SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – I – SBSA1303 – COMPUTER ARCHITECTURE

1.1 NUMBER SYSTEM

Number systems are the technique to represent numbers in the computer system architecture. Whatever value we are store and getting from computer memory has defined in the number system. Computers can understand the following types of number.

1. Decimal Number - base 10
2. Binary Number – Base 2
3. Octal Number - Base 8
4. Hexadecimal Number – base 16
5. Decimal Number System

Decimal number are from 0 to 9, binary number are 0's and 1's. Octal number system starts from 0 to 7. Hexadecimal Number System starts from 0 to 15. In this hexadecimal system from 10 to 15 represented as A to F.

Decimal Number System

Decimal number system is base 10 number system. The digits from 0 to 9.

Decimal to Binary Conversion

Steps to follow for conversion:

- Divide each digit by 2, keep track of the reminder
- Find the reminder reaches 0 LSB(Least Significant Bit)
- E.g. $10_{10} = 1010_2$

Decimal to Octal Conversion

Steps to follow for conversion:

-Divide each digit by 8, keep track of the reminder

- Find the reminder reaches 0 LSB(Least Significant Bit)
- E.g. $10_{10} = 12_8$

Decimal to Hexadecimal Conversion

Steps to follow for conversion:

-Divide each digit by 16, keep track of the reminder

- Find the reminder reaches 0 LSB(Least Significant Bit)
- E.g. $20_{10} = A4_{16}$

Binary Number System

Digital Computers represent all kinds of data and information in form the binary number system.

Binary Number System consists of two digits 0 and 1. Base is 2.

Binary to Decimal Conversion

Steps to follow for conversion:

- Multiply each digit by 2^n, where n is "weight" of the bit.
- N is the position of the bit. Starting from 0.
- Add the result.
- E.g $1010_2 = 10_{10}$

Binary to Octal Conversion

Steps to follow for conversion:

- Group binary digits in a 3 bits from on right side (Most Significant Bit)
- Convert to octal digits.
- E.g $101011100_2 = 534_8$

Binary to Hexadecimal Conversion

Steps to follow for conversion:

- Group binary digits in a 4 bits from on right side (Most Significant Bit)
- Convert to hexadecimal digits.
- E.g $101011100_2 = 15C_{16}$

Octal Number System

Octal Number system is the base 8 number system. Starting from 0 to 7. The number after 7 is 10.

Octal to Decimal Conversion

Steps to follow for conversion:

-Multiply each digit by 8^n, where n is "weight" of the bit.
- N is the position of the bit. Starting from 0.
- Add the result.
- E.g $534_8 = 348_{10}$

Octal to Binary Conversion

Steps to follow for conversion:

- Group binary digits in a 3 bits from on right side (Most Significant Bit)
- Convert to octal digits.
- E.g $534_8 = 101011100_2$

Octal to Hexadecimal Conversion

Steps to follow for conversion:

- Convert the octal to binary.
- Group binary digits in a 4 bits from on right side (Most Significant Bit)
- Convert to octal digits.
- E.g $534_8 = 15C_{16}$

Hexadecimal Number System

Octal Number system is the base 16 number system. Starting from 0 to 9. The number is 10 - 15 represents as A - F.

Hexadecimal to Decimal Conversion

Steps to follow for conversion:

-Multiply each digit by 16^n , where n is "weight" of the bit.

- N is the position of the bit. Starting from 0.
- Add the result.
- E.g $15C_{16} = 348_{10}$

Hexadecimal to Binary Conversion

Steps to follow for conversion:

- Group binary digits in a 4 bits from on right side (Most significant Bit)
- Convert to octal digits.
- E.g $15C_{16} = 101011100_2$

Hexadecimal to Octal Conversion

Steps to follow for conversion:

- Group binary digits in a 4 bits from on right side (Most Significant Bit)
- Convert to octal digits.
- E.g $15C_{16} = 534_8$

2. COMPLEMENTS

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. There are two types of complements, i.e.

a) r's-complement
b) (r – 1)' s-complement

1's-Complement

To find 1's-complement of a number replace all 0's with 1's and all 1's with 0's. The 1's complement of a number is always 1 less than the 2's-complement of a number.

E.g. 1. 1's-complement of 1011010.

- Replacing all 1's with 0's and all 0's with 1's.
- The 1's-complement of 1011010 is 0100101.

E.g. 2. 1's-complement of 0.0101

- Replace all 1's with 0's and all 0's with 1's.
- The 1's-complement of 0.0101 is 0.1010.

E.g.3. Find the subtraction $(1110101 - 1001101)_2$ using the 2's-complement method.

Minuend = 1110101

Subtrahend = 1001101

Minuend = 1110101

1's Complement of Subtrahend = 0110010

2's-complement of subtrahend = 0110010 +1= 0110011

1110101 + 0110011 = 1 0101000

Here, an end carry occurs, hence discard it. The result of $(1110101 - 1001101)_2$ is $(0101000)_2$.

3. SIGNED MAGNITUDE OF BINARY NUMBERS

A signed binary number consists of a sign, either positive or negative and magnitude. In a signed magnitude representation of binary numbers, the most significant digit is zero for the representation of positive binary number and one for the representation of negative binary numbers. This most significant digits (0 or 1) represents whether the number is positive or negative and the magnitude is the value of the numbers**.**

3.1. DECIMAL SIGNED NUMBERS

Decimal values of the positive and negative signed magnitude numbers can be determined by the summation of the weights of all the magnitude bits, where there are 1's and ignoring all other bits, where there are zeros (0).

E.g. Express the decimal equivalent of signed binary number 10011100 expressed in its sign magnitude form.

Solution: There are seven magnitude bits and one sign bit. Separating sign bits and magnitude bits sign bit = 1, which means that the magnitude of the number is negative.

Magnitude bits = 0011100, assigning the weights to the bits, we get

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

0 0 1 1 1 0 0

Summing the weights together where 1 exists and ignoring where 0 exists, we get,

$2^4 + 2^3 + 2^2 = 16 + 8 + 4 = 28.$

Adding sign magnitude bit to the solution for the signed magnitude binary number (1 0011100) is (–28).

3.2 . BINARY CODE

The digits 1 and 0 used in binary reflect the on and off states of a transistor. Each instruction is translated into machine code - simple binary codes that activate the CPU.

Programmers write computer code and this is converted by a translator into binary instructions that the processor can execute.

Different Types of Binary Code:

1. Binary Coded Decimal or 8421 Code
2. 2421 Code
3. 5211 Code
4. Gray Code (Reflected Code)
5. Error - Detection Code

Convert to BCD to Excess-3

0000 + 0011 = 0011

01 0011 = 0100

Convert Binary to Gray Code

BC GC

0011 0010

Most Significant Bit

Keep MSB , like 0 means 0

Both are same, like 0 and 0 means 0

like 1 and 1 means 0

Difference, like 0 and 1 means 1

Binary codes for the decimal digits

Decimal digit	(BCD) 8421	Excess-3	84-2-1	2421	(Biquinary) 5043210
0	0000	0011	0000	0000	0100001
1	0001	0100	0111	0001	0100010
2	0010	0101	0110	0010	0100100
3	0011	0110	0101	0011	0101000
4	0100	0111	0100	0100	0110000
5	0101	1000	1011	1011	1000001
6	0110	1001	1010	1100	1000010
7	0111	1010	1001	1101	1000100
8	1000	1011	1000	1110	1001000
9	1001	1100	1111	1111	1010000

Fig.1.1 Binary codes for the decimal numbers

4. Error Correction Code

Error Correction codes detect the error, if it is occurred during transmission of the original data bit stream.

E,g. Parity code, Hamming code.

Error correction codes − are used to correct the errors present in the received data bit stream so that, we will get the original data.

4.1. Parity Code

One of the common way to detect the error is Parity bit. A parity bit is extra bit included with a message to make the total number of one's transmitted either odd or Even.

Two Types in the Parity Code:

1. Odd Parity
2. Even Parity

If an odd parity selected, the total number of 1's in the message bit and parity bit is odd , the P bit.

Parity bit

Odd parity		Even parity	
Message	P	Message	P
0000	1	0000	0
0001	0	0001	1
0010	0	0010	1

Fig.1.2 . Parity bit

5. BINARY LOGIC

Binary Logic Deals with binary that on two discrete values and with operations that assume logical meaning. The two variables take may be called by different names.

e.g. True or false , yes or no

There are basic three logical operations : AND, OR and NOT

AND = X.Y

OR = X+ Y

NOT = X′

Truth Tables of Logical Operations

AND			OR			NOT	
x	y	x y	x	y	x + y	x	x′
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Fig 1.3. Truth Tables of Logical operations

5.1 LOGIC GATES- TRUTH TABLE

AND GATE

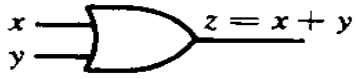
(a) Two-input AND gate

Fig. 1.4. Logical Diagram of AND gate

2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 1.5. Truth Table of AND gate

OR GATE



(b) Two-input OR gate

Fig.1.6. Logical Diagram of the OR gate

2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Fig.1.7. Truth Table of the OR gate NOT GATE

NOT gate or inverter

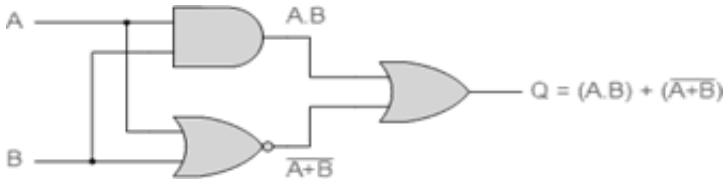
Fig.1.8 Logical Diagram of the NOT gate

NOT gate	
A	$\overline{A}$
0	1
1	0

Fig.1.9. Truth Table of the NOT gate

5.2.BOOLEAN ALGEBRA

The system consists of an AND Gate, a NOR Gate and finally an OR Gate. The expression for the AND gate is A.B, and the expression for the NOR gate is A+B. Both these expressions are also separate inputs to the OR gate which is defined as A+B. Thus the final output expression is given as:



The output of the system is given as Q = (A.B) + (A+B), but the notation A+B is the same as the De Morgan´s notation A.B, Then substituting A.B into the output expression gives us a final output notation of Q = (A.B)+(A.B), which is the Boolean notation for an Exclusive-NOR Gate as seen in the previous section.

Inputs		Intermediates		Output
B	A	A.B	A + B	Q
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

6. Simplification of Boolean function with Map method

2.6.3 Converting Expressions in Standard SOP or POS Forms

Sum of products form can be converted to standard sum of products by ANDing the terms in the expression with terms formed by ORing the literal and its complement which are not present in that term. For example for a three literal expression with literals A, B and C, if there is a term AB, where C is missing, then we form term $(C + \bar{C})$ and AND it with AB. Therefore, we get $AB(C + \bar{C}) = ABC + AB\bar{C}$

Steps to convert SOP to standard SOP form

Step 1 : Find the missing literal in each product term if any.

Step 2 : AND each product term having missing literal/s with term/s form by ORing the literal and its complement.

Step 3 : Expand the terms by applying distributive law and reorder the literals in the product terms.

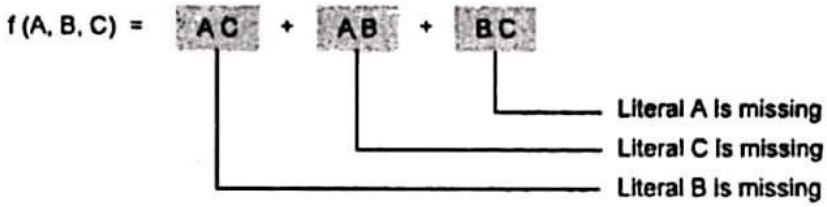
Step 4 : Reduce the expression by omitting repeated product terms if any. Because $A + A = A$.

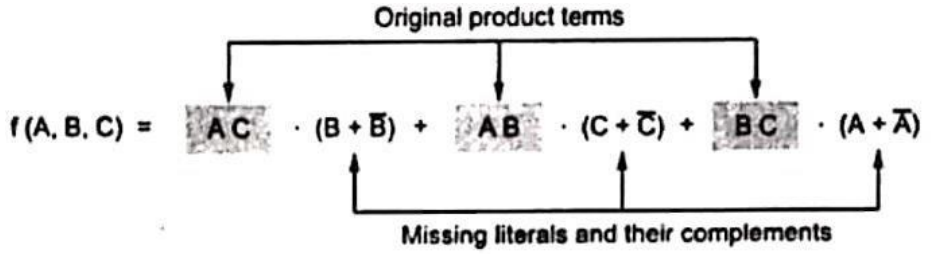
Example 2.3 : *Convert the given expression in standard SOP form.*

$$f(A, B, C) = AC + AB + BC$$

Solution :

Step 1 : Find the missing literal/s in each product term





2.6.4 M-Notations : Minterms and Maxterms

Each individual term in standard SOP form is called **minterm** and each individual term in standard POS form is called **maxterm**. The concept of minterms and maxterms allows us to introduce a very convenient shorthand notations to express logical functions. Table 2.10 gives the minterms and maxterms for a three literal/variable logical function where the number of minterms as well as maxterms is $2^3 = 8$. In general, for an n-variable logical function there are 2^n minterms and an equal number of maxterms.

Variables			Minterms	Maxterms
A	B	C	m_i	M_i
0	0	0	$\bar{A}\bar{B}\bar{C} = m_0$	$A + B + C = M_0$
0	0	1	$\bar{A}\bar{B}C = m_1$	$A + B + \bar{C} = M_1$
0	1	0	$\bar{A}B\bar{C} = m_2$	$A + \bar{B} + C = M_2$
0	1	1	$\bar{A}BC = m_3$	$A + \bar{B} + \bar{C} = M_3$
1	0	0	$A\bar{B}\bar{C} = m_4$	$\bar{A} + B + C = M_4$
1	0	1	$A\bar{B}C = m_5$	$\bar{A} + B + \bar{C} = M_5$
1	1	0	$AB\bar{C} = m_6$	$\bar{A} + \bar{B} + C = M_6$
1	1	1	$ABC = m_7$	$\bar{A} + \bar{B} + \bar{C} = M_7$

Table 2.10 Minterms and maxterms for three variables

2.8 Karnaugh Map Minimization

We have seen that for simplification of Boolean expressions by Boolean algebra we need better understanding of Boolean laws, rules and theorems. During the process of simplification we have to predict each successive step. For these reasons, we can never be absolutely certain that an expression simplified by Boolean algebra alone is the simplest possible expression. On the other hand, the map method gives us a systematic approach for simplifying a Boolean expression. The map method, first proposed by Veitch and modified by Karnaugh, hence it is known as the **Veitch diagram** or the **Karnaugh map.**

2.8.1 One-Variable, Two-Variable, Three-Variable and Four-Variable Maps

The basis of this method is a graphical chart known as Karnaugh map (K-map). It contains boxes called cells. Each of the cell represents one of the 2^n possible products that can be formed from n variables. Thus, a 2-variable map contains $2^2 = 4$ cells, a 3-variable map contains $2^3 = 8$ cells, and so forth. Fig. 2.3 shows outlines of 1, 2, 3 and 4 variable maps.

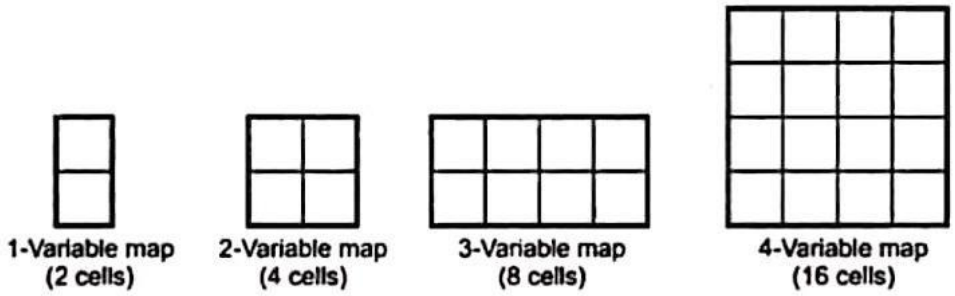


Fig. 2.3 Outlines of 1, 2, 3 and 4 variable Karnaugh maps

2.8.2 Plotting a Karnaugh Map

We know that logic function can be represented in various forms such as truth table, SOP Boolean expression and POS boolean expression. In this section we will see the procedures to plot the given logic function in any form on the Karnaugh map.

2.8.2.1 Representation of Truth Table on Karnaugh Map

Fig. 2.6 shows K-maps plotted from truth tables with 2, 3 and 4-variables. Looking a the Fig. 2.6 we can easily notice that the terms which are having output 1, have th corresponding cells marked with 1s. The other cells are marked with zeros.

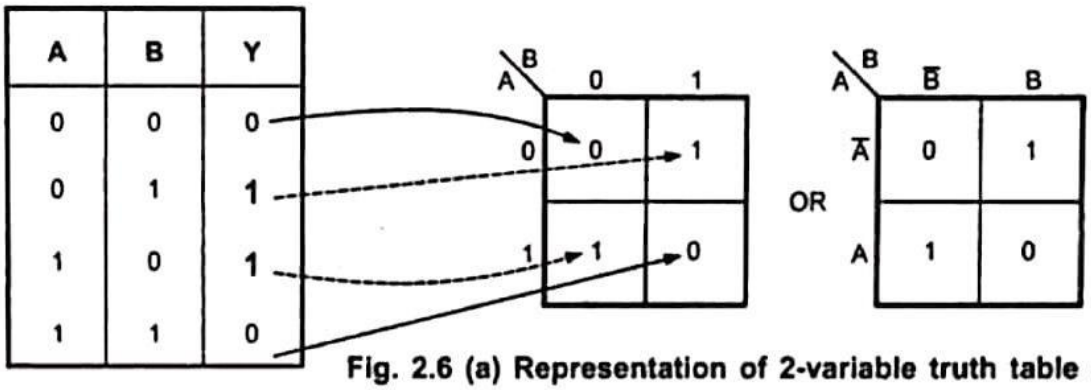


Fig. 2.6 (a) Representation of 2-variable truth table on K-map

Note : The student can varify the data in each cell by checking the data in the column Y for particular row number and the data in the same cell number in the K-map.

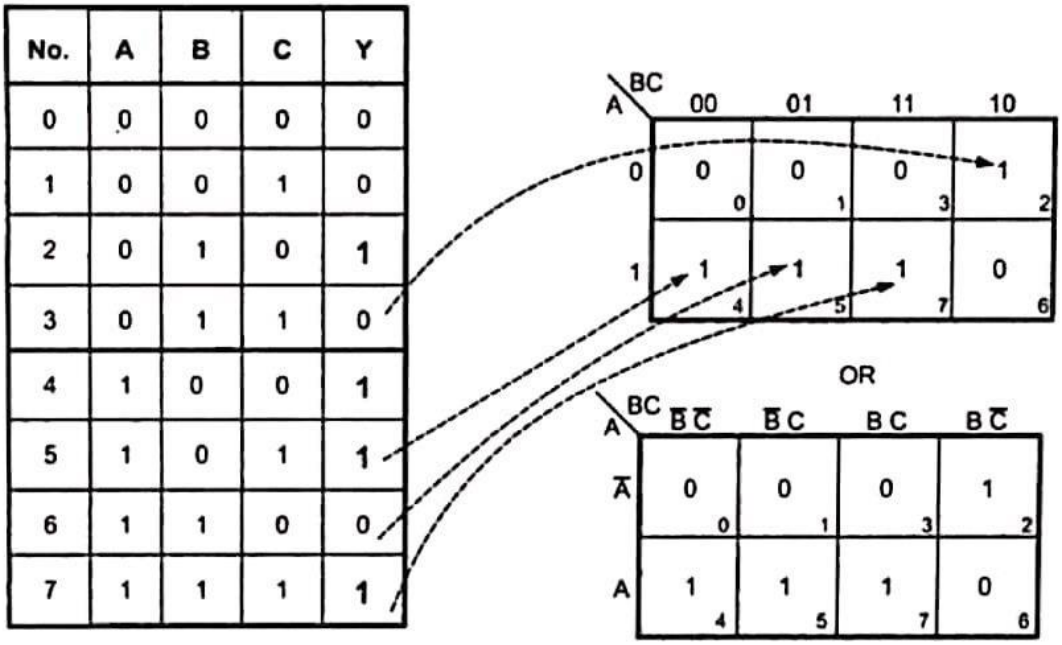
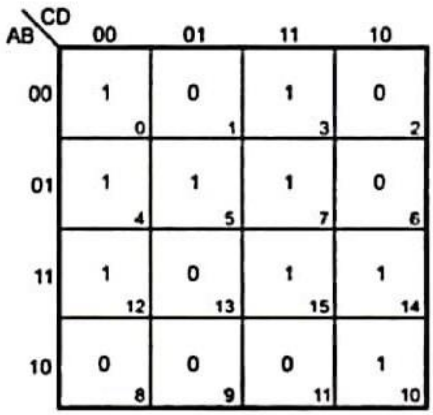


Fig. 2.6 (b) Representation of 3-variable truth table on K-map

No.	A	B	C	D	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1



OR

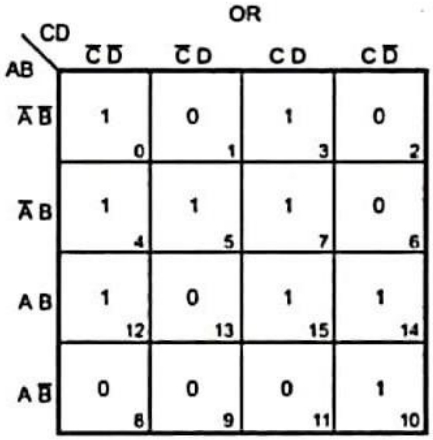


Fig. 2.6(c) Representation of 4-variable truth table on K-map
Fig. 2.6 Plotting truth table on K-map

2.8.2.2 Representing Standard SOP on K-map

A Boolean expression in the sum of products form can be plotted on the Karnaugh map by placing a 1 in each cell corresponding to a term (minterm) in the sum of products expression. Remaining cells are filled with zeros. This is illustrated in the following examples.

Example 2.10 : *Plot Boolean expression* $Y = A\bar{B}\bar{C} + ABC + \bar{A}\bar{B}C$ *on the Karnaugh map.*

Solution : The expression has 3 variables and hence it can be plotted using 3-variable map as shown below.

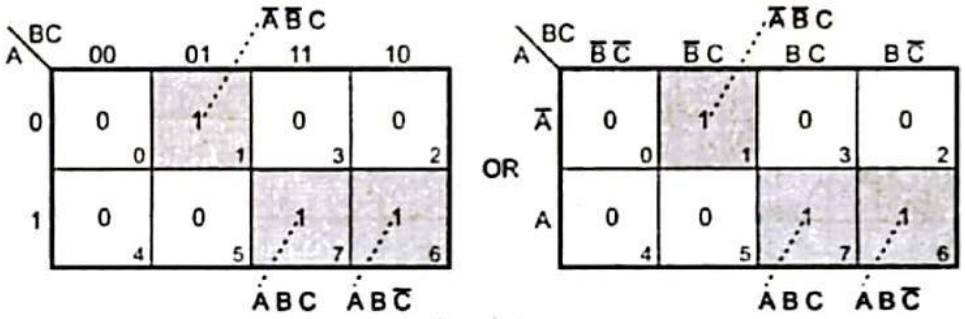


Fig. 2.7

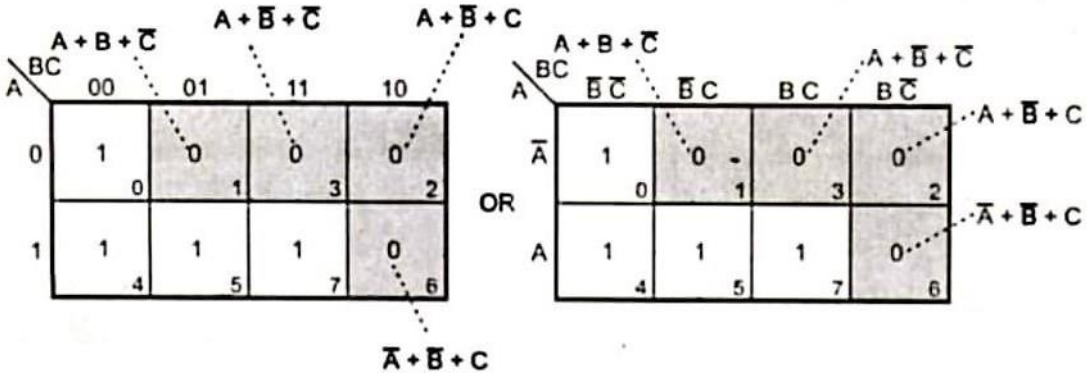
2.8.2.3 Representing Standard POS on K-map

A Boolean expression in the product of sums can be plotted on the Karnaugh map by placing a 0 in each cell corresponding to a term (maxterm) in the expression. Remaining cells are filled with ones. This is illustrated in the following examples.

Example 2.12 : *Plot Boolean expression* $Y = (A + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + \bar{B} + C)(A + B + \bar{C})$ *on the Karnaugh map.*

Solution : The expression has 3 variables and hence it can be plotted using 3-variable map as shown below.

$(A + \bar{B} + C) = M_2, (A + \bar{B} + \bar{C}) = M_3, (\bar{A} + \bar{B} + C) = M_6, (A + B + \bar{C}) = M_1$



2.8.3 Grouping Cells for Simplification

In the last section we have seen representation of Boolean function on the Karnaugh map. We have also seen that minterms are marked by 1s and maxterms are marked by 0s. Once the Boolean function is plotted on the Karnaugh map we have to use grouping technique to simplify the Boolean function. The grouping is nothing but combining terms in adjacent cells. Two cells are said to be adjacent if they conform the single change rule. The simplification is achieved by grouping adjacent 1s or 0s in groups of 2^i, where $i = 1, 2, .., n$ and n is the number of variables. When adjacent 1s are grouped then we get result in the sum of products form; otherwise we get result in the product of sums form. Let us see the various grouping rules.

2.8.3.1 Grouping Two Adjacent Ones (Pair)

Fig. 2.11 (a) shows the Karnaugh map for a particular three variable truth table. This K-map contains a pair of 1s that are horizontally adjacent to each other; the first represents $\overline{A}\,\overline{B}C$ and the second represents $\overline{A}BC$. Note that in these two terms only the B variable

appears in both normal and complemented form ($\overline{A}$ and C remain unchanged). Thus these two terms can be combined to give a resultant that eliminates the B variable since it appears in both uncomplemented and complemented form. This is easily proved as follows :

$$Y = \overline{A}\,\overline{B}C + \overline{A}\,BC$$

$$= \overline{A}\,C(\overline{B} + B) \qquad \text{Rule 6 : } [A + \overline{A} = 1]$$

$$= \overline{A}\,C$$

This same principle holds true for any pair of vertically or horizontally adjacent 1s. Fig. 2.11 (b) shows an example of two vertically adjacent 1s. These two can be combined to eliminate A variable since it appears in both its uncomplemented and complemented forms. This gives result

$$Y = \overline{A}\,B\,C + A\,B\,C = B\,C$$

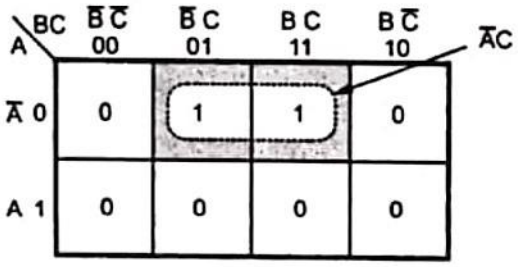


Fig. 2.11(a)

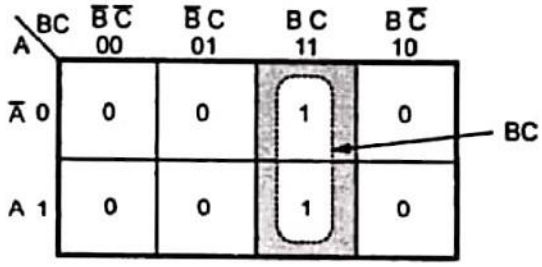


Fig. 2.11(b)

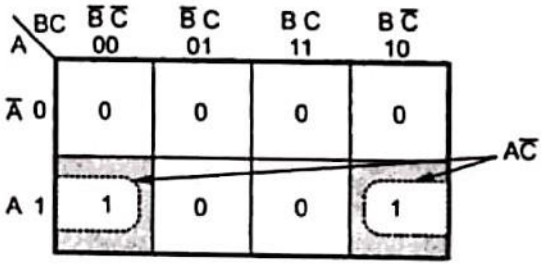


Fig. 2.11 (c)

In a Karnaugh map the leftmost column and rightmost column are considered to be adjacent. Thus, the two 1s in these columns with a common row can be combined to eliminate one variable. This is illustrated in Fig. 2.11 (c).

Here variable B has appeared in both its complemented and uncomplemented forms and hence eliminated as follows :

$$Y = A\,\overline{B}\,\overline{C} + A\,B\,\overline{C}$$

$$= A\,\overline{C}(\overline{B} + B) \qquad \text{Rule 6 : } [\overline{A} + A = 1]$$

$$= A\,\overline{C}$$

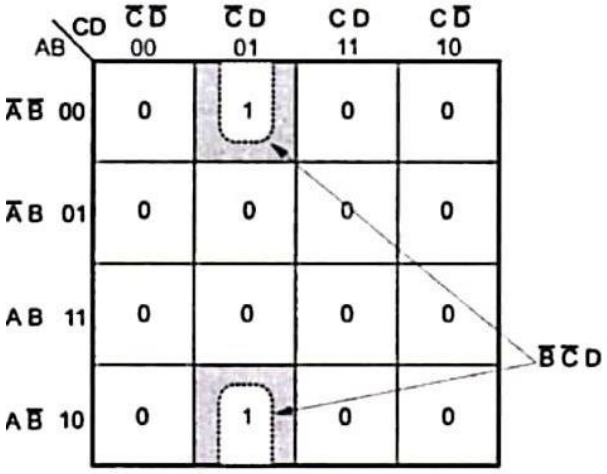


Fig. 2.11 (d)

Let us see another example shown in Fig. 2.11 (d). Here two 1s from top row and bottom row of some column are combined to eliminate variable A, since in a K-map the top row and bottom row are considered to be adjacent.

$$Y = \overline{A}\,\overline{B}\,\overline{C}\,D + A\,\overline{B}\,\overline{C}\,D$$
$$= \overline{B}\,\overline{C}\,D\,(\overline{A} + A)$$
$$= \overline{B}\,\overline{C}\,D$$

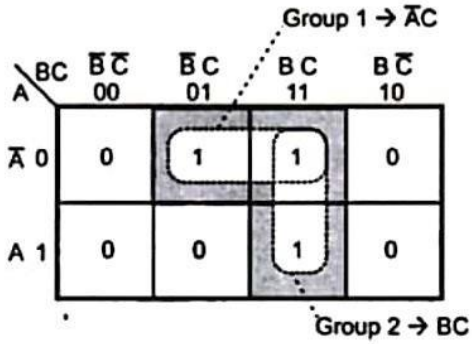


Fig. 2.11 (e)

Fig. 2.11 (e) shows a Karnaugh map that has two overlapping pairs of 1s. This shows that we can share one term between two pairs.

$$Y = \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + A\,B\,C$$
$$= \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + \overline{A}\,B\,C + A\,B\,C \qquad \text{Rule 5 : } [A + A = A]$$
$$= \overline{A}\,C\,(\overline{B} + B) + B\,C\,(\overline{A} + A)$$
$$= \overline{A}\,C + B\,C$$

Fig. 2.11 (f) shows a K-map where three group of pairs can be formed. But only two pairs are enough to include all 1s present in the K-map. In such cases third pair is not required.

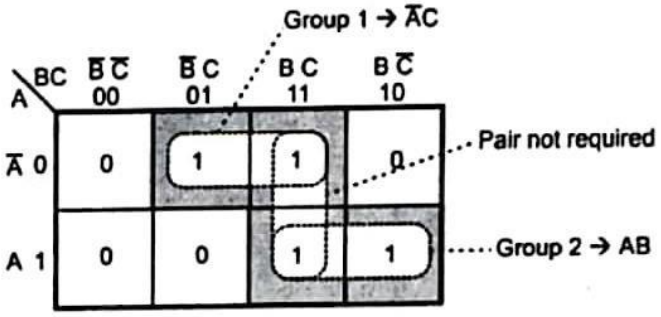


Fig. 2.11 (f) Examples of combining pairs of adjacent ones

$$Y = \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + A\,B\,C + A\,B\,\overline{C}$$

$$= \overline{A}\,C\,(\overline{B} + B) + A\,B\,(C + \overline{C}) \qquad \text{Rule 6 : } [\overline{A} + A = 1]$$

$$= \overline{A}\,C + A\,B$$

2.8.3.2 Grouping Four Adjacent Ones (Quad)

In a Karnaugh map we can group four adjacent 1s. The resultant group is called Quad. Fig. 2.12 shows several examples of quads. Fig. 2.12 (a) shows the four 1s are horizontally adjacent and Fig. 2.12 (b) shows they are vertically adjacent.

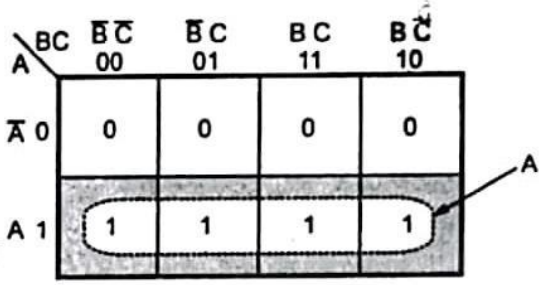


Fig. 2.12 (a) Y = A

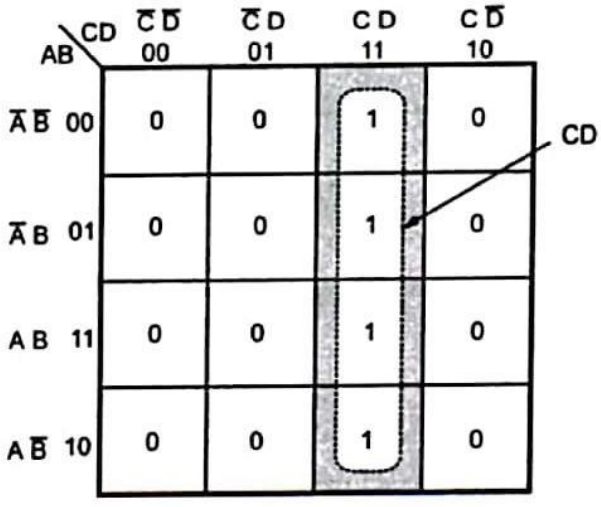


Fig. 2.12 (b) Y = CD

A K-map in Fig. 2.12 (c) contains four 1s in a square, and they are considered adjacent to each other. The four 1s in Fig. 2.12 (d) are also adjacent, as are those in Fig. 2.12 (e) because, as mentioned earlier, the top and bottom rows are considered to be adjacent to each other and the leftmost and rightmost columns are also adjacent to each other.

Simplification of Sum of Products Expressions (Minimal Sums)

From the above discussion we can outline generalized procedure to simplify Boolean expressions as follows :

1. Plot the K-map and place 1s in those cells corresponding to the 1s in the truth table or sum of product expression. Place 0s in other cells.
2. Check the K-map for adjacent 1s and encircle those 1s which are not adjacent to any other 1s. These are called isolated 1s .
3. Check for those 1s which are adjacent to only one other 1 and encircle such pairs.
4. Check for quads and octets of adjacent 1s even if it contains some 1s that have already been encircled. While doing this make sure that there are minimum number of groups.
5. Combine any pairs necessary to include any 1s that have not yet been grouped.
6. Form the simplified expression by summing product terms of all the groups.

To get familiar with these steps we will solve some examples.

➠ **Example 2.14 :** *Minimize the expression* $Y = A\overline{B}\,C + \overline{A}\,\overline{B}\,C + \overline{A}BC + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C}$

Solution :

Step 1 : Fig 2.14 (a) shows the K-map for three variables and it is plotted according to the given expression.

Step 2 : There are no isolated 1s.

Step 3 : 1 in the cell 3 is adjacent only to 1 in the cell 1. This pair is combined and referred to as group 1.

Step 4 : There is no octet, but there is a quad. Cells 0, 1, 4 and 5 form a quad. This quad is combined and referred to as group 2.

Step 5 : All 1s have already been grouped.

Step 6 : Each group generates a term in the expression for Y. In group 1 B variable is eliminated and in group 2 variables A and C are eliminated and we get,

$Y = \overline{A}C + \overline{B}$

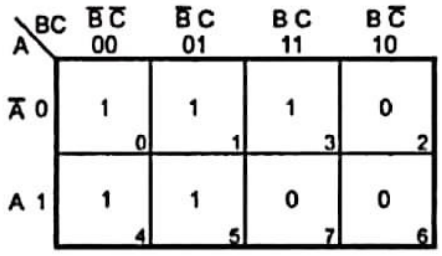


Fig. 2.14 (a)

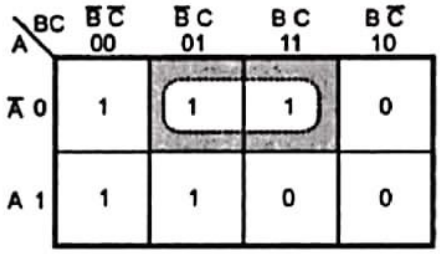


Fig. 2.14 (b)

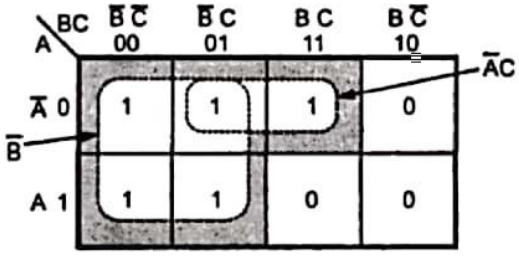


Fig. 2.14 (c)

2.8.6 Simplification of Product of Sums Expressions (Minimal Products)

In the above discussion, we have considered the Boolean expression in sum of products form and grouped 2, 4, and 8 adjacent ones to get the simplified Boolean expression in the same form. In practice, the designer should examine both the sum of products and product of sums reductions to ascertain which is more simplified. We have already seen the representation of product of sums on the Karnaugh map. Once the expression is plotted on the K-map instead of making the groups of ones, we have to make groups of zeros. Each group of zero results a sum term and it is nothing but the **prime implicate**. The technique for using maps for POS reductions is a simple step by step process and it is similar to the one used earlier.

1. Plot the K-map and place 0s in those cells corresponding to the 0s in the truth table or maxterms in the products of sum expression.
2. Check the K-map for adjacent 0s and encircle those 0s which are not adjacent to any other 0s. These are called isolated 0s.
3. Check for those 0s which are adjacent to only one other 0 and encircle such pairs.
4. Check for quads and octets of adjacent 0s even if it contains some 0s that have already been encircled. While doing this make sure that there are minimum number of groups.
5. Combine any pairs necessary to include any 0s that have not yet been grouped.
6. Form the simplified SOP expression for $\bar{F}$ by summing product terms of all the groups.

(**Note** : The simplified expression is in the complemented form because we have grouped 0s to simplify the expression.)

7. Use DeMorgan's theorem on $\bar{F}$ to produce the simplified expression in POS form.

To get familiar with these steps we will solve some examples.

Example 2.21 : *Minimize the expression*

$$Y = (A + B + \bar{C}) (A + \bar{B} + \bar{C}) (\bar{A} + \bar{B} + \bar{C}) (\bar{A} + B + C)(A + B + C)$$

Solution : $(A + B + \bar{C}) = M_1$, $(A + \bar{B} + \bar{C}) = M_3$, $(\bar{A} + \bar{B} + \bar{C}) = M_7$,

$(\bar{A} + B + C) = M_4$, $(A + B + C) = M_0$

Step 1 : Fig. 2.21 (a) shows the K-map for three variable and it is plotted according to given maxterms.

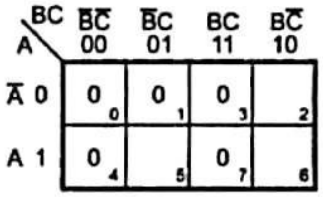


Fig. 2.21 (a)

Step 2 : There are no isolated 0s.

Step 3 : 0 in the cell 4 is adjacent only to 0 in the cell 0 and 0 in the cell 7 is adjacent only to 0 in the cell 3. These two pairs are combined and referred to as group 1 and group 2 respectively.

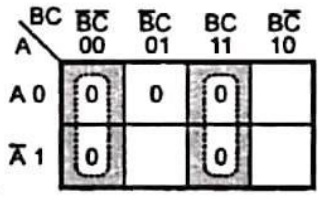


Fig. 2.21 (b)

Step 4 : There are no quads and octets.

Step 5 : The 0 in the cell 1 can be combined with 0 in the cell 3 to form a pair. This pair is referred to as group 3.

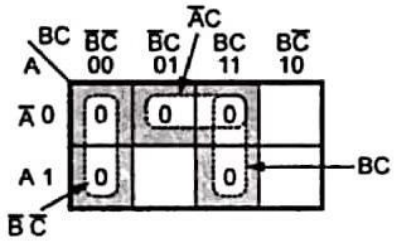


Fig. 2.21 (c)

Step 6 : In group 1 and in group 2, A is eliminated, where as in group 3 variable B is eliminated and we get

$$\overline{Y} = \overline{B}\,\overline{C} + BC + \overline{A}C$$

Step 7 :

$$Y = \overline{Y} = \overline{\overline{B}\,\overline{C} + BC + \overline{A}C}$$

$$= \left(\overline{\overline{B}\,\overline{C}}\right)\left(\overline{BC}\right)\left(\overline{\overline{A}C}\right)$$

$$= \left(\overline{\overline{B}} + \overline{\overline{C}}\right)\left(\overline{B} + \overline{C}\right)\left(\overline{\overline{A}} + \overline{C}\right)$$

$$= (B + C)\left(\overline{B} + \overline{C}\right)\left(A + \overline{C}\right)$$

It is possible to directly write the expression for Y by using DeMorgans theorem for each minterm as follows :

$$\overline{Y} = \overline{B}\,\overline{C} + BC + \overline{A}\,C \rightarrow Y = (B + C)\left(\overline{B} + \overline{C}\right)\left(A + \overline{C}\right)$$

2.9.2 Don't Care Conditions in Logic Design

In this section, we see the example of incompletely specified Boolean function. Let us see the logic circuit for an even parity generator for 4-bit BCD number. The Table 2.15 shows the truth table for even-parity generator. The truth table shows that the output for last six input conditions cannot be specified, because such input conditions does not occur when input is in the BCD form.

Example 2.25 : *Find the reduced SOP form of the following function.*

$$f(A, B, C, D) = \sum m(1, 3, 7, 11, 15) + \sum d(0, 2, 4).$$

Solution :

AB \ CD	$\overline{C}\,\overline{D}$ 00	$\overline{C}D$ 01	CD 11	$C\overline{D}$ 10
$\overline{A}\,\overline{B}$ 00	X 0	1 1	1 3	X 2
$\overline{A}B$ 01	X 4	0 5	1 7	0 6
AB 11	0 12	0 13	1 15	0 14
$A\overline{B}$ 10	0 8	0 9	1 11	0 10

AB \ CD	$\overline{C}\,\overline{D}$ 00	$\overline{C}D$ 01	CD 11	$C\overline{D}$ 10
$\overline{A}\,\overline{B}$ 00	1	1	1	1
$\overline{A}B$ 01	0	0	1	0
AB 11	0	0	1	0
$A\overline{B}$ 10	0	0	1	0

CD

To form a quad of cells 0, 1, 2 and 3 the don't care conditions 0 and 2 are replaced by 1s.

The remaining don't care condition is replaced by 0 since it is not required to form any group. With these replacements we get the simplified equation as

$$f(A, B, C, D) = \underbrace{\overline{A}\,\overline{B}}_{\text{Group 1}} + \underbrace{CD}_{\text{Group 2}}$$

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – II – SBSA1303 – COMPUTER ARCHITECTURE

ADDER
- Half Adder
- Full Adder

SUBTRACTOR
- Half Subtractor
- Full Subractor

DECODER
ENCODER
MULTIPLEXER
DEMULTIPLEXER

2.1 ADDER

Adder is digital Logic Circuit in digital Electronics , used for addition of numbers.
In many computers and components like Processors, adders are not only used in arithmetic logic unit, and other operations like calculate the addresses, increment and decrement operators.
Two types: 1. Half Adder, 2. Full Adder
Have a look of single digit Addition,

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10 (with carry 1)

2.1.1. HALF ADDER

Half Adder adds two single digits. Digit A and Digit B. It has two output Sum(S) and carry (C)
The carry signal represents an overflow into the next digit of multi- digit Addition. Fig2.1 Shows the logical diagram of the half adder, which is the combination of the OR and AND gates. The OR gate output considered as Sum and the AND gat output considered as carry. Fig.2.2 shows the truth table of the half adder.

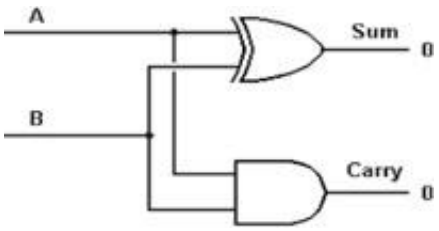


Fig 2.1 . Logical Circuit of the half adder

Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Fig.2.1.Truth Table

2.1.2. FULL ADDER

A Full Adder adds binary numbers and accounts for adding carried in as well as out. A one bit Full Adder has three inputs, often written as A,B and C_{in}.It has two outputs Sum (S) and Carry (C_{out}).The Main Difference between half adder and full adder is Full adder has three inputs and Two outputs.

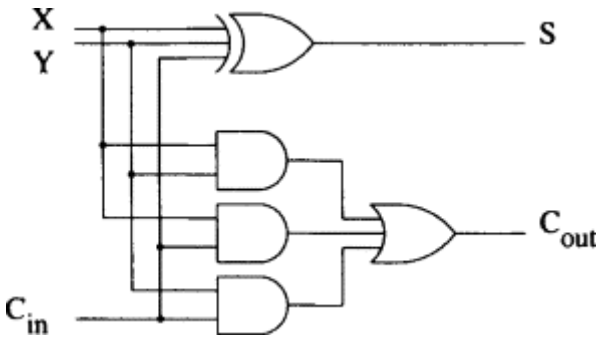


Fig.2.3. Logical Circuit of the Full adder

The Above diagram shows the full adder logical diagram, it is a combination of the two OR gates and three AND gates.

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
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	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Fig.2.4. Truth Table of the Full Adder

2.2. SUBTRACTOR

The Subtraction of the two number by taking the complement of the subtrahend and adding it to the minuend. By this method, the subtraction operation becomes an addition operation requiring full-adder for its machine implementation. Subtraction operation is each subtrahend bit of the number is subtracted from the minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, 1 is borrowed from the next significant bit. The fact that a 1 has been borrowed from the next significant bit.

2.2.1. HALF SUBTRACTOR

Half subtractor is a combination circuit with two inputs and two outputs (difference and borrow). It produces the difference between the two binary bits at the input and also produces an output (Borrow) to indicate if a 1 has been borrowed. In the subtraction (A-B), A is called as Minuend bit and B is called as Subtrahend bit.

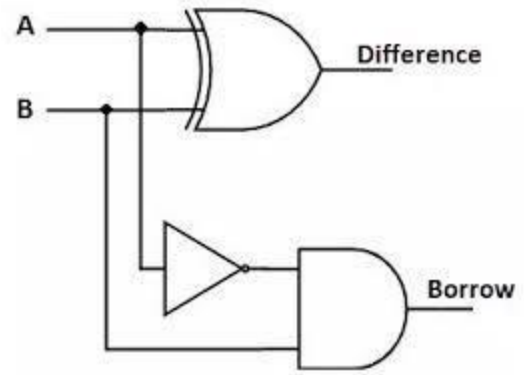


Fig.2.5. Half Subtractor Logical Circuit

Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Fig.2.6.Truth Table of Half Subtractor

2.2.2. FULL SUBTRACTOR

The full subtractor is a combinational circuit with three inputs A,B,C and two output D and C'. A is the 'minuend', B is 'subtrahend', C is the 'borrow' produced by the previous stage, D is the difference output and C' is the borrow output.

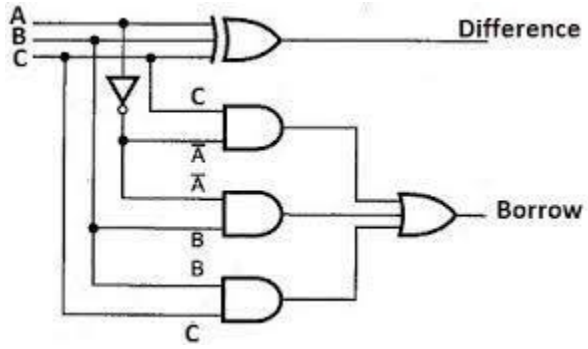


Fig.2.7. Logical Circuit of Full Subtractor

Inputs			Outputs	
A	B	$Borrow_{in}$	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Fig.2.8. Truth Table of Full Subtractor

2.3. ENCODER

Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder. An encoder has n number of input lines and m number of output lines. An encoder produces an m bit binary code corresponding to the digital input number. The encoder accepts an n input digital word and converts it into an m bit another digital word.

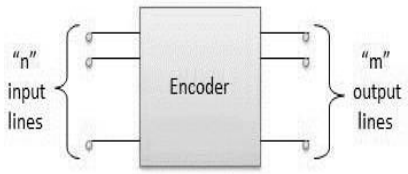


Fig.2.9. Block Diagram of the Encoder

Examples of Encoder

- Priority encoders
- Decimal to BCD encoder
- Octal to binary encoder
- Hexadecimal to binary encoder

2.3.1. PRIORITY ENCODER

This is a special type of encoder. Priority is given to the input lines. If two or more input line are 1 at the same time, then the input line with highest priority will be considered. There are four input D_0, D_1, D_2, D_3 and two output Y_0, Y_1. Out of the four input D_3 has the highest priority and D_0 has the lowest priority. if $D_3 = 1$ then $Y_1\ Y_1 = 11$ irrespective of the other inputs. Similarly if $D_3 = 0$ and $D_2 = 1$ then $Y_1\ Y_0 = 10$ irrespective of the other inputs.

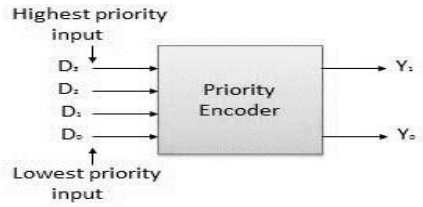


Fig.2.10. Block Diagram of the Priority Encoder

Highest	Inputs		Lowest	Outputs	
D_3	D_2	D_1	D_0	Y_0	Y_1
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

Fig.2.11. Truth Table of the Priority Encoder

2.4. MULTIPLEXER

Multiplexer is a special type of combinational circuit. There are n-data inputs, one output and m select inputs with $2^m = n$. It is a digital circuit which selects one of the n data inputs and routes it to the output. The selection of one of the n inputs is done by the selected inputs. Depending on the digital code applied at the selected inputs, one out of n data sources is selected and transmitted to the single output Y.

E is called the strobe or enable input which is useful for the cascading. It is generally an active low terminal that means it will perform the required operation when it is low. This is called as Data Selector.

Types of Multiplexer :

2 : 1 multiplexer

4 : 1 multiplexer

16 : 1 multiplexer

32 : 1 multiplexer

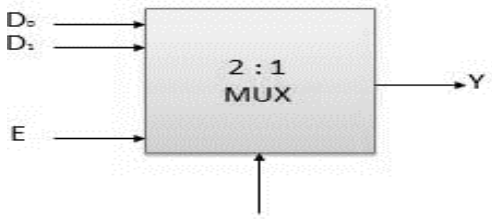


Fig. 2.12. Block diagram of the Multilexer

Enable	Select	Output
E	S	Y
0	x	0
1	0	D_0
1	1	D_1

x = Don't care

Fig.2.13. Truth table of the Multiplexer

2.5. DEMULPLEXER

A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs. It has only one input, n outputs, m select input. At a time only one output line is selected by the select lines and the input is transmitted to the selected output line.

Demultiplexers in multiple variations.

1: 2 demultiplexer
1 : 4 demultiplexer
1 : 16 demultiplexer
1 : 32 demultiplexer

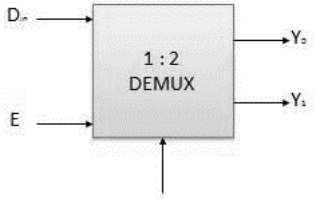


Fig. 2.14. Block Diagram of Demultiplexer

Enable	Select	Output
E	S	Y0 Y1
0	x	0 0
1	0	0 D_{in}
1	1	D_{in} 0

x = Don't care

Fig.2.15. Truth Table of the Demultiplexer

2.6. SEQUENTIAL LOGIC

The combinational circuit does not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit. But sequential circuit has memory so output can vary based on input. This type of circuits uses previous input, output, clock and a memory element.

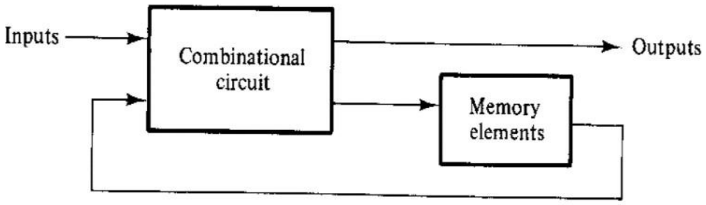


FIGURE 6-1
Block diagram of a sequential circuit

Fig.2.16. Block diagram of a Sequential Circuit

The following all are the sequential logic circuits,

- FLIPFLOP
- R-S FLIP FLOP
- M-S JK FLIPFLOP
- DELAY FLIPFLOP
- TOGGLE FLIPFLOP
- REGISTERS
- COUNTERS

2.7. FLIP FLOP

Flip flop is a sequential circuit which generally samples its inputs and changes its outputs only at particular instants of time and not continuously. Flip flop is said to be edge sensitive or edge triggered rather than being level triggered like latches.

2.7.1. S-R FLIP FLOP

It is basically S-R latch using NAND gates with an additional enable input. It is also called as level triggered SR-FF. For this, circuit in output will take place if and only if the enable input (E) is made active. In short this circuit will operate as an S-R latch if E = 1 but there is no change in the output if E = 0.

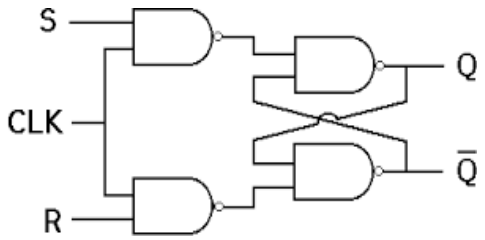


Fig. 2.17. Logical Diagram of S-R FlipFlop

Inputs			Outputs		Comments
E	S	R	Q_{n+1}	$\overline{Q_{n+1}}$	
1	0	0	Q_n	$\overline{Q_n}$	No change
1	0	1	0	1	Rset
1	1	0	1	0	Set
1	1	1	x	x	Indeterminate

Fig.2.18. Truth Table of S-R FlipFlop

S-R FLIP FLOP – OPERATIONS

S = R = 0 : No change

If S = R = 0 then output of NAND gates 3 and 4 are forced to become 1.

Hence R' and S' both will be equal to 1. Since S' and R' are the input of the basic S-R latch using NAND gates, there will be no change in the state of outputs.

S = 0, R = 1, E = 1

Since S = 0, output of NAND-3 i.e. R' = 1 and E = 1 the output of NAND-4 i.e. S' = 0.

Hence Qn+1 = 0 and Qn+1 bar = 1. This is reset condition.

S =0 1, R = 0, E = 1

Output of NAND-3 i.e. R' = 0 and output of NAND-4 i.e. S' = 1.

Hence output of S-R NAND latch is Qn+1 = 1 and Qn+1 bar = 0. This is the reset condition.

S = 1, R = 1, E = 1

As S = 1, R = 1 and E = 1, the output of NAND gates 3 and 4 both are 0 i.e. S' = R' = 0. Hence the Race condition will occur in the basic NAND latch.

7.2. MASTER SLAVE JK FLIPFLOP

Master slave JK FF is a cascade of two S-R FF with feedback from the output of second to input of first. Master is a positive level triggered. But due to the presence of the inverter in the clock line, the slave will respond to the negative level. Hence when the clock = 1 (positive level) the master is active and the slave is inactive. Whereas when clock = 0 (low level) the slave is active and master is inactive.

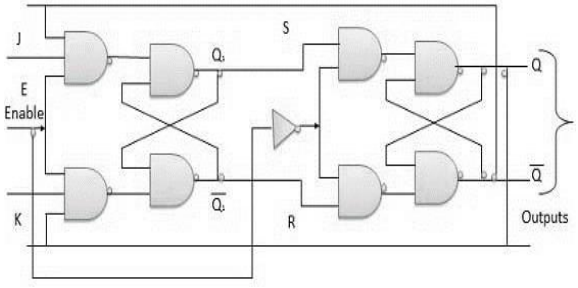


Fig.2.19.Logical Diagram of M-S FlipFlop

Inputs			Outputs		Comments
E	J	K	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	Q_n	$\overline{Q}_n$	No change
1	0	1	0	1	Rset
1	1	0	1	0	Set
1	1	1	$\overline{Q}_n$	Q_n	Toggle

Fig.2.20. Truth Table of M-S FlipFlop

MS JK FLIP FLOP –OPERATIONS

- **J = K = 0 (No change)**
  When clock = 0, the slave becomes active and master is inactive. But since the S and R inputs have not changed, the slave outputs will also remain unchanged. Therefore outputs will not change if J = K =0.
- **J = 0 and K = 1 (Reset)**
  Clock = 1 − Master active, slave inactive. Therefore outputs of the master become $Q_1 = 0$ and Q_1 bar = 1. That means S = 0 and R =1.Clock = 0 − Slave active, master inactive. Therefore outputs of the slave become Q = 0 and Q bar = 1.
  Again clock = 1 − Master active, slave inactive. Therefore even with the changed outputs Q = 0 and Q bar = 1 fed back to master, its output will be Q1 = 0 and Q1 bar = 1. That means S = 0 and R = 1. Hence with clock = 0 and slave becoming active the outputs of slave will remain Q = 0 and Q bar = 1. Thus we get a stable output from the Master slave.
- **J = 1 and K = 0 (Set)**
  Clock = 1 − Master active, slave inactive. Therefore outputs of the master become $Q_1 = 1$ and Q_1 bar = 0. That means S = 1 and R =0. Clock = 0 − Slave active, master inactive. Therefore outputs of the slave become Q = 1 and Q bar = 0. Again clock = 1 − then it can be shown that the outputs of the slave are stabilized to Q = 1 and Q bar = 0.
- **J = K = 1 (Toggle)**
  Clock = 1 − Master active, slave inactive. Outputs of master will toggle. So S and R also will be inverted. Clock = 0 − Slave active, master inactive. Outputs of slave will toggle.
  These changed output are returned back to the master inputs. But since clock = 0, the master is still inactive. So it does not respond to these changed outputs. This avoids the multiple toggling which leads to the race around condition. The master slave flip flop will avoid the race around condition.

2.7.2. DELAY FLIP FLOP / D FLIPFLOP

Delay Flip Flop or D Flip Flop is the simple gated S-R latch with a NAND inverter connected between S and R inputs. It has only one input. The input data is appearing at the output after some time.

Due to this data delay between i/p and o/p, it is called delay flip flop. S and R will be the complements of each other due to NAND inverter. Hence S = R = 0 or S = R = 1, these input condition will never appear. This problem is avoid by SR = 00 and SR = 1 conditions.

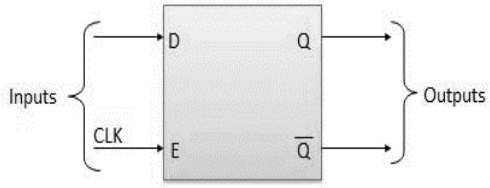


Fig.2.21. Block Diagram of D- FlipFlop

Inputs		Outputs		Comments
E	D	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	1	Rset
1	1	1	0	Set

Fig.2.22. Truth Table of the D- FlipFlop

D FLIPFLOP – Operations

E = 0

Latch is disabled. Hence no change in output.

E = 1 and D = 0

If E = 1 and D = 0 then S = 0 and R = 1. Hence irrespective of the present state, the next state is Qn+1 = 0 and Qn+1 bar = 1. This is the reset condition.

E = 1 and D = 1

If E = 1 and D = 1, then S = 1 and R = 0. This will set the latch and Q_{n+1} = 1 and Q_{n+1} bar = 0 irrespective of the present state.

7.4. TOGGLE FLIP FLOP / T FLIP FLOP

Toggle flip flop is basically a JK flip flop with J and K terminals permanently connected together. It has only input denoted by T as shown in the Symbol Diagram. The symbol for positive edge triggered T flip flop is shown in the Block Diagram.

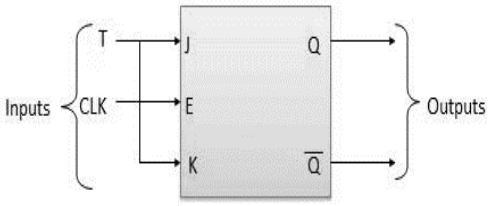


Fig.2.23. Block Diagram of T- FlipFlop

Inputs		Outputs		Comments
E	T	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	Q_n	$\overline{Q}_n$	No change
1	1	$\overline{Q}_n$	Q_n	Toggle

Fig.2.24.Truth Table of T-FlipFlop

T FLIP FLOP – Operations

T = 0, J = K = 0

The output Q and Q bar won't change

T = 1, J = K = 1

Output will toggle corresponding to every leading edge of clock signal.

2.8. DIGITAL REGISTERS

Flip-flop is a 1 bit memory cell which can be used for storing the digital data. To increase the storage capacity in terms of number of bits, we have to use a group of flip-flop. Such a group of flip-flop is known as a **Register**.

The **n-bit register** will consist of **n** number of flip-flop and it is capable of storing an **n-bit** word.

The binary data in a register can be moved within the register from one flip-flop to another. The registers that allow such data transfers are called as **shift registers**. There are four mode of operations of a shift register.

- Serial Input Serial Output
- Serial Input Parallel Output
- Parallel Input Serial Output
- Parallel Input Parallel Output

2.8.1. Serial Input Serial Output

Let all the flip-flop be initially in the reset condition i.e. $Q_3 = Q_2 = Q_1 = Q_0 = 0$. If an entry of a four bit binary number 1 1 1 1 is made into the register, this number should be applied to D_{in} bit with the LSB bit applied first. The D input of FF-3 i.e. D_3 is connected to serial data input D_{in}. Output of FF-3 i.e. Q_3 is connected to the input of the next flip-flop i.e. D_2 and so on.

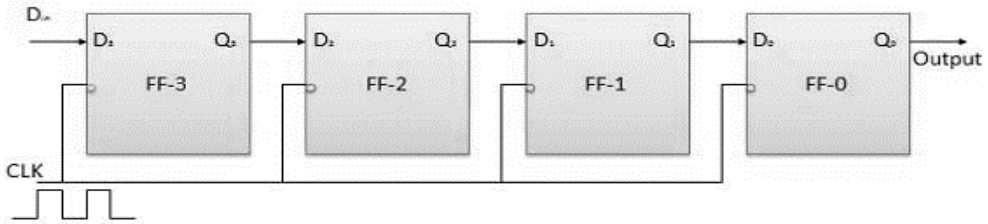


Fig.2.24. Block Diagram of the Serial In Serial Out

Operation: Before application of clock signal, let Q3 Q2 Q1 Q0 = 0000 and apply LSB bit of the number to be entered to Din. So Din = D3 = 1. Apply the clock. On the first falling edge of clock, the FF-3 is set, and stored word in the register is Q3 Q2 Q1 Q0 = 1000.

	CLK	$D_{in} = Q_3$	$Q_3 = D_2$	$Q_2 = D_1$	$Q_1 = D_0$	Q_0
Initially			0	0	0	0
(i)	↓	1	1	0	0	0
(ii)	↓	1	1	1	0	0
(iii)	↓	1	1	1	1	0
(iv)	↓	1	1	1	1	1

Direction of data travel

Fig.2.25. Truth Table of SISO

2.8.2.Serial Input Parallel Output

In such types of operations, the data is entered serially and taken out in parallel fashion.Data is loaded bit by bit. The outputs are disabled as long as the data is loading.As soon as the data loading gets completed, all the flip-flops contain their required data, the outputs are enabled so that all the loaded data is made available over all the output lines at the same time.4 clock cycles are required to load a four bit word. Hence the speed of operation of SIPO mode is same as that of SISO mode.

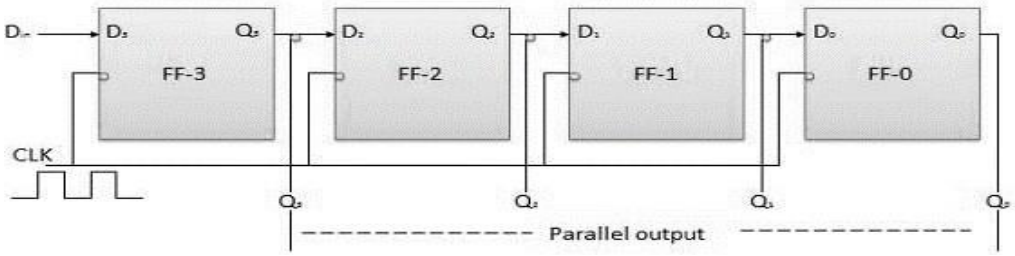


Fig.2.25. Block Diagram of SIPO

2.8.3.Parallel Input Serial Output (PISO)

Data bits are entered in parallel fashion. The circuit shown below is a four bit parallel input serial output register. Output of previous Flip Flop is connected to the input of the next one via a combinational circuit. The binary input word B_0, B_1, B_2, B_3 is applied though the same combinational circuit. There are two modes in which this circuit

can work namely - shift mode or load mode.

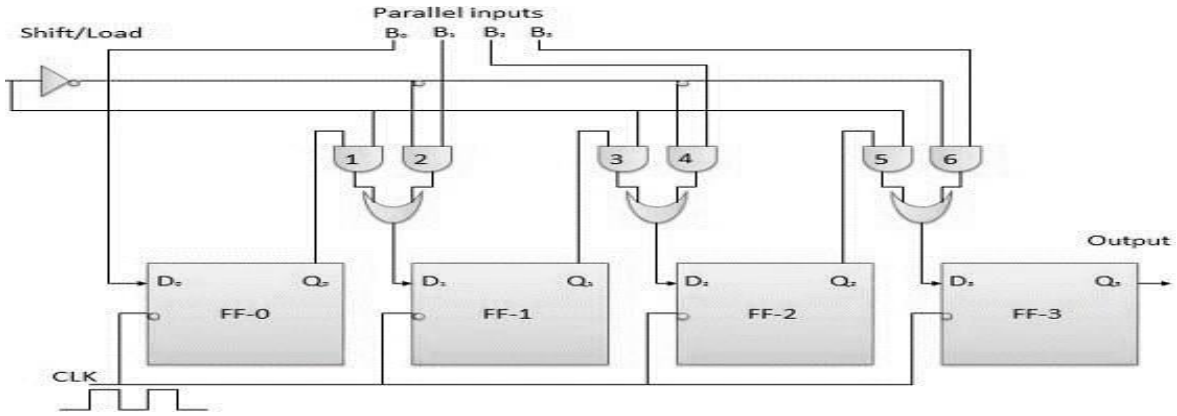


Fig.2.26. Parallel Input Serial Output

Load Mode

When the shift/load bar line is low (0), the AND gate 2, 4 and 6 become active they will pass B_1, B_2, B_3 bits to the corresponding flip-flops. On the low going edge of clock, the binary input B_0, B_1, B_2, B_3 will get loaded into the corresponding flip-flops. Thus parallel loading takes place.

Shift mode

When the shift/load bar line is low (1), the AND gate 2, 4 and 6 become inactive. Hence the parallel loading of the data becomes impossible. But the AND gate 1,3 and 5 become active. Therefore the shifting of data from left to right bit by bit on application of clock pulses. Thus the parallel in serial out operation takes place.

2.8.4.Parallel Input Parallel Output (PISO)

The 4 bit binary input B_0, B_1, B_2, B_3 is applied to the data inputs D_0, D_1, D_2, D_3 respectively of the four flip-flops. As soon as a negative clock edge is applied, the input binary bits will be loaded into the flip-flops simultaneously. The loaded bits will appear simultaneously to the output side. Only clock pulse is essential to load all the bits.

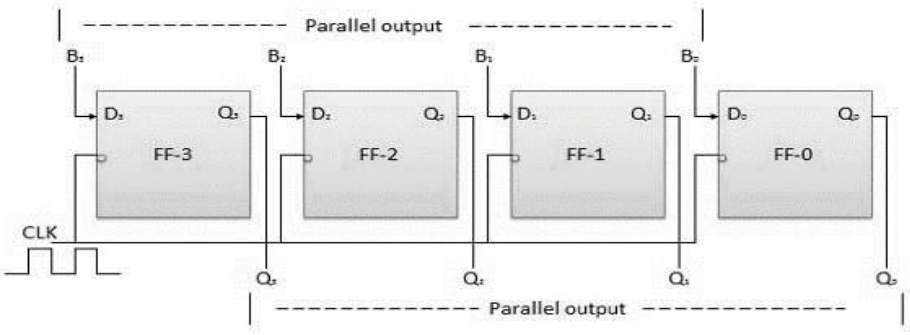


Fig.27. Block Diagram of Parallel Input Parallel Output

2.9. BIDIRECTIONAL SHIFT REGISTER

If a binary number is shifted left by one position then it is equivalent to multiplying the original number by 2. Similarly if a binary number is shifted right by one position then it is equivalent to dividing the original number by 2.Hence if we want to use the shift register to multiply and divide the given binary number, then we should be able to move the data in either left or right direction.Such a register is called bi-directional register. A four bit bi-directional shift register is shown in fig2.28. There are two serial inputs namely the serial right shift data input DR, and the serial left shift data input DL along with a mode select input (M).

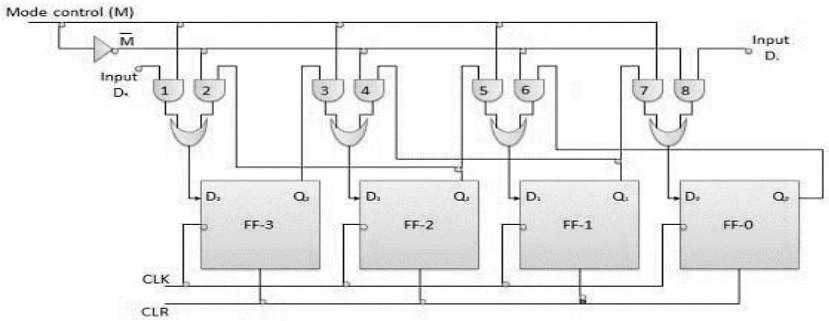


Fig.2.28. Block Diagram of bidirectional shift register

Operations :

With M = 1 − Shift right operation

If M = 1, then the AND gates 1, 3, 5 and 7 are enabled whereas the remaining AND gates 2, 4, 6 and 8 will be disabled.

The data at D_R is shifted to right bit by bit from FF-3 to FF-0 on the application of clock pulses. Thus with M = 1 we get the serial right shift operation.

With M = 0 − Shift left operation

When the mode control M is connected to 0 then the AND gates 2, 4, 6 and 8 are enabled while 1, 3, 5 and 7 are disabled.

The data at D_L is shifted left bit by bit from FF-0 to FF-3 on the application of clock pulses. Thus with M = 0 we get the serial right shift operation.

2.10. UNIVERSAL SHIFT REGISTER

A shift register which can shift the data in only one direction is called a unidirectional shift register. A shift register which can shift the data in both directions is called a bi-directional shift register. Applying the same logic, a shift register which can

shift the data in both directions as well as load it parallely, is known as a universal shift register.

The shift register is capable of performing the following operation −

- Parallel loading
- Left Shifting
- Right shifting

The mode control input is connected to logic 1 for parallel loading operation whereas it is connected to 0 for serial shifting. With mode control pin connected to gr ound, the universal shift register acts as a bi-directional register.

For serial left operation, the input is applied to the serial input which goes to AND gate-1 shown in figure. Whereas for the shift right operation, the serial input is applied to D input.

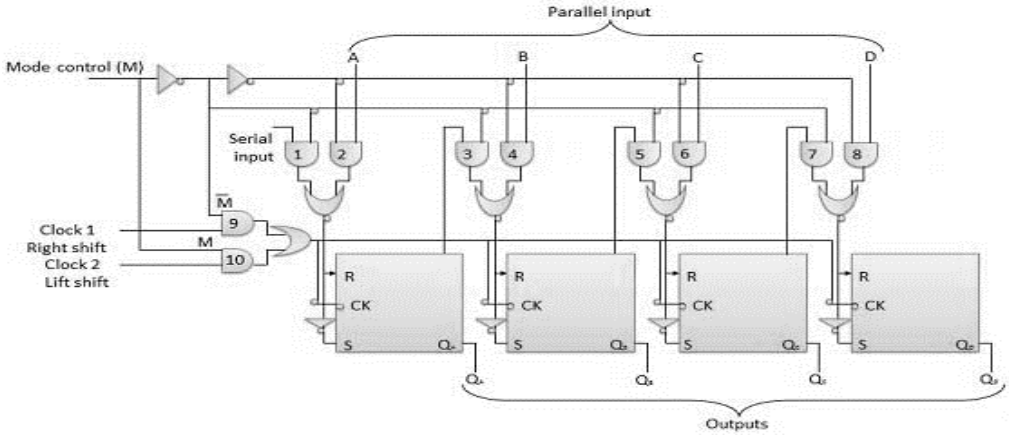


Fig.2.29. Block Diagram UNIVERSAL SHIFT REGISTER

2.11.COUNTERS

Counter is a sequential circuit. A digital circuit which is used for a counting pulses is known counter. Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied.

Counters are of two types:

- Asynchronous or ripple counters.
- Synchronous counters.

2.11.1. ASYNCHRONOUS OR RIPPLE COUNTERS

The logic diagram of a 2-bit ripple up counter is shown in figure. The toggle (T) flip-flop are being used. But we can use the JK flip-flop also with J and K connected permanently to logic 1. External clock is applied to the clock input of flip-flop A and Q_A output is applied to the clock input of the next flip-flop i.e. FF-B.

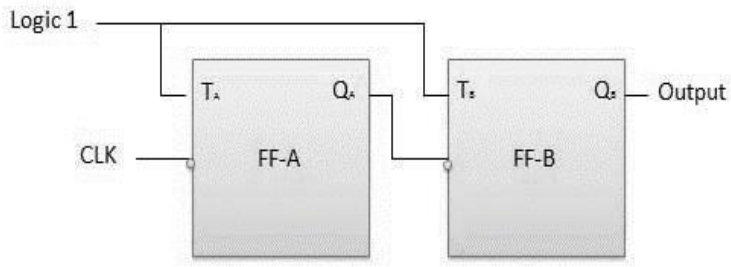


Fig.2.30. Block Diagram of Asynchronous Or Ripple Counters

Operations :

Initially let both the FFs be in the reset state

$Q_BQ_A = 00$ initially

After 1st negative clock edge.

As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will be equal to 1.Q_A is connected to clock input of FF-B. Since Q_A has changed from 0 to 1, it is treated as the positive clock edge by FF-B. There is no change in Q_B because FF-B is a negative edge triggered FF. $Q_BQ_A = 01$ after the first clock pulse.

After 2nd negative clock edge

On the arrival of second negative clock edge, FF-A toggles again and $Q_A = 0$.

The change in Q_A acts as a negative clock edge for FF-B. So it will also toggle, and Q_B will be 1. $Q_BQ_A = 10$ after the second clock pulse.

After 3rd negative clock edge

On the arrival of 3rd negative clock edge, FF-A toggles again and Q_A become 1 from 0.

Since this is a positive going change, FF-B does not respond to it and remains inactive.

So Q_B does not change and continues to be equal to 1.

$Q_BQ_A = 11$ after the third clock pulse.

After 4th negative clock edge

On the arrival of 4th negative clock edge, FF-A toggles again and Q_A becomes 1 from 0.

This negative change in Q_A acts as clock pulse for FF-B. Hence it toggles to change Q_B from 1 to 0. $Q_BQ_A = 00$ after the fourth clock pulse.

Clock	Counter output		State number	Deciimal Counter output
	Q_B	Q_A		
Initially	0	0	—	0
1st	0	1	1	1
2nd	1	0	2	2
3rd	1	1	3	3
4th	0	0	4	0

Fig.2.30. Truth Table of the Ripple Counter

2.11.2. SYNCHRONOUS COUNTERS

If the "clock" pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter.

2-bit Synchronous up counter

The J_A and K_A inputs of FF-A are tied to logic 1. So FF-A will work as a toggle flip-flop. The J_B and K_B inputs are connected to Q_A.

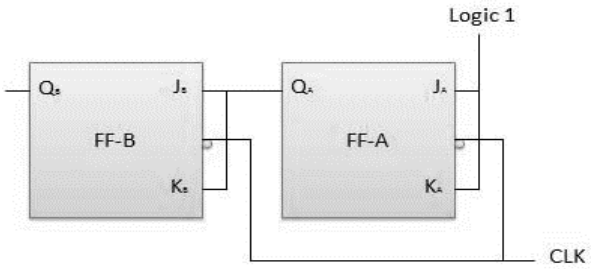


Fig.2.31. Block Diagram of Synchronous Counters

Operations:

Initially let both the FFs be in the reset state

Q_BQ_A = 00 initially.

After 1st negative clock edge

As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will change from 0 to 1.

But at the instant of application of negative clock edge, Q_A , $J_B = K_B = 0$. Hence FF-B will not change its state. So Q_B will remain 0.

Q_BQ_A = 01 after the first clock puls.

After 2nd negative clock edge

On the arrival of second negative clock edge, FF-A toggles again and Q_A changes from 1 to 0.But at this instant Q_A was 1. So $J_B = K_B = 1$ and FF-B will toggle. Hence Q_B changes from 0 to 1.

Q_BQ_A = 10 after the second clock pulse.

After 3rd negative clock edge

On application of the third falling clock edge, FF-A will toggle from 0 to 1 but there is no change of state for FF-B.

Q_BQ_A = 11 after the third clock pulse.

After 4th negative clock edge

On application of the next clock pulse, Q_A will change from 1 to 0 as Q_B will also change from 1 to 0.

Q_BQ_A = 00 after the fourth clock pulse.

2.12. CLASSIFICATION OF COUNTERS

Depending on the way in which the counting progresses, the synchronous or asynchronous counters are classified as follows :

- Up counters
- Down counters
- Up/Down counters

Up counter and down counter is combined together to obtain an UP/DOWN counter. A mode control (M) input is also provided to select either up or down mode. A combinational circuit is required to be designed and used between each pair of flip-flop in order to achieve the up/down operation.

Type of up/down counters:

- UP/DOWN ripple counters
- UP/DOWN synchronous counter

2.12.1. UP/DOWN Ripple Counter

In the UP/DOWN ripple counter all the FFs operate in the toggle mode. So either T flip-flops or JK flip-flops are to be used. The LSB flip-flop receives clock directly. But the clock to every other FF is obtained from (Q = Q bar) output of the previous FF.

UP counting mode (M=0) − The Q output of the preceding FF is connected to the clock of the next stage if up counting is to be achieved. For this mode, the mode select input M is at logic 0 (M=0).

DOWN counting mode (M=1) − If M = 1, then the Q bar output of the preceding FF is connected to the next FF. This will operate the counter in the counting mode.

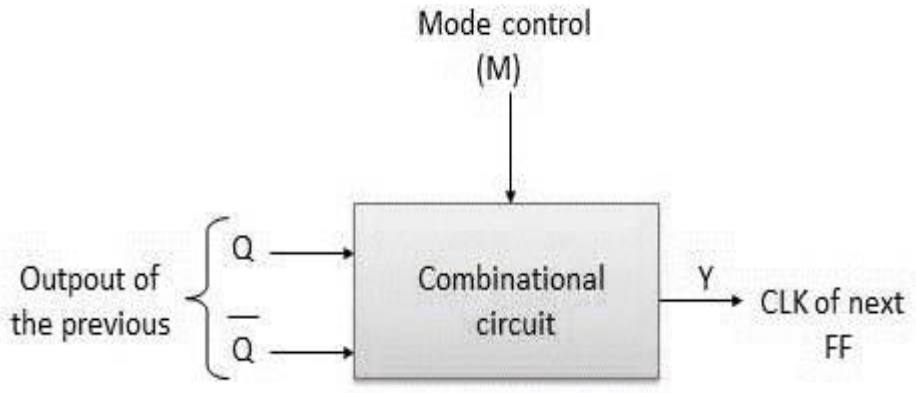


Fig.2.31. Combinational Circuit of UP/Down Ripple Counter

Inputs			Outputs	
M	Q	$\overline{Q}$	Y	
0	0	0	0	Y = Q for up counter
0	0	1	0	
0	1	0	1	
0	1	1	1	
1	0	0	0	$Y = \overline{Q}$ for up counter
1	0	1	1	
1	1	0	0	
1	1	1	1	

Fig.2.32. Truth Table of Ripple Counter

Operations:

Case 1 − With M = 0 (Up counting mode)

If M = 0 and M bar = 1, then the AND gates 1 and 3 in fig. will be enabled whereas the AND gates 2 and 4 will be disabled.

Hence Q_A gets connected to the clock input of FF-B and Q_B gets connected to the clock input of FF-C.

These connections are same as those for the normal up counter. Thus with M = 0 the circuit work as an up counter.

Case 2: With M = 1 (Down counting mode)

If M = 1, then AND gates 2 and 4 in fig. are enabled whereas the AND gates 1 and 3 are disabled.

Hence Q_A bar gets connected to the clock input of FF-B and Q_B bar gets connected to the clock input of FF-C.

These connections will produce a down counter. Thus with M = 1 the circuit works as a down counter.

The 2-bit ripple counter is called as MOD-4 counter and 3-bit ripple counter is called as MOD-8 counter. So in general, an n-bit ripple counter is called as modulo-N counter. Where, MOD number = 2^n.

Type of modulus

- 2-bit up or down (MOD-4)
- 3-bit up or down (MOD-8)
- 4-bit up or down (MOD-16)

Application of counters

- Frequency counters
- Digital clock
- Time measurement
- A to D converter
- Frequency divider circuits
- Digital triangular wave generator.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – III – SBSA1303 – COMPUTER ARCHITECTURE

3.1. MEMORY

A memory is just like a human brain. It is used to store data and instructions. Computer memory is the storage space in the computer, where data is to be processed and instructions required for processing are stored. Memory have divided into small space called cells. Each cell have its unique (own) address.

Address will various from 0 to memory size -1.

Example, If your computer have 32words 32* 1024 = 32,768 words. Memory size will be 0 to 32,767.

Memory organization Hierarchy

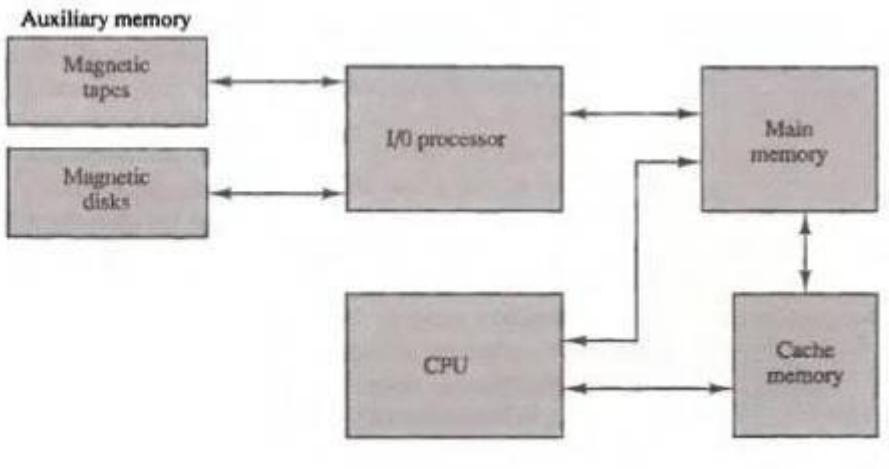


Fig.3.1. Memory Hierarchy in a Computer System

MEMORY TYPES

- Cache Memory
- Virtual Memory
- Auxiliary Memory
- Associative Memory

3.1.1. CACHE MEMORY

Cache memory is a very high speed semiconductor memory which can speed up the CPU. It acts as a buffer between the CPU and the main memory.

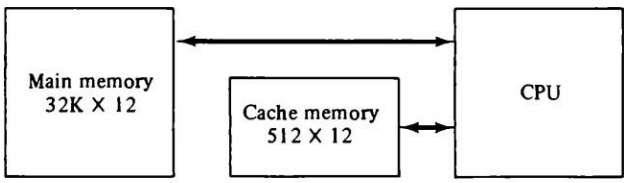


Figure 12-10 Example of cache memory.

Fig.3.2. Example of Cache Memory

Operation of Cache Memory:

When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the fast memory. If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word.

A block of words containing the one just accessed is then transferred from main memory to cache memory. The block size may vary from one word (the one just accessed) to about 16 words adjacent to the one just accessed. In this manner, some data are transferred to cache so that future references to memory find the required words in the fast cache memory. The performance of cache memory is frequently measured in terms of a quantity called hit ratio.

The transformation of data from main memory to cache memory is referred to as a mapping process.

Three types of mapping procedures are of practical interest when considering the organization of cache memory:

1. Associative mapping
2. Direct mapping
3. Set-associative mapping

Associative Mapping

The fastest and most flexible cache organization uses an associative memory.

The associative memory stores both the address and content (data) of the memory word.

The address value of 15 bits is shown as a five-digit octal number and its corresponding 12 -bit word is shown as a four-digit octal number. A CPU address of 15 bits is placed in the argument register and the 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.

If no match occurs, the main memory is accessed for the word.

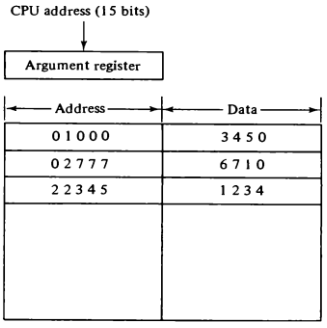


Fig.3.3. Associative Mapping Cache

Direct Mapping

Associative memories are expensive compared to random-access memories because of the added logic associated with each cell.

The CPU address of 15 bits is divided into two fields. The nine least significant bits constitute the index field and the remaining six bits form the tag field.

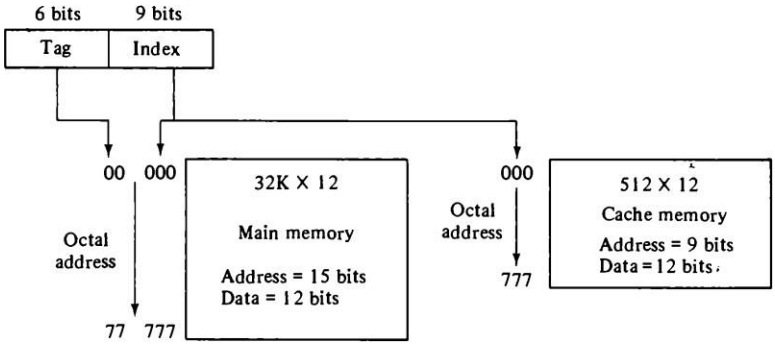


Fig.3.4. Addressing relationship between Main and Cache Memory

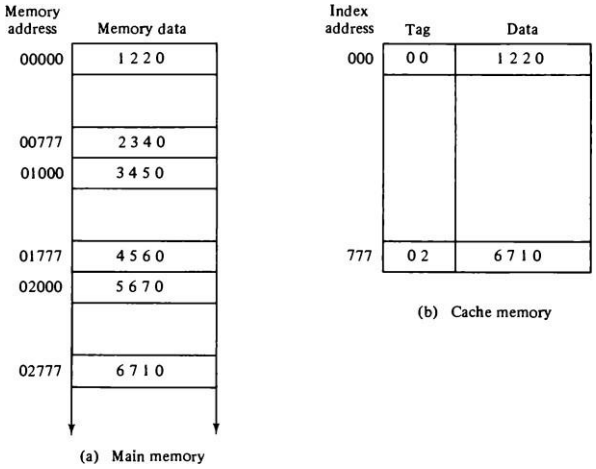


Fig.3.5. Direct Mapping cache organization

The main memory needs an address that includes both the tag and the index bits.

The number of bits in the index field is equal to the number of address bits required to access the cache memory. There are 2' words in cache memory and 2" words in main memory.

The n-bit memory address is divided into two fields: k bits for the index field and n - k bits for the tag field. The direct mapping cache organization uses the n-bit address to access the main memory and the k-bit index to access the cache.

Set-Associative Mapping

A third type of cache organization, called set-associative mapping, is an improvement over the direct mapping organization in that 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.

The size of cache memory is 512 x 36. It can accommodate 1024 words of main memory since each word of cache contains two data words. In general, a set-associative cache of set size k will accommodate k words of main memory in each word of cache.

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 then compared with both tags in the cache to determine if a match occurs.

The comparison logic is done by an associative search of the tags in the set similar to an associative memory search: thus the name "set-associative." The hit ratio will improve as the set size increases because more words with the same index but different tags can reside in cache.

ADVANTAGES :

- Cache memory is faster than main memory.
- It consumes less access time as compared to main memory.
- It stores the program that can be executed within a short period of time.It stores data for temporary use.

DISADVANTAGES

- Cache memory has limited capacity.
- It is very expensive.

3.3. ASSOCIATIVE MEMORY

The memory unit accessed by content is called an associative or Content Addressable memory (CAM). This type of memory accessed simultaneously and parallel on the basis of data content rather than the 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 a word.

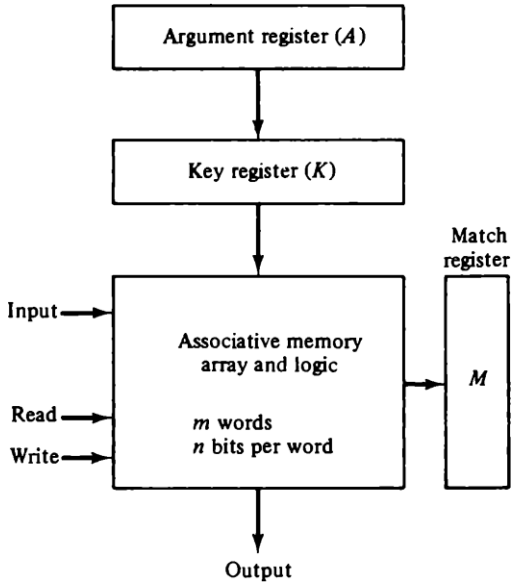


Fig.3.6. Block Diagram of Associative Memory

A	101 111100	
K	111 000000	
Word 1	100 111100	no match
Word 2	101 000001	match

Fig.3.7. Word Comparison

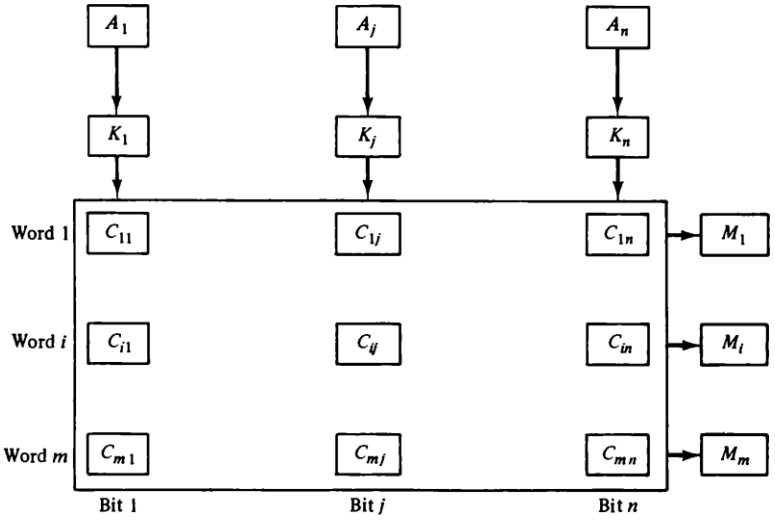


Fig.3.8. Associative Memory of m word, n cells per word

3.4.VIRTUAL MEMORY

Virtual memory is the separation of logical memory from physical memory. This separation provides large virtual memory for programmers when only small physical memory is available.

Virtual memory is used to give programmers the illusion that they have a very large memory even though the computer has a small main memory. It makes the task of programming easier because the programmer no longer needs to worry about the amount of physical memory available.

An address used by a programmer will be called a virtual address, and the set of such addresses the address space. An address in main memory is called a location or physical address.

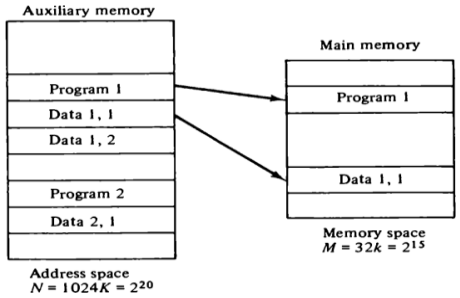


Fig.3.9.Relation Between Address and memory space in a virtual world

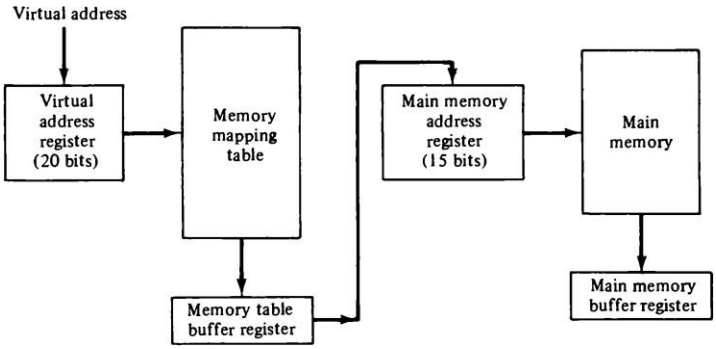


Fig.3.10. Memory table for mapping a virtual address

Consider a computer with a main-memory capacity of 32K words (K = 1024). Fifteen bits are needed to specify a physical address in · memory since 32K = 215• Suppose that the computer has available auxiliary memory for storing 220 = 1024K words.
Thus auxiliary memory has a capacity for storing information equivalent to the capacity of 32 main memories. Denoting the address space by N and the memory space by M, we then have for this example N = 1024K and M = 32K.

3.5. PERIPHERAL DEVICES

Input or output devices that are connected to computer are called peripheral devices. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be the part of computer system. These devices are also called peripherals.
Example: *Keyboards*, *display units* and *printers* are common peripheral devices.

Primary memory holds only those data and instructions on which the computer is currently working. It has a limited capacity and data is lost when power is switched off. It is generally made up of semiconductor device.
These memories are not as fast as registers. The data and instruction required to be processed resides in the main memory. It is divided into two subcategories RAM and ROM.
There are three types of peripherals:

Input peripherals : Allows user input, from the outside world to the computer.
Example: Keyboard, Mouse etc.
Output peripherals: Allows information output, from the computer to the outside world. Example: Printer, Monitor etc.
Input-Output peripherals: Allows both input(from outised world to computer) as well as, output(from computer to the outside world). E.g. Touch screen etc.

3.6. INTERFACES

Interface is a shared boundary between two separate components of the computer system which can be used to attach two or more components to the system for communication purposes.

There are two types of interface:

- CPU Interface
- I/O Interface

3.7. INPUT-OUTPUT INTERFACE

Peripherals connected to a computer need special communication links for interfacing with CPU. In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers. These components are called input-output interface units.

They provide communication links between processor bus and peripherals. They provide a method for transferring information between internal system and input-output devices.

3.7.1. MODES OF I/O DATA TRANSFER

Data transfer between the central unit and I/O devices can be handled in generally three types of modes which are given below:

- Programmed I/O
- Interrupt Initiated I/O
- Direct Memory Access

PROGRAMMED I/O

Programmed I/O instructions are the result of I/O instructions written in computer program. Each data item transfer is initiated by the instruction in the program.

Usually the program controls data transfer to and from CPU and peripheral. Transferring data under programmed I/O requires constant monitoring of the peripherals by the CPU.

In the programmed I/O method the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is time consuming process because it keeps the processor busy needlessly.

This problem can be overcome by using **interrupt initiated I/O**. In this when the interface determines that the peripheral is ready for data transfer, it generates an interrupt. After receiving the interrupt signal, the CPU stops the task which it is processing and service the I/O transfer and then returns back to its previous processing task.

3.8. DIRECT MEMORY ACCESS

Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This technique is known as DMA. In this, the interface transfer data to and from the memory through memory bus. A DMA controller

manages to transfer data between peripherals and memory unit.

Many hardware systems use DMA such as disk drive controllers, graphic cards, network cards and sound cards etc. It is also used for intra chip data transfer in multicore processors.
In DMA, CPU would initiate the transfer, do other operations while the transfer is in progress and receive an interrupt from the DMA controller when the transfer has been completed.

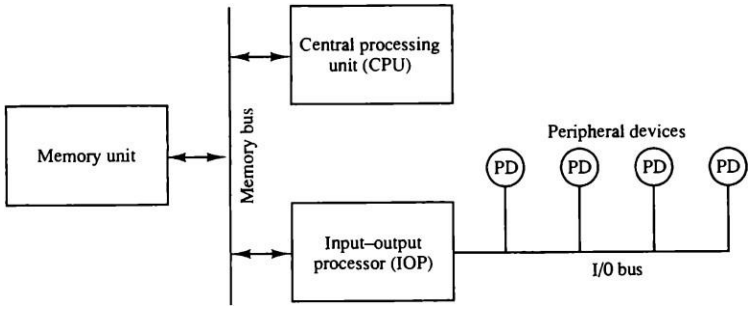


Fig. 3.11. Block Diagram of a Computer with I/O Processor

3.9. PRIORITY INTERRUPT

A priority interrupt is a system which decides the priority at which various devices, which generates the interrupt signal at the same time, will be serviced by the CPU.
The system has authority to decide which conditions are allowed to interrupt the CPU, while some other interrupt is being serviced.

Generally, devices with high speed transfer such as *magnetic disks* are given high priority and slow devices such as *keyboards* are given low priority. When two or more devices interrupt the computer simultaneously, the computer services the device with the higher priority first.

3.9.1. TYPES OF INTERRUPTS

Hardware Interrupts

When the signal for the processor is from an external device or hardware then this interrupts is known as hardware interrupt. Let us consider an example: when we press any key on our keyboard to do some action, then this pressing of the key will generate an interrupt signal for the processor to perform certain action.
Interrupt can be of two types:

Maskable Interrupt

The hardware interrupts which can be delayed when a much high priority interrupt has occurred at the same time.

Non Maskable Interrupt

The hardware interrupts which cannot be delayed and should be processed by the processor immediately.

3.10. SOFTWARE INTERRUPTS

The interrupt that is caused by any internal system of the computer system is known as a **software interrupt**. It can also be of two types:

Normal Interrupt

The interrupts that are caused by software instructions are called **normal software interrupts**.

Exception

Unplanned interrupts which are produced during the execution of some program are called **exceptions**, such as division by zero.

3.11. DAISY CHAINING PRIORITY

The interrupt priority consists of serial connection of all the devices which generates an interrupt signal. The device with the highest priority is placed at the first position followed by lower priority devices and the device which has lowest priority among all is placed at the last in the chain.

In daisy chaining system all the devices are connected in a serial form. The interrupt line request is common to all devices. If any device has interrupt signal in low level state then interrupt line goes to low level state and enables the interrupt input in the CPU.

When there is no interrupt the interrupt line stays in high level state. The CPU respond to the interrupt by enabling the interrupt acknowledge line. This signal is received by the device 1 at its PI input. The acknowledge signal passes to next device through PO output only if device 1 is not requesting an interrupt.

This problem is solved by following mechanism:

- Strobe
- Handshaking

Data is transferred from source to destination through data bus in between.

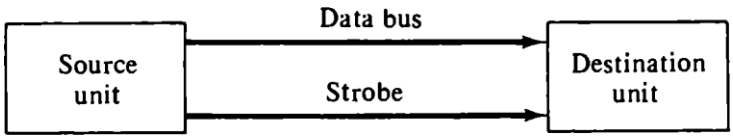
Strobe Mechanism:

Source initiated Strobe – When source initiates the process of data transfer. Strobe is just a signal.

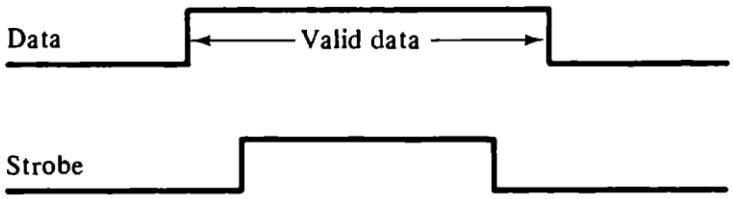
(i) First, source puts data on the data bus and on the strobe signal.

(ii) Destination on seeing the ON signal of strobe, read data from the data bus.

(iii) After reading data from the data bus by destination, strobe gets OFF.



(a) Block diagram



(b) Timing diagram

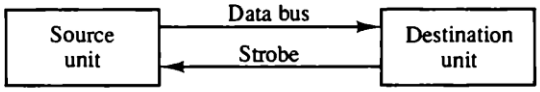
Fig.3.12. Source Initiated strobe

Destination initiated signal – When destination initiates the process of data transfer.

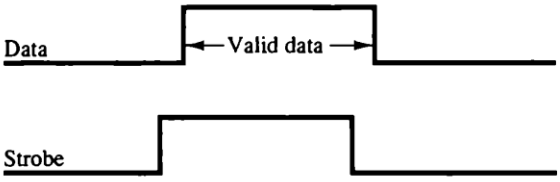
(i) First, the destination ON the strobe signal to ensure the source to put the fresh data on the data bus.

(ii) Source on seeing the ON signal puts fresh data on the data bus.

(iii) Destination reads the data from the data bus and strobe gets OFF signal.



(a) Block diagram



(b) Timing diagram

Fig.3.13. Destination Initiated Signal

Problems faced in Strobe based asynchronous input output

In Source initiated Strobe, it is assumed that destination has read the data from the data bus but their is no surety. In Destination initiated Strobe, it is assumed that source has put the data on the data bus but their is no surety. This problem is overcome by **Handshaking**.

Handshaking

Handshaking Mechanism

When source initiates the data transfer process. It consists of signals:

DATA VALID: if ON tells data on the data bus is valid otherwise invalid.

DATA ACCEPTED: if ON tells data is accepted otherwise not accepted.

(i) Source places data on the data bus and enable Data valid signal.

(ii) Destination accepts data from the data bus and enable Data accepted signal.

(iii) After this, disable Data valid signal means data on data bus is invalid now.

(iv) Disable Data accepted signal and the process ends.

Now there is surety that destination has read the data from the data bus through data accepted signal.

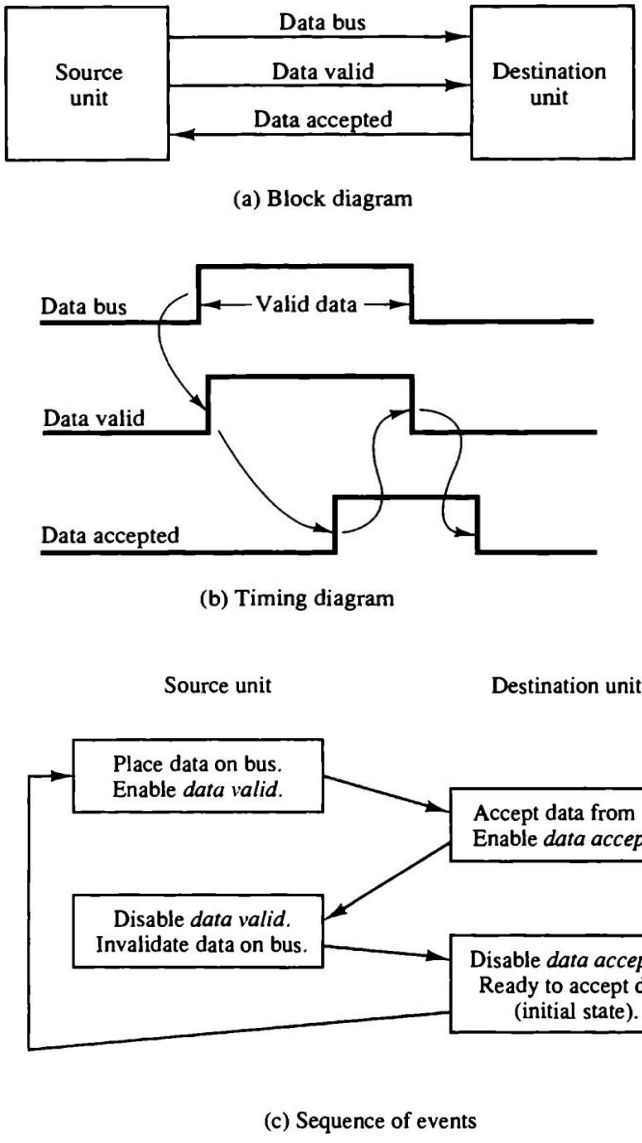


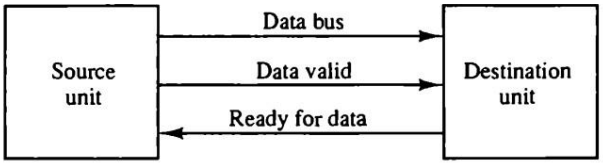
Fig.3.14. Source Initiated transfer using handshaking

(i) When destination is ready to receive data, Request for Data signal gets activated.

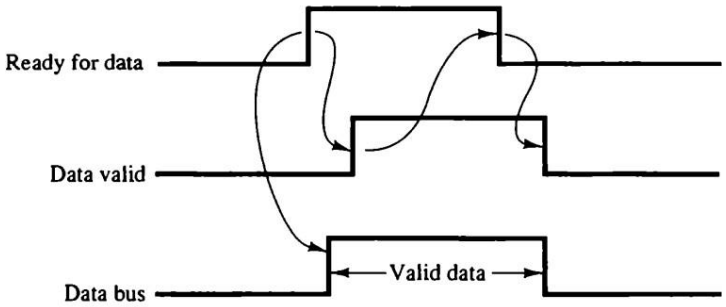
(ii) source in response puts data on the data bus and enabled Data valid signal.

(iii) Destination then accepts data from the data bus and after accepting data, disabled Request for Data signal.

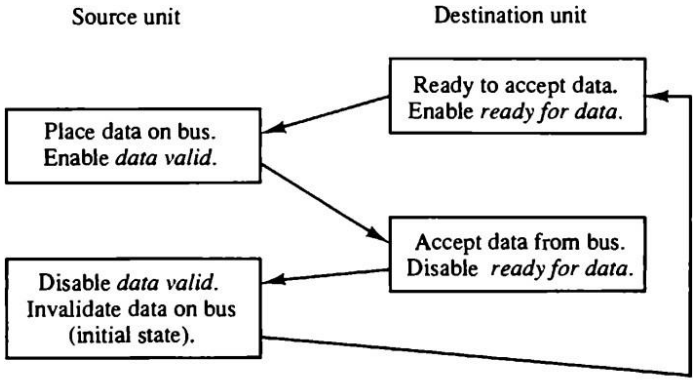
(iv) At last, Data valid signal gets disabled means data on the data bus is no more valid data. Now there is surety that source has put the data on the data bus through data valid signal.



(a) Block diagram



(b) Timing diagram



(c) Sequence of events

Fig.3.15. Destination Initiated transfer using Hanshaking

3.12. SERIAL COMMUNICATION

Serial Communication is a communication technique used in telecommunications wherein data transfer occurs by transmitting data one bit at a time in a sequential order over a computer bus or a communication channel. It is the simplest form of communication between a sender and a receiver.

A data communication processor is an I/O processor that distributes and collects data from numerous remote terminals connected through telephone and other communication lines to the computer. It is a specialized I/O processor designed to communicate with data communication networks.

Such a communication network consists of variety of devices such as printers, display devices, digital sensors etc. serving many users at once.

The data communication processor communicates with each terminal through a single pair of wire. It also communicates with CPU and memory in the same manner as any I/O processor does.

MODEM

In a Data Communication Network, the remote terminals are connected to the data communication processor through telephone lines or other wires.

Such telephone lines are specially designed for voice communication and computers use them to communicate in digital signals, therefore some conversion is required. These conversions are called modem (modulator-demodulator). A modem converts digital signal into audio tones to be transmitted over telephone lines and also converts audio tones into digital signal for machine use.

Modes Of Transmission

- Simplex
- Half Duplex
- Full Duplex

Types of Protocols

Character Oriented Protocol

It is based on the binary code of character set. The code is mostly used in ASCII. It includes upper case and lower case letters, numerals and variety of special symbols. The characters that control the transmission is called communication control characters.

Bit Oriented Protocol

It does not use characters in its control field and is independent of any code. It allows the transmission of serial bit stream of any length without the implication of character boundaries.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

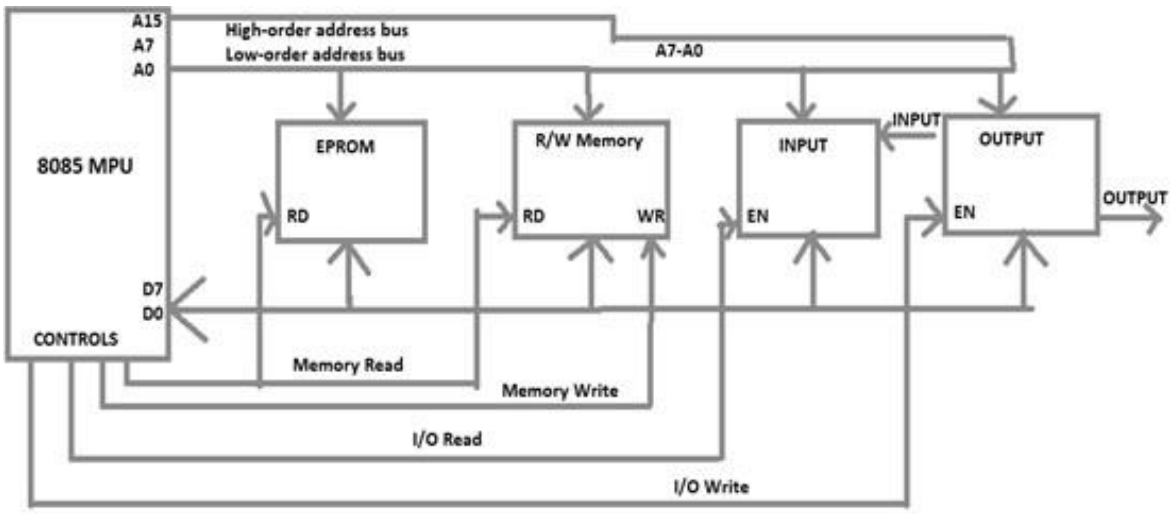
SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – IV – SBSA1303 – COMPUTER ARCHITECTURE

4.1. MICRO COMPUTER

Microcomputer is used to describe a system that includes a minimum of a microprocessor, program memory, data memory, and input–output (I/O). ..... Thus, a microcomputer system can be anything from a large computer having hard disks, floppy disks, and printers to a single-chip embedded controller.



A Microcomputer System

4.2. ASSEMBLY LANGUAGE PROGRAMMING

Assembly language programming is low-level **programming** using some basic syntax to represent machine **code** for a specific CPU.

An **assembler** is used to translate the **assembly code** into the machine **code** for the target computer.

A **program** created from assembly can be more efficient and faster than a program created with a compiler.

Each family of processors has its own set of instructions for handling various operations such as getting input from keyboard, displaying information on screen and performing various other jobs. These set of instructions are called 'Machine Language Instructions'.

Assemply Language Program

2000 LDA 2050 A<-[2050]

2003 MOV H, AH<-A

2004 LDA 2051 A<-[2051]

2007 ADD HA<-A+H

2006 MOV L, AL←A

2007 MVI A 00A←00

2009 ADC A A←A+A+carry

200A MOV H, AH←A

200B SHLD 3050 H→3051, L→3050

200 EHLT

4.3. MICROPROCESSORS

It is an 8-bit microprocessor designed by Intel in 1977 using NMOS(N-channel metal-oxide semiconductor is a microelectronic circuit) technology.

It has the following configuration −

8-bit data bus

16-bit address bus, which can address upto 64KB

A 16-bit program counter

A 16-bit stack pointer

Six 8-bit registers arranged in pairs: BC, DE, HL

Requires +5V supply to operate at 3.2 MHZ single phase clock

It is used in washing machines, microwave ovens, mobile phones, etc.

8085 Microprocessor – Functional Units

4.3.1. 8085 consists of the following functional units :

Accumulator

- It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

Arithmetic and logic unit

- As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

General purpose register

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H L. Each register can hold 8-bit data.

These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

STACK POINTER

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

Temporary Register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

Flag Register

R

- It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops:

D7	D6	D5	D4	D3	D2	D1	D0

- Sign (S),Zero (Z),Auxiliary Carry (AC),Parity (P),Carry (C)

Its bit position is shown in the following table :

- It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops:

- Sign (S),Zero (Z),Auxiliary Carry (AC),Parity (P),Carry (C)

Its bit position is shown in the following table :

S	Z		AC		P		CY

Instruction Register And Decoder

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

Timing And Control Unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits .

- Control Signals: READY, RD, WR, ALE

- Status Signals: S0, S1, IO/M
- DMA Signals: HOLD, HLDA
- RESET Signals: RESET IN, RESET OUT

4.4. INTERRUPT CONTROL

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

4.5. SERIAL INPUT/OUTPUT CONTROL

It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

Address Buffer And Address-Data Buffer

The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

Address Bus And Data Bus

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

8085 MICROPROCESSOR ARCHITECTURE

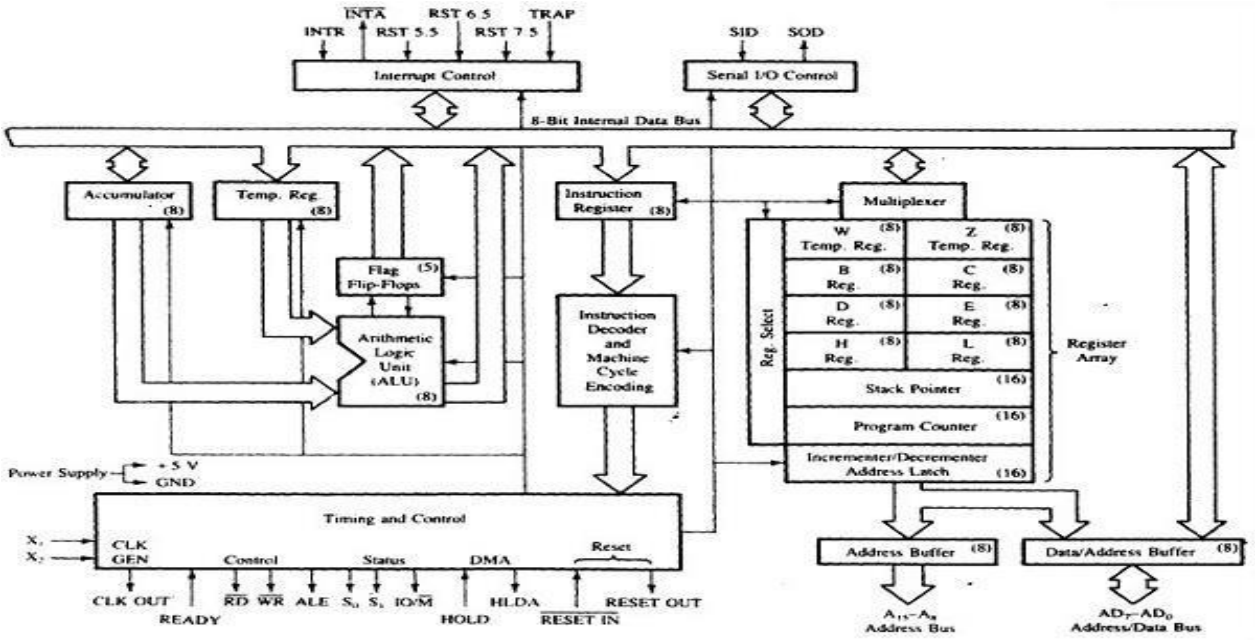


Fig.4.1. Architecture diagram of Microprocessor

The pins of a 8085 microprocessor can be classified into seven groups −

Address bus

- A15-A8, it carries the most significant 8-bits of memory/IO address.

Data bus

- AD7-AD0, it carries the least significant 8-bit address and data bus.

Control And Status Signals

These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.

Three control signals are RD, WR & ALE.

RD − This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.

WR − This signal indicates that the data on the data bus is to be written into a selected memory or IO location.

ALE − It is a positive going pulse generated when a new operation is started

by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three status signals are IO/M, S0 & S1.

IO/M

- This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

S1 & S0

- These signals are used to identify the type of current operation.

POWER SUPPLY

- There are 2 power supply signals − VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

CLOCK SIGNALS

There are 3 clock signals, i.e. X1, X2, CLK OUT.

X1, X2 − A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.

CLK OUT − This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & Externally Initiated Signals

- Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

INTA − It is an interrupt acknowledgment signal.

RESET IN − This signal is used to reset the microprocessor by setting the program counter to zero.

RESET OUT − This signal is used to reset all the connected devices when the microprocessor is reset.

READY − This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.

HOLD − This signal indicates that another master is requesting the use of the address and data buses.

HLDA (HOLD Acknowledge) − It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

SERIAL I/O SIGNALS

There are 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.

SOD (Serial output data line) − The output SOD is set/reset as specified by the SIM instruction.

SID (Serial input data line) − The data on this line is loaded into accumulator whenever a RIM instruction is executed.

8085 INSTRUCTION SET

Instruction set classified into 5 categories:

- CONTROL INSTRUCTIONS
- LOGICAL INSTRUCTIONS
- BRANCING ARTHMETIC INSTRUCTIONS
- DATA TRANSFER INSTRUCTIONS

Control Instructions

- NOP, HLT, DI,EI,RIM, SIM

Logical Instructions

- CMP,CPI,ANA,ANI,XRA,XRI,RLC,
- RRC, RAL,RAR,CMA,CMC,STC

Branching Instructions

- JMP,RET,PCHL,RST

Arithmetic Instructions

- ADD,ADC,ADI,ACI,LXI,DAD,SUB,SBB,SUI,SBI,INR,DCX,
- DAA

Data Transfer Instructions

- MOV,MVI,LDA,LDAX,LXI,LHLD,STA,STAX,SHLD,XCHG,
- SPHL,PUSH,POP,OUT,IN

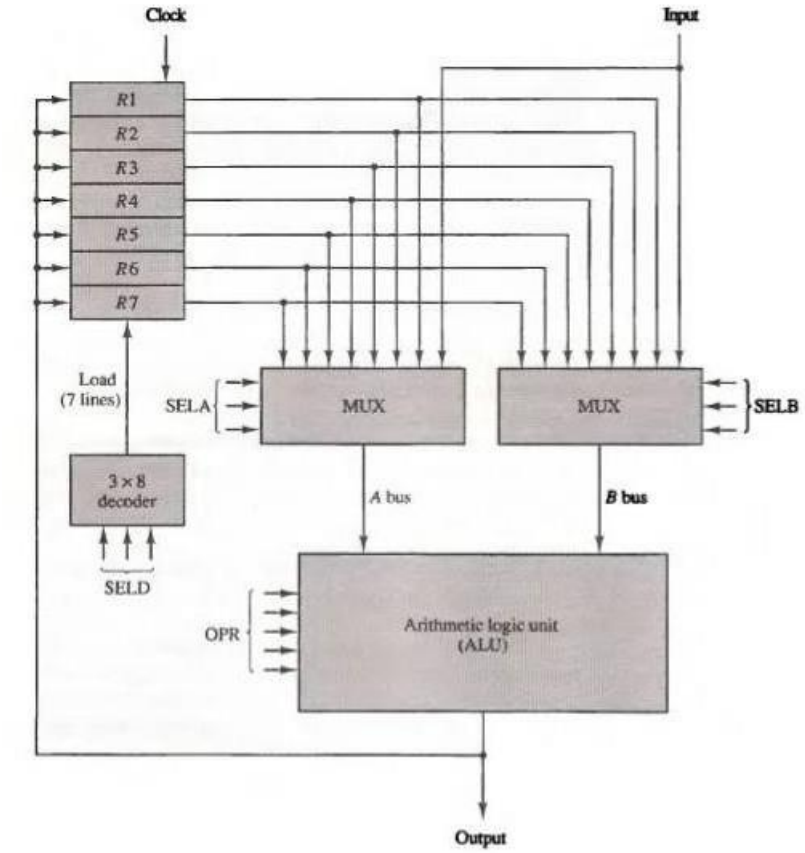
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – V– SBSA1303 – COMPUTER ARCHITECTURE

5.1. GENERAL PURPOSE REGISTERS



The output of each register is connected to two multiplexers (MUX) to form the two buses A and B.

The selection lines in each multiplexer select one register or the input data for the particular bus.

The A and B buses form the inputs to a common arithmetic logic unit (ALU). The operation selected in the ALU determines the arithmetic or logic micro operation that is to be performed.

The result of the micro operation is available for output data and also goes into the inputs of all the registers.The register that receives the information from the output bus is selected by a decoder. The decoder activates one of the register load inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination register.

The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system.

For example, to perform the operation

R1<--R2 + R3

the control must provide binary selection variables to the following selector inputs:

1. MUX A selector (SELA): to place the content of R2 into bus A.

2. MUX B selector (SELB): to place the content of R3 into bus B.

3. ALU operation selector (OPR): to provide the arithmetic addition A+ B. 4. Decoder destination selector (SELD): to transfer the content of the output bus into R 1.

5.1.1.CONTROL WORD

The combined value of a binary selection inputs specifies the control word.

It consist of four fields SELA,SELB,and SELD or SELREG contains three bit each and SELOPR field contains four bits thus the total bits in the control word are 13-bits.

The three bit of SELA select a source registers of the a input of the ALU.

The three bits of SEL B select a source registers of the b input of the ALU.

The three bits of SEL D or SEL REG select a destination register using the decoder.

The four bits of SEL OPR select the operation to be performed by ALU.

5.2. STACK ORGANIZATION

A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.

The operation of a stack can be compared to a stack of trays. The last tray placed on top of the stack is the first to be taken off.

The stack in digital computers is essentially a memory unit with an address register that can count only (after an initial value is loaded into it). The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.

The physical registers of a stack are always available for reading or writing. It is the content of the word that is inserted or deleted.

The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.

The physical registers of a stack are always available for reading or writing. It is the content of the word that is inserted or deleted.

Types of Stack

- Register Stack
- Memory Stack
- Register Stack
- A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers.

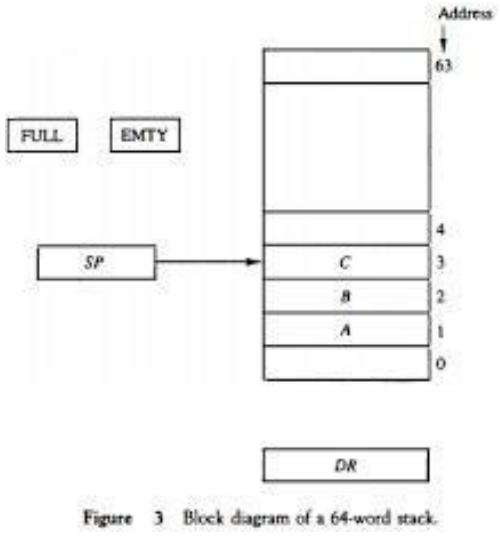


Figure 3 Block diagram of a 64-word stack.

Fig.5.2. Register Stack

The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B, and C, in that order.

Item C is on top of the stack so that the content of SP is now 3. To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP.

Item B is now on top of the stack since SP holds address 2. To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack. Note that item C has been read out but not physically removed.

Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with a push operation.

The push operation is implemented with the following sequence of microoperations:

SP <-SP + 1 - Increment stack pointer

M[SP]<-DR - Write item on top of the stack

If (SP = 0) then (FULL <--1) - Check if stack is full

EMTY <--0 - Mark the stack not empty

A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequence of microoperations:

DR <--M[SP] -Read item from the top of stack

SP<--SP - 1 - Decrement stack pointer

If (SP = 0) then (EMTY <--1) -Check if stack is empty

FULL <--0 - Mark the stack not full

The top item is read from the stack into DR. The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1.

This condition is reached if the item read was in location L Once this item is read out, SPis decremented and reaches the value 0, which is the initial value of SP.

Note that if a pop operation reads the item from location 0 and then SP is decremented, SP changes to 111111, which is equivalent to decimal 63.

In this configuration, the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when FULL = 1 or popped when EMTY = 1.

5.2.1.Memory Stack

The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer.

A portion of computer memory partitioned into three segments: program, data, and stack.

The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data.

The stack pointer SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory.

PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or pop items into or & from the stack.

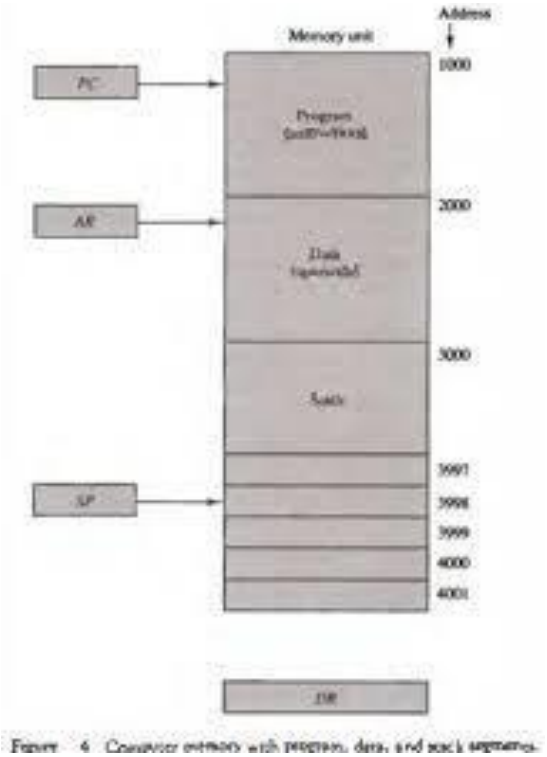


Fig.5.3. Memory Stack

The initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 40,00 the second item is stored at address 39,99 and the last address that can be used for the stack Is 3000. No provisions are available for stack limit checks.

We assume that the items in the stack communicate with a data register DR. A new item is inserted with the push operation as follows:

SP<-SP - 1

[SP]<-DR

The stack pointer is decremented so that it points at the address of the next word. A memory write operation inserts the word from DR into the top of the stack.

A new item is deleted with a pop operation as follows:

DR <-M[SP]

SP<-SP + 1

The top item is read from the stack into DR. The stack pointer is then incremented to point at the next item in the stack.

5.3. INSTRUCTION FORMATS

The physical and logical structure of computers is normally described in reference manuals provided with the system. Such manuals explain the internal construction of the CPU, including the processor registers available and their logical capabilities.

They list all hardware-implemented instructions, specify their binary code format, and provide a precise definition of each instruction.

The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

An operation code field that specifies the operation to be performed. An address field that designates a memory address or a processor register. A mode field that specifies the way the operand or the effective address is determined.

The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift.

The most common operations available in computer instructions are enumerated and discussed.

Three types of CPU organizations:

Single accumulator organization.

– All operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field.

– E.g ADD X

General register organization.

– The instruction format in this type of computer needs three register address fields.

– E.g. ADD R1, R2, R3

Stack organization.

– Computers with stack organization would have PUSH and POP instructions which require an address field.

– E.g PUSH X

Three Address Instructions

Two Address Instructions

One Address Instructions

5.3.1.Three Address Instructions

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.

E.g. X = (A + B) • (C + D)

ADD	R1, A, B	R1<-- M[A] + M[B]
ADD	R2, C, D	R2<-- M[C] + M[D]
MOL	X, R1, R2	M[X] <-- R1•R2

The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions.

The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

5.3.2TWO ADDRESS INSTRUCTIONS

Two-address instructions are the most common in commercial computers.

MOV R1, A R1<--M[A]

ADD R1, B R1 <-- R1 + M[B]

MOV R2, C R2<-- M[C]

ADD R2, D R2<-- R2 + M[D]

MUL R1,R2 R1 <-- R1• R2

MOV X, R1 M[X] <-- R1

The MOV instruction moves or transfers the operands to and from memory and processor registers.

The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

5.3.3. One Address Instructions

One-address instructions use an implied accumulator (AC) register for all data manipulation.

For multiplication and division there is a need for a second register.

LOAD	A	AC<-M[A]
ADD	B	AC <-AC + M[B]
STORE	T	M[T] <-AC
LOAD	C	AC<-M[C]
ADD	D	AC<-AC + M[D]
MUL	T	AC<-AC•M[T]
STORE	X	M[X] <-AC

All operations are done between the All operations are done between the AC register and a memory operand.

T is the address of a temporary memory location required for storing the intermediate result.

5.3.4. Zero Address Instructions

A stack-organized computer does not use an address field for the instructions ADD and MUL.

The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.

PUSH	A	TOS<-A
PUSH	B	TOS<-B
ADD		TOS<-(A+B)
PUSH	C	TOS <-C
PUSH	D	TOS<-D
ADD		TOS<-(C+D)

MUL TOS<-(C +D)•(A+B)

POP X M[X]<-TOS

The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

5.4. ADDRESSING MODES

Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:

1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.

2. To reduce the number of bits in the addressing field of the instruction.

The control unit of a computer is designed to go through an instruction cycle that is divided into three major phases:

1. Fetch the instruction from memory.

2. Decode the instruction.

3. Execute the instruction.

Program Counter (PC) There is one register in the computer called the program counter or PC that keeps track of the instructions in the program stored in memory.

Mode Field : The mode field is used to locate the operands needed for the operation.

Implied Mode: In this mode the operands are specified implicitly in the definition of the instruction. For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.

E.g. CLC (used to reset Carry flag to 0)

Immediate addressing mode: In this mode data is present in address field of instruction .Designed like one address instruction format.

E.g. MOV AL, 35H (move the data 35H into AL register)

Register mode: In register addressing the operand is placed in one of 8 bit or 16 bit general purpose registers. The data is in the register that is specified by the instruction. Here one register reference is required to

access the data. E.g. MOV AX,CX (move the contents of CX register to AX register)

Register Indirect Mode : In this mode, the instruction specifies the register whose contents give us the address of operand which is in memory.

Thus, the register contains the address of operand rather than the operand itself.

E.g. MOV AX, [BX](move the contents of memory location s addressed by the register BX to the register AX)

Auto Increment Mode: Effective address of the operand is the contents of a register specified in the instruction.

After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location.

E.g. Add R1, (R2)+ // OR

R1 = R1 +M[R2]

R2 = R2 + d

Direct Addressing/Absolute Addressing Mode : The operand's offset is given in the instruction as an 8 bit or 16 bit displacement element.

In this addressing mode the 16 bit effective address of the data is the part of the instruction.

E.g ADD AL,[0301] //add the contents of offset address 0301 to AL

Indirect Address Mode: In this mode the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

EFFECTIVE ADDRESS = address part of instruction + content of CPU register.

E.g : MOV AX, [SI +05]

Based Indexed Addressing: The operand's offset is sum of the content of a base register BX or BP and an index register SI or DI.

E.g.: ADD AX, [BX+SI]

PC Relative Addressing Mode: PC relative addressing mode is used to implement intra segment transfer of control, In this mode effective address is obtained by adding displacement to PC.EA= PC + Address field value PC= PC + Relative value.

Base Register Addressing Mode: Base register addressing mode is used to implement inter segment transfer of control.

In this mode effective address is obtained by adding base register value to address field value.

EA= Base register + Address field value.

PC= Base register + Relative value.

5.5. DATA TRNSFER AND MANIPULATION

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. The data manipulation instructions in a typical computer are usually divided into three basic types:

1. Arithmetic instructions

2. Logical and bit manipulation instructions

3. Shift instructions

5.5.1. Arithmetic Instructions

The four basic arithmetic operations are addition, subtraction, multiplication, and division. Most computers provide instructions for all four operations. Some small computers have only addition and possibly subtraction instructions.

The multiplication and division must then be generated by means of software subroutines.

Increment ,Decrement, Add ,Subtract ,Multiply, Divide, Add with carry ,Subtract with borrow Negate (2's complement).

ADDI Add two binary integer numbers

ADDF Add two floating-point numbers

ADDD Add two decimal numbers in BCD

The number of bits in any register is of finite length and therefore the results of arithmetic operations are of finite precision

5.5.2. Logical and bit manipulation instructions

Logical instructions perform binary operations on strings of bits stored in registers.

They are useful for manipulating individual bits or a group of bits that represent binary-coded information. The logical instructions consider each bit of the operand separately and treat it as a Boolean variable.

Clear ,Complement, AND, OR, Exclusive-OR, Clear, carry, Set carry ,Complement carry, Enable interrupt, Disable interrupt.

The complement instruction produces the 1's complement by inverting all the bits of the operand. The OR instruction is used to set a bit or a selected group of bits of an operand.

5.5.3. Shift Instructions

Instructions to shift the content of an operand are quite useful and are often provided in several variations.

Shifts are operations in which the bits of a word are moved to the left or right.

Some computers have a multiple-field format for the shift instructions.

One field contains the operation code and the others specify the type of shift and the number of times that an operand is to be shifted. A possible instruction code format of a shift instruction may include five fields as follows: OP REG TYPE RL COUNT

Here OP is the operation code field; REG is a register address that specifies the location of the operand.

Logical shift right, Logical shift left, Arithmetic shift right, Arithmetic shift left, Rotate right, Rotate left, Rotate right through carry, Rotate left through carry.

5.6. PROGRAM CONTROL

Instructions are always stored in successive memory locations. When processed in the CPU, the instructions are fetched from consecutive memory locations and executed.

Each time an instruction is fetched from memory, the program counter is incremented so that it contains the address of the next instruction in sequence.

After the execution of a data transfer or data manipulation instruction, control returns to the fetch cycle with the program counter containing the address of the instruction next in sequence.

A program control type of instruction, when executed, may change the address value in the program counter and cause the flow of control to be altered.

Branch and jump instructions may be conditional or unconditional.

An unconditional branch instruction causes a branch to the specified address without any conditions.

The conditional branch instruction specifies a condition such as branch if positive or branch if zero.

Branch, Jump, Skip, Call, Return, Name, Compare (by subtraction), Test (by ANDing).

Status Bit Conditions

Status bits are also called condition-code bits or flag bits.

The four status bits are symbolized by C. S, Z, and V.

The bits are set or cleared as a result of an operation performed in the ALU.

Bit C (carry) is set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry is 0.

Bit S (sign) is set to 1 if the highest-order bit F, is 1. It is set to 0 if the bit is 0.

Bit Z (zero) is set to 1 if the output of the ALU contains all O's. It is cleared to otherwise. In other words, Z = 1 if the output is zero and Z = 0 if the output is not zero.

Bit V (overflow) is set to 1 if the exclusive-OR of the last two carries is equal to 1, and cleared to 0 otherwise.

Conditional Branch Instructions

BZ BNZ BC BNC BP BM BY BNV

Subroutine Call And Return

A subroutine is a self-contained sequence of instructions that performs a given computational task.

The instruction that transfers program control to a subroutine is known by different names.

The most common names used are call subroutine, jump to subroutine, branch to subroutine, or branch and save address.

A call subroutine instruction consists of an operation code together with an address that specifies the beginning of the subroutine.

5.7. PROGRAM INTERRUPT

There are three major types of interrupts that cause a break in the normal execution of a program.

1. External interrupts

2. Internal interrupts

3. Software interrupts

5.8. REDUCED INSTRUCTION SET COMPUTER (RISC)

The instruction set chosen for a particular computer determines the way that machine language programs are constructed.

Early computers had small and simple instruction sets, forced mainly by the need to minimize the hardware used to implement them.

As digital hardware became cheaper with the advent of integrated circuits, computer instuctions tended to increase both in number and complexity. Many computers have instruction sets that include more than 100 and sometimes even more than 200 instructions.

A computer with a large number of instructions is classified as a complex instruction set computer, abbreviated CISC.

RISC Characteristics The concept of RISC architecture involves an attempt to reduce execution time by simplifying the instruction set of the computer.

The major characteristics of a RISC processor are:

1. Relatively few instructions

2. Relatively few addressing modes

3. Memory access limited to load and store instructions

4. All operations done within the registers of the CPU

5. Fixed-length, easily decoded instruction format

6. Single-cycle instruction execution

7. Hardwired rather than micro programmed control

5.9. PARALLEL PROCESSING

Parallel processing is a term used to denote a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.

The system may have two or more ALUs and be able to execute two or more instructions at the same time. The system may have two or more processors operating concurrently.

The purpose of parallel processing is to speed up the computer processing capability and increase its throughput, that is, the amount of processing that can be accomplished during a given interval of time.

Parallel processing at a higher level of complexity can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously.

Parallel processing is established by distributing the data among the multiple functional units.

For example: the arithmetic, logic, and shift operations can be separated into three units and the operands diverted to each unit under the supervision of a control unit.

Processor with Different Functional Unit

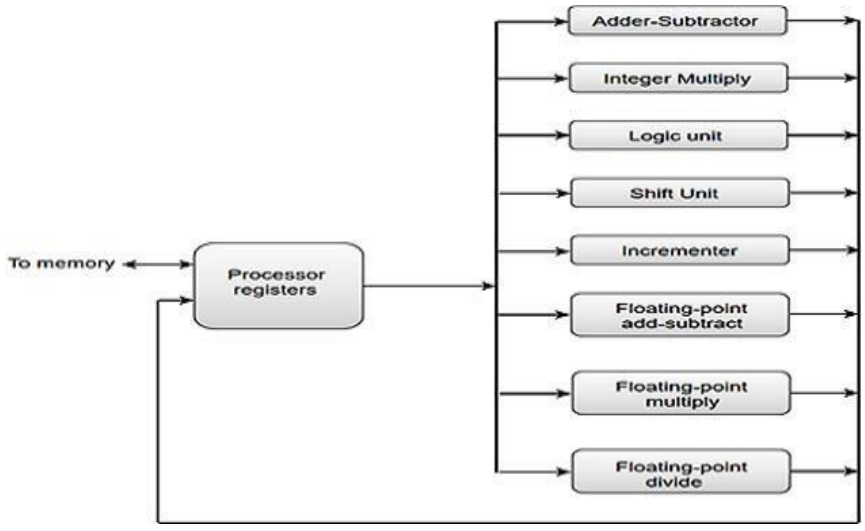


Fig.5.6. Processor with Different Functional Units

There are a variety of ways that parallel processing can be classified.

It can be considered from the internal organization of the processors, from the interconnection structure between processors, or from the flow of information through the system.

One classification introduced by M. J. Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.

The normal operation of a computer is to fetch instructions from memory and execute them in the processor.

The sequence of instructions read from memory constitutes an instruction stream. The operations performed on the data in the processor constitutes a data stream.

Parallel processing may occur in the instruction stream, in the data stream, or in both.

Flynn's classification divides computers into four major groups as follows:

1. Single instruction stream, single data stream (SISD)
2. Single instruction stream, multiple data stream (SIMD)
3. Multiple instruction stream, single data stream (MISD)
4. Multiple instruction stream, multiple data stream (MIMD)

SISD represents the organization of a single computer containing a control unit, a processor unit, and a memory unit.

Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities.

Parallel processing in this case may be achieved by means of multiple functional units or by pipeline processing.

SIMD represents an organization that includes many processing units under the supervision of a common control unit.

All processors receive the same instruction from the control unit but operate on different items of data.

The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.

MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

MIMD organization refers to a computer system capable of processing several programs at the same time. Most multiprocessor and multicomputer systems can be classified in this category.

Flynn's classification depends on the distinction between the performance of the control unit and the data-processing unit.

- 1. Pipeline processing
- 2. Vector processing
- 3. Array processors

Pipeline processing is an implementation technique where arithmetic sub operations or the phases of a computer instruction cycle overlap in execution.

Vector processing deals with computations involving large vectors and matrices. Array processors perform computations on large arrays of data.

5.10. PIPELINING

Pipelining is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.

The sub operations performed in each segment of the pipeline are as follows:

R1<--A,, R2<--B, Input A, and B,

R3<--R1•R2, R4<--C, Multiply and input C,

R5<--R3 + R4 Add C; to product

The five registers are loaded with new data every clock pulse

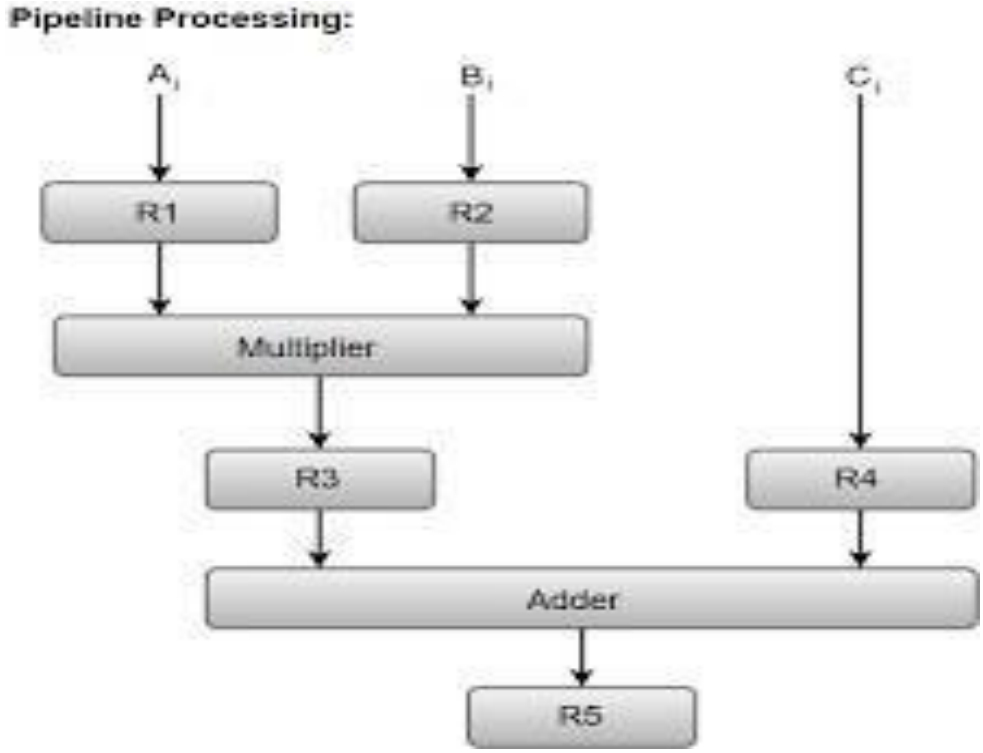


Fig.5.7. Pipeline Processing

TABLE 1 Content of Registers in Pipeline Example

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

The first clock pulse transfers A1 and 81 into R 1 and R2. The second dock pulse transfers the product of R 1 and R2 into R3 and C1 into R4.

The same clock pulse transfers A2 and B2 into R 1 and R2. The third clock pulse operates on all three segments simultaneously.

It places A, and B, into R1 and R2, transfers the product of R1 and R2 into R3, transfers C, into R4, and places the sum of R3 and R4 into RS.

It takes three clock pulses to fill up the pipe and retrieve the first output from RS.

Arithmetic Pipeline

Instruction Pipeline

5.10.1.ARITHMETIC PIPELINE

Pipeline arithmetic units are usually found in very high speed computers.

They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems. The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.

E.g : $X=A \times 2^a$ $Y = B \times 2^b$

A and B are two fractions that represent the mantissas and a and b are the exponents. The floating-point addition and subtraction can be performed in four segments.

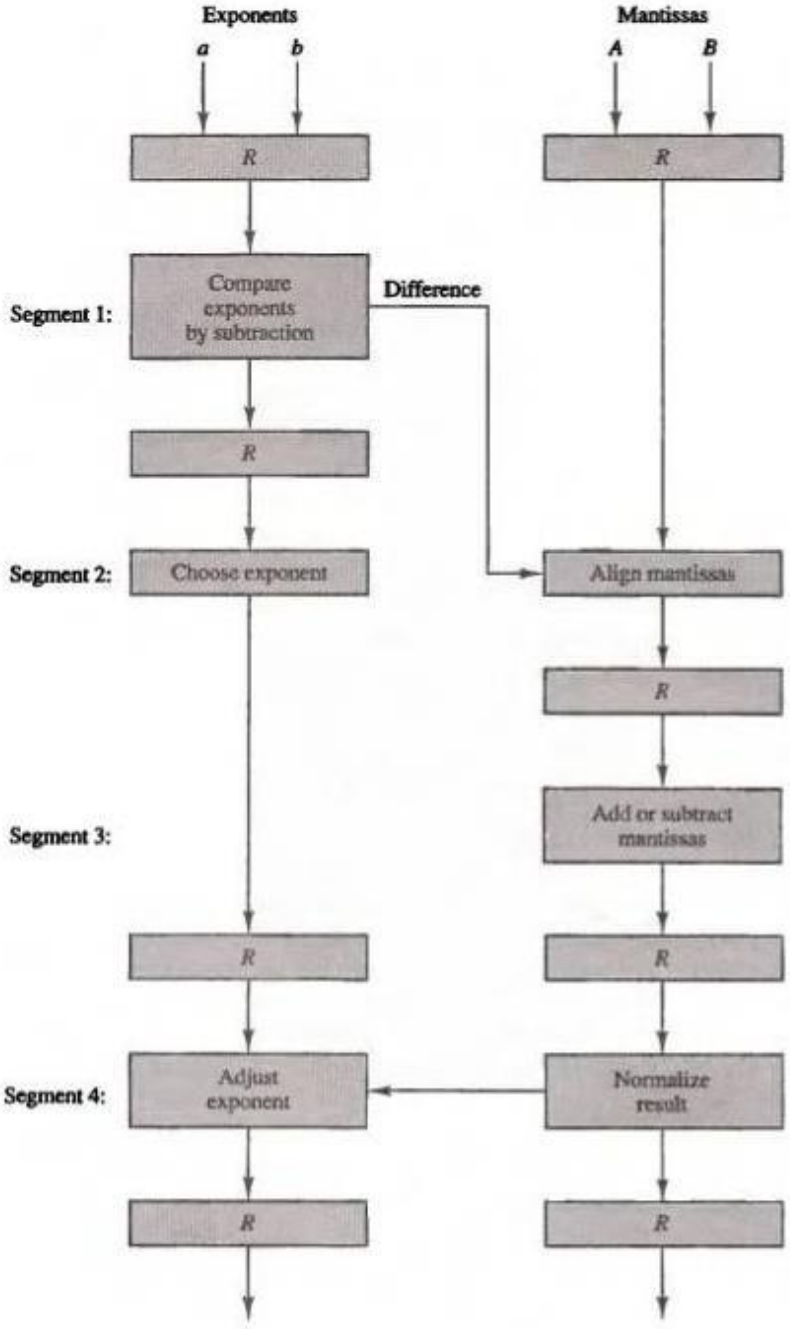


Figure 9-6 Pipeline for floating-point addition and subtraction.

The sub operations that are performed in the four segments are:

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result.

The exponents are compared by subtracting them to determine their difference.

The larger exponent is chosen as the exponent of the result.

The exponent difference determines how many times the mantissa associated with the smaller exponent must be shifted to the right. This produces an alignment of the two mantissas. It should be noted that the shift must be designed as a combinational circuit to reduce the shift time.

The two mantissas are added or subtracted in segment 3.

The result is normalized in segment 4. When an overflow occurs, the mantissa of the sum or difference is shifted right and the exponent incremented by one.

If an underflow occurs, the number of leading zeros in the mantissa determines the number of left shifts in the mantissa and the number that must be subtracted from the exponent.

5.10.2 INSTRUCTION PIPELINE

Pipeline processing can occur not only in the data stream but in the instruction stream as well.

An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments. This causes the instruction fetch and execute phases to overlap and perform simultaneous operations. Pipeline processing can occur not only in the data stream but in the instruction stream as well.

An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.

This causes the instruction fetch and execute phases to overlap and perform simultaneous operations. Computer with an instruction fetch unit and an instruction execution unit designed to provide a two-segment pipeline.

The instruction fetch segment can be implemented by means of a first-in, first-out (FIFO) buffer. This is a type of unit that forms a queue rather than a stack.

Whenever the execution unit is not using memory, the control increments the program counter and uses its address value to read consecutive instructions from memory.

The instructions are inserted into the FIFO buffer so that they can be executed on a first-in, first-out basis.

Computers with complex instructions require other phases in addition to the fetch and execute to process an instruction completely.

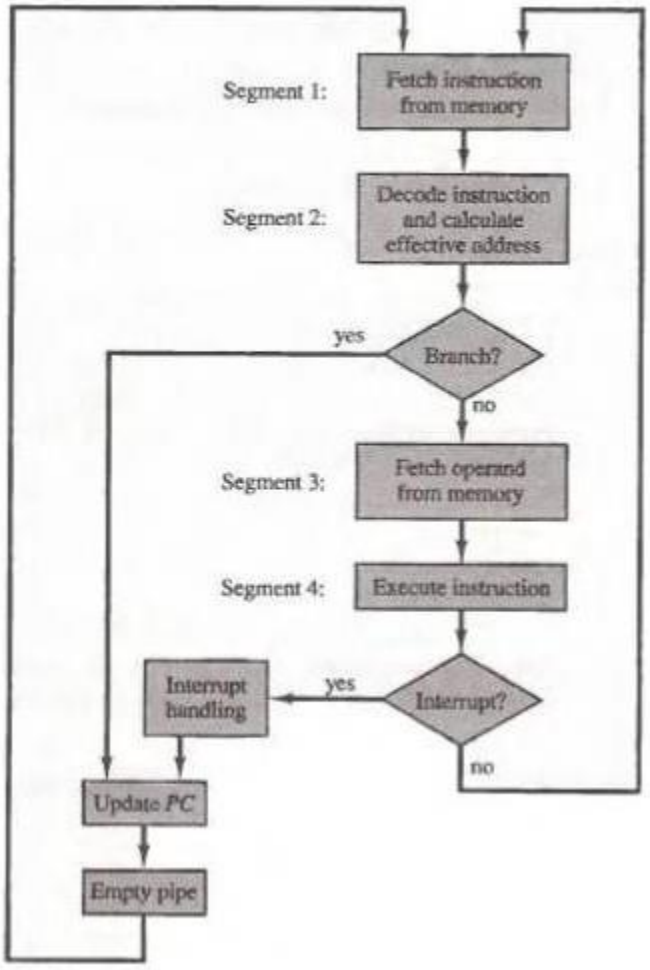


Fig.5.9. Instruction Pipelininig

In the most general case, the computer needs to process each instruction with the following sequence of steps.

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Calculate the effective address.

4. Fetch the operands from memory.

5. Execute the instruction.

6. Store the result in the proper place.

There are certain difficulties that will prevent the instruction pipeline from operating at its maximum rate. Different segments may take different times to operate on the incoming information. Some segments are skipped for certain operations.

1. Fl is the segment that fetches an instruction.

2. DA is the segment that decodes the instruction and calculates the effective

address.

3. FO is the segment that fetches the operand.

4. EX is the segment that executes the instruction.

For example, a register mode instruction does not need an effective address calculation.

Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Fig.5.10. Timing of Instruction Pipeline

Two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory.

Memory access conflicts are sometimes resolved by using two memory buses for accessing instructions and data in separate modules.

5.11. VECTOR PROCESSING

There is a class of computational problems that are beyond the capabilities of a conventional computer.

These problems require vast number of computations on multiple data items, that will take a conventional computer (with scalar processor) days or even weeks to complete.

Such complex instructions, which operates on multiple data at the same time, requires a better way of instruction execution, which was achieved by Vector processors.

Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions like **ADD A to B, and store into C** are not practically efficient.

Addresses are used to point to the memory location where the data to be operated will be found, which leads to added overhead of data lookup. So until the data is found, the CPU would be sitting ideal, which is a big performance issue.

Hence, the concept of **Instruction Pipeline** comes into picture, in which the instruction passes through several sub-units in turn. These sub-units perform various independent functions,

For example : the **first** one decodes the instruction, the **second** sub-unit fetches the data and the **third** sub-unit performs the math itself.

Therefore, while the data is fetched for one instruction, CPU does not sit idle, it rather works on decoding the next instruction set, ending up working like an assembly line.

Vector processor, not only use Instruction pipeline, but it also pipelines the data, working on multiple data at the same time.

A normal scalar processor instruction would be ADD A, B, which leads to addition of two operands, but what if we can instruct the processor to ADD a group of numbers(from 0 to n memory location) to another group of numbers(lets say, n to k memory location). This can be achieved by vector processors.

In vector processor a single instruction, can ask for multiple data operations, which saves time, as instruction is decoded once, and then it keeps on operating on different data items.

- Applications of Vector Processors

– Computer with vector processing capabilities are in demand in specialized applications. The following are some areas where vector processing is used:

– Petroleum exploration.

– Medical diagnosis.

– Data analysis.

- Weather forecasting.
- Aerodynamics and space flight simulations.
- Image processing.
- Artificial intelligence.

TYPES OF ARRAY PROCESSORS

There are basically two types of array processors:

1. Attached Array Processors
2. SIMD Array Processors

5.11.1. ATTACHED ARRAY PROCESSORS

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks.

It achieves high performance by means of parallel processing with multiple functional units.

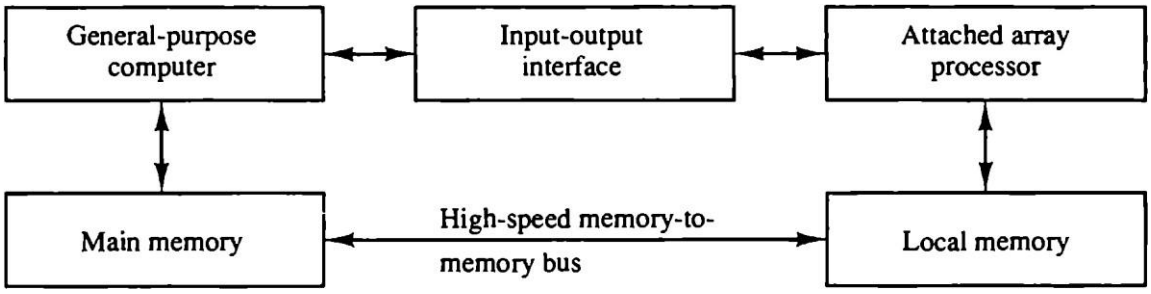


Fig.5.11. Attached Array Processor with Host Computer

5.11.2. SIMD ARRAY PROCESSORS

SIMD is the organization of a single computer containing multiple processors operating in parallel.

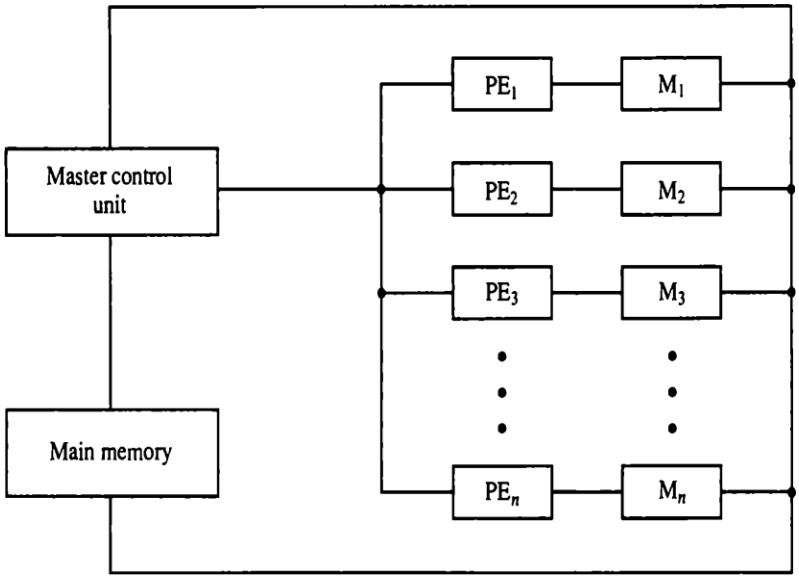


Fig.5.11. SIMD Processors

The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams.

A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an ALU and registers.

The master control unit controls all the operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed.

The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instructions are send to all PE's simultaneously and results are returned to the memory.

The best known SIMD array processor is the ILLIAC IV computer developed by the Burroughs corps. SIMD processors are highly specialized computers. They are only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.