro processing unit (MPU) is defined here as a group of devices that can perform these functions with the necessary set of control signals. This term is similar to the term central processing unit (CPU). However, later microprocessors include most of the necessary circuitry to perform these operations on a single chip. Therefore, the terms MPU and microprocessor often are used synonymously.

3.2.1 MICROPROCESSOR-INITIATED OPERATIONS & 8085 BUS ORGANIZATION

The MPU performs primarily four operations

1. Memory READ: reads data (or instructions) from memory.
2. Memory WRITE: reads data (or instructions) into memory.
3. I/O READ: accepts data from input devices.
4. I/O WRITE: send data to output devices.



This watermark does not appear in the registered version - http://www.clicktoconvert.com

Figure 3.3 The 8085 Bus Structure

To communicate with a peripheral (or memory location), the MPU needs to perform the following steps:

STEP 1: Identify the peripheral or the memory location (with its address)
STEP 2: Transfer data.
STEP 3: Provide timing or synchronization signals.

The 8085 MPU performs these functions using three sets of communication lines called buses: the address bus, the data bus and the control bus (figure3.2). These buses are together to form one group called the system bus**.**

Address Bus

1. The address bus is a group of 16 lines generally identified as A_0 to $A_{15.}$
2. The address bus is unidirectional; bits flow in one direction from the MPU to peripheral devices.
3. The MPU uses to perform the first function: identifying the peripheral or a memory location.

Data Bus

1. Data bus is group of eight lines used for data flow.
2. These lines are bidirectional;
3. The MPU uses the data bus to perform the second function: Transferring data.

Control Bus

1. It comprised of various single lines that carries synchronization signals.
2. It performs the third function: providing time signals.

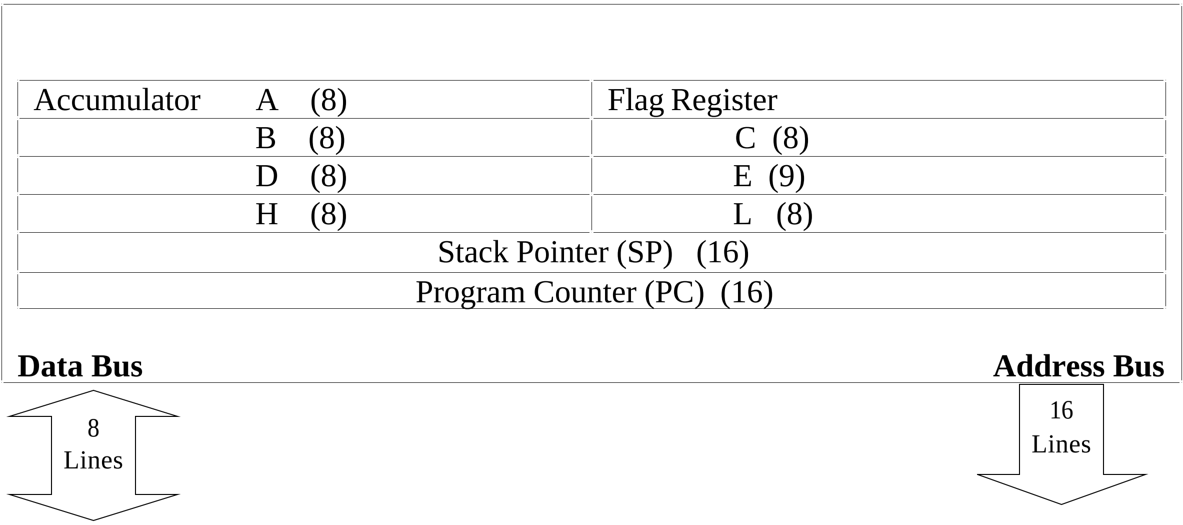
3.2.2. INTERNAL DATA OPERATIONS AND 8085 REGISTERS

The internal architecture of the 8085 microprocessor determines how and what operations to be performed with data. The operations are

1. Store 8-bit data.
2. Perform ALU OPERATIONS.
3. Test for conditions
4. Sequence the execution of instructions.
5. Store data temporarily during execution in the defined R/W memory locations called the stack.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

This watermark does not appear in the registered version - http://www.clicktoconvert.com



Accumulator A (8)	Flag Register
B (8)	C (8)
D (8)	E (9)
H (8)	L (8)
Stack Pointer (SP) (16)	
Program Counter (PC) (16)	

Figure3.4 The 8085 Programmable Registers

Registers

1) The 8085 have 6 general-purpose registers to perform the first operations: store 8-bit data.
2) These registers are identified by B, C, D, E, H and L as shown in Figure.
3) They can be combined as register pairs-BC, DE, and HL- to perform 16 bit operations.
4) These registers are programmable, meaning that a programmer can use them to load and transfer data from the registers by using instructions.
5) E.g. MOVE B, C – copies the data from C to B.

Accumulator

1) Accumulator is an 8 –bit register.
2) This register is used to store 8-bit data and in performing Arithmetic and Logical Operations.
3) The result of operations is stored in accumulator (A).

[

Flags

1) The ALU includes 5 flip-flops that are set or reset.
2) The microprocessor is used to perform the third operation: namely testing for data conditions

Program Counter (PC)

1) This 16 – bit register deals with fourth operations, sequence in the execution of instructions.
2) This register is the memory pointer.
3) Memory location has 16 bit addresses.

Stack Pointer (SP)

The stack pointer is also a 16-bit register used as a memory pointer.

1) I t points to the memory location in R/W memory called the stack.
2) The beginning of the stack is defined by loading the 16 bit address in the stack pointer (register).

This watermark does not appear in the registered version - http://www.clicktoconvert.com

3.2.3. PERIPHERAL OR EXTERNALLY INITIATED OPERATIONS

External devices can initiate the following operations, for which individual pins on the microprocessor chip are assigned:

- Reset
- Interrupt
- Ready
- Hold

1) Reset: When the reset is activated, all internal operations are suspended and the program counter is cleared (it holds 0000H). The program execution can again begin at the zero memory address.

2) Interrupt: The microprocessor can be interrupted from the normal execution of instructions and asked to execute some other instructions called service e routine. The microprocessor resumes its operation after completing the service routine.

3) Ready: The 8085 has a pin called READY. If the signal at this READY pin low, the microprocessor enters into a wait state.

4) Hold: When the HOLD pin is activated by an external signal, the microprocessor relinquishes control of the buses and allows the external peripheral to use them.

3.3 FUNCTIONAL DIAGRAM AND PINOUT DIAGRAM

3.3.1 FUNCTIONAL DIAGRAM OF 8085

The internal architecture of the 8085 includes the ALU (Arithmetic Logic Unit), timing and control unit, instruction register and decoder, register array, interrupt control and serial I/O control.

The ALU

The Arithmetic Logic Unit performs the computing functions; it includes the accumulator, the temporary register, arithmetic logic circuits and five flags. The temporary register is used to hold data during an arithmetic/logic operation. The result is stored in the accumulator and the flags (flip-flops) are set or reset according to the result of the operation.

The flags are affected by the arithmetic and logic operations in the ALU. In most of these operations result is stored in the accumulator. Therefore the flags generally reflect data conditions in the accumulator with some exceptions. The descriptions and conditions of the flags are as follows:

This watermark does not appear in the registered version - http://www.clicktoconvert.com

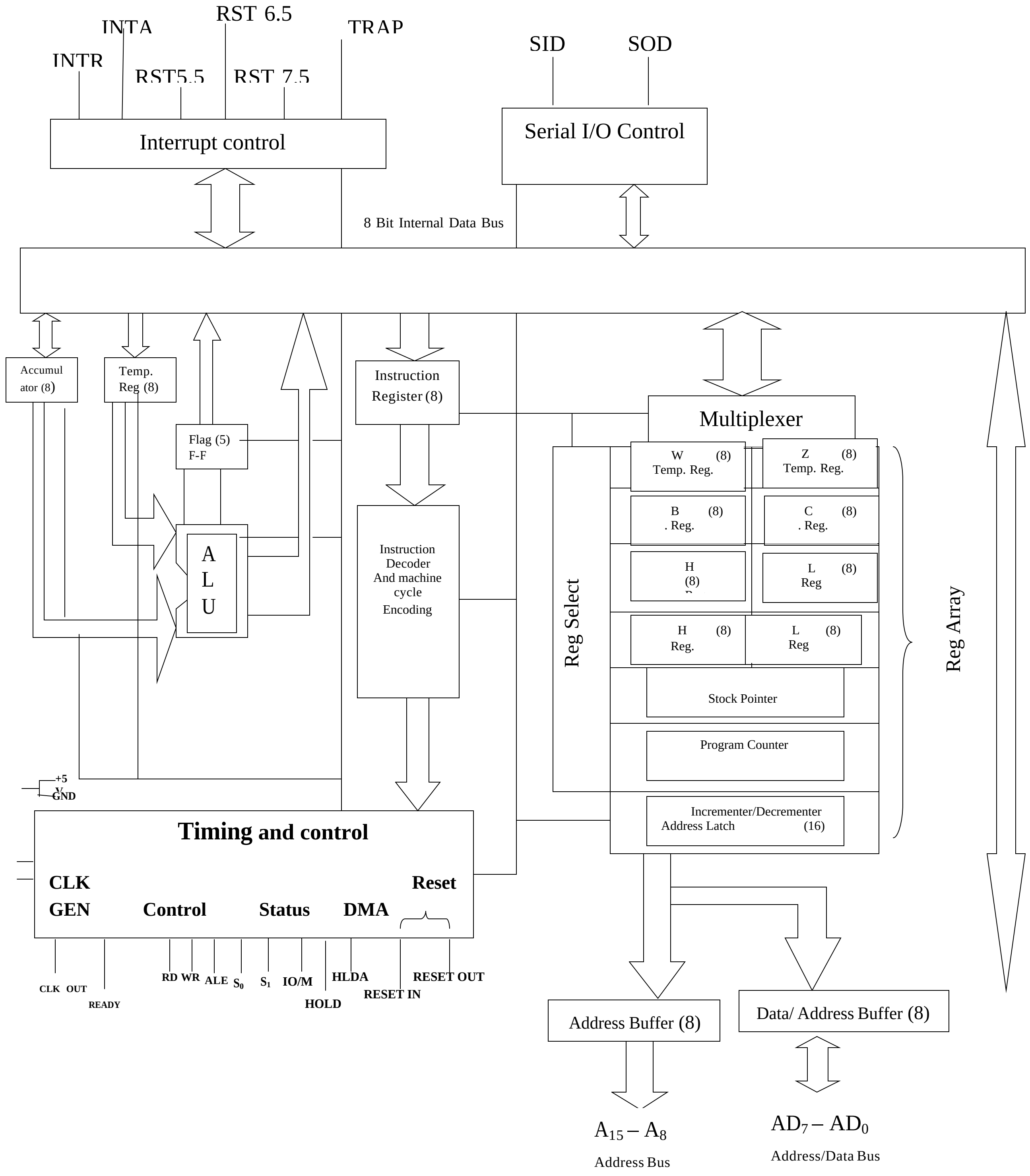


Figure 3.5 The 8085A Microprocessor: Functional Block Diagram

This watermark does not appear in the registered version - http://www.clicktoconvert.com

S-sign flag

After execution of an arithmetic or logic operation if bit D_7 of the result is 1, the sign flag is set. This flag is used with signed numbers. In a given byte if D_7 is 1,the number will be viewed as a negative number; if it is zero the number will be considered positive. In arithmetic operations with signed numbers, the bit D_7 is reserved for indicating the sign and the remaining seven bits are used to represent the magnitude of a number.

Z-Zero flag

The zero flag is set if the ALU operations results in 0, and the flag is reset if the result is not 0.This flag is modified by the results in the accumulator as well as in the other registers.

AC-Auxiliary Carry flag

In an arithmetic operation, when a carry is generated by digit D_3 and passed on to digit $D_{4,}$ the AC flag is set. It is used only internally for BCD operations.

P—Parity flag

After an arithmetic or logic operation, if the result as an even number of 1s the flag is set. If it as an odd number of 1s the flag is reset.

CY-Carry flag:

If an arithmetic operation results in a carry the carry flag is set otherwise it is reset. The carry flag also serves as a borrow flag for subtraction.

The bit positions reserved for these flags in the flag   in the flag register are as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

Among the five flags, the AC flag is used internally for BCD arithmetic; the instruction set does not include any conditional jump instructions based on the AC flag. Of the remaining four flags, the Z and CY flags are those most commonly used.

Timing and Control Unit

This unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between the microprocessor and the peripherals. The control signals are similar to a sync pulse in an oscilloscope. The RD and WR signals are sync pulses indicating the availability of data on the data bus.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Instruction Register and Decoder

The instruction register and the decoder are part of the ALU. When an instruction is fetched from memory, it is loaded in the instruction register. The decoder decodes the instruction and establishes the sequence of events to follow. The instruction register is not programmable and cannot be accessed through any instruction.

Register Array

Two additional registers, called temporary registers W and Z, are included in the register array. These registers are used to hold 8-bit data during the execution of some instructions. However, because they are used internally, they are not available to the programmer.

3.3.2 PINOUT DIAGRAM OF 8085

The 8085A (commonly known as the 8085) is an 8-bit general-purpose microprocessor capable of addressing 64k of memory. The device has forty pins, requires a +5V single power supply, and can operate with a 3-MHz single-phase clock. The 8085 A-2 version can operate at the maximum frequency of 5 MHz. The 8085 is an enhanced version of its predecessor, the 8080A; its instruction set is upward-compatible with that of the 8080A, meaning that the 8085 instruction set includes all the 8080A will be executed by the 8085, but the 8085 and the 8080A are not pin-compatible.

Figure 3.1 shows the logic pin out of the 8085 microprocessor. All the signals can be classified into six groups 1) address bus, 2) data bus, 3) control and status signals, 4) power supply and frequency signals, 5) externally initiated signals, and 6) serial I/O ports.

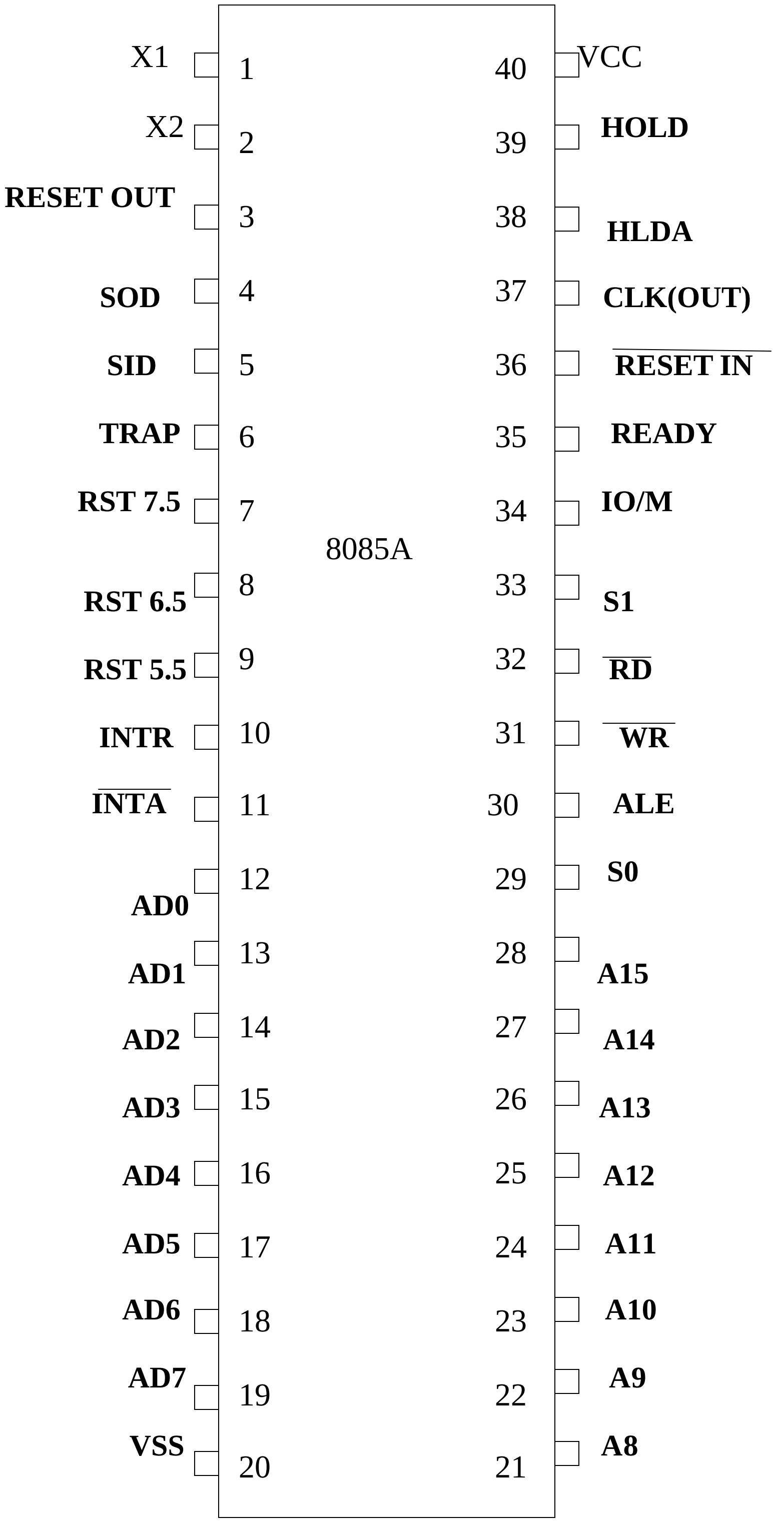
Address Bus

The 8085 have eight signal lines, A15-A8, which is unidirectional and used as the high-order address bus.

Multiplexed Address/Data Bus

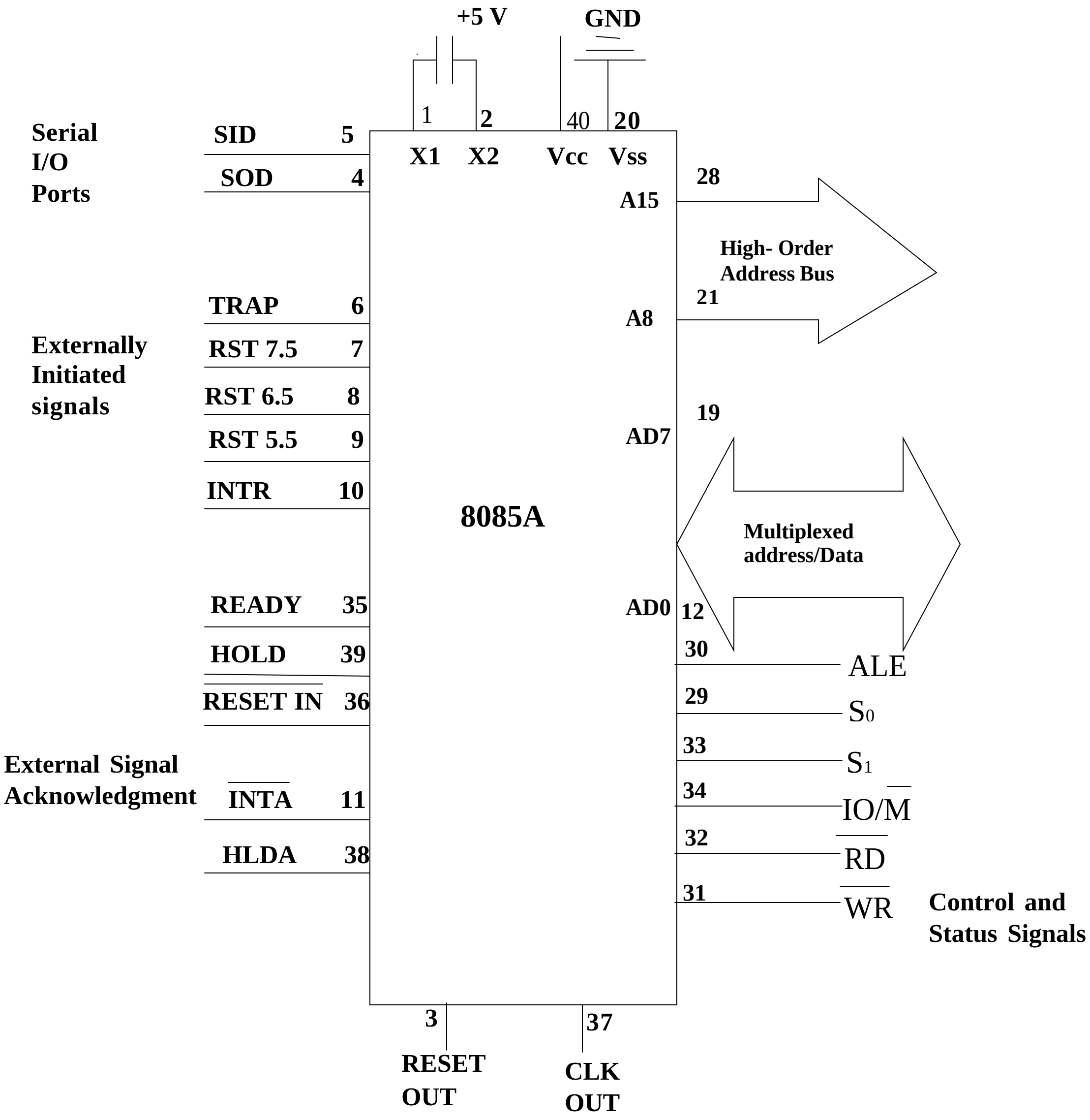
The signal lines AD7-AD0 are bi-directional: they serve a dual purpose. They are used as the low-order address bus as well as the data bus. In executing an instruction, during the earlier part of the cycle, these lines are used as the low-order address bus. During the later part of the cycle, these lines are used as the data bus. (This is also known as multiplexing the bus.) However, the low-order address bus can be separated from these signals by using a latch.

This watermark does not appear in the registered version - http://www.clicktoconvert.com



(a) The 8085 Microprocessor Pin out Diagram

This watermark does not appear in the registered version - http://www.clicktoconvert.com



(b) The 8085 Microprocessor Signals

Figure 3.6 The 8085 Microprocessor Pin out And Signals

Control And Status Signals

This group of signals includes two control signals (RD and WR), three status signals, (IO/M, s1 and s0) to identify the nature of the operation, and one special signal (ALE) to indicate the beginning of the operation. These signals are as follows:

This watermark does not appear in the registered version - http://www.clicktoconvert.com

- ALE – Address Latch Enable: This is a positive going pulse generated every time the 8085 begins an operation (machine cycle); it indicates that the bits on AD7 – AD0 are address bits. This signal is used primarily to latch the low-order address from the multiplexed bus and generate a separate set of eight address lines, AD7 – A0.
- RD – Read: This is a Read Control Signal (active low). This signal indicates that the selected I/O or memory device is to be read and data are available on the data bus.
- WR – Write: This is a Write control signal (active low). This signal indicates that the data on the data bus are to be written into a selected memory or I/O location.
- IO/M: This is a status signal used to differentiate between I/O and memory operations. When it is high, it indicates an I/O operation; when it is low, it indicates a memory operation. This signal is combined with RD (Read) and WR Write) to generate I/O and memory control signals.
- S1 and S0: These status signals, similar to IO/M, can identify various operations, but they are rarely used in small systems. (All the operations and their associated status signals are listed in Table 3.1 for reference.)

Power Supply And Clock Frequency
The power supply and frequency signals as follows:

- Vcc: +5V power supply.
- Vss: Ground Reference.
- X1, X2: A crystal (or RC, LC network) is connected at these two pins. The frequency is internally divided by two; therefore, to operate a system at 3 MHz, the crystal should have a frequency of 6 MHz.
- CLK (OUT) – Clock Output: This signal can be used as the system clock for other devices.

Table 3.1 8085 Machine Cycle Status and Control Signals

	Status			
Machine Cycle	**IO/$\overline{M}$**	**S_1**	**S_0**	**Control Signals**
Opcode Fetch	0	1	1	$\overline{RD}$ = 0
Memory Read	0	1	0	$\overline{RD}$ = 0
Memory Write	0	0	1	$\overline{WR}$ = 0
I/O Read	1	1	0	$\overline{RD}$ = 0
I/O Write	1	0	1	$\overline{WR}$ = 0
Interrupt Acknowledge	1	1	1	$\overline{INTA}$ = 0
Halt	Z	0	0	$\overline{RD}$, $\overline{WR}$ =Z and $\overline{INTA}$ =1
Hold	Z	X	X	
Reset	Z	X	X	

NOTE: Z = Tri-state (high impedance)
X = Unspecified

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Externally Initiated Signals, Including Interrupts

The 8085 have five interrupt signals that can be used to interrupt a program execution. One of the signals, INTR (Interrupt Request), is identical to the 8080A microprocessor interrupt signal (INT); the others are enhancement to the 8080A. The microprocessor acknowledges an interrupt request by the INTA (Interrupt Acknowledge) signal.

In addition to the interrupts, three pins – RESET, HOLD and READY – accept the externally initiated signals as inputs. To respond to the HOLD request, it has one signal called HLDA (Hold Acknowledge). The RESET is again described below, and others are instead in Table 3.2 for reference.

- ✓ RESER IN: When the signal on this pin goes low, the program counter is set to zero, the buses are tri-stated, and the MPU is reset.
- ✓ RESET OUT: This signal indicates that the MPU is being reset. This signal can be used to reset other devices.

Serial I/O Ports

The 8085 have two signals to implement the serial transmission: SID (Serial Input Data) and SOD (Serial Output Data).

TABLE 3.2 8085 Interrupts and Externally Initiated Signals

• INTR (Input)	Input Request: This is used as a general-purpose interrupt; is similar to the INT signal of the 8080A
• INTA (Output)	Interrupt Acknowledge: This is used to acknowledge as interrupt.
• RST 7.5 (Inputs) RST 6.5 RST 5.5	Restart Interrupt: These are vectored interrupts and transfer RST the program controls to specific memory locations. They have higher priorities than the INTR interrupt. Among these three, the priority order is 7.5, 6.5 and 5.5.
• TRAP (Input)	This is a non-mask able interrupt and has the highest priority.
• HOLD (Input)	This signal indicates that a peripheral such as a DMA (Direct Memory Access) controller is requesting the use of the address and data buses.
• HLDA (Output)	Hold Acknowledge: This signal acknowledges the HOLD request.
• READY (Input)	This signal is used to delay the microprocessor Read or Write Cycles until a slow-responding peripheral is ready to send or accept data. When this signal goes low, the microprocessor waits for an integral number of clock cycles until it goes high

This watermark does not appear in the registered version - http://www.clicktoconvert.com

3.4 ADDRESSING MODES OF 8085

The 8085 has five Addressing modes

1. **Direct**—Instructions using this mode specify the effective address as a part of the instruction. These instructions contain 3 bytes, with the first byte as the OP code following by 2 bytes of address of data (the low-order byte of the address in byte 2, the high-order byte of the address in byte 3), Consider LDA 2035H. This instruction loads accumulator with the content of memory location 2035_{16}.This mode is also called the absolute mode.

2. **Register**—This mode specifies the register pair that contains data. For example, MOV B, C moves the contents of register C to register B.

3. **Register Indirect—**This mode contains a register pair which store the address of data (the high-order byte of the address in the pair, and the low-order byte in the second). As an example, LDAX B loads the accumulator with the contents of a memory location addressed by B, C register pair.

4. **Implied or Inherent—**The instructions using this mode have no operands. Example include STC (set the carry flag).

5. **Immediate—**For an 8-bit datum, this mode uses 2 bytes, with the first byte as the OP code, followed by 1 byte of data. On the other hand, for 16-bi data, this instruction contains 3 bytes, with the first as the OP code followed by 2 byte of data. For example, MVI B, 05 loads B with the value 5, and LXI H, 2050 loads H with 20h and L with 50H.

A JUMP instruction interprets the address that it would branch to in the following ways.

1. **Direct—**The JUMP instruction, such a JZ ppqq, uses direct addressing and contain 3 byte. The first byte is the OP code, followed by 2 byte of the 16-bit address where it would branch to unconditionally or based on a condition if satisfied. For example JMP 2020 unconditionally branches to location 2020H.

2. **Implied or Inherent Addressing –**This JUMB instruction using the mode is 1 byte long. A 16-bit register pair contains the address of the next instruction to be executed. The instruction PCHL unconditionally branches to a location address by H, L pairs.

Self Check Exercise: 1

1. Say True or False

(a) Initial portion of the boot program of a microcomputer resides on the Ram.
True/False

(b)The physical memory size of a computer whose address bus is of 24 lines can be 16 MB.
True/False

(c) Both the address and the data bus should be unidirectional only. True/False

(d) Today’s microprocessor’s are more powerful than the first mainframe computer
True/False

This watermark does not appear in the registered version - http://www.clicktoconvert.com

2. List the basic design objectives of a Microprocessor.

…………………………………………………………………………………………………
…………………………………………………………………………………………………

3.5 INSTRUCTION SET OF 8085

The 8085 use a 16-bit address. Since 8085 is a byte- addressable machine, it follows that it can directly address 65,536(216) distinct memory locations. The addressing structure of the 8085 processor is shown in the figure 3.7.

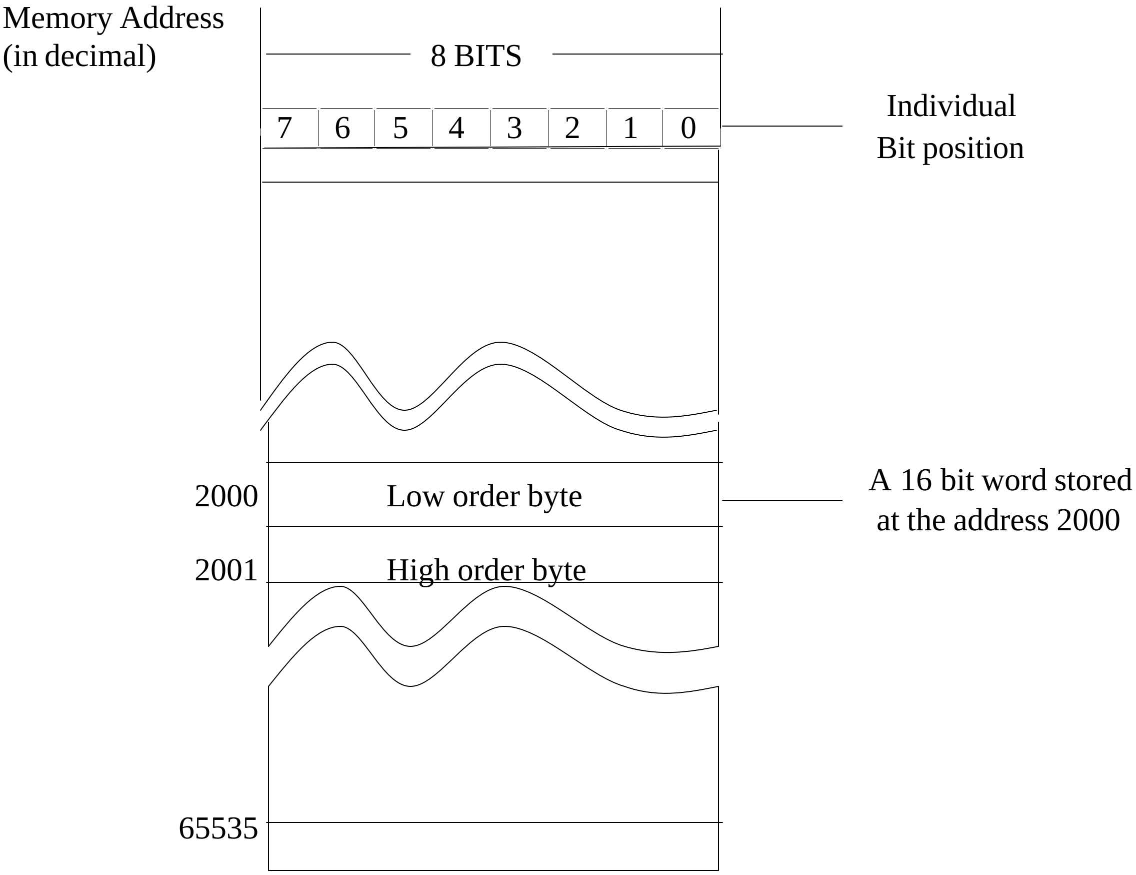


Figure 3.7 8085 Addressing Structures

3.5.1 INSTRUCTION CLASSIFICATION

- An instruction is a binary pattern designed inside a microprocessor to perform a specific function.
- The entire group of instructions, called the instruction set, determines what functions the microprocessor can perform.
- The instructions can be classified into five functional categories:
    1) Data transfer operations
    2) Arithmetic operations
    3) Logical operations
    4) Branching operations
    5) Machine control operations

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Data Transfer (Copy) Operations

- This group of instructions copies data from a location called a source to another location, called destination, without modifying the contents of the source.
- The term data *transfer* is used for this copying function.
- The various types of data transfer are listed below:

1) Between registers
2) Specific data byte to a register or a memory location
3) Between a memory location and a register
4) Between an I/O device and the accumulator

Arithmetic Operations

These instructions perform arithmetic operations such as addition, subtraction, increment and decrement.

1) ADDITION: Any 8 – bit number, or the contents of a register, or the contents of a memory location can be added to the contents of the accumulator and sum is stored in the accumulator.
2) SUBTRACTION: Any 8 – bit number, or the contents of a register, or the contents of a memory location can be added to the contents of the accumulator and sum is stored in the accumulator.
3) INCREMENT/DECREMENT: The 8-bit contents of a register or a memory location can be incremented or decremented.

Logical Operations

These instructions perform various logical operations with the contents of the accumulator.

1) AND, OR, EXCLUSIVE –OR: Any 8 – bit number, or the contents of a register, or the contents of a memory location can be logically AND ed, Or ed, or Exclusive – OR ed with the contents of the accumulator and sum is stored in the accumulator.
2) Rotate: Each bit in the accumulator can be shifted either left or right to the next position.
3) Compare: Any 8 – bit number, or the contents of a register, or the contents of a memory location can be compared for equality, greater than, or less than, with the contents of the accumulator.
4) Complement: The contents of the accumulator can be complemented; all 0s are replaced by 1s and all 1s are replaced by 0s.

Branching Operations

This group of instructions alters the sequence of program execution either conditionally or unconditionally.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1.JUMP: Conditional jumps are an important aspect of the decision making process in programming. These instructions test for a certain condition (e.g., Zero or Carry flag) and alter the program sequence when the condition is met. In addition, the instruction set includes an instruction called unconditional jump.

2.CALL, RETURN AND RESTART: These instructions change the sequence of a program either by calling a subroutine or returning from a subroutine or returning from a subroutine. The conditional Call and Return instructions also can test conditional flags.

3.MACHINE CONTROL OPERATIONS: These instructions control machine functions such as Halt, Interrupt or do nothing

3.5.2 INSTRUCTION FORMAT

An instruction is a command to the microprocessor to perform a given task on specified data. Each instruction has two parts: one is the task to be performed, called the operation code (opcode), and the second is the data to be operated on, called the operand. The operand (or data) can be specified in various ways. It may include 8-bit data, an internal register, a memory location, or 8-bit address. In some instructions, the operand is implicit.

3.5.2.1 Instruction Word Size

The 8085 instruction set is classified into the following three groups according to word size:

1. One-word or 1-byte instructions.
2. Two-word or 2-byte instructions.
3. Three-word or 3-byte instructions.

In the 8085,”byte” and “word” are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms are of bytes rather than words.

One-Byte Instructions

A 1-byte instruction includes the opcode and the operand in the same byte. For example: Each bit in the accumulator.

These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand B is specified and the accumulator is assumed. Similarly, in the third instruction, the accumulator is assumed to be the implicit operand. These instructions are stored in 8-bit binary format in memory; each requires one memory location.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Task	Opcode	Operand	Binary Code	Hex Code
Copy the contents Of accumulator in Register C.	MOV	C,A	01001111	4FH
Add the contents of register B to the contents of the accumulator	ADD	B	10000000	80H
Invert (complement) each bit in the accumulator.	CMA		00101111	2FH

Two-Byte Instructions

In a 2-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. For example:

Task	Opcode	Operand	Binary Code	Hex Code	
Load First Byte an 8-bit data byte in the accumulator Data	MVI	A, Data	0011 1110 DATA	3E Data	First Byte Second Byte

Assume the data byte is 32H. The assembly language instruction is written as

Mnemonics	HexCode
MVI A, 32H	3E 32H

This instruction would require two memory locations to store in memory.

Three-Byte Instructions

In a 3-byte instruction, the first byte specifies the opcode, and address and the following two bytes specify the 16-bit address. Note that second byte is the high-order address. For example:

TASK	OPCODE	OPERAND	BINARY CODE	HEX CODE	BYTE
Transfer the program sequence to the memory location.	JMP	2085H	1100 0011 1000 0101 0010 0000	C3 85 20	FIRST BYTE SECOND BYTE THIRD BYTE

This instruction would require three memory locations to store in memory.

These commands are in many ways similar to our everyday conversation. For example, while eating in a restaurant, we make the following requests and orders:

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1. Pass (the) butter
2. Pass (the) bowl
3. (Let us) eat.
4. I will have combination 17(on the menu)
5. I will have what Susie ordered.

The first request specifies the exact item; it is similar to the instruction for loading a specific data byte in a register. The second request mentions the bowl rather than the contents, even though one is interested in the contents of the bowl. It is similar to the instruction MOV C, A where registers (bowls) are specified rather than data. The third suggestion (let us eat) assumes that one knows what to eat. It is similar to the instruction Complement, which implicitly assumes that the one knows what to eat. It is similar to the instruction Complement, which implicitly assumes that the operand is the accumulator. In the fourth sentence the location of the item on the menu is specified and not the actual item. It is similar to the instruction: transfer the data byte from the location 2050H. The last order (what Susie ordered) is specified indirectly. It is similar to an instruction that specifies a memory location through the contents of a register pair.

These various ways of specifying data are called the addressing modes. Although microprocessor instructions require one or more words to specify the operands, the notations and conventions used in specifying the operands have very little to do with the operation of the microprocessor. The mnemonic letters used to specify a command are chosen (somewhat arbitrarily) by the manufacturer. When an instruction is stored in memory, it is stored in binary code; the only code the microprocessor is capable of reading and understanding. The conventions used in specifying the instructions are valuable in terms of keeping uniformity in different programs and in writing assemblers. The important point to remember is that the microprocessor neither reads nor understands mnemonics or hexadecimal numbers.

3.5.2.2 OPCODE FORMAT

To understand operation codes, we need to examine how an instruction is designed into the microprocessor. This information will be useful in reading a user's manual, in which operation codes are specified in binary format and 8-bits are divided in various groups.

In the design of the 8085 microprocessor chip, all operations, registers, and status flags are identified with a specific code. For example, all internal registers are identified as follows:

Code	**Register**	**Code**	**Register Pair**
000	B	00	BC
001	C	01	DE
010	D	10	HL
011	E	11	AF OR SP
100	H		
101	L		
111	A		
110	Reserved for memory related operation		

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Some of the operation codes are identified as follows:

Function	Operation code
1. Rotate each bit of the accumulator left by one position	00000111 = 07H(8-bit opcode) to the
2. Add the contents of a register to the to the accumulator	10000 SSS (5-bit opcode-3 bits are reserved for a register)

This instruction is completed by adding the code of the register. For example,

Add	:	10000
Register B	:	000
To A	:	Implicit
Binary Instruction	:	$\underbrace{10000}_{\text{Add}}\ \underbrace{000}_{\text{Reg.B}} = 80H$

In assembly language, this is expressed as

Op code	**Operand**	**Hex Code**
ADD	B	80H

3. MOVE (Copy) The content of Register Rs (source) to register Rd (destination)	01 2-bit Opcode For MOVE	DDD Reg. Rd	SSS Reg. Rs

MOVE(copy) the content:		0	1	
To register C	:	0	0	1 (DDD)
From register A	:	1	1	1 (SSS)

Binary Instruction : $\underbrace{0\quad 1}_{\text{opcode}}\ \underbrace{0\quad 0\quad 1\quad 1\quad 1\quad 1}_{\text{Operand}}$ ——— 4FH

In assembly language, this is expressed as

Opcode	Operand	Hex Code
MOV	C,A	4F

Please note that first register is the destination and the second register is the source is the source – from A to C—which appears reversed for a general pattern from left to right. Typically in the 8085 user's manual the data transfer (copy) instruction is shown as follows.

MOV				r1,		r2	
0	1	D	D	D	S	S	S

3.5.3 OVERVIEW OF THE 8085 INSTRUCTION SET

The 8085 microprocessor instruction set has 74 operation codes that result in 246 instructions. The set includes all the 8080A instructions plus two additional instructions (SIM and RIM, related to serial I/O).

This watermark does not appear in the registered version - http://www.clicktoconvert.com

It is an overwhelming experience for a beginner to study these instructions. You are strongly advised not to attempt to read all these instructions at one time. However, you should be able to grasp an overview of the set by examining the frequently used instruction listed below.

The following notations are used in the description of the instructions.

R = 8085 8-bit register	(A,B,C,D,E,H,L)
M = Memory register (location)	

Rs = Register source
Rd = Register destination
	(A,B,C,D,E,H,L)

Rp = Register pair	(BC,DE,HL,SP)
( ) = Contents of	

1. Data transfer (copy) instructions: From register to register.
Load an 8-bit number in a register.
Between memory and register.
Between I/O and accumulator.
Load 16-bit number in a Register pair.

Mnemonics	Tasks
MOV Rd,Rs	Copy data from source register Rs in to destination register Rd
MVI R, 8-bit	Load 8-bit data in a register
OUT 8-bit (Port address)	Send (write) data byte from the accumulator to an output device
IN 8-bit (Port address)	Accept (read) data byte from an input device and place it in the accumulator
LXI Rp ,16-bit	Load 16-bit in a register pair
MOV R,M	Copy the data byte from a memory location (source) in to a register
LDAX Rp	Copy the data byte in to the accumulator from the memory location indicated by a register pair
LDA 16-bit	Copy the data byte in to the accumulator from the memory location specified by 16-bit address
MOV M,R	Copy the data byte from the register in to memory location
STAX Rp	Copy the data byte from the accumulator in to the memory location indicated by a register pair
STA 16-bit	Copy the data byte from the accumulator in the memory location specified by 16-bit address

2. Arithmetic instructions: Add
Subtract
Increment (Add 1)
Decrement (Subtract 1)

This watermark does not appear in the registered version - http://www.clicktoconvert.com

ADD R	Add the contents of a register to the contents of the accumulator.
ADI 8-bit	Add 8-bit data to the contents of the accumulator.
SUB R	Subtract the contents of a register from the contents of the accumulator.
SUI 8-bit	Subtract 8-bit data from the contents of the accumulator.
INR R	Increment the contents of a register .
DCR R	Decrement the contents of a register .
INX Rp	Increment the contents of a register pair.
DCX Rp	Decrement the contents of a register pair.
ADD M	Add the contents of a memory location to the contents of the accumulator.
SUB M	Subtract the contents of a memory location from the contents of the accumulator.
INR M	Increment the contents of a memory location.
DCR M	Decrement the contents of a memory location.

3. Branch instructions: Change the program sequence unconditionally.
Change the program sequence if specified data conditions are met.

JMP 16-bit address	Change the program sequence to the location specified by the 16-bit address.
JZ 16-bit address	Change the program sequence to the location specified by the 16-bit address if the Zero flag is set.
JNZ 16-bit address	Change the program sequence to the location specified by the 16-bit address if the Zero flag is reset.
JC 16-bit address	Change the program sequence to the location specified by the 16-bit address if the Carry flag is set.
JNC 16-bit address	Change the program sequence to the location specified by the 16-bit address if the Carry flag is reset.
CALL 16-bit address	Change the program sequence to the location of a subroutine.
RET	Return to the calling program after completing the subroutines sequence.

4. .Logical instructions: AND
OR
X-OR
Compare
Rotate

This watermark does not appear in the registered version - http://www.clicktoconvert.com

AND R/M	Logically AND the contents of register/memory with the contents of the accumulator.
ANI 8-bit	Logically AND the 8-bit data with the contents of the accumulator.
ORA R/M	Logically OR the contents of register/memory with the contents of the accumulator.
ORI 8-bit	Logically OR the 8-bit data with the contents of the accumulator.
XRA R/M	Exclusive-OR the contents of register/memory with the contents of the accumulator.
XRI 8-bit	Exclusive-OR the 8-bit data with the contents of the accumulator.
CMA	Complement the contents of the accumulator.
RLC	Rotate each bit in the accumulator to the left position
RAL	Rotate each bit in the accumulator including the carry to the left position.
RRC	Rotate each bit in the accumulator to the right position.
RAR	Rotate each bit in the accumulator including the carry to the right position.
CMP R/M	Compare the contents of register/memory with the contents of the accumulator for less than, equal to, or more than.
CPI 8-bit	Compare 8-bit data with the contents of the accumulator for less than,equal to,or more than

5. Machine control instructions:

HLT	Stop processing and wait.
NOP	Do not perform any operations.

This set of instructions is a representative sample; it does not include various instructions related to 16-bit data operations, additional Jump instructions, and conditional Call and Return instructions.

Microprocessor Communication and Bus Timings

To understand the functions of various signals of the 8085, we should examine the process of communication (reading from and writing into the memory) between the microprocessor and memory and the timings of these signals in relation to the system clock. The first step in the communication process is reading from memory or fetching An instruction. This can be easily understood using an analogy of how a package is picked up from your house by a shipping company such as Federal Express. The steps are as follows:

1. A courier gets the address from the office; he or she drives the pickup van, finds the street, and looks for your house number.
2. The courier rings the bell
3. Somebody in the house opens the door and gives the package to the courier, and the courier returns to the offices with the package.
4. The internal office staff disposes the package according to the instructions given by the customer.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Now let us examine the steps in the following example of how the microprocessor fetches or gets a machine code from memory.

Step 1: The microprocessor places the 16-bit memory address from the program counter (PC) on the address bus. In our analogy, this is the equivalent of our courier getting on the road to find the address.

Step 2: The control unit sends the control signal RD to enable the memory chip. This is similar to ringing the doorbell in our analogy of a package pick up.

Step 3: The byte from the memory location is placed on the data bus.

Step 4: The byte is placed in the instruction decoder of the microprocessor, and the task is carried out according to the instruction.

The above four steps are similar to the steps listed in our analogy of the package pickup.

3.6 INPUT/OUTPUT SCHEMES

The 8085 I/O transfer techniques are discussed. The 83855/87855 and 8155/8156 I/O Ports and 8085 SID and SOD lines are also discussed.
8085 Programmed I/O

There are two I/O instructions in the 8085, IN and OUT. These instructions are 2 bytes long. The first byte defines the OP code of the instruction and the second byte specifies the I/O Port number. Execution of the IN PORT instruction causes the 8085 to send one byte of data from the accumulator into a specified I/O Port.

The 8085 can access I/O Ports using either standard I/O or memory mapped I/O. In standard I/O. the 8085 inputs or outputs data using IN or OUT instructions. In memory mapped I/O, the 8085 maps I/O ports as memory addresses. Hence, LDA addr or STA addr instructions are used to input or output data to or from the 8085. The 8085's programmed I/O capabilities are obtained via the support chips, namely, 8355/8755 and 8155/8156. The 8355/8755 contains 2K byte ROM/EPROM and two bit I/O ports (Ports A and B).

The 8155/8156 contains 256 byte RAM, two 8 bit and one 6 bit I/O ports, and a 14 bit programmable timer. The only difference between the 8155 and 8156 is that chip enable is LOW on the 8155 and HIGH on the 8156.

3.6.1 8085 INTERRUPT SYSTEM

The 8085 chip has five interrupt pins, namely, TRAP, RST7.5, RST 6.5, RST 5.5, AND INTR. If the signals on these interrupt pins go to HIGH simultaneously, then TRAP will be serviced first followed by RST 7.5, RST 6.5, RST 5.5 and INTR. Note that once an interrupt is serviced, all the interrupts except TRAP are disabled. They can also be enabled or disabled simultaneously by executing the EI or DI instruction respectively. The 8085 interrupts are:

This watermark does not appear in the registered version - http://www.clicktoconvert.com

1. TRAP – TRAP is a non-maskable interrupt. That is, it cannot be enabled or disabled by an instruction. In order for the 8085 to service this interrupt, the signal on the TRAP pin must have a sustained HIGH level with a low to high transition. If this condition occurs, then the 8085 complete execution of the current instruction pushes the program counter onto the stack, and branches to location 0024 (interrupt address vector for the TRAP). Note that the TRAP interrupt is cleared by the falling edge of the signal on the pin.

2. RST7.5 – RST7.5 is a maskable interrupt. This means that it can be enabled or disabled using the SIM or EI/DI instruction. The 8085 responds to the RST 7.5 interrupt when the signal on the RST 7.5 pin has a low to high transition. In order to service RST 7.5, the 8085 completes execution of the current instruction, pushes the program counter onto the stack, and branches to $003C_{16}$. The 8085 remember the RST 7.5 interrupt by setting an internal D flip flop by the leading edge.

3. RST 6.5 – RST 6.5 is a maskable interrupt. It can be enabled or disabled using the SIM or EI/DI instruction. RST 6.5 is HIGH level sensitive. In order to service this interrupt the 8085 completes execution of the current instruction, saves the program counter onto the stack, and branches to location 0034_{16}.

4. RST 5.5 - RST 5.5 is a maskable interrupt. It can be enabled or disabled using the SIM or EI/DI instruction. RST 6.5 is HIGH level sensitive. In order to service this interrupt the 8085 completes execution of the current instruction, saves the program counter onto the stack, and branches to location $002C_{16}$.

5. INTR – INTR is a maskable interrupt. It can be enabled or disabled using the SIM or EI/DI instruction. This is also called the handshake interrupt. INTR is HIGH level sensitive. When no other interrupts are active and the signal on the INTR pin is HIGH, the 8085 completes execution of the current instruction, and generates an interrupt acknowledge, $\overline{\text{INTA}}$, LOW pulse on the control bus. The 8085 then expect either a 1 byte CALL or a 3 – byte CALL on the data line. This instruction must be provided by external hardware. In other words, the INTA can be used to enable tristate buffer. The output of this buffer can be connected to the 8085 data lines. The buffer can be designed to provide the appropriate op code on the data lines. Note that the occurrence of INTA turns off the 8085 interrupt system in order to avoid multiple interrupts from a single device. Also note that there are eight RST instructions. Each of these RST instructions has a vector address.

3.6.2 8085 DMA

The intel 8257 DMA controller chip is a 40 pin DIP and is programmable. It is compatible with the 8085 microprocessor. The 8257 is a four channel DMA controller with priority logic built into the chip. This means that the 8257 provide for DMA transfers for a maximum of up to devices via the DMA request lines DRQ0 to DRQ3. Associated with each DRQ is a DMA acknowledge. Note that the DACK signals are active LOW. The 8257 uses the 8085 HOLD pin in order to take over the system bus. After initializing the 8257 by the 8085, the

This watermark does not appear in the registered version - http://www.clicktoconvert.com

8257 perform the DMA operation in order to transfer a block of data of up to 16,384 bytes between the memory and a peripheral without involving the microprocessor. A typical 8085 – 8257 is shown in the **figure** An I/O device when enabled by the 8085, can request a DMA transfer by raising the DMA request (DRQ) line of one of the channels of the 8257. In response, the 8257 will send a HOLD request (HOLD) to the 8085. The 8257 waits for the HOLD Acknowledge (HLDA) from the 8085. On receipt of HLDA from the 8085, the 8257 generates a LOW on the DACK lines for the I/o devices. Note that DACK is used as a chip select bit for the I/O device. The 8257 sends the READ or WRITE control signals and data are transferred between the I/O and memory. On completion of the data transfer, the DACK0 is set to HIGH, and the HRQ line is reset to LOW in order to transfer control of the bus to the 8085. The 8257 utilizes four clock cycles in order to transfer 8 bits of data.

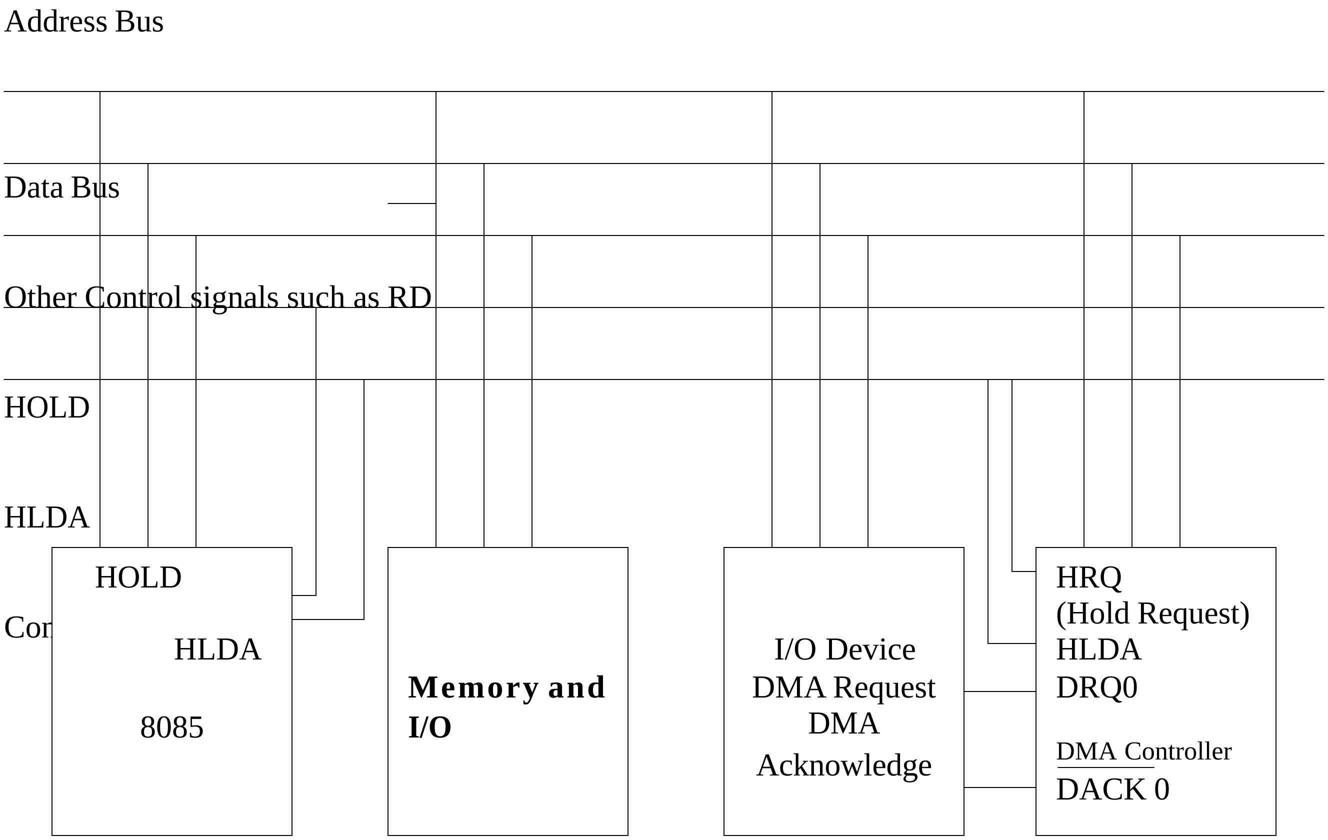


Figure 3.8 An 8085 – 8257 Interface

The 8257 has three main registers. These are a 16 bit DMA address register, a terminal count register and a status register. Both address and terminal count registers must be initialized before DMA operation. The DMA address register is initialized with the starting address of the memory to be written into or read from. The low order 14 bits of the terminal count register are initialized with the value (n-1), where n is the desired number of DMA cycles. A terminal count (TC) pin on the 8257 is set to HIGH in order to indicate to the peripheral device that the present DMA cycle is the last cycle. An 8-bit status register in the 8257is used to indicate which channels have attained a terminal count.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

3.6.3 8085 SID AND SOD LINE

Serial I/O is extensively used for data transfer between a peripheral device and the microprocessor. Since microprocessors perform internal operations in parallel, conversion of data from parallel to serial and vice versa is required to provide communication between the microprocessor and the serial I/O. The 8085 provide serial I/O capabilities via SID (Serial Input Data) and SOD (Serial Output Data) lines.

One can transfer data to or from the SID or SOD lines using the instruction RIM and SIM. After executing the RIM instruction, the bits in the accumulator are interpreted as follows:

1. Serial input bit is bit 7 of the accumulator.
2. Bits 0 to 6 are interrupt masks, the interrupt enable bit, and pending interrupts.

The SIM instruction sends the contents of the accumulator to the interrupt mask register and serial output line. Therefore, before executing the SIM, the accumulator must be loaded with proper data. The contents of the accumulator are interpreted as follows:

1. Bit 7 of the accumulator is the serial output bit.
2. The SOD enable bit is bit 6 of the accumulator. This bit must be 1 in order to output bit 7 of the accumulator to the SOD lines.
3. Bits 0 to 5 are interrupt masks, enables and resets.

3.7 INTERFACES AND PERIPHERALS

Printer interface

Microprocessors are typically interfaced to two types of printers: serial and parallel. Serial printers print one character at a time, while parallel printers print a number of characters on a single line so fast that they appear to b printed simultaneously. Depending on the character generation technique used, printers can be classified as impact or non-impact. In impact printers, the print head strikes the printing medium, such as paper, directly, in order to print a character. In non-impact printers, thermal or electrostatic methods are used to print a character.

Printers can also be classified based on the character formation technique used. For example, character printers' use completely formed characters for character generation, While matrix printers use dots or lines to create character.

There are two ways of interfacing the printer to a microcomputer. These are

1. Direct microcomputer control
2. Indirect microcomputer control using a special chip called the Printer Controller

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The direct microcomputer control interfaces the printer via its I/O ports and utilizes mostly software. The microcomputer performs all the functions required for printing the alphanumeric character.

Indirect microcomputer control, on the other hand, utilizes a printer control chip such as the Intel 8295 Dot Matrix Printer Controller. The benefits of each technique depend on the specific application.

The direct microcomputer approach provides an inexpensive interface and can be appropriate when the microcomputer has a light load. The indirect microcomputer approach, on the other hand may be useful when the microcomputer has a heavy load and cost is not a major.

Printer Interface Using Direct Microcomputer Control

The steps involved in starting a printing sequence by the microcomputer are provided below:

1. The microcomputer must turn the Main Drive motor (MDM) ON by sending a HIGH output to the MDM.
2. The microcomputer is required to detect a HIHG at the HOME micro switch. The will ensure that the print head is at the left-hand margin of the print area.
3. The microcomputer is then required to send five bytes of the data for an alphanumeric character in sequence to energize the solenoids. Each solenoid requires a pulse of about 400 ms to generate a dot on the paper. A pause of about ms is required between these pulses to provide a space between dots.

Keyboard Interface

A common method of entering programs into a microcomputer is via a keyboard. A popular way of displaying results by the microcomputer is by using seven segment displays. The main functions to be performed for interfacing a keyboard are

- Sense the key actuation
- Debounce the key
- Decode the key

A keyboard is arranged in rows and columns fig 3.9 shows a 2X2 keyboard interfaced to a typical microcomputer. In this figure the columns are normally at a high level. A key actuation is sensed by sending a low to each row one at a time via PA0 and PA1 of port A. The two columns can then be input via PB2 and PB3 of port B to see whether any of the normally HIGH columns are pulled LOW by a key action. If so, row can be checked individually to determine the row in which the key is down. A row and column code in which the key is pressed can thus be found.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The next step is to debounce the key. Key bounce occurs when a key is pressed or released- it bounces for a short time before making the contact. When this bounce occurs, it may appear to the microcomputer that the same key has been actuated several times instead of just once. This problem can be eliminated by reading the keyboard after 20 ms and then verifying to see if it is still down. If it is, then the key actuation is valid.

The next step is to translate the row and column code into a more popular code such as hexadecimal or ASCII. This can easily be accomplished by a program. There are certain characteristics associated with keyboard actuation which must be considered while interfacing to a micro computer. Typically, these are two-key lock out and n-key roll over. The two-key lock takes into account only one key pressed. An additional key pressed and released does not generate any codes. The system is simple to implement and most often used. However, it might slow down the typing since each key must be fully released before the next one is pressed down. On the other hand, the N-key rollover will ignore all keys pressed until only one remains down.

Now let us elaborate on the interfacing characteristics of typical displays. The following functions are to be typically performed for displays:

- Output the appropriate display code.
- Output the code via right entry or left entry into the display if there is more than one display.

The above functions can easily be realized by a microcomputer program. If there is more than one display, they are typically arranged in rows. A row of four displays is shown in fig… In the figure, one has the option of outputting the display code via right entry or left entry. If it is entered via right entry, then the code for the most significant digit of the four digit display should be output first, then the next digit code, and so on. Note that the first digit will be shifted three times, the next digit twice, the next digit once, and the last digit (least significant digit in this case) does not need to be shifted. The shifting operations are so far that visually all four digits will appear on the display simultaneously. If the displays are entered via left entry, then the least significant digit must be output first and the rest of the sequence is similar to the right entry.

Two techniques are typically used to interface a hexadecimal display to the microcomputer. These are non-multiplexed and multiplexed. In non-multiplexed methods, each hexadecimal display digit is interfaced to the microcomputer via an I/O port.

BCD to seven-segment conversion is done in software. The microcomputer can be programmed to output to the two display digits in sequence. However, the microcomputer executes the display instruction sequence so fast that the displays appear to the human eye at the same time.

Fig 3.9 illustrates the multiplexing method of interfacing the two hexadecimal displays to the microcomputer. In the multiplexing scheme, seven-segment code is sent to all displays simultaneously. However, the segment to be illuminated is grounded.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

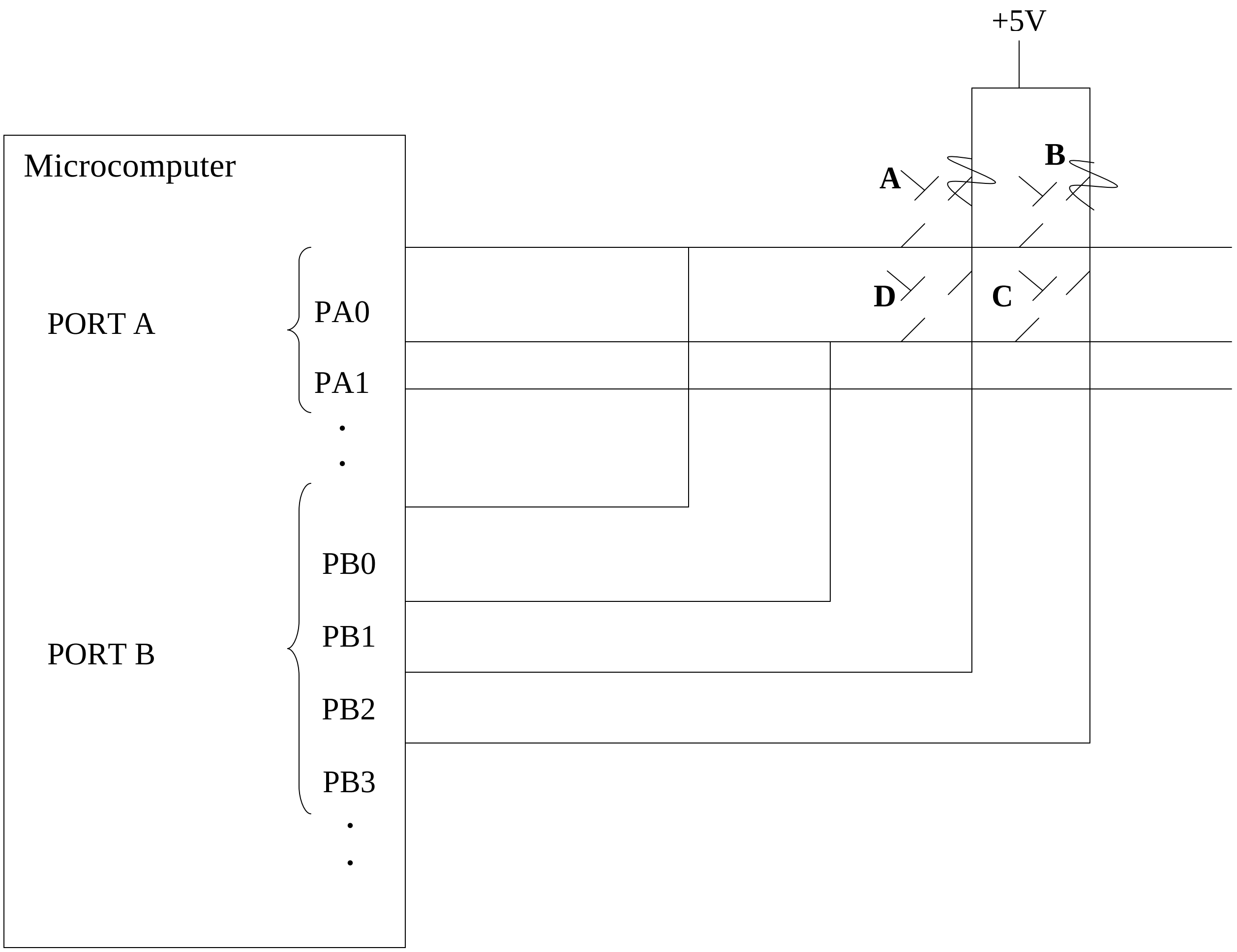


FIGURE 3.9 A 2X2 keyboard interfaced to a microprocessor

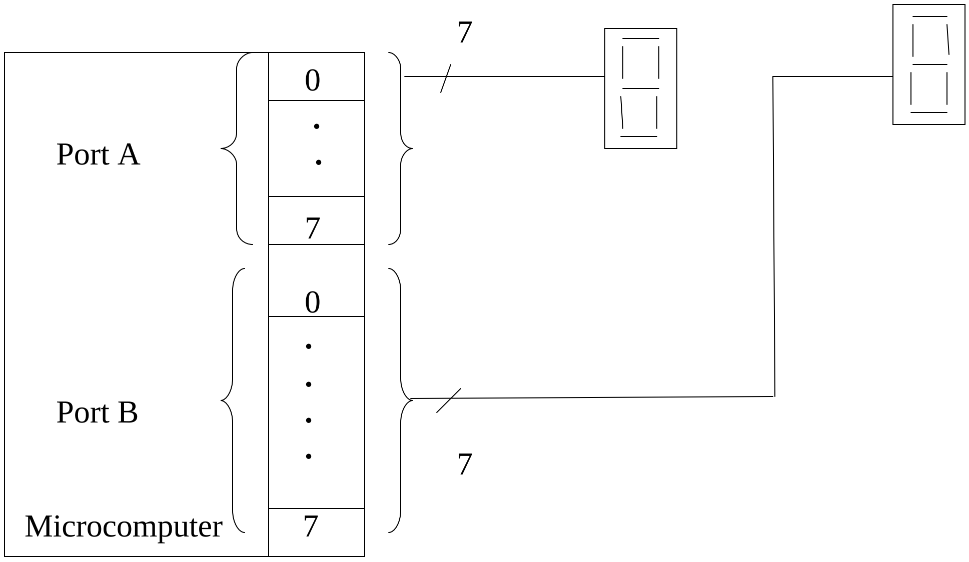


FIGURE 3.10 Non-multiplexed hexadecimal displays

This watermark does not appear in the registered version - http://www.clicktoconvert.com

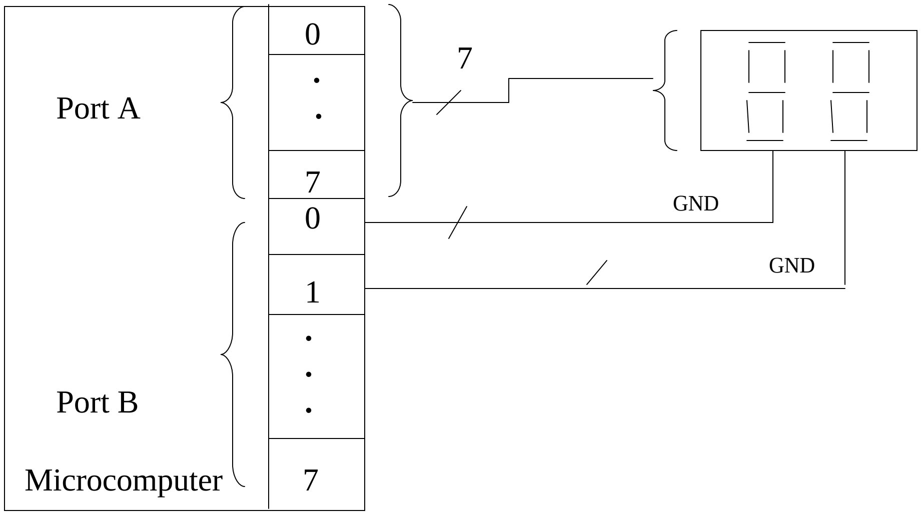


Figure 3.11 Multiplexed displays

The 8255A Programmable Peripheral Interface

The 8255A is a widely used, programmable, parallel I/O device. It can be programmed to transfer data under various conditions, from simple I/O interrupt I/O. It is flexible, versatile, and economical (when multiple I/O ports are required), but somewhat complex. It is an important general-purpose I/O device that can be used with almost any microprocessor.

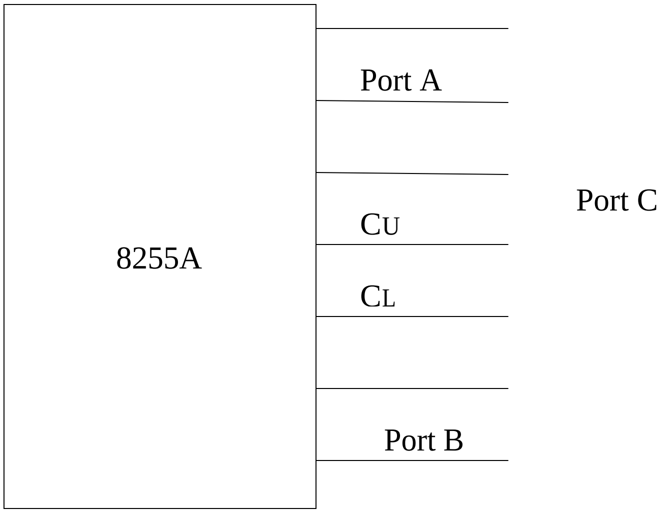


Figure 3.12 8255A I/O Ports

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The 8255A has 24 I/O pins that can be grouped primarily in two 8-bit parallel ports: A and B, with the remaining eight bits as port C. The eight bits of port C can be used as individual bits or be grouped in two 4-bit ports: CUPPER(CU) and CLOWER(CL), as into Figure 15.1.The function of these port are defined by writing a control word in the control register.

All the functions of the 8255A, classified according to two modes: the Bit Set/Reset (BSR) mode and the I/O mode. The BSR mode is used to set or reset the bits in port C. The I/O mode is further divided into three modes 0.Mode 1, and mode 2. In Mode 0, all port function as simple I/O ports. Mode 1 is a handshake mode whereby ports A and/or B use bits from port C as handshake signals. In the handshake mode, two type of I/O data transfer can be implemented: status check and interrupt. In Mode 2,port A can be set up for bi-directional data transfer using handshake signal from port C, and port B can be set up either in mode 0 or mode 1.

All the functions of the 8255A, classified according to two modes: the Bit Set/Reset (BSR) mode and the I/O mode. The BSR mode is used to set or reset the bits in port C. The I/O mode is further divided into three modes 0.Mode 1, and mode 2. In Mode 0, all port function as simple I/O ports. Mode 1 is a handshake mode whereby ports A and/or B use bits from port C as handshake signals. In the handshake mode, two type of I/O data transfer can be implemented: status check and interrupt. In Mode 2,port A can be set up for bi-directional data transfer using handshake signal from port C, and port B can be set up either in mode 0 or mode 1.

Block Diagram of the 8255A

The block diagram in Fig shows two 8-bit port (A and B), two 4-bit ports (CU and CL), and the data bus buffer, and control logic. This block diagram includes all the elements of a programmable device: port C performs function similar to that of the status register in addition to providing handshake signal.

CONTROL LOGIC

The control logic has six lines. Their functions to providing handshake signal

- **RD (Read):** This control signal enables the read operation. When the signal is low the MPU reads data from a selected I/O port of 8255A.
- **WR (Write):** This control signal enables the write operation. When the signal is low the MPU writes in to a selected I/O port of control register.
- **RESET (Reset):** This is an active high signal; it clears the controls register and sets all port in the input modes.
- **CS, A0, and A1:** These are device select signals. CS is connected to a decoded address and A0 and A1 are generally connected to MPU address lines A0 and A1, respectively.

BSR (Bit Set/Reset) Mode

The BSR modes is concerned only with the eight bits of port C, which can be set or rest by writing an appropriate control word in the control register. A control word with bit D7=0 is recognized as a BSR control word, and it does not alter any previously transmitted control word with bit D7=1: thus the I/O operation of ports A and B are not affected by a BSR control word. In the BSR mode, individual bits of port c can be used for application such as an on/off switch.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

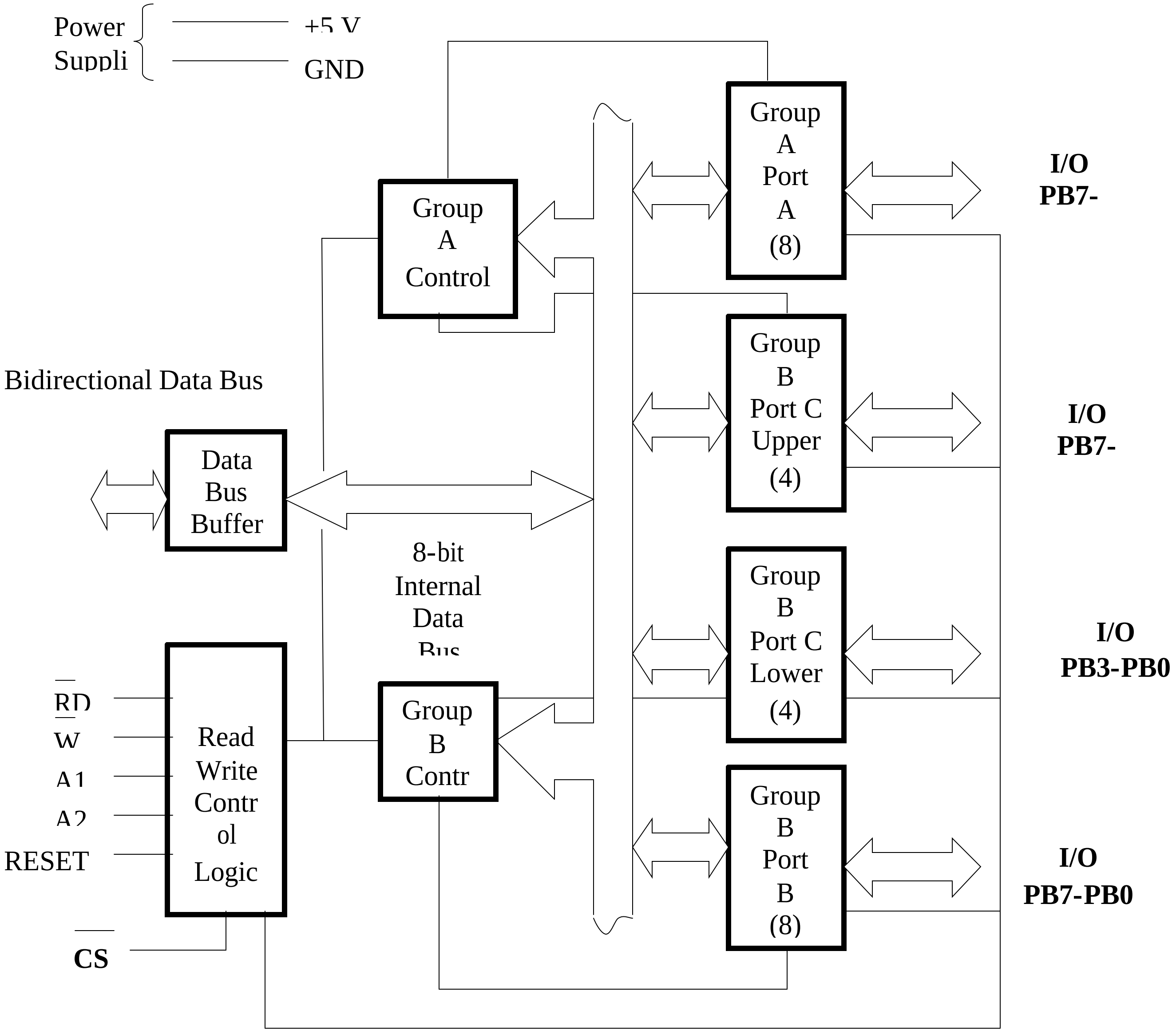


Figure 3.13 Block Diagram of the 8255A

Self Check Exercise : 2

1. State True or False:

(a) CALL instruction should be followed by a RET instruction True/False

(b) Conditional JUMP instruction require one of the flags to be tested. True/False

(c) The instruction ADD AD, ARRAY [BP+SI] is incorrect. True/False

This watermark does not appear in the registered version - http://www.clicktoconvert.com

2. Compare two address field micro instruction with one address field micro instruction. Which of them is more commonly used?

…………………………………………………………………………………………………..
…………………………………………………………………………………………………..
…………………………………………………………………………………………………..

3.8 LET US SUM UP

In this unit, we have studied about one the most popular series of microprocessors, viz Intel 8085.8085 serves as a base to all its successors. The successors of 8085 have all the features of their parent, plus a lot more advanced features. All the programs on 8085 can be directly run on any of its successors.

3.9 Lesson - End Activities

(1) Explain about the microprocessor initiated operation and 80 85 Bus organization.
(2) Discuss about the addressing modes of 80 85 Microprocessor.
(3) Explain about the instru8ction format in detail.

3.10 Points for Discussion

(1) Identify the instruction format and addressing modes for any two sample progress.
(2) Discuss the functional diagram and pin out diagram for an Microprocessor.

3.11 Check Your Progress: Model Answers

Self Check Exercise: 1

1.

(a) False
(b) True
(c) False
(d) True

2. More throughput, address capability, powerful instruction set and addressing modes, pipelined operations and virtual memory management scheme.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Self Check Exercise: 2

1.

a) False
b) True
c) false

2.

Two address field micro instructions	One address filed micro instructions
Micro program counter is not needed	Microprogram counter is needed
Wastes lot of control memory space	Smaller in comparison to two address
Very simple to implement	Slightly more complex to implement than two address.

3.12 REFERENCES

1. Microprocessors and Microcomputer-Based System Design – Mohamed Rafiquzzaman, second Edition.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

UNIT IV INPUT OUTPUT ORGANIZATION

CONTENTS:

4.0 AIMS AND OBJECTIVES

At the end of the unit you will be able to know how:

- Identify the structure of Input/Output Module.
- Describes the three types of Input/Output techniques, Viz., Programmed Input/Output Interrupt driven and Direct Memory Access and
- Define an Input/ Output Processor

4.1 INTRODUCTION

In this unit we will discuss briefly about Input Output Interfaces, then move on to the data transfer and priority interrupt, then we will discuss Direct Memory Access and at the end we will discuss about the Input/Output Processors which were quite common in mainframe computers. It introduces you to the world Input Output organization.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

4.2 INPUT – OUTPUT INTERFACE

Input output interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with CPU.

Difference between the central computer and each peripheral are:

1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.
2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU and consequently a synchronization mechanism may be needed.
3. Data codes and formats in peripherals differ from the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer systems include special hardware components between the CPU and peripherals to supervise and synchronize all input and output transfers. These components are called interface units because they interface between the processor bus and the peripheral device.

Each device may have its own controller that supervises the operations of the particular mechanism in the peripheral.

4.2.1 I/ O Bus and Interface Modules

A typical communication link between the processor and several peripherals is shown in Figure 4.1. The I/O bus consists of data lines, address lines and control lines. The magnetic disk, printer, and terminal are employed in practically any general-purpose computer. The magnetic tape is used in some computers for backup storage. Each peripheral device is has associated with it an interface unit. Each interface decodes the address and control received from the I/O Bus, interprets them for the peripheral, and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller that operates the particular electromechanical devices. For example, the printer controller controls the paper motion, the print timing and the selection of printing characters. A controller may be housed separately or may be physically integrated with the peripheral.

The I/O bus from the processor is attached to all peripheral interfaces. To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address lines. When the interface detects its own address, it activates the path between the bus lines and the device that it controls. All peripherals whose address does not correspond to the address in the bus are disabled by their interface.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

At the same time that the address is made available in the address lines, the processor provides a function code in the control lines. The interface selected responds to the function code and proceeds to execute it. The function code is referred to as an I/O command ad is in essence an instruction that is executed in the interface and its attached peripheral unit. The interpretation of the command depends on the peripheral that the processor is addressing. There are four types of commands that an interface may receive. They are classified as control, status, data output and data input.

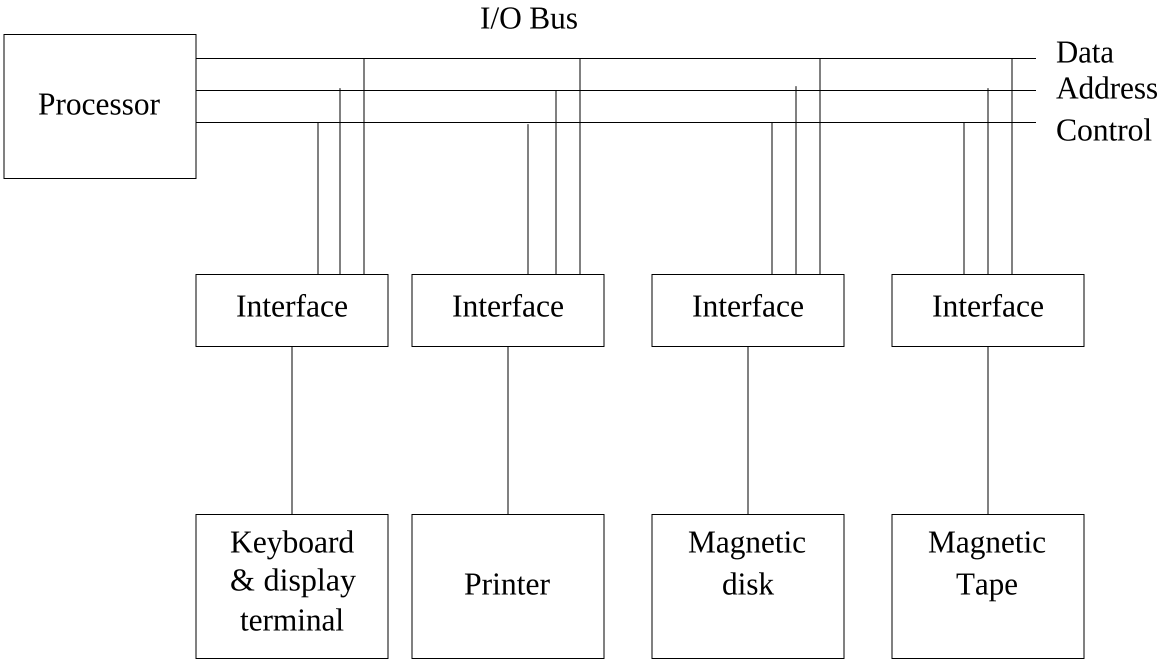


Figure 4.1 Connection of I/O bus to input – output devices

A control command is issued to activate the peripheral and to inform it what to do. For example, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction. The particular control command issued depends on the peripheral, and each peripheral receives its own distinguished sequence of control commands, depending on its mode of operations.

A status command is used to test various status conditions in the interface and the peripheral. For example, the computer may wish to check the status of the peripheral before a transfer is initiated. During the transfer, one or more errors occur which are detected by the interface. These errors are designated by setting bits in a status register that the processor can read at certain intervals.

A data output command causes the interface to respond by transferring data from the bus into one of its registers. Consider an example with a tape unit. The computer starts the tape moving by issuing a control command. The processor then monitors the status of the tape by means of a status command. When the tape is in the correct position, the processor issues a data output command. The interface responds to the address and command and transfers the information from the data lines in the bus to its buffer register. The interface then communicates with the tape controller and sends the data to be stored on tape.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The data input command is the opposite of the data output. In this case the interface receives an item of data from the peripheral and places it in its buffer register. The processor checks if data are available by means of a status command and then issues a datainput command. The interface places the data on the data lines, where they are accepted by the processor.

4. 2.2 I/O BUS VERSUS MEMORY BUS

In addition to communicating with I/O, the processor must communicate with memory unit. Memory bus contains data, address and read/write control lines. Three ways for computer buses that can be used to communicate with memory and I/O:

1. Use two separate buses, one for memory and other for I/O.
2. Use one common bus for both memory and I/O but have separate control lines for each.
3. Use one common bus for memory and I/O with common control lines.

In the first method, the computer has independent sets of data, address and control buses, one for accessing memory and other for I/O. this is done in computers that provide a separate I/O processor (IOP) in addition to the CPU.

The memory communicates with both the CPU and the IOP through a memory bus and also with input and output devices through a separate I/O bus with its own address, data and control lines. The purpose of the IOP is to provide an independent pathway for the transfer of information between the external devices and the internal memory. The I/O processor is sometimes called a data channel.

4.2.3 ISOLATED VERSUS MEMORY MAPPED I/O

Many computers use one common bus to transfer information between memory or I/O and the CPU. The distinction between a memory transfer and I/O transfer is made through separate read and write lines.

The CPU specifies whether the address on the address lines is for a memory word or for an interface register by enabling one of two possible read or writ lines. The I/O read and I/O writes control lines are enabled during an I/O transfer. The memory read and memory write control lines are enabled during a memory transfer.

This configuration isolates all I/O interface addresses from the address assigned to memory and is referred to as the isolated I/O method for assigning addresses in a common bus.

ISOLATED I/O

In the isolated I/O configuration, the CPU has distinct input and output instructions and each of these instructions are associated with the address of an interface register. When the CPU fetches and decodes the operation code of an input or output instruction, it places the address

This watermark does not appear in the registered version - http://www.clicktoconvert.com

associated with the instruction into the common address lines. At the same time, it enables the I/O read (for input) or I/O write (for output) control line. This informs the external components that are attached to the common bus that the address in the address lines is for an interface register and not for a memory word.

When the CPU is fetching an instruction or an operand from memory, it places the memory address on the address lines and enables the memory read or memory write control lines. This informs the external components that the address is for a memory word and not for an I/O interface.

MEMORY MAPPED I/O

The isolated I/O method isolates memory and I/O addresses so that memory address values are not affected by interface address assignment since each has its own address space. The other alternative is to use the same address space for both memory and I/O. This is the case in computers that employ only one set of read and write signals and do not distinguish between memory and I/O addresses. This configuration is referred to as memory mapped I/O.

The computer treats an interface register as being part of the memory system. The assigned addresses for interface registers cannot be used for memory words, which reduce the memory address, range available. In memory mapped I/O organization, there are no specific inputs or output instructions. The CPU can manipulate I/O data residing in interface registers with the same instructions that are used to manipulate memory words.

Each interface is organized as a set of registers that respond to read and write requests in the normal address space. Typically, a segment of the total address space is reserved for interface registers, but in general, they can be located at any address as long as there is not also a memory word that responds to the same address. Computers with memory mapped I/O can use memory type instructions to access I/O data. It allows the computer to use the same instructions for either input-output transfers or for memory transfers.

ADVANTAGE

Load and store instructions used for reading and writing from memory can be used to input and output data from I/O registers.

4.2.4 EXAMPLE OF I/O INTERFACE

An example of I/O interface unit is shown in block diagram form in Figure 4.2.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

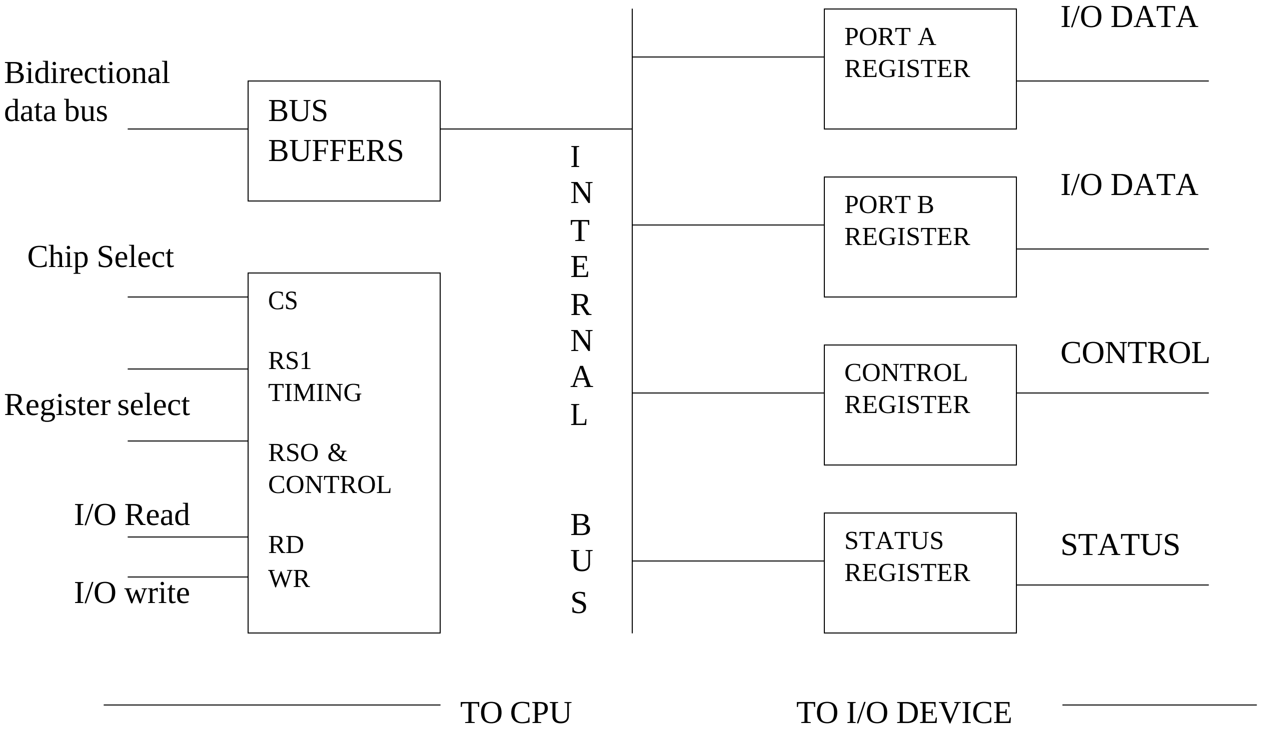


Figure. 4.2 Example of I/O Interface Unit

It consists of two data registers called ports, a control register, a status register, bus buffers and timing and control circuits. The interface communicates with the CPU through the data bus. The chip select and register select inputs determine the address assigned to the interface. The I/O read and write are two control lines that specify an input or output respectively. The four registers communicate directly with the I/O device attached to the interface.

The I/O data to and from the device can be transferred into either port A or B. the interface may operate with an output device or with an input device or with a device that requires both input and output. If the interface is connected to a printer, it will only output data and if it services a character reader, it will only input data.

A magnetic disk unit transfers data in both directions but not at the same time, so the interface can use bidirectional lines. A command is passed to the I/O device by sending a word to the appropriate interface register. The control is sent to the control register, status information is received from the status register and data are transferred to and from ports A and B registers.

Thus the transfer of data, control and status information is always via the common data bus. The distinction between data, control or status information is determined from the particular interface register with which the CPU communicates. The control register receives control information from the CPU. By loading appropriate bits into the control register, the interface and the I/O device attached to it can be placed in a variety of a operating modes.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

For example, a status bit may indicate that port A has received a new data item from the I/O device. Another bit in the status register may indicate that a parity error has occurred during the transfer. The interface registers communicate with CPU through the bidirectional data bus. The address bus selects the interface unit through the chip select and the two register select inputs.

The circuit enables the chip select (CS) input when the interface is selected by the address bus. The two register select inputs RS1 and RS0 are usually connected to the two least significant lines of the address bus. These two inputs select one of the four registers in the interface as specified in the table accompanying the diagram.

The content of the selected registered is transfer into the CPU via the data bus when the I/O read signal is enabled. The CPU transfers binary information into the selected register via the data bus when the I/O write input is enabled.

4.3 ASYNCHRONOUS DATA TRANSFER

The internal operations in a digital system are synchronized by means of clock pulses supplied by a common pulse generator. If the interface registers and CPU shares a common clock, the transfer is said to be synchronous.

Clock pulses are applied to all registers within a unit and all data transfers among internal registers occur simultaneously during the occurrence of a clock pulse. If the internal timing in each unit is independent from the other, each uses its own private clock for internal registers. In this case, two units are said to be synchronous to each other.
Methods

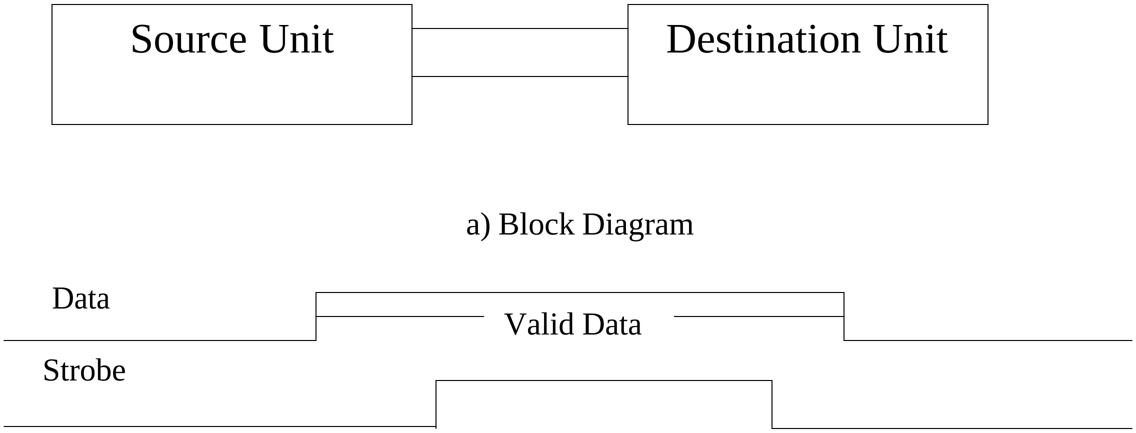
Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted. Two ways of achieving this,

1. Strobe control
2. Handshaking

4.3.1 STROBE CONTROL

The strobe control method of asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either the source or the destination unit. Strobe pulse is supplied by one of the units to indicate to the other unit when the transfer has to occur.

This watermark does not appear in the registered version - http://www.clicktoconvert.com



a) Block Diagram

b) Timing Diagram

4. 3 Source initiated Strobe for data transfer

The Figure 4.3 shows a data transfer initiated by the destination unit activates the strobe pulse informing the source to provide the data. The source unit responds by placing the requested binary information on the data bus. The data must be valid and remain in the bus long enough for the destination unit to accept it. The falling edge of the strobe pulse can be used again to trigger a destination register. The destination unit then disables the strobe. The source removes the data from the bus after a predetermined time interval.

The strobe pulse is actually controlled by the clock pulse in the CPU. The CPU is always in control of the buses and informs the external units how to transfer data. For example, the strobe of Figure 4.4 could be a memory write control signal from the CPU to a memory unit.

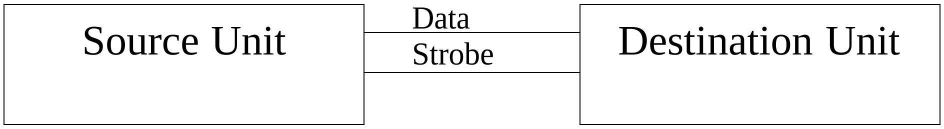
The source being the CPU, places a word on the data bus and informs the memory unit, which is the destination, that this is the write operation. Figure 4.2 (a) shows a source-initiated transfer.

The data bus carries the binary information from source unit to the destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available in the bus.

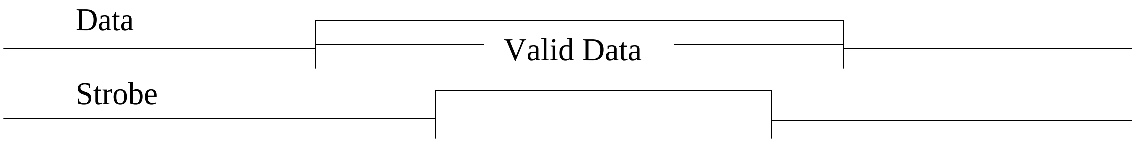
As shown in the timing diagram of Figure 4.3 (b). The source unit first places the data on the data bus. The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data. The destination unit uses the falling edge of the strobe pulse to transfer the contents of the data bus into one of its internal registers. The source removes the data from the bus a brief period after it disables its strobe pulse.

Actually, the source does not have to change the information in the data bus. The fact that the strobe signal is disabled indicates that the data bus does not contain valid data. New valid data will be available only after the strobe is enabled again.

This watermark does not appear in the registered version - http://www.clicktoconvert.com



a) Block diagram



b) Timing Diagram

Figure. 4.4 Destination – Initiated Strobe for data Transfer

The strobe of Figure 4.4(b) could be a memory-read control signal from CPU to a memory unit. The destination, the CPU, initiates the read operation to inform the memory, which is the source, to place a selected word into the data bus.

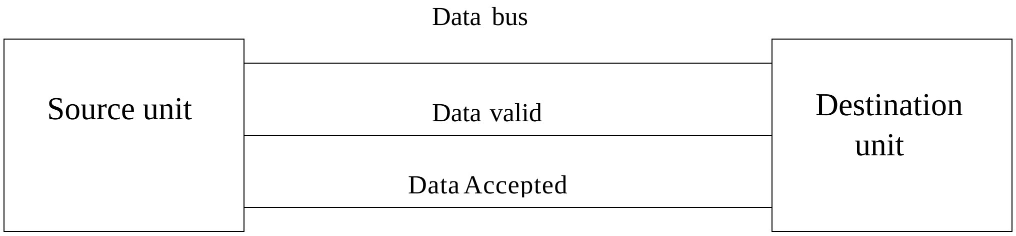
Strobe transfer → data transfer between CPU and an interface unit.
Handshaking → data transfer between an interface and an I/O device.

4.3.2 HANDSHAKING

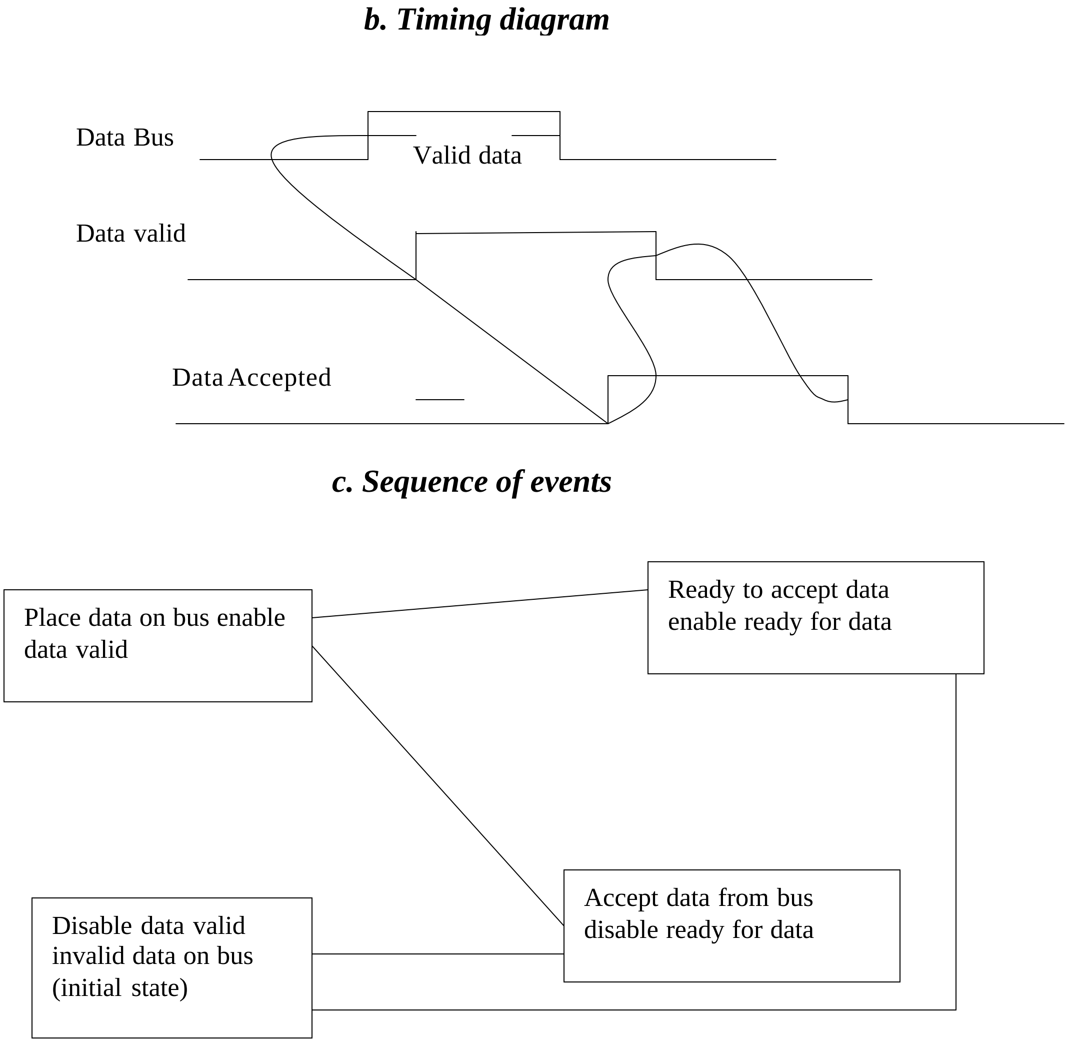
This is commonly used to accompany each data being transferred with a control signal that indicates the presence of data in a bus. The unit receiving the data item responds with another control signal to acknowledge receipt of the data. This type of agreement between two independent units is referred to as handshaking.

The basic principle of the two-wire handshaking method of data transfer is as follows: One control line is in the same direction as the dataflow in the bus from the source to the destination. It is used by the source unit to inform the destination unit whether there are valid data in the bus. The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept data.

a. Block diagram



This watermark does not appear in the registered version - http://www.clicktoconvert.com


b. Timing diagram

c. Sequence of events

Data Bus
Valid data
Data valid
Data Accepted
Place data on bus enable data valid
Ready to accept data enable ready for data
Accept data from bus disable ready for data
Disable data valid invalid data on bus (initial state)

4.5 Source-initiated transfer using handshaking

Two handshaking lines are,

Data valid → generated by source unit.
Data accepted → generated by destination unit.

The timing diagram shows the exchanged of signals between the two units. The sequence of events listed in Figure 4.5 (c) and shows the four possible states that the system can be at any given time.

SOURCE INITIATED TRANSFER

The source unit initiates the transfer by placing the data on the bus and enabling its data valid signal. The data accepted signal is activated by the destination unit after it accepts the data from the bus. The source unit then disables its data valid signal, which invalidates the data on the bus. The destination unit then disables its data accepted signal and the system goes into the initial state.

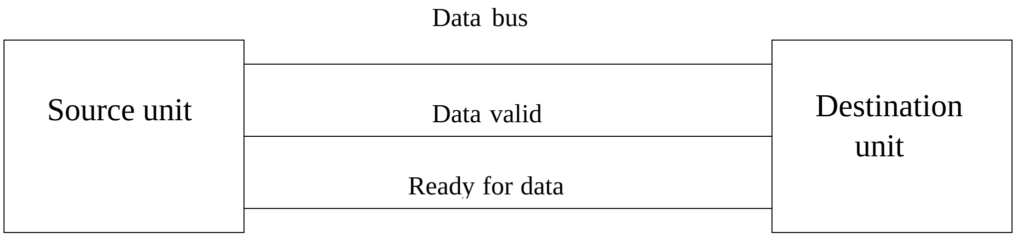
This watermark does not appear in the registered version - http://www.clicktoconvert.com

The source does not send the next data item until after the destination unit shows readiness to accept new data by disabling data accepted signal. This scheme allows arbitrary delays from one state to the next state. It permits each unit to respond at its own data transfer rate.

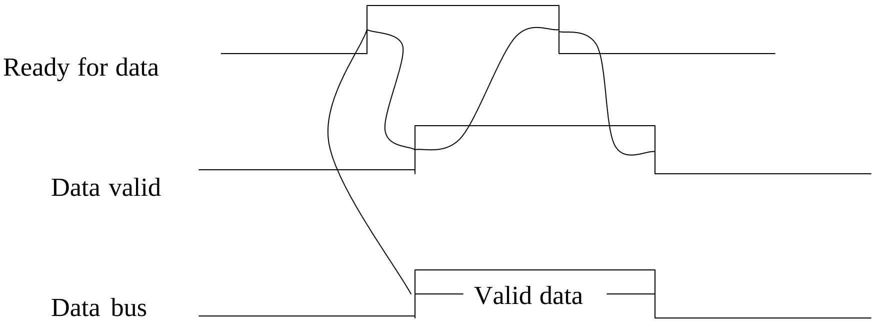
DESTINATION INITIATED TRANSFER

The destination initiated transfer using handshaking lines is shown in Figure 4.6. The source unit does not place data on the bus until after it receives the ready for data signal from the destination unit. The sequence of events in both cases would be identical if we consider the ready for data signal as the complement of data accepted. The only difference between the source-initiated and destination-initiated transfer is in their choice of initial state.

a. Block diagram

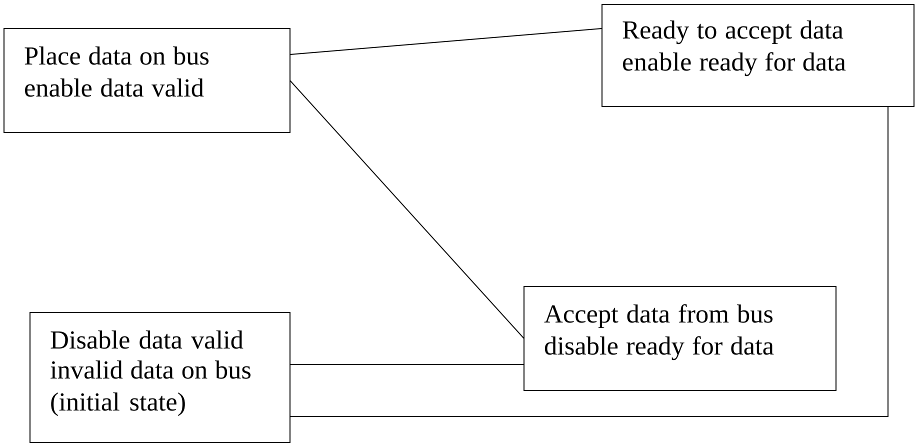


b. Timing diagram



This watermark does not appear in the registered version - http://www.clicktoconvert.com

c. Sequence of events



4.6 Destination-initiated transfer using handshaking

ADVANTAGES OF HANDSHAKING MECHANISM

The handshake scheme provides high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units.

TIMEOUT MECHANISM

If either source or destination is faulty, the data transfer will not be completed. Such an error can be detected by means of a timeout mechanism. It produces an alarm if the data transfer is not completed within a predetermined time. It is implemented by means of an internal clock that starts counting time when the unit enables one of its handshaking control signals. If the return handshake signal does not respond within a given time period, the unit assumes that an error has occurred.

Self-Check Exercise: 1

1. Say True or False:

(a) The devices are normally connected directly to system bus. True/ False
(b) Input/Output Module is needed only for slower I/O device. True/ False
(c) Data buffering is helpful for smoothing out the speed difference between CPU and input/output devices. True/ False

2. What is device Controller?

…………………………………………………………………………………………………
…………………………………………………………………………………………………
…………………………………………………………………………………………………

This watermark does not appear in the registered version - http://www.clicktoconvert.com

4.4 PRIORITY INTERRUPT

A priority interrupt is a system that establishes a priority over the various sources to destination which condition is to be serviced first when two or more requests arrive simultaneously. The system may also determine which conditions are permitted to interrupt the computer while another interrupt is being serviced. Higher-priority interrupt levels are assigned to requests, which, if delayed or interrupted, could have serious consequences. Devices with high-speed transfer such as magnetic disks are given high-priority, and slow devices such as keyboards receive low-priority. When two devices interrupt the computer at the same time, the computer services the device, with the higher priority first.

Establishing the priority of simultaneous interrupt can be done by software or hardware. A polling procedure is used to identify the highest-priority source by software means. In this method there is one common branch address for all interrupts. The program that takes care of interrupts begins at the branch address and polls the interrupt sources in sequence. The order in which they are tested determines the priority of each interrupt.

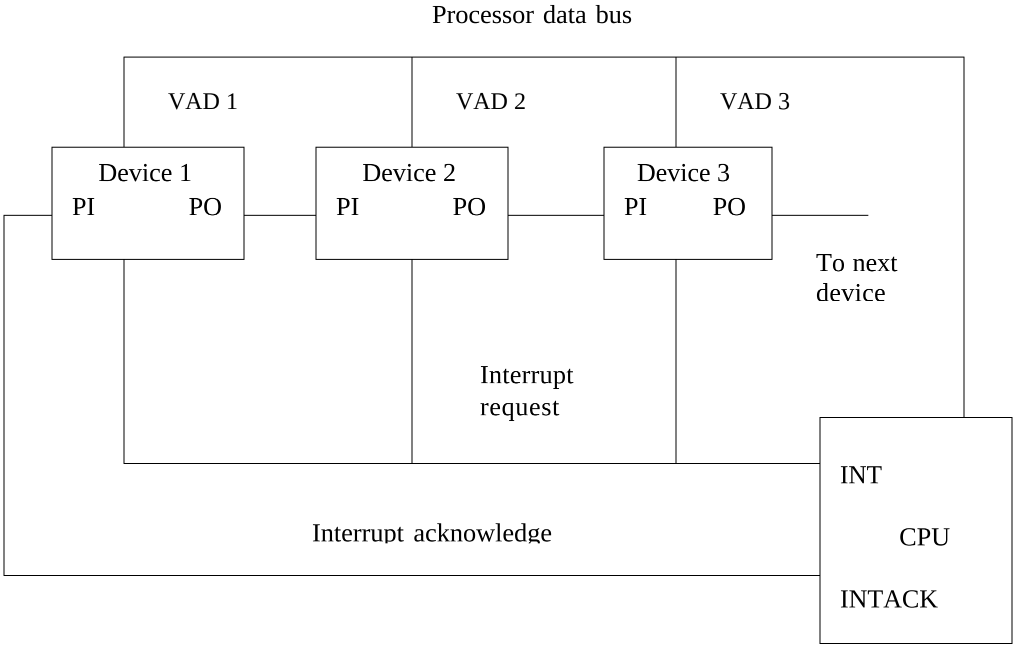
The highest-priority source is tested first, and if its interrupt signal is on, control branches to a service routine for this source. Otherwise the next-lower priority source is tested and so on. Thus the initial service routine for all interrupts consists of a program that tests the interrupt sources in sequence and branches to one of many possible service routines. The particular service routine reached belongs to the highest-priority device among all devices that interrupted the computer. The disadvantage of the software method is that if there are many interrupts, the time required to poll them can exceed the time available to service the I/O device. In this situation a hardware priority-interrupt unit can be used to speed up the operation.

A hardware priority-interrupt unit functions as an overall manager in an interrupt system environment. It accepts interrupt requests from many sources, determines which of the incoming request has the highest priority and issues an interrupt request to the computer based on this determination. To speed up the operation each interrupt source has its own interrupt vector to access its own service routine directly. Thus no polling is required because priority interrupt unit. The hardware priority function can be established by either a serial or a parallel connection of interrupt lines. The serial connection is also known as the daisy chaining method.

4.4.1 DAISY CHAINING METHOD

The daisy chaining method of establishing priority consists of a serial connection of all devices that request an interrupt. The device with the highest priority is placed in the first position, followed by lower priority devices up to the device with the lowest priority, which is placed last in the chain. This method of connection between these devices and the CPU is known as daisy chain priority interrupt.

This watermark does not appear in the registered version - http://www.clicktoconvert.com



4.7 Daisy chain method

The interrupt request line is common to all devices and forms a wired logic connection. If any device has to interrupt signal in the low-level state, the interrupt line goes to the low-level state and enables the interrupt input in the CPU. When no interrupts are pending, the interrupt line stays in the high-level state and no interrupts are recognized by the CPU responds to an interrupt request by enabling the interrupt acknowledge line. This signal is received by device 1 at its PI (Priority In) input. The acknowledge signal passes on to the next device through the PO (Priority out) output only if device 1 is not requesting an interrupt. If device 1 has a pending interrupt, it blocks the acknowledge signal from the next device by placing a 0 in the PO output. It then proceeds to insect its own interrupt vector address (VAD) into the data bus for the CPU to use during the interrupt cycle.

A device with a 0 in its PI input generates a 0 in its PO output to inform the next-lower-priority device that the acknowledge signal has been blocked. A device that is requesting an interrupt and has a 1 in its PI input will interrupt the acknowledge signal by placing a 0 in its PO output. If the device does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output. Thus the device with PI-1 and PO=0 is the one with the highest priority that is requesting an interrupt and the device places its VAD on the data bus.

The daisy chain arrangement gives the highest priority to the device that receives the interrupt acknowledge signal from the CPU. The farther the device is from the first position, the lower is its priority.

4.4.2 PARALLEL PRIORITY INTERRUPT

The parallel priority interrupt method uses a register whose bits are set separately by the interrupt signal from each device priority is established according to the position of the bits in the

This watermark does not appear in the registered version - http://www.clicktoconvert.com

register. In addition to the computer register, the circuit may include a mask register whose purpose is to control the status of each interrupt request. The mask register can be programmed to disable lower priority interrupts while a higher priority device is being serviced. It can also provide a facility that allows a high priority device to interrupt the CPU while a lower priority device is being serviced.

The priority logic for a system of four interrupts sources. It consists of an interrupt register whose individual bits are set by external conditions and cleared by program instructions. The magnetic disk, being a high-speed device, is given the highest priority. The printer has the next priority, followed by a character reader and a keyboard. The mask register has the same number of bits as the interrupt register. By means of program instructions, it is possible to set or reset any bit in the mask register. Each interrupt bit and its corresponding mask bit are applied to an AND gate to produce the four inputs to a priority encoder. In this way an interrupt is recognized only if its corresponding mask bit is set to 1 by the program. The priority encoder generates two bits of the vector address, which is transferred to the CPU.

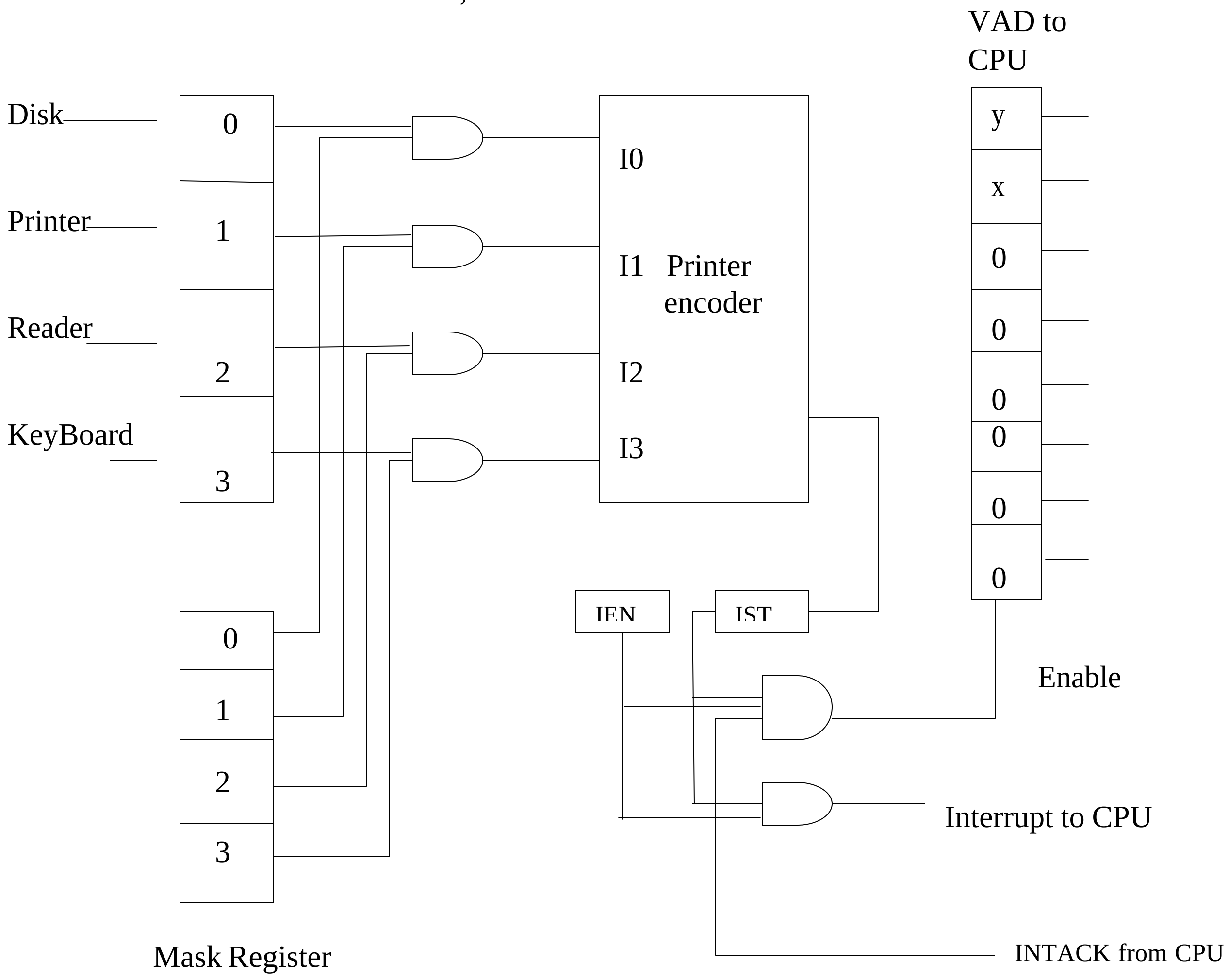


Figure 4.8 Priority Interrupt Hardware

This watermark does not appear in the registered version - http://www.clicktoconvert.com

4. 5 DIRECT MEMORY ACCESS (DMA)

The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called direct memory access (DMA).

During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

BUS REQUEST

The bus request (BR) input is used by the DMA controller to request the CPU to relinquish control of the buses.

BUS GRANT

CPU activates the bus grant (BG) output to inform the external DMA that the buses are in a high impedance state.

BURST TRANSFER

In DMA burst transfer, a block sequence consisting of a number of memory words is transferred in a continuous burst while the AMA controller is master of the memory buses.

CYCLE STEALING

Cycle stealing allows the DMA controller to transfer one data word at a time, after which it must return control of the buses to the CPU.

In addition, it needs an address register, a word count register, and a set of address lines. The address register and address lines are used for direct communication with the memory. The word count register specifies the number of words that must be transferred. The data transfer may be close directly between the device and memory under control of the DMA.

The Figure 4.9 shows the block diagram of a typical DMA controller. The unit communicates with the CPU via the data bus and control lines. The CPU through the address bus selects the registers in the DMA by enabling the DS (DMA Select) and RS (Register Select) inputs. The RD (read) and WR (write) inputs are bi-directional. When the BG (bus grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers.

When BG=1 the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control. The DMA Controller has three registers.

1. Address register
2. Word count register
3. Control register.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

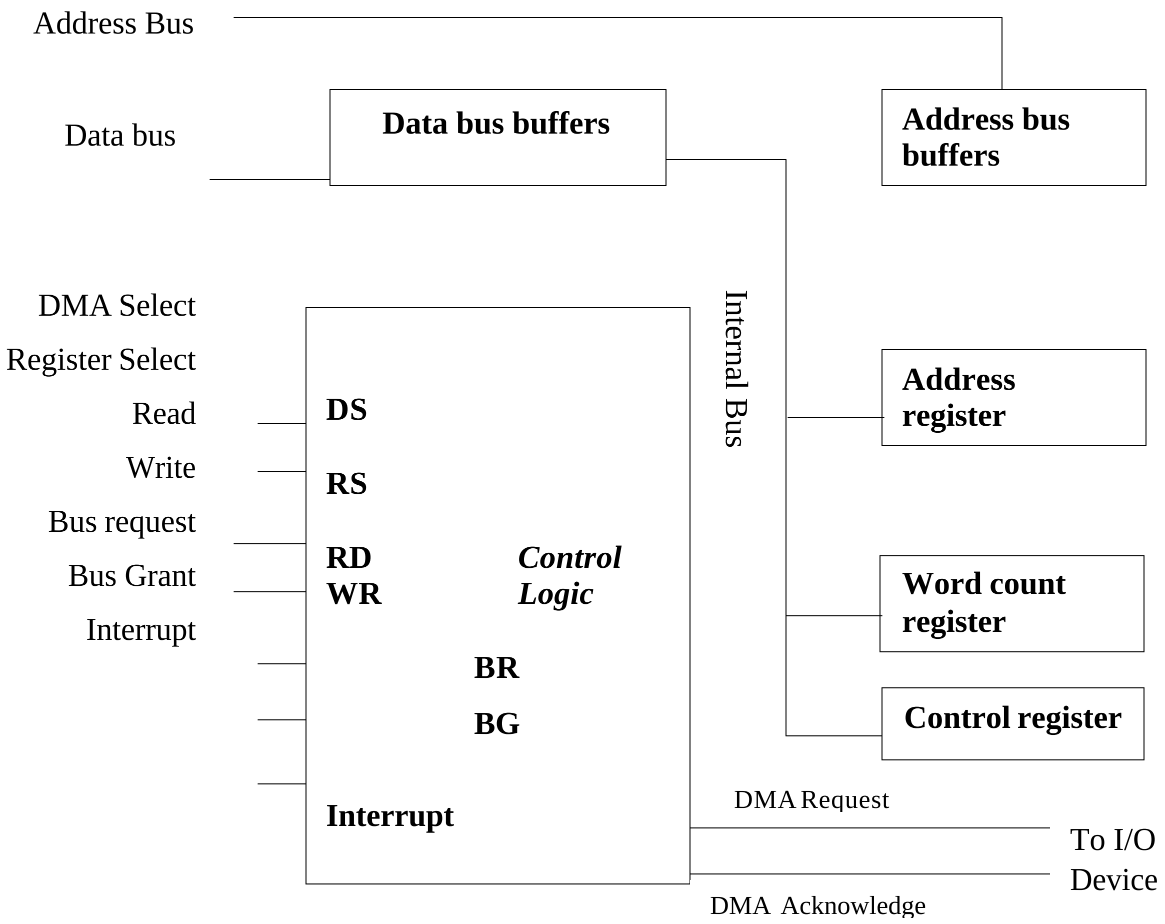


Figure 4.9 Block Diagram of DMA Controller

ADDRESS REGISTER

Address register contains an address to specify the desired location in memory.

WORD COUNT REGISTER

Word count register holds the numb4er of words to be transferred.

CONTROL REGISTER

The control register specifies the mode of transfer.

DMA Transfer

The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initiates the DMA through the data bus. When the peripheral device sends a DMA request, the DMA controller activates the BR line, informing the CPU to relinquish the buses.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The CPU responds with its BG line. The DMA then puts the current value of its address register into the address bus, initiates the RD or WR signal and sends a DMA acknowledge to the peripheral device. The direction of transfer depends on the status of the BG line.

When BG=0, the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When BG=1 the RD and WR are output lines from the DMA controller to the random access memory to specify the read or write operation for the data. When the peripheral device receives a DMA receives a word from the data bus. The peripheral unit can then communicate with memory through the data bus for direct transfer between the two units while the CPU is momentarily disabled.

For each word that is transferred, the DMA increments its address registers and decrements its word count registers.

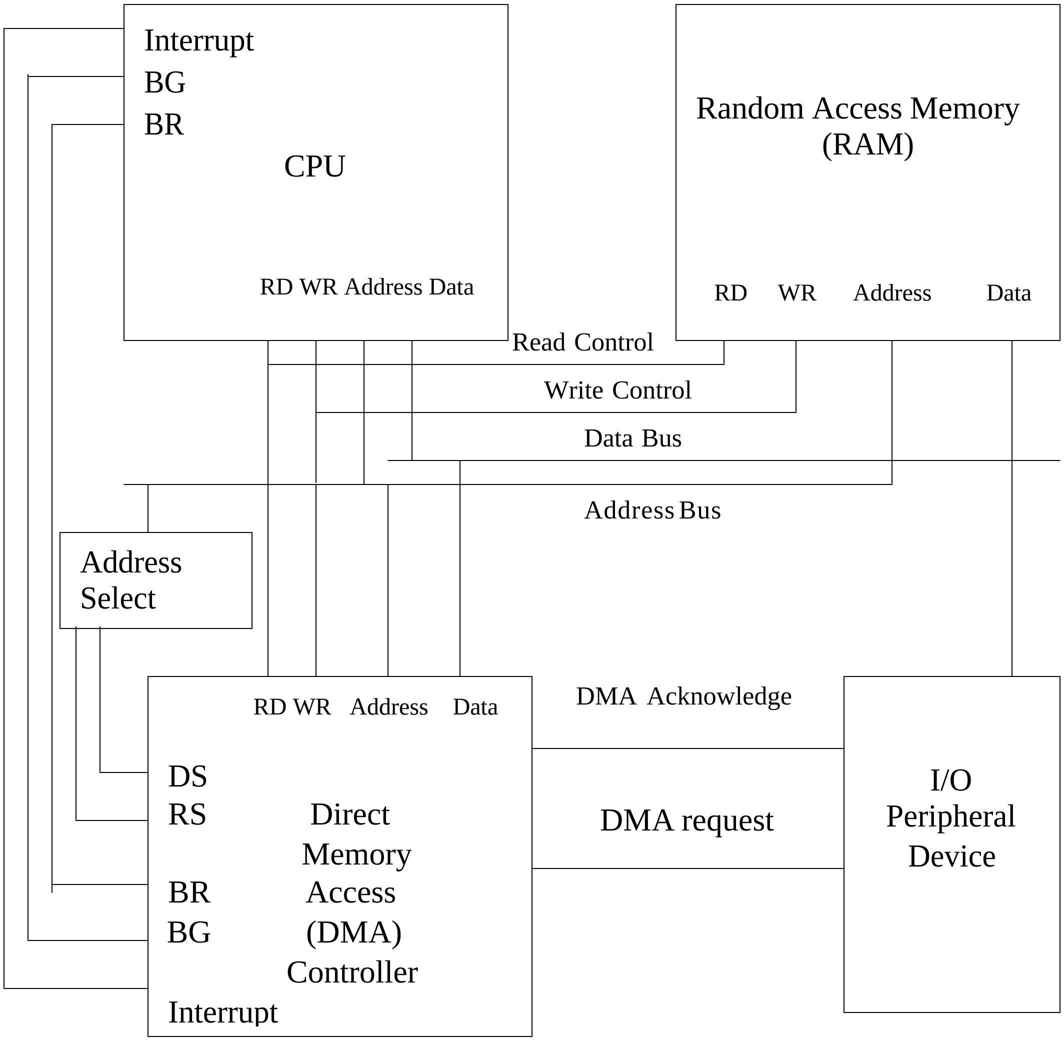


Figure 4.10 DMA transfer in a computer system.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

If the word count registers reaches zero, the DMA stops any further transfer and removes its bus request. It also informs the CPU of the termination by means of an interrupt. A DMA controller may have more than one channel. In this case, each channel has a request and acknowledges pair of control signals, which are connected to separate peripheral devices. A priority among the channels may be established so that channels with high priority are serviced before channels with high priority are serviced before channels with lower priority. DMA transfer is very useful in many applications. It is used for fast transfer of information between magnetic disks and memory. It is also useful for updating the display in an interactive terminal..

4.6 INPUT-OUTPUT PROCESSOR (IOP)

An input-output processor (IOP may be classified as a processor with direct memory access capability that communicates with I/O devices. In this configuration, the computer system can be divided into a memory unit, and a number of processors comprised of the CPU and one or more IOPs. Each takes care of input and output tasks.

The block diagram of a computes with two processor is shown in Figure 4.11.

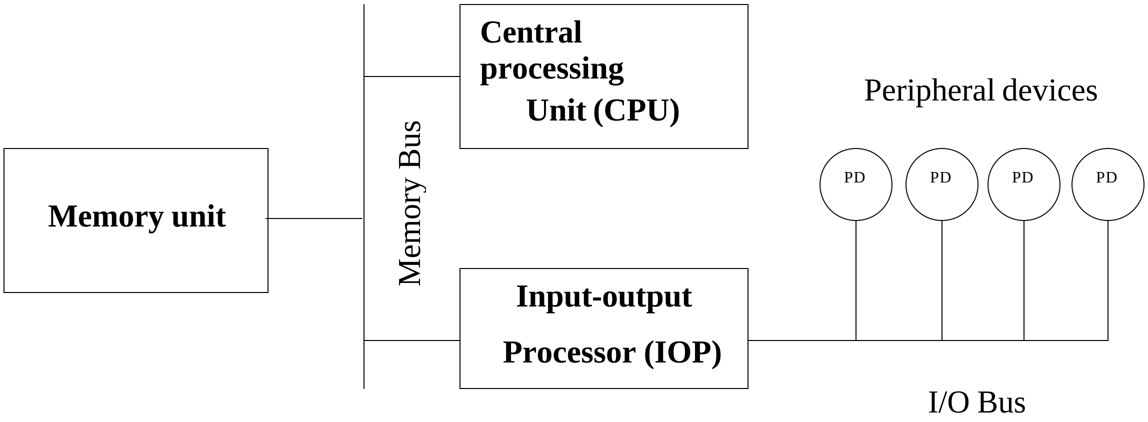


Figure 4.11 Block diagram of a computer with I/O processor.

The memory unit occupies a central position and can communicate with each processor by means of direct access memory. The CPU is responsible for processing data needed in the solution of computational tasks. The IOP provides a path for transfer of data between various peripheral devices and the memory unit. The CPU is usually assigned the task of initiating the I/O program. From then on the IOP operates independent of the CPU and continues to transfer data from external devices and memory.

4.6.1 CPU-IOP COMMUNICATION

The communication between CPU and IOP may take different forms, depending on the particular computer considered. In most cases the memory unit acts as a message center where each processor leaves information for the other. The cooperation of a typical IOP is specified by an example, the method by which the CPU and IOP communicate. The sequence of operations may be carried out as shown in the flowchart.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

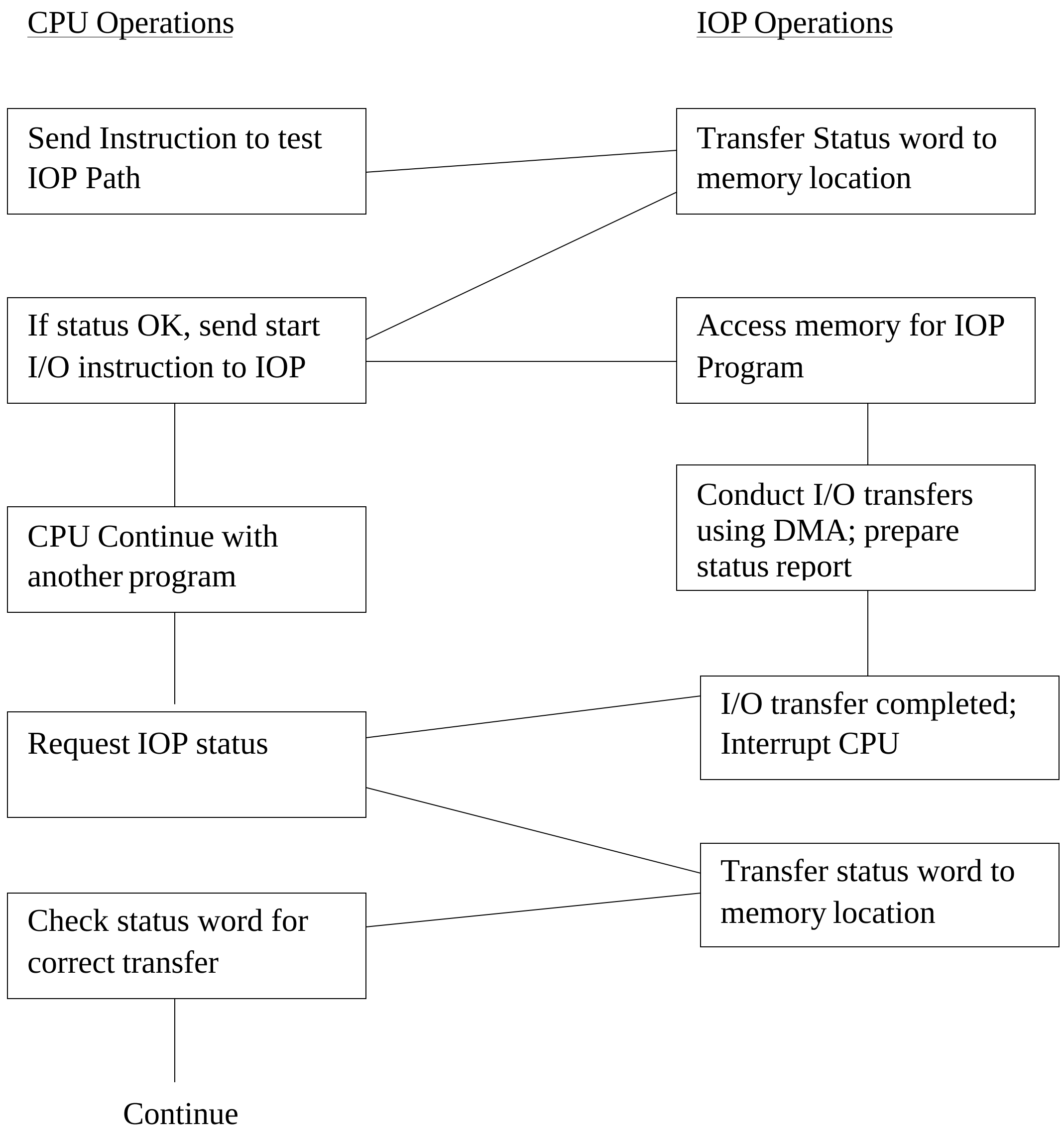

CPU Operations
IOP Operations
Send Instruction to test IOP Path
Transfer Status word to memory location
If status OK, send start I/O instruction to IOP
Access memory for IOP Program
Conduct I/O transfers using DMA; prepare status report
CPU Continue with another program
I/O transfer completed; Interrupt CPU
Request IOP status
Transfer status word to memory location
Check status word for correct transfer
Continue

Figure 4.12 CPU – IOP Communication

The CPU sends an instruction to test the IOP path. The IOP responds by inserting a status word in memory for the CPU to check. The bits of the status word indicate the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer or device ready for I/O transfer. The CPU refers to the status word in memory to decide what too do next. If all is in order, the CPU sends the instruction to start I/O transfer. The memory address received with this instruction tells the IOP where to find its program. The CPU can now continue with another program while the IOP is busy with the I/O program. Both programs refer to memory by means of its program it sends an interrupt request to the CPU. The CPU responds to the interrupt by issuing an instruction to read the status from the IOP. The IOP responds by placing the contents of its status report into specified memory locations. The status word indicates whether the transfer has been completed or if any errors occurred during the transfer.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

From inspection of the bits in the status word, the CPU determines if the I/O operation was completed satisfactorily without errors.

The IOP takes care of all data transfers between several I/O units and the memory while the CPU is processing another program. The IOP and CPU are competing for the use of use of memory, so the number of devices that can be in operation is limited by the access time of the memory.

Self-Check Exercise: 2

Say True or False

1. I/O mapped I/O scheme requires no additional line from CPU to I/O device except for system bus. True / False
2. In the transparent DMA the cycles are stolen only when CPU is not using the bus. True / False
3. Most of the I/O processors have its own memory except for register or a simple buffer area. True / False
4. Parallel interfaces are commonly used for connecting printers to a computer. True / False

4.7 LET US SUM UP

In this unit, we have covered the five major points, which include

- ❖ Input Output Interface
- ❖ Asynchronous Data Transfer
- ❖ Priority Interrupt
- ❖ Direct Memory Access
- ❖ Input Output Processor

Input output interface provides a method for transferring information between internal storage and external I/O devices.

The internal operations in a digital system are synchronized by means of clock pulses supplied by a common pulse generator. If the interface registers and CPU shares a common clock, the transfer is said to be synchronous.

A priority interrupt is a system that establishes a priority over the various sources to destination which condition is to be serviced first when two or more requests arrive simultaneously.

The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called direct memory access (DMA).

An input-output processor (IOP may be classified as a processor with direct memory access capability that communicates with I/O devices.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

4.8 LESSON – END ACTIVITIES

(1) Explain about Input and Output Interfaces.
(2) Discuss about Asynchronous data transfer in detail.
(3) Explain the Design of DMA Controller for Data Transfer.

4.9 POINTS FOR DISCUSSION

(1) Mode of Data transfer – Three types; Discussion is detail.
(2) Demonstrate how CPU and I/O have communication for Data Transfer.

4.10 CHECK YOUR PROGRESS: MODEL ANSWERS

Self-Check Exercise 1

1. (a) False
   (b) False
   (c) True

2. A device controller is an I/O Module, which interacts with the I/O devices as per the instructions, provided by the CPU.

Self-Check Exercise 2

1. False
2. True
3. True
4. True

4.11 REFERENCES

1. Stallings, William, Computer Organization and Architecture, Fourth Edition, Prentice Hall of India, 1997
2. Mano, M. Morris, Somputer System Architecture, Third Edition, Prentice Hall of India, 1993.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

UNIT V

MEMORY ORGANIZATION

5.0 AIMS AND OBJECTIVE

At the end of this unit, you will be able to:

- Describe the key characteristics of memory system
- Distinguish among the various types of random access memories.
- Describe the importance of cache memory and other high speed memories.

5.1 INTRODUCTION

In this unit we will discuss about the main memory, Associative memory, cache memory and virtual memory.

5.2 MEMORY HIERARCHY

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The total memory capacity of a computer can be visualized as being a hierarchy of components. The memory hierarchy system consists of all storage devices employed by a computer system from the software but high capacity auxiliary memory devices to a relatively fast main memory to an even very smaller and very faster buffer memory accessible to the high-speed processor logic.

The following Figure 5. 1 illustrates the components in a typical memory hierarchy.

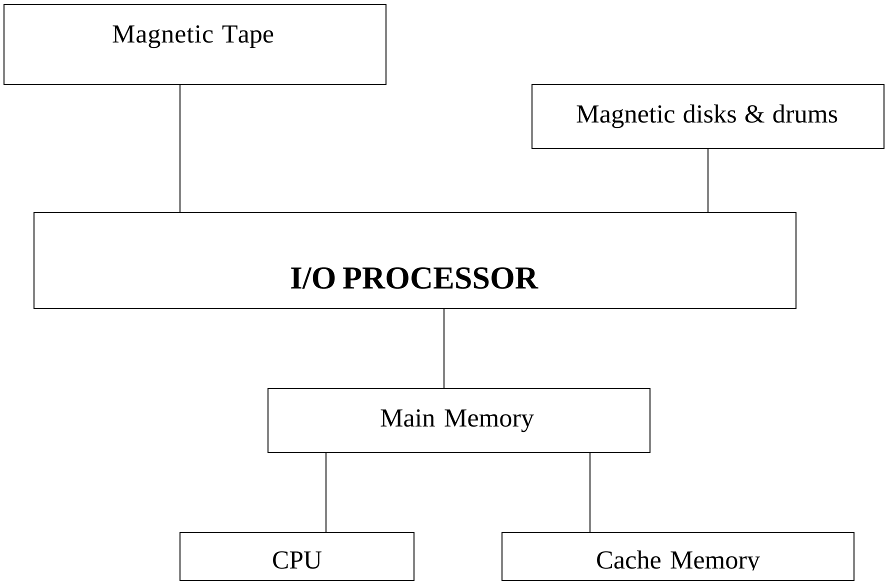


Figure 5.1 Storage hierarchy in a large system

Multiprogramming refers to the existence of many programs in different parts of main memory at the same time. For example, suppose a program is being executed in the CPU and the I/O transfer is required. The CPU initiates the I/O transfer by using I/O processor. This leaves the CPU free to execute another program.

The part of the operating system that supervises the flow of information between all storage devices is called "Memory Management System". The memory management system distributes program and data to various levels in the memory hierarchy. They are

- Batch Mode
- Time sharing mode

In a Batch mode, each user prepares his program off-line and submits it to the computer center. An operator loads all programs into the computer where they are executed. The operator retrieves the printed output and returns it to the user.

In a time-sharing mode, many users communicate with the computer via remote terminals. Because of slow human response compared to computer speeds, the computer can respond to multiple users at, seemingly at the same time.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

A major concept common to both batch and time – sharing modes is their use of Multiprogramming. Processor logic is usually faster than main memory access time with the result that processing speed is mostly limited by the speed of main memory.

A technique used to compensate for the mismatch in operating systems is to employ an extremely fast, small memory between CPU and main memory whose access time is close to processor logic propagation delays. This type of memory is called is called a "buffer" and sometimes a "Cache Memory".

The cache memory is used to store segments of programs currently being executed in the CPU. In a computer system where the demand for service is high, it is customary to run all programs in one of two possible modes. At the bottom of the hierarchy are the relatively slow magnetic tapes used to store removable files. Above if are the magnetic disks or drums used as backup storage. The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary devices through an I/O processor.

When programs not residing in main memory are needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space for currently used programs and data according to their expected frequency of storage. The objective of the memory management system is to adjust the frequency with which the various memories are referenced to provide an efficient method of transfers between levels so as to maximize the utilization of all computer components.

5.3 MAIN MEMORY

The main memory is the central storage unit in a computer system. It is a relatively large and fast memory used to store programs and data during the computer operation. The principal technology used for the main memory is based on semiconductor integrated circuits. Integrated circuit RAM chips are available in two possible operating modes, static and dynamic. The static RAM consists essentially of internal flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit. The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors. The capacitors are provided inside the chip by MOS transistors. The stored charges on the capacitors tend to discharge with time and the capacitors must be periodically recharged by refreshing the dynamic memory. Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge. The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip. The static RAM is easier to use and has shorter read and write cycles.

Most of the main memory in a general-purpose computer is made up of RAM integrated circuit chips, but a portion of the memory may be constructed with ROM chips. Originally, RAM was used to refer to a random-access memory, but now it is used to designate a read/write memory to distinguish it from a read-only memory, although ROM is also random access. RAM is used for storing the bulk of the programs and data that are subject to change. ROM is used for storing programs that are permanently resident in the computer and for tables of constants that do not change in value once the production of the computer is completed.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Among other things, the ROM portion of main memory is needed for storing an initial program called a bootstrap loader. The bootstrap loader is a program whose function is to start the computer software operating when power is turned on. Since RAM is volatile, its contents are destroyed when power is turned off. The contents of ROM remain unchanged after power is turned off and on again. The start up a computer consists of turning the power on and starting the execution of an initial program. Thus when power is turned on, the hardware of the computer sets the program counter to first address of the bootstrap loader. The bootstrap program loads a portion of the operating system from disk to; main memory and control is then transferred to the operating system, which prepares the computer for general use.

RAM and ROM chips are available in a variety of sizes. If the memory needed for the computer is larger than the capacity of one chip, it is necessary to combine a number of chips to form the required memory size. To demonstrate the chip interconnection, we will show an example of a 1024 * 8 memory constructed with 128 * 8 RAM chips and 512 * 8 ROM chips.

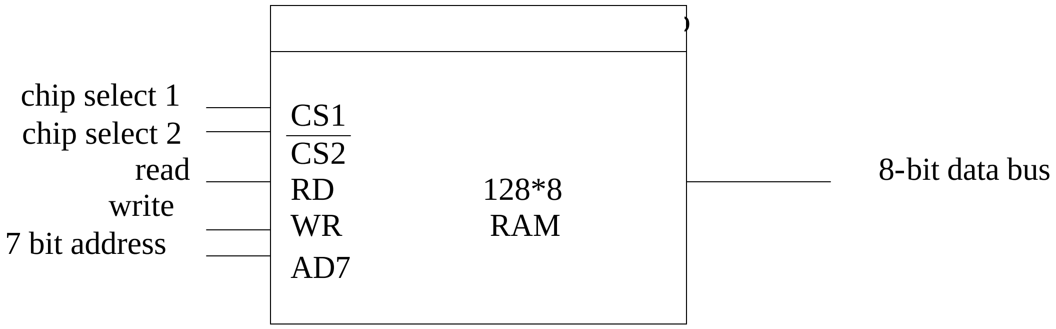
RAM and ROM Chips

A RAM chip is better suited for communication with the CPU if it has one or more control inputs that selected the chip only when needed. Another common feature is a bidirectional data bus that allows the transfer of data either from memory to CPU during a read operation or from CPU to memory during a write operation. A bidirectional bus can be constructed with three-state buffers. A three-state buffer output could be placed in one of three possible states: a signal equivalent to logic 1, a signal equivalent to logic 0, or high impedance state. The logic 1 and 0 are normal digital signals. The high impedance state behaves like an open circuit, which means that the output does not carry a signal and has no logic significance. The block diagram if a RAM chip is shown in the Figure 5.2(a).

The capacity of the memory is 128 words of eight bits per word. This requires a 7-bit address and an 8-bit bidirectional data bus. The read and write inputs specify the memory operation and the two chips select (CS) control inputs are for enabling the chip only when it is select the chip facilitates the decoding of the address lines when multiple chips are used in the microcomputer. The read and write inputs are sometimes combined into one line labeled R/W. When the chip is selected, the two binary states in this line specify the two operations of read or write.

The function table listed in Figure. 5.2(b) specifies the operation of the RAM chip. The unit is in operation only when CS1 = 1 and CS$\overline{2}$ = 0. The bar on top of the second select variable indicates that this input is enabled when it is equal to 0. If the chip select inputs are not enabled, or if they are enabled but the read or write inputs are not enabled, the memory is inhibited and its data bus is in a high-impedance state. When CS1 = 1 and CS2 = 0, the memory can be placed in a write or read mode. When the WR in-put is enabled, the memory stores a byte from the data bus into a location specifies by the address input lines. When the RD input is enabled, the content of the selected byte is placed into the data bus. The RD and WR signals control the memory operation as well as the bus buffers associated with the bidirectional data bus.

This watermark does not appear in the registered version - http://www.clicktoconvert.com



5.2 (a) Block Diagram

CS1	**$\overline{\text{CS2}}$**	**RD**	**WR**	**MEMORY FUNCTION**	**STATE OF DATA BUS**
0	0	X	X	Inhibit	High-Impedance
0	1	X	X	Inhibit	High-Impedance
1	0	0	0	Inhibit	High-Impedance
1	0	0	1	Write	Input data to RAM
1	0	1	X	Read	Output data to RAM
1	1	X	X	Inhibit	High -Impedance

5.2 (b) Function table

A ROM chip is organized externally in a similar manner. However, since a ROM can only read, the data bus can only be in an output mode. The block diagram of a ROM chip is shown in Figure. 5.3. For the same-size chip, it is possible to have more bits of ROM than in RAM, because the internal binary cells in ROM occupy less space than of RAM. For this reason, the diagram specifies a 512-byte ROM, while the RAM has only 128 bytes.

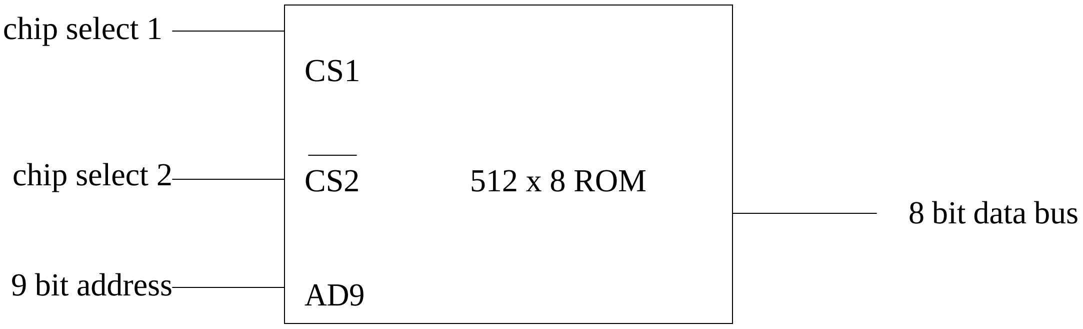


Figure. 5.3 Typical ROM Chip

The nine address lines in the ROM chip spec~~ify~~ any one of the 512 bytes stored in it. The two chip select inputs must be CS1 = 1 and CS2 = 0 for the unit to operate. Otherwise, the data bus is in a high-impedance state. There is no need for a read or write control because the unit can only read. Thus when the chip is enabled by the two select inputs, the byte selected by the address lines appears on the data bus.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Memory Address Map

The designer of a computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM. The interconnection between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available. The addressing of memory can be established by means of a table that specifies the memory address assigned to each chip. The table, called a memory address map, is a pictorial representation of assigned address space for each chip in the system.

To demonstrate with a particular example, assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM. The RAM and ROM chips to be used are specified in Figure.5. 2 and 5. 3. The memory address map for this configuration is shown in Table 5.1. The component column specifies whether a RAM or a ROM chip is used. The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip. The address bus lines are listed in the third column. Although there are 16 lines in the address bus, the table shows only 10 lines because the other 6 are not used in this example and are assumed to be zero. The small x's under the address bus lines designate those lines that must be connected to the address inputs in each chip. The RAM chips have 128 bytes and need seven address lines. The ROM chip has 512 bytes and needs 9 address lines. The x's are always assigned to the low-order bus lines: lines 1 through 7 for the RAM and lines 1 through 9 for the ROM. It is now necessary to distinguish between four RAM chips by assigning to each a different address. For this particular example we choose bus lines 8 and 9 to represent four distinct binary combinations. Note that any other pair of unused bus lines can be chosen for this purpose. The table clearly shows that the nine low-order bus lines constitute a memory space for RAM equal to $2^{9\,=}$ 512 bytes. The distinction between a RAM and ROM address is done with another bus line. Here we choose line 10 for this purpose. When line 10 is 0, the CPU selects a RAM, and when this line is equal to 1, it selects the ROM.

		ADDRESS BUS									
COMPONENT	HEXA DECIMAL ADDRESS	10	9	8	7	6	5	4	3	2	1
RAM 1	0000-007F	0	0	0	X	X	X	X	X	X	X
RAM 2	0080-00FF	0	0	1	X	X	X	X	X	X	X
RAM 3	0100-017F	0	1	0	X	X	X	X	X	X	X
RAM 4	0180-01FF	0	1	1	X	X	X	X	X	X	X
ROM	0200-03FF	1	X	X	X	X	X	X	X	X	X

TABLE 5.1 Memory Address Map for Microcomputer

The equivalent hexadecimal address for each chip is obtained from the information under the address bus assignment. The address bus lines are subdivided into groups of four bits each so that each group can be represented with a hexadecimal digit. The first hexadecimal digit represents lines 9 to 12, but lines 11 and 12 are always 0. The range of hexadecimal addresses for each component is determined from the x's associated with it. These x's represent a binary number that can range from an all-1's value.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Memory Connection to CPU

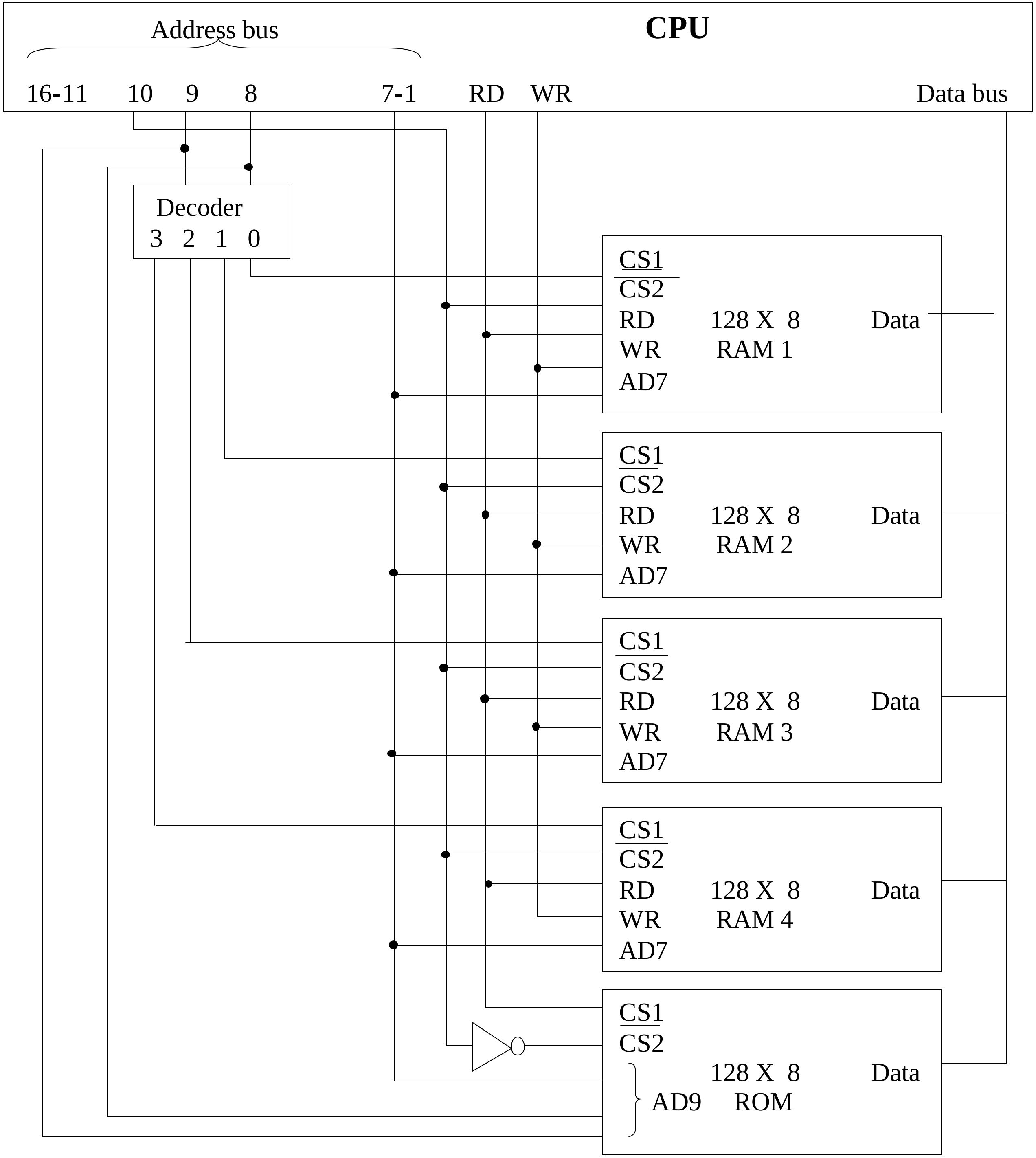
RAM and ROM chips are connected to a CPU through the data and address buses. The low-order lines in the address bus select the byte within the chips and other lines in the address bus select a part9cular chip through its chip select inputs. The connection of memory chips to the CPU is shown in Figure. 5.4. This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM. It implements the memory map of Table 5.4. Each RAM receives the seven low-order bits of the address bus to select one of 128 possible bytes. The particular RAM chip selected is determines from lines 8 and 9 in the address bus. This is done through a 2 * 4 decoder whose outputs go to the CS1 inputs in each RAM chip is selected. When 01, the second RAM chip is selected, and so on. The RD and WR outputs from the microprocessor are applied to the inputs of each RAM chip.

The selection between RAM and ROM is achieved through bus line 10. The RAMs are selected when the bit in this line is 0 and the ROM when the bit is 1. The other chip select input in the ROM is connected to the RD control line for the ROM chip to be enabled only during a read operation. Address bus lines 1 to 9 are applied to the input address of ROM without going through the decoder. This assigns addresses 0 to 511 to RAM and 512 to 1023 to ROM. The data bus of the RAMs can transfer information in both directions.

The example just shown gives an indication of the interconnection complexity that can exist between memory chip and the CPU. The more chips that are connected, the more external decoders are required for selection among the chips. The designer must establish a memory map that assigns addresses to the various chips from which required connections are determined.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Figure 5.4 Memory connections to the CPU



This watermark does not appear in the registered version - http://www.clicktoconvert.com

Self check Exercise: 1

1. Say True or False

(a) The secondary memory is slower than that of main memory but has a larger capacity.
True / False

(b) In Random Access Memory any memory location can be accessed independently.
True / False

2. State the differences between:

a. Compare and Contrast RAM and ROM.

…………………………………………………………………………………………......
…………………………………………………………………………………………......
…………………………………………………………………………………………......
…………………………………………………………………………………………......

5.4 ASSOCIATIVE MEMORY

Many data processing applications require the search of items in a table to be stored in memory. The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address.

A memory unit access by content is called an associative memory or Content Addressable Memory (CAM). This type of memory is accessed simultaneously and in parallel on the basis of data content rather than specific address or location.

When a word is written in an associative memory, no address is given. The memory is capable of finding an empty unused location to store the word. When a word is to be read from an associative memory, the content of the word or part of the word is specified.

The memory locates all words, which match the specified content, and marks them for reading. Because of its organization, the associative memory is uniquely suited to do parallel searches by data association. Moreover, searches can be done on an entire word or on a specific field within a word. Associative memories are used in applications where the search time is very critical and must be very short.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

5.4.1 Hardware Organization

The block diagram of an associative memory is shown in the following Figure. 5.5.

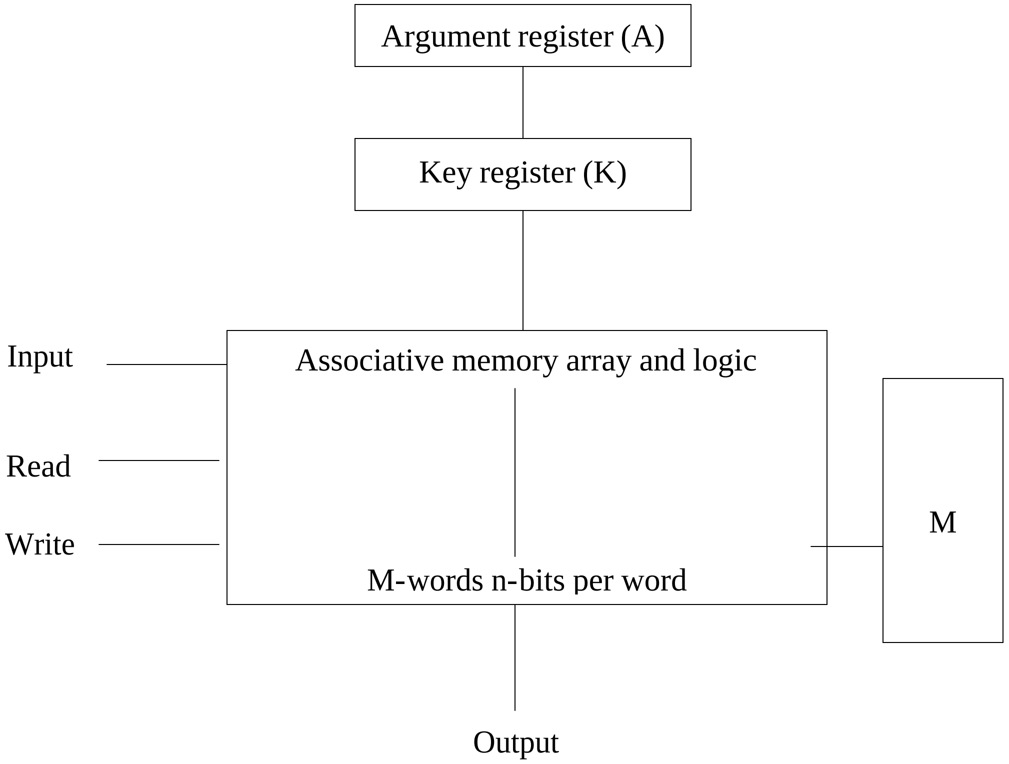


Figure. 5.5 Block Diagram of Associative Memory

The associative memory consists of a memory array and logic for m words with n bits per word. The argument register A and key register K each have n bits, one for each have n bits, one for each bit of a word. The argument register A and key register K each have n bits, one for each bit of a word. The match register M has m bits, one for each memory word.

Each word in memory is compared in parallel with the content of the argument register. The word that matches the bits of the argument register set a corresponding bit in the match register. Reading is accomplished by a sequential process or access to memory for those words whose corresponding bits in the match register have been set. The key register provides a mask for choosing a particular field or key in the argument word.

The entire argument is compared with each memory word if the key register contains all 1's. Otherwise, only those bits in the argument that have 1's in their corresponding position of the key register are compared.

Example

A	101	111100	
K	111	000000	
Word1	100	111100	=> no match
Word2	101	000001	=> match

This watermark does not appear in the registered version - http://www.clicktoconvert.com

Word2 matches the unmasked argument field because the three left-most bits of the argument and the word are equal. The cells in the array are marked by the letter C with two subscripts. The first subscript gives the word number and second specifies the bit position of the logic in the word.

Figure 5.6 Associative memory of an m word, n cells – per word

The relation between the memory array and external registers in an associative memory is shown in Figure. 5.6

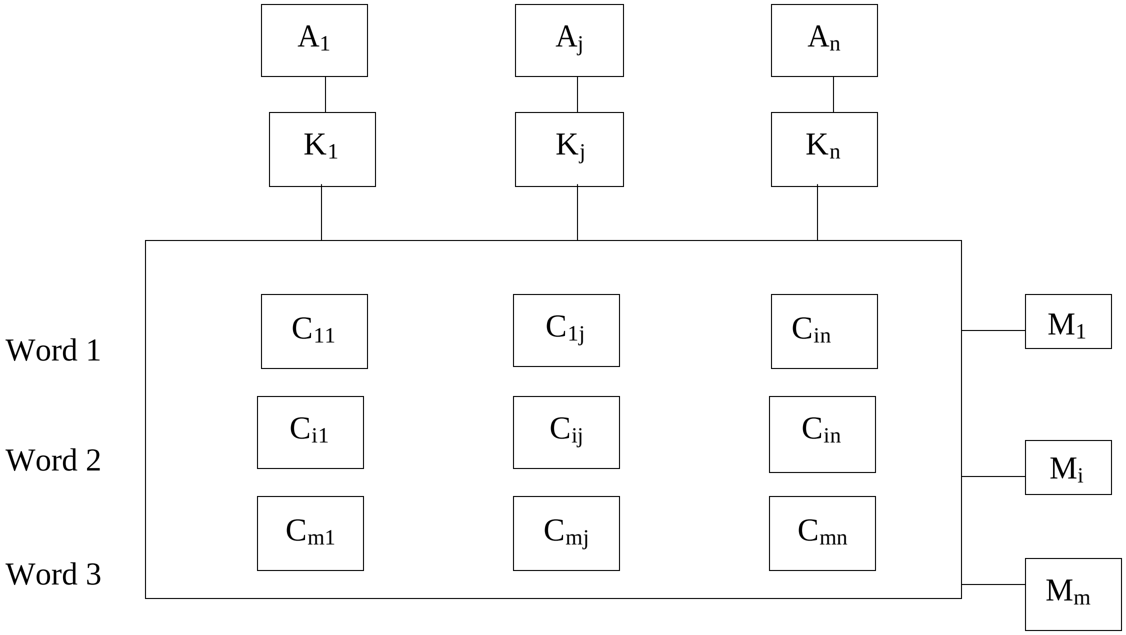


Figure 5.6 Associative memory of an m word, n cells – per word

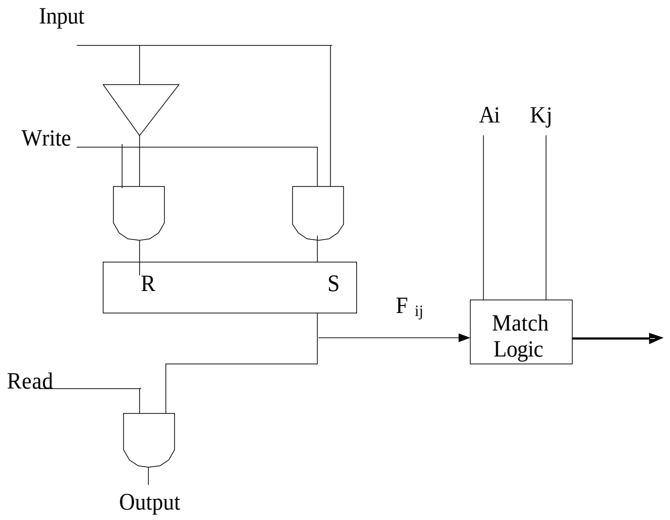


Figure 5.7 One Sell of Associative Memory.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The Internal organization of a typical cell C_{ij} is shown in Figure. 5.7. Each cell consists of a flip-flop storage element F_{ij} and the circuits for reading, writing and matching the cell. The Input bit is transferred into the storage cell during a write operation. The bit stored is read out during a read operation.

5.4.2 Match Logic

The match logic for each word can be derived from the comparison algorithm of two binary numbers. First, we neglect the K bits and compare the argument in A with the bits stored in the cells of the words.

Word I is equal to the argument in A if

$A_j = F_{ij}$ for j=1,2…n

Two bits are equal if they are both 1 and 0

The equality of two bits can be expressed logically by the Boolean function

$$X_j = A_j F_{ij} + A_j' F_{ij}'$$

For a word I to be equal to the argument in A we must have all X_j variables equal to 1. This is the condition for setting the corresponding match bit M_i to 1. The function for this condition is $M_i = x_1 x_2 x_3 \ldots x_n$ and constitute the AND operation of all pairs of matched bits.

We now include the key bit K_j in the comparison logic. The requirement is that if K_j=0, the corresponding bits of A_j and need no comparison. Only when K_j=1 must be compared. This requirement is achieved by OR ing each term with K_j. Thus

$$X_j + K_j' = \begin{cases} x_j \text{ if } k_j = 1 \\ 1 \text{ if } k_j = 0 \end{cases}$$

When K_j =1, we have K_j'=0 and $x_j + 0 = x_j$.

When K_j=0, then k_j '=1 and x_j+1=1

The match logic for word I in an associative memory can now be expressed by the following Boolean function.

$$M_i = (x_1 + k_1')\,(x_2 + k_2')\,(x_3 + k_3') \ldots (x_n + k_n')$$

If we substitute the original definition of x_j, the above Boolean function can be expressed as follow

$$M_i = \mathop{}_{j=1}^{n} (A_j F_{ij} + A_j' F_{ij}' + k_j')$$

Where   is a product symbol designating the AND operation of all n terms.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

5.4.3 Read Operation

If more than one word in memory matches the unmasked argument field, all the matched words will have 1's in the corresponding bit position of the match register.

It is then necessary to scan the bits of the match register one at a time. The matched words are read in sequence by applying a read signal to each word line whose corresponding M_i bit is a 1.

If only one word may match the unmasked argument field, then connect output Mi directly to the read line in the same word position, the content of the matched word will be presented automatically at the output lines and no special read command signal is needed. Furthermore, if we exclude words having zero content, then all zero output will indicate that no match occurred and that the searched item is not available in memory.

5.4.4 Write Operation

If the entire memory is loaded with new information at once, then the writing can be done by addressing each location in sequence. The information is loaded prior to a search operation. If unwanted words have to be deleted and new words inserted one at a time, there is a need for a special register to distinguish between active an inactive words. This register is called "Tag Register".

A word is deleted from memory by clearing its tag bit to 0.

5.5 Cache Memory

The references to memory at any given interval of time tent to be contained within a few localized areas in memory. This is called "Locality of reference". If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a fast small memory is referred to as "Cache Memory".

The performance of the cache memory is measured in terms of a quality called "Hit Ratio". When the CPU refers to memory and finds the word in cache, it produces a hit. If the word is not found in cache, it counts it as a miss.

The ratio of the number of hits divided by the total CPU references to memory (hits + misses) is the hit ratio. The hit ratios of 0.9 and higher have been reported. The average memory access time of a computer system can be improved considerably by use of cache.

It is placed between the CPU and main memory. It is the faster component in the hierarchy and approaches the speed of CPU components. Most frequently accessed instruction and data are kept in the fast cache memory. Although the cache is only a small fraction of the size of main memory, a large fraction of memory requests will be found due to locality of reference.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

When the CPU needs to access memory, the cache is examined. If it is found in the cache, it is read very quickly. If it is not found in the cache, the main memory is accessed. A block of words containing the one just accessed is then transferred from main memory to cache memory.

For example,

A computer with cache access time of 100ns, a main memory access time of 1000ns and a hit of 0.9 produce an average access time of 200ns. This is a considerable improvement over a similar computer without a cache memory, whose access time is 1000ns.

The basic characteristic of cache memory is its fast access time. Therefore, very little or no time must be wasted when searching for words in the cache. The transformation of data from main memory to cache memory is referred to as a “Mapping Process”. There are three types of mapping procedures are available.

- Associative Mapping
- Direct Mapping
- Self – Associative Mapping.

Consider the following memory organization Figure.5.8 to show mapping procedures of the cache memory.

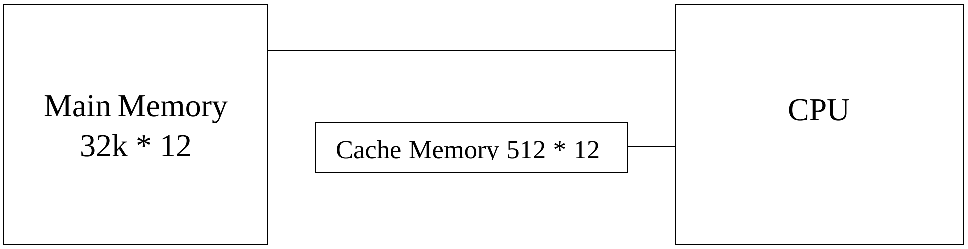


Figure 5.8 Example of Cache Memory

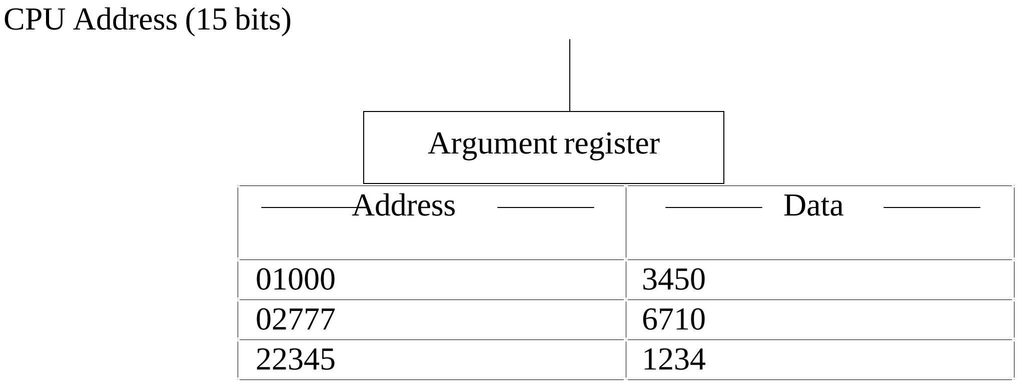
- The main memory can stores 32k word of 12 bits each.
- The cache is capable of storing 512 of these words at any given time.
- For every word stored in cache, there is a duplicate copy in main memory.
- The CPU communicates with both memories
- If 1st sends a 15 – bit address to cache.
- If there is a hit, the CPU accepts the 12 bit data from cache
- If there is a miss, the CPU reads the word from main memory and the word is then transferred to cache.

5.5.1 Associative Mapping

The associative mapping stores both the address and content (data) of the memory word. The diagram, given below shows 3 words presently stored in cache. The address value of 15 bits it’s shown as a 5 digit octal number and its corresponding 12 bits is shown as 4 digit octal number

This watermark does not appear in the registered version - http://www.clicktoconvert.com

A CPU address of 15 bits is placed in the argument register and associative memory is searched for a matching address. If the address is found, the corresponding 12 bit data is read and sent to the CPU.



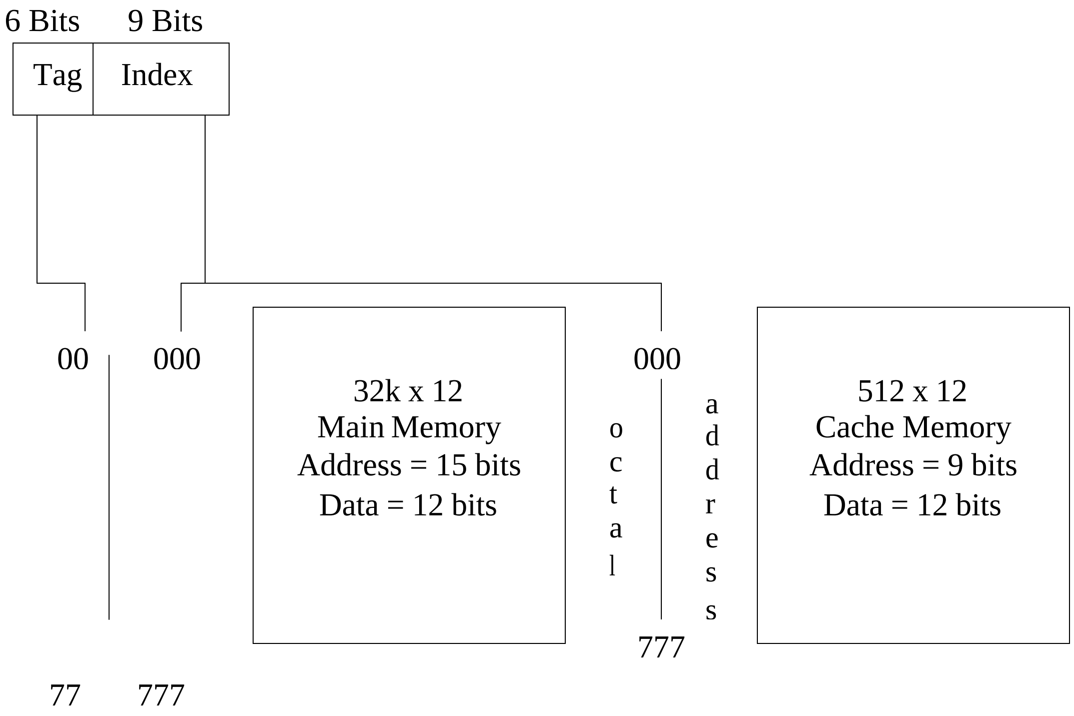
Address	Data
01000	3450
02777	6710
22345	1234

Figure. 5.9 Associative Mapping Cache (all numbers in octal)

If no match occurs, the main memory is accessed for the word. The address – data pair is then transferred to associative cache memory. If the cache is full, it must be displayed, using replacement algorithm. FIFO may be used.

5.5.2 Direct Mapping

Here, using RAM for the cache is presented in the following Figure 5.10.



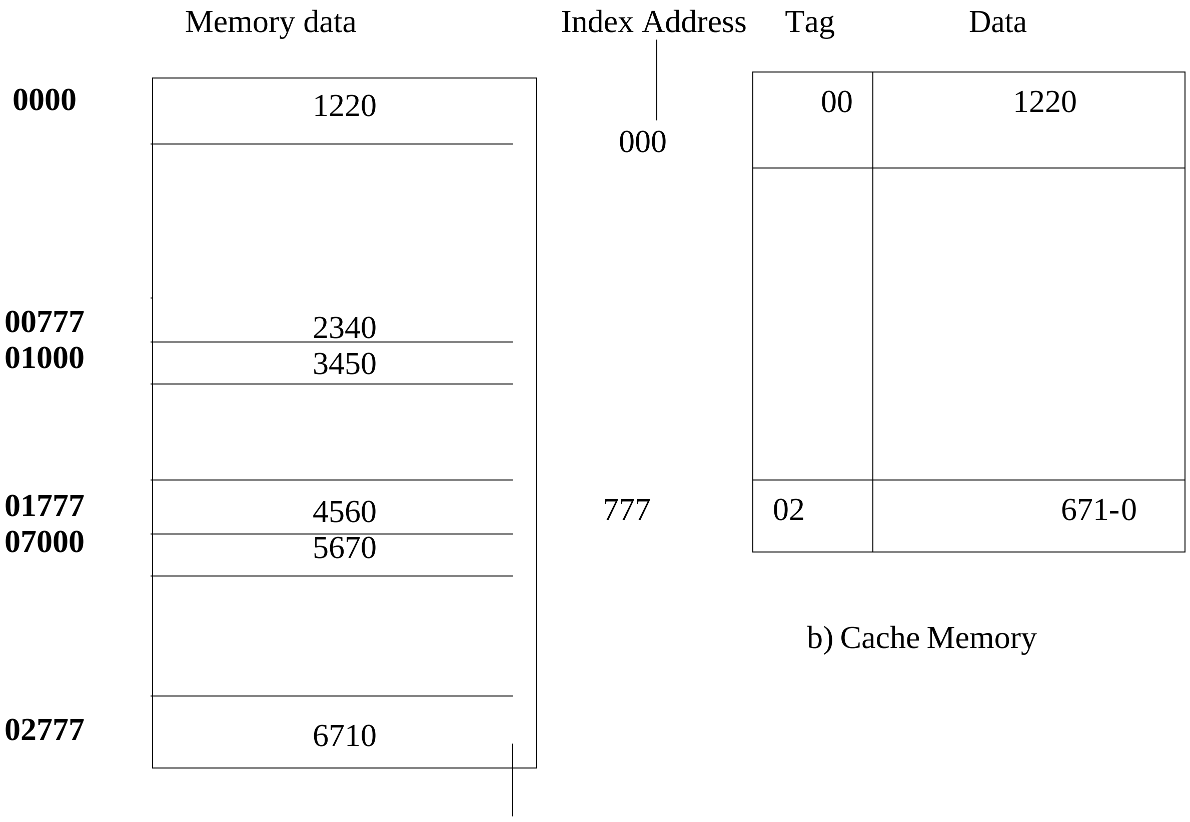
5.10 Addressing relationships between main and cache memories

This watermark does not appear in the registered version - http://www.clicktoconvert.com

The 15-bit CPU address is divided into two fields. The 9 least significant bits constitute the index field and the remaining 6 bits form the tag fields. The main memory needs an address but includes both the tag and the index bits. The cache memory requires the index bit only i.e., 9. There are 2^k words in the cache memory & 2^n words in the main memory.

Eg: $k = 9, n = 15$

The internal organization of the words in the cache is shown in the Figure. 5.11**.** Each word in cache consists of the data word and it associated tag. When a new word is brought into cache, the tag bits store along data. When the CPU generates a memory request, the index field is used in the address to access the cache. The tag field of the CPU address is equal to tag in the word from cache; there is a hit, otherwise miss.



b) Cache Memory

a) Main Memory

Figure 5.11 Direct mapping Cache Organization

The word at address zero is presently stored in the cache (index = 000, tag = 00, data = 1220). The CPU now wants to access the word at address 0200. The index address is 000. So it is used to access the cache. The two tags are then compared.

The cache tag is 00 but the address tag is 02, which does not produce match. Therefore the main memory is accessed and the data word 5670 is transferred to CPU. The cache word at

This watermark does not appear in the registered version - http://www.clicktoconvert.com

index address 000 is then replaced with a tag of 02 and data of 5670. The direct mapping uses a block size of 1 word or 8 words.

5.5.3 Set – Associative Mapping

In set – Associative mapping, each word of cache can store two or more words of memory under the same index address. Each data word is stored together with its tag and the number of tag – data items in one word of cache is said to form a set.

Index	Tag	Data		Tag	Data
000	01	3450		02	5670
777	02	6710		00	2340

Figure 5.12 Self – associative mapping cache with set size of two

Each index address refers to two data words and their associated tags. Each tag requires 6 bits & each data word has 12 bits, so the word length is 2(6+12) =36 bits. An index address of 9 bits can accommodate 512 words. It can accommodate 1024 words. When the CPU generates a memory request, the index value of the address is used to access the cache.
The tag field of the CPU address is compared with both tags in the cache.

The most common replacement algorithms are

- Random replacement
- FIFO
- Least Recently Used (LRU)

5.5.4 Writing into cache

An important aspect of cache organization is concerned with memory write requests. When the CPU finds a word in cache during a read operation, the main memory is not involved in the transfer. However, if the operation is a write, there are two ways that the system can proceed.

The simplest and most commonly used procedure is to update main memory with every memory write operation, with cache memory being update in parallel if it contains the word at the specified address. This is called the “Write-through method”.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

This method has the advantage that main memory always contains the same data as the cache. The second procedure is called the "write-back method". In this method only the cache location is updated during a write operation. The location is then marked by a flag so that later when the word is removed from the cache it is copied into main memory.

The reason for the write-back method is that during the time a word resides in the cache, it may be updated several times. Analytical results indicate that the number of memory writes in a typical program ranges between 10 and 30 % of the total references to memory.

Self-Check Exercise 2

1. The direct mapping of main memory to cache is achieved by mapping a block of main memory to any slot of cache.

True / False

2. A replacement algorithm is needed only for associative and set associative mapping of cache.

True / False

3. Write policy is not needed for instruction cache.

True / False

5.6 VIRTUAL MEMORY

In a memory hierarchy system, programs and data are first stored in auxiliary memory. Portions of program or data are brought into main memory, as they are needed by the CPU.

Virtual memory is a concept used in some large computer systems that permit the user to construct his programs as though he had a large memory space, equal to the totality of auxiliary memory. Each address, which is referenced by the CPU, goes through an address mapping from FC:\WINDOWS\hinhem.scrthe so- called virtual address to an actual address in main memory.

A virtual memory system provides a mechanism for translating program generated address into correct main memory locations. This is done dynamically, while programs are being executed in the CPU.

The translation or mapping is handled automatically by the hardware by means of a mapping table.

5.6.1 Address space and Memory space

An address used by a programmer will be called a virtual address and the set of such address is the "address space". An address in main memory is called a "location" or "physical address". The set of such locations is called the "memory space". Thus, the address space is the set of address generated by programs as they reference instructions and data. The most computers the address and memory spaces are identical. The address space is allowed to be larger than the memory space in computer with virtual memory. In a multiprogramming computer system, programs and data are transferred to and from auxiliary memory and main memory based on demands on the CPU.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

In the Figure. 5.13, program 1 is currently being executed in the CPU. Program 1 and a portion of its associated a data are moved from auxiliary memory into main memory. In a virtual memory system, the programmer is told that he has the total address space at his disposal. Moreover, the address field of an instruction code will consist of 20 bits but physical memory addresses must be specified with only 15 bits.

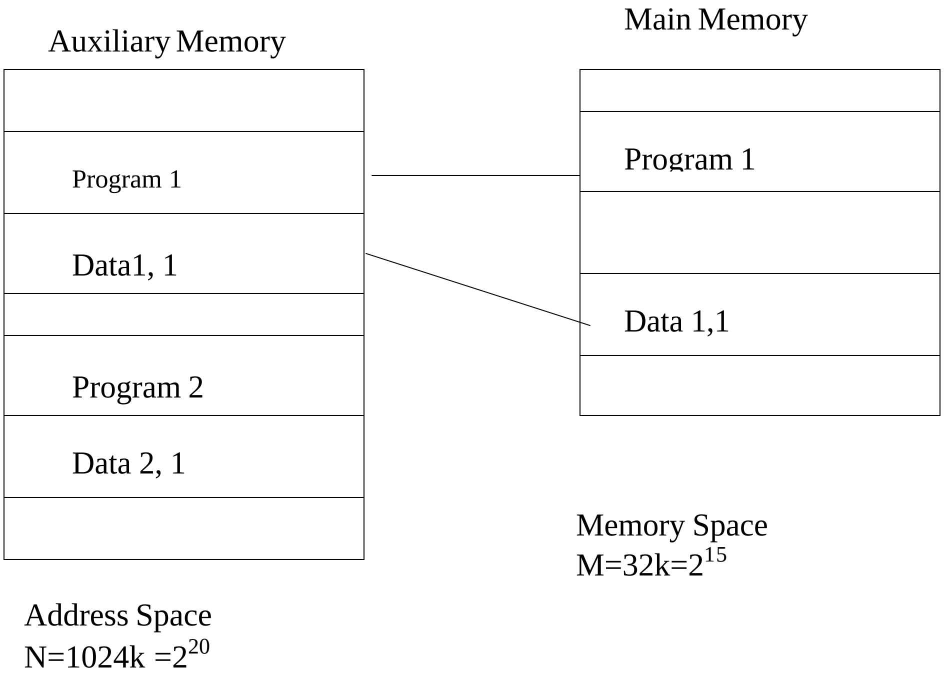


Figure 5.13 Relation between address and memory space in a virtual memory system

A table is needed to map a virtual address of 20 bits to a physical address of 15 bits. The mapping table may be stored in a separate memory or in main memory.

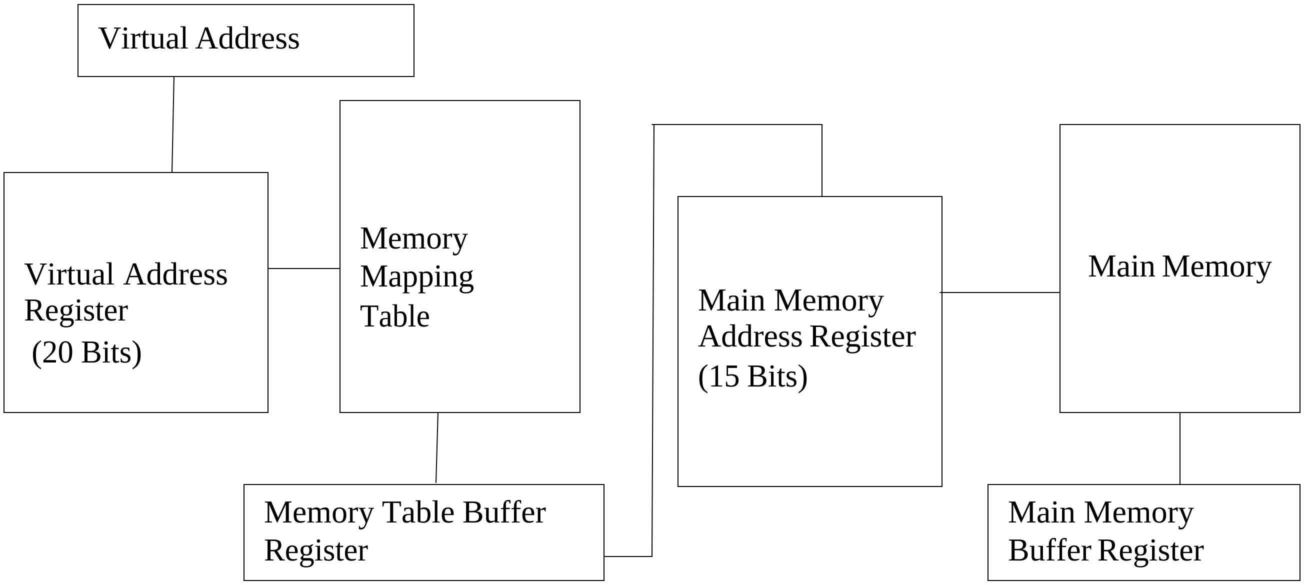

Virtual Address
Virtual Address Register (20 Bits)
Memory Mapping Table
Memory Table Buffer Register
Main Memory Address Register (15 Bits)
Main Memory
Main Memory Buffer Register

Figure 5.14 Memory table for mapping a virtual address

This watermark does not appear in the registered version - http://www.clicktoconvert.com

5.6.2 Address Mapping using Pages

Introduction in the address space and the memory space are each divided into groups of equal size. The group in the physical memory called blocks, 64 to 4096 words. The group in the auxiliary memory is called pages.

For example,

Address space is divided into 1024 pages and main memory is divided into 32 blocks. The program is divided into pages. Positions of programs are moved from auxiliary memory to main memory. The terms page refers to group of address space of the same size.

Consider a computer with an address space of 8k and a memory space of 4k. If we split each into groups of 1k words we obtain 8 pages and 4 blocks.

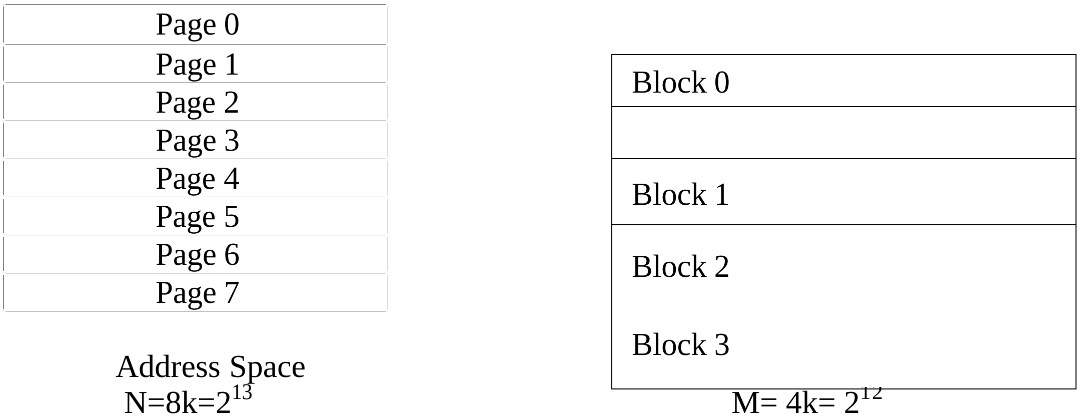


Figure 5.15 Address space and memory space split into groups of 1k words

At any time, 4 pages may reside in main memory. Each virtual address is represented by two numbers. A page number address and a line within the page. In a computer with 2^p words per page, p bits are used to specify a line address and the remaining bits of the virtual address specify the page number. A virtual address has 13 bits. Since each page consists of 2^{10} = 1024 words, the high order 3 bits of a virtual address will specify one of the 8 pages and the low order 10 bits give the line address within the page.

5.6.3 Associative memory page table

A random access memory –page table is inefficient with respect to storage utilization. In the example of Figure.5.16 we observe that eight words of memory are needed; one for each page, but at least four words will always be marked empty because main memory cannot accommodate more than four blocks. In general a system with n pages and m blocks would require a memory- page table of n locations of which up to n blocks will be marked with block.(In general a system with n pages and m blocks would require a memory page table of) numbers and all others will be empty. As a second numerical examples space of 32k words. In each page or block contains 1k words than the number of pages is 1024 and the number of blocks 32. the capacity of the memory page table must be 1024 words and only 32 locations may have a presence bit equal to 1 . at any given time, at least 992 locations will be empty and not in use.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

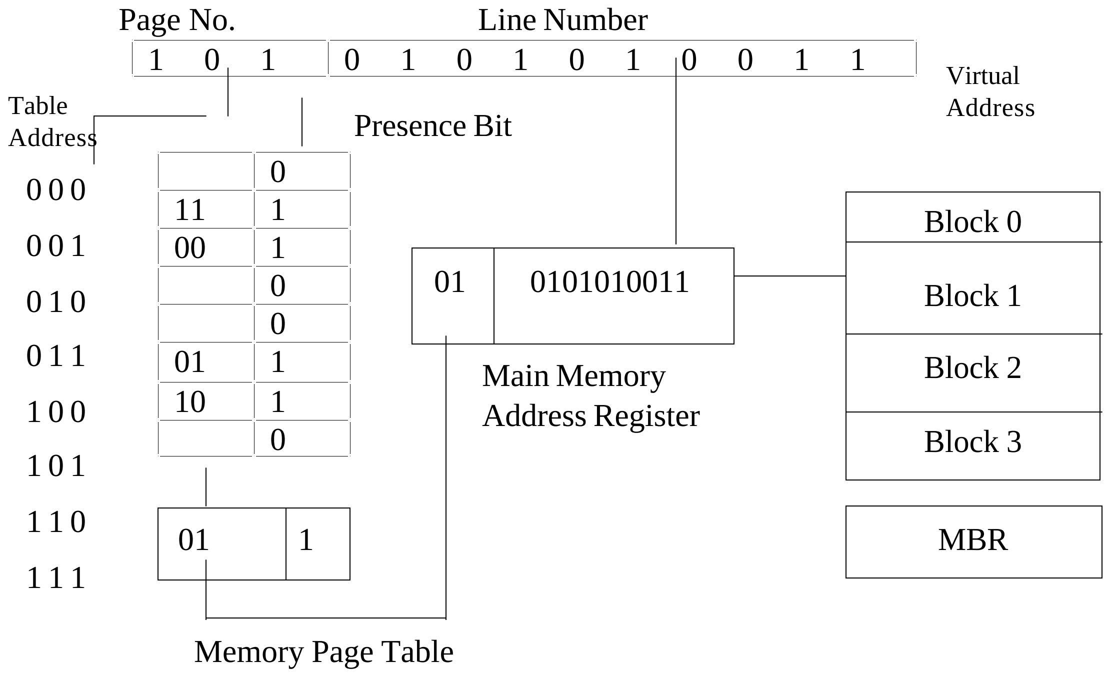


Figure.5.16 Memory Table in a paged memory

A more efficient way to organize the page table would be to construct it with a number of words equal to the number of blocks in main memory. In this way, the size of the memory is reduced and each location is fully utilized. This method can be implemented by means of an associative memory with each word in memory containing a page number together with its corresponding block numbers. The page field in each word is compared with the page number in the virtual address. If a match occurs the words is read from memory and its corresponding block number is extracted.

Consider again the case pf eight pages and four blocks as in the example of Figure.5.17 we replace the random access memory-page table with a associative memory of four words as shown in the Figure (an associative memory page table). Each entry in the associative memory array consists of two fields. The first three bits specify a field for storing the page number. The last two bits constitute a field for storing the block numbers. The virtual address is placed in the argument registers. The page number bits in the argument register are compared with all page numbers in the page field of the associative memory. In the page number id found the 5-bits word is read-out from memory. The corresponding block number, being in the same words is transferred to the main memory address register. If no match occurs a call to the operating systems id generated to bring the required page from auxiliary memory.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

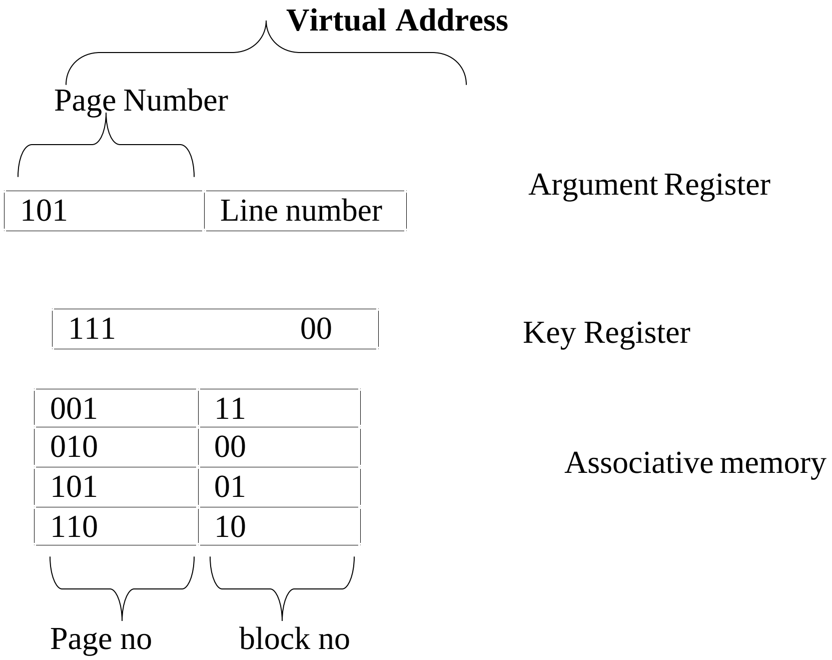


Figure. 5.17 An Associative Memory Page Table

5.6.4 Page replacement

A virtual memory system is a combination of hardware and software techniques. The memory management software system handles all the software operations for the efficient utilization of memory. It must decide

(1) Which page in memory ought to be removed to make room for a new page
(2) When a new page is to be transferred from auxiliary memory and
(3) Where the page is to be placed in the main memory. The hardware mapping mechanism and the memory management software together constitute the architectural of a virtual memory.

When a program starts execution one or more pages are transferred into main memory and the page table is set to indicate checks position. The program is executed from main memory until if attempt to reference a page that is still in auxiliary memory. This condition is called page fault when page fault. When page fault occurs, the execution of the present program is suspended until the required page is brought into main memory. Since loading a page from auxiliary memory to main memory is basically on I/O operation, the operating system assigns this task to the I/O processor. In the mean time control is transferred to the program in memory that is waiting to be processed in the CPU. Later when the memory block has been assigned and the transfer completed the original program can resume its operation.

When a page fault occurs in a virtual memory system it signifies that the page referenced by the CPU is not in main memory. A new page is then transferred from auxiliary memory to main memory. If the main memory is full, it would be necessary to remove a page from a

This watermark does not appear in the registered version - http://www.clicktoconvert.com

memory block to make room for the new page. The policy for the choosing pages to remove is determined from the replacement algorithm that is used. The goal of a replacement policy is to try to remove the page least likely to be referenced in the immediate future.

Two of the most common development algorithms used are the first in first out (FIFO) and the least recently used (LRU). The FIFO algorithms select for replacement the page that has been in the memory the longest time. Each time a page is loaded into memory, its identification number is pushed into a FIFO will be full whenever memory has no more empty blocks. When a new page must be loaded, the page least recently brought in is removed. The page to be removed is easily determined because its identification number is at the top of the FIFO stack. The FIFO replacement policy has the advantage of being easy to employment. It has the disadvantage that under certain circumstances pages are removed and loaded from memory too frequently.

The LRU policy is more difficult to implement but has been more attractive on the assumption that the least recently used page is better candidate for removal than the least recently loaded page as in FIFO. The LRU algorithm can be implemented by associating a counter with every page that is in main memory. When a page is referenced, its associated counted is set to zero. At fixed intervals of time, the counters associated with all pages presently in memory are incremented by 1. The least recently used page is the page with the highest count. The counters are often called aging registers as their count indicates their age, that is, how long ago their associated pages have been referenced.

Self-Check Exercise: 3

1. What is the use of Virtual Memory?

…………………………………………………………………………………………………
…………………………………………………………………………………………………
………………………………………………………………………………………..

2. List the various Page Replacement Algorithms.

…………………………………………………………………………………………………
…………………………………………………………………………………………………
………………………………………………………………………………………..

5.7 LETS US SUM UP

Thus we have taken a complete view of the memory system of computer system along with the various technologies. The unit has outlined the importance of the memory system, the memory hierarchy, the main memory and its technologies.

5.8 LESSON – END ACTIVITIES

1. Explain about memory hierarchy in detail.
2. Explain the reed and write operations of Association Memory.
3. Explain about need for Secondary memory.

This watermark does not appear in the registered version - http://www.clicktoconvert.com

5.9 POINT FOR DISCUSSION

(1) Compare the different types of memory and identify the limitation and advantages of each memory organization.

5.10 CHECK YOUR PROGRESS: MODEL ANSWERS

Self-Check Exercise 1

1. (a) True
   (b) True

2.

RAM	ROM
Read Write Memory	Read Only Memory
Semi conductor memories	Semi conductor memories
Volatile	Non volatile
Can be used by user program and system programs	Can be used for micro organisms

Self Check Exercise : 2

1. False
2. True
3. True

Self Check Exericse: 3

1. A virtual memory system provides a mechanism for translating program generated address into correct main memory locations. This is done dynamically, while programs are being executed in the CPU.

2. FIFO, LRU,RANDOM Page Replacement.

5.10 REFERENCES

1. Computer system Architecture, M. Morris Mano, Pearson Education Publication, Third Edition.