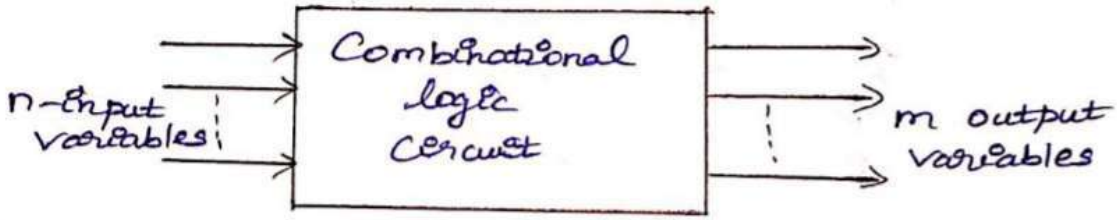
UNIT I

COMBINATIONAL CIRCUIT DESIGN

A combinational circuit consists of input variables, logic gates and output variables. The combinational circuit accepts n-input binary variables and generates output variables depending on the logical combination of gates.



(fig) Block diagram of a combinational circuit.

Design Procedure :-

The design of combinational circuits starts from the outline of the problem statement and ends in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be easily obtained. The design procedure involves following steps:.

1. The problem definition.
2. The determination of number of available input variables and required output variables.
3. Assigning letter symbols to input and output variables.
4. The derivation of truth table indicating the relationships between i/p and o/p variables.
5. Obtain simplified Boolean expression for each output.
6. Obtain the logic diagram.

Problem :-

1.) A majority gate is a digital circuit whose o/p is equal to 1 if the majority of inputs are 1's. The o/p is 0 otherwise. Using a truth table, find the Boolean function implemented by a 3-input majority gate. Simplify the function and implement with gates.

Solution :-

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

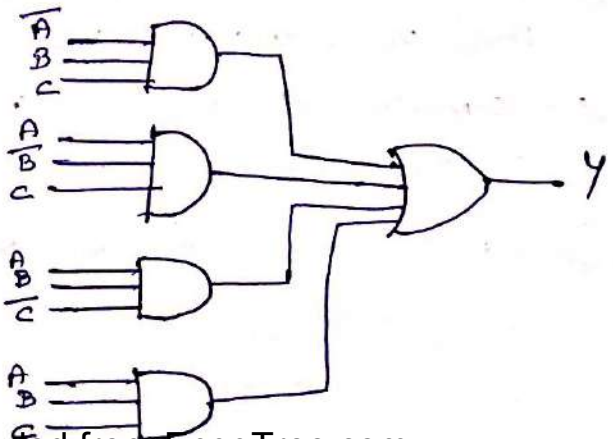
← Truth Table.

K-map simplification

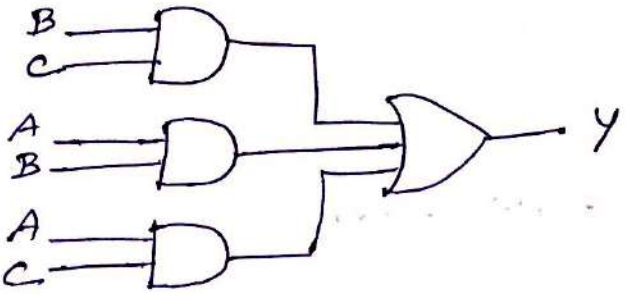
A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$Y = BC + AB + AC.$$

Implementation with 3-input gates :-



Implementation after simplification:-



Design of Adders :-

Digital computers perform various arithmetic operations. The most basic operation is the addition of two binary digits. The four possible elementary operations are,

$0+0=0$ $\quad\quad$ $1+0=1$

$0+1=1$ $\quad\quad$ $1+1=10_2$.

The higher significant bit of this result is called a carry, and lower significant bit is called sum. The logic circuit which performs this operation is called a half-adder. The circuit which performs addition of three bits (two significant bits and a previous carry) is a full adder.

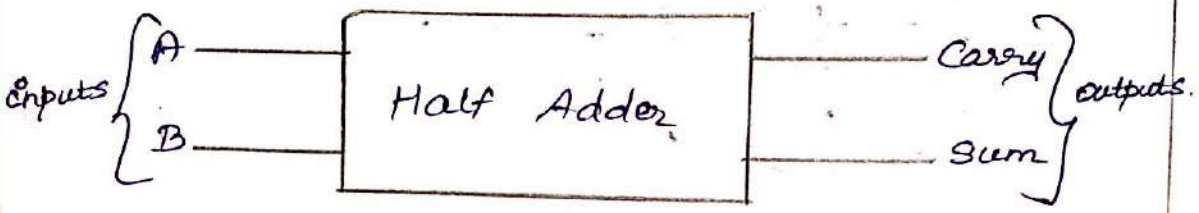
Half Adder :-

The half adder operation needs two binary inputs : augend and addend bits; and two binary outputs : sum and carry.

Truth Table:-

Inputs		Outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Block Schematic of half adder :-



K-map Simplification for carry and sum :-

For carry

A \ B	0	1
0	0	0
1	0	(1)

For sum

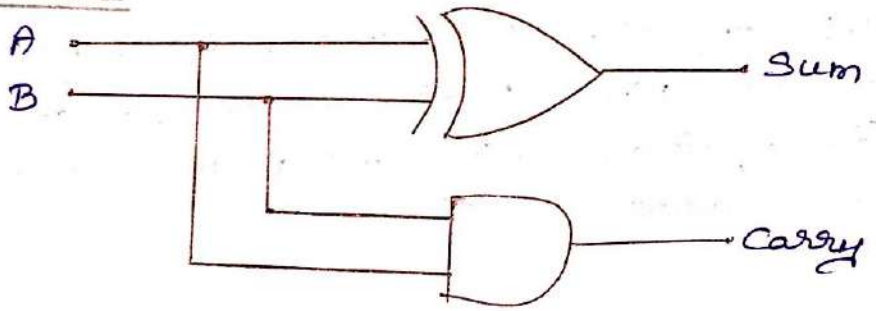
A \ B	0	1
0	0	(1)
1	(1)	0

$Carry = AB$

$Sum = A\bar{B} + \bar{A}B$

$Sum = A \oplus B$

Logic Diagram :-



Limitations :-

→ In multidigit addition we have to add two bits along with the carry of previous digit addition.

→ Effectively such addition requires addition of three bits.

→ This is not possible with half-adder.

→ Hence half adders are not used in practice.

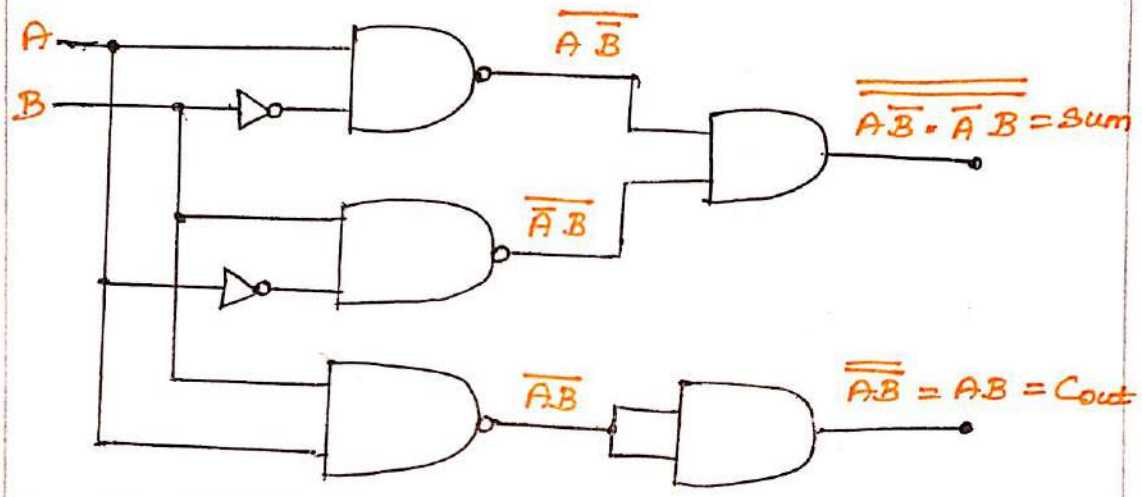
Eg : Design half Adder using NAND Gates.

Soln : For half Adder;

$$Sum = A\bar{B} + \bar{A}B \qquad Cout = AB$$

$$= \overline{\overline{A\bar{B} + \bar{A}B}} \qquad Cout = \overline{\overline{AB}}$$

$$= \overline{\overline{A\bar{B}} \cdot \overline{\bar{A}B}}$$



FULL ADDER :

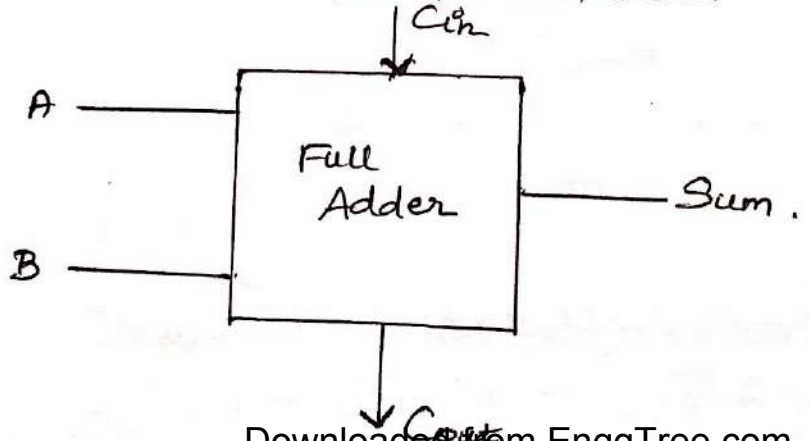
A full adder is a combinational circuit that forms the arithmetic sum of three input bits.

→ It consists of three inputs and two outputs.

→ Two of the i/p variables, denoted by A and B, represent the two significant bits to be added.

→ The 3rd i/p Cin, represents the carry from the previous lower significant position.

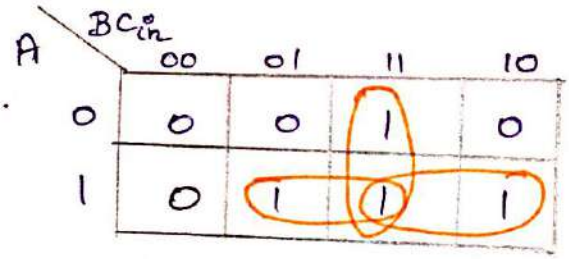
Block Schematic of full adder :



Truth Table for full adder:-

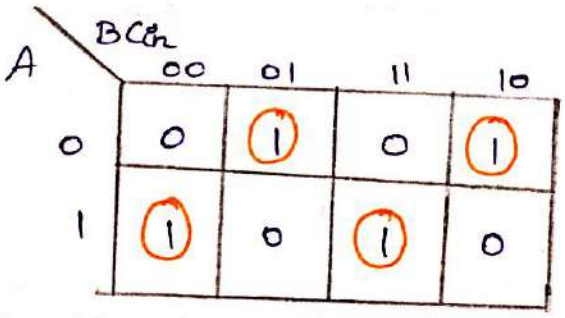
Inputs			Outputs	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-map for carry and sum:-



⇐ For carry (C_{out})

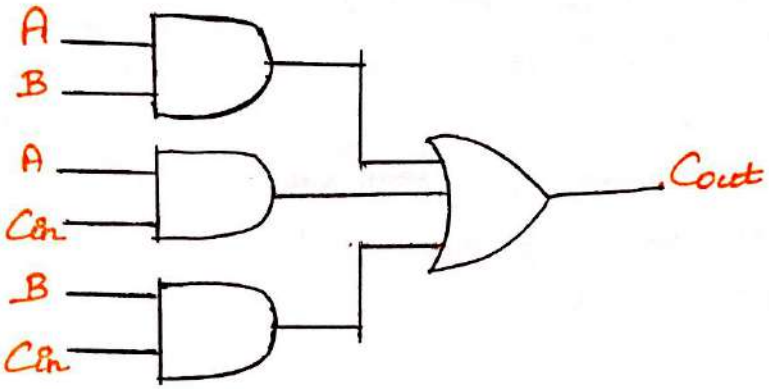
$$C_{out} = AB + AC_{in} + BC_{in}$$

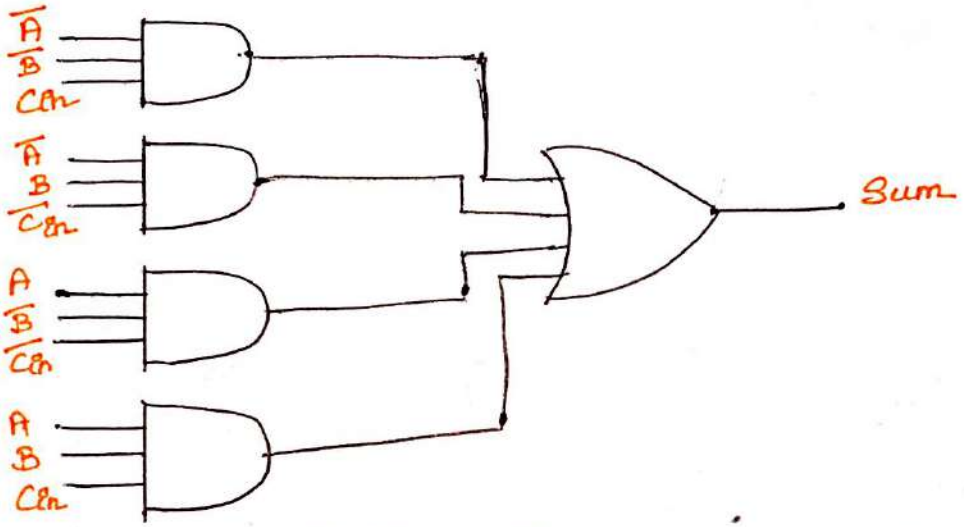


⇐ For Sum

$$Sum = \bar{A}\,\bar{B}\,C_{in} + \bar{A}\,B\,\bar{C}_{in} + A\,\bar{B}\,\bar{C}_{in} + ABC_{in}$$

Logic diagram :-





$$Sum = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

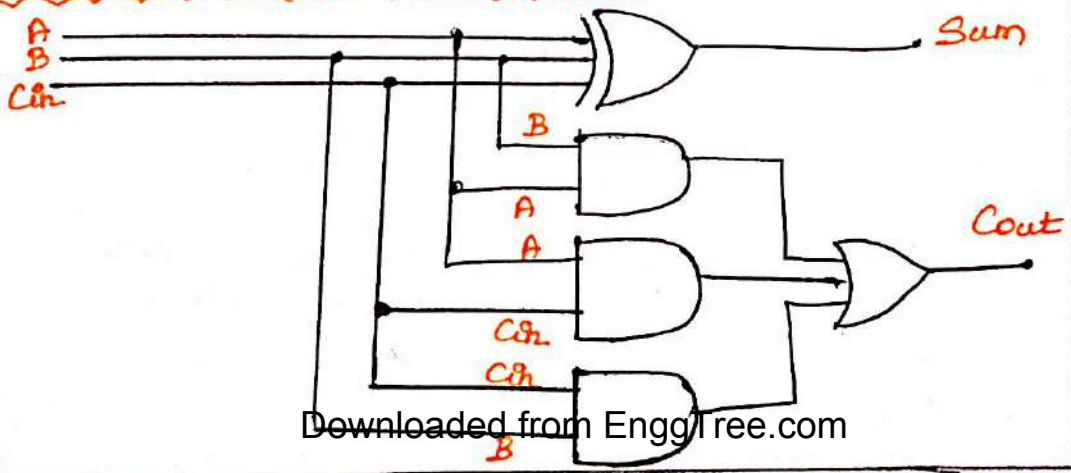
$$= C_{in}(\bar{A}\bar{B} + AB) + \bar{C}_{in}(\bar{A}B + A\bar{B})$$

$$= C_{in}(A \odot B) + \bar{C}_{in}(A \oplus B)$$

$$= C_{in}(\overline{A \oplus B}) + \bar{C}_{in}(A \oplus B)$$

$$\boxed{Sum = C_{in} \oplus (A \oplus B)}$$

Implementation of full adder :-



Full Adder Using two half Adders :-

→ A full adder can also be implemented with two half adders and one OR-gate.

→ The sum output from the second half adder is the exclusive-OR of C_{in} and the output of the first half-adder, giving,

$$C_{out} = AB + AC_{in} + BC_{in}$$

$$= AB + AC_{in}(B + \overline{B}) + BC_{in}(A + \overline{A})$$

$$= AB + ABC_{in} + A\overline{B}C_{in} + ABC_{in} + \overline{A}BC_{in}$$

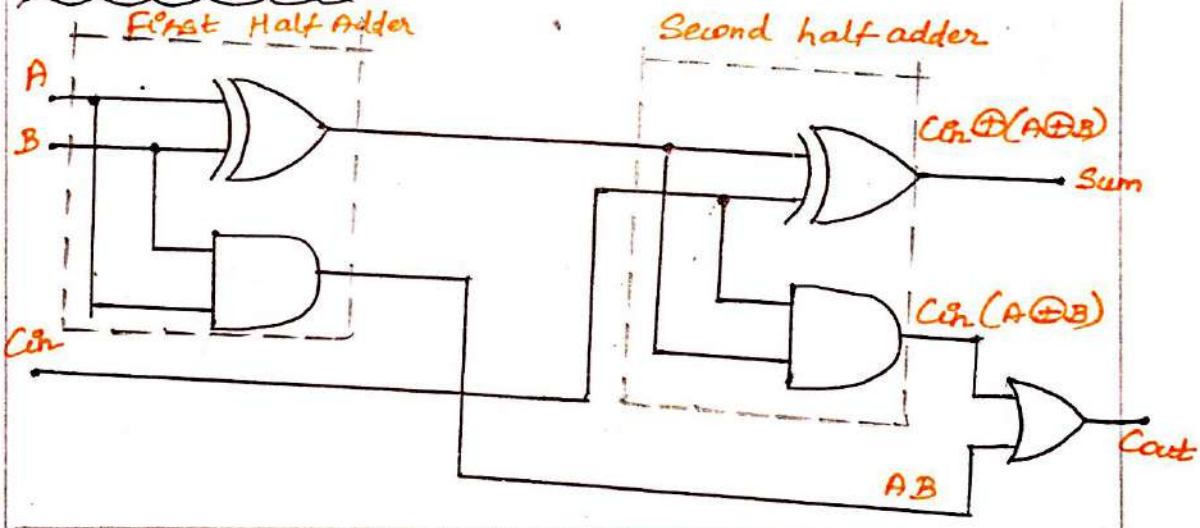
$$= AB(1 + C_{in} + C_{in}) + A\overline{B}C_{in} + \overline{A}BC_{in}$$

$$= AB + A\overline{B}C_{in} + \overline{A}BC_{in}$$

$$= AB + C_{in}(A\overline{B} + \overline{A}B)$$

$$C_{out} = AB + C_{in}(A \oplus B)$$

Implementation :-



Subtractor :-

* Subtrahend bit is subtracted from the minuend bit.
* Minuend bit is smaller than the subtrahend bit, hence 1 is borrowed.

$0 - 0 = 0$

$0 - 1 = 1$ with 1 borrow.

$1 - 0 = 1$

$1 - 1 = 0$

Half Subtractor :-

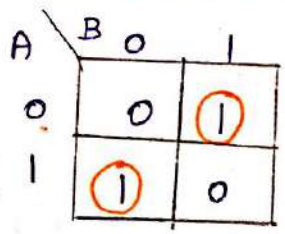
* A half-subtractor is a combinational circuit that subtracts two-bits and produces their difference.
* It also has an output to specify if a 1 has been borrowed.
* Let 'A' be minuend bit and 'B' be subtrahend bit.

Truth Table :-

Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-map :-

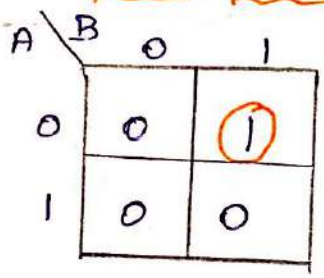
For difference



$Difference = A\bar{B} + \bar{A}B$

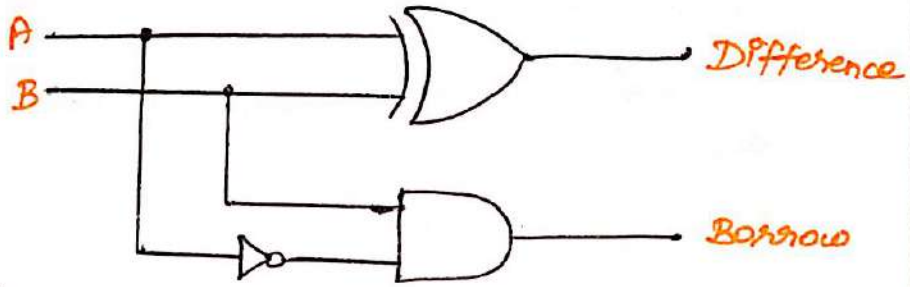
$= A \oplus B$

For borrow.



$Borrow = \bar{A}B$.

Logic Diagram :-



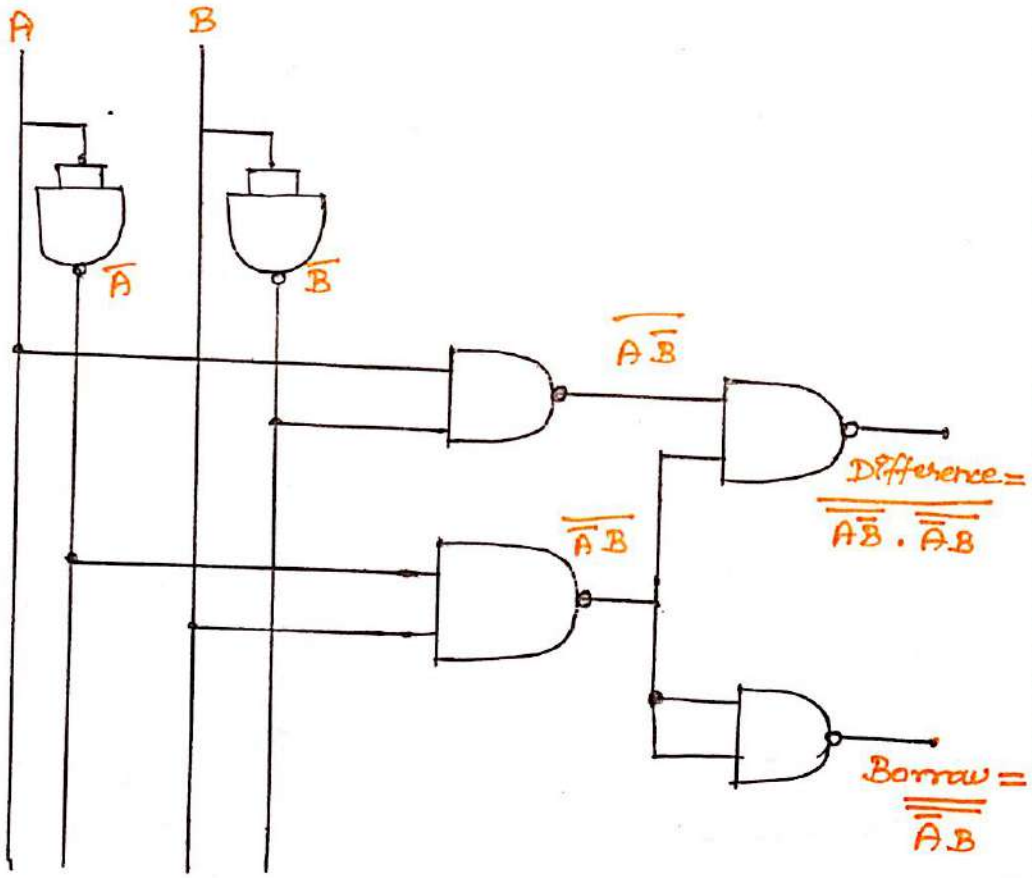
Limitations :-

→ In multidigit subtraction, we have to subtract two bits along with the borrow of the previous digit subtraction.

→ Effectively such subtraction requires subtraction of three bits.

→ This is not possible with half-subtractor.

Implementation :-



Full Subtractor :-

A full subtractor is a combinational circuit that performs a subtraction between two bits, taking into ~~and~~ account borrow of the LSB.

It has 3 i/ps and 2 o/ps.

i/p ⇒ A, B & B_{in}

o/p ⇒ D and B_{out}.

Inputs			Outputs	
A	B	B_{in}	D	B_{out}
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

K-map for Difference :-

A \ $B B_{in}$	00	01	11	10
0	0	(1)	0	(1)
1	(1)	0	(1)	0

$$D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in}$$

$$= B_{in}(\bar{A}\bar{B} + AB) + \bar{B}_{in}(\bar{A}B + A\bar{B})$$

$$= B_{in}(A \odot B) + \bar{B}_{in}(A \oplus B)$$

$$= B_{in}\overline{(A \oplus B)} + \bar{B}_{in}(A \oplus B)$$

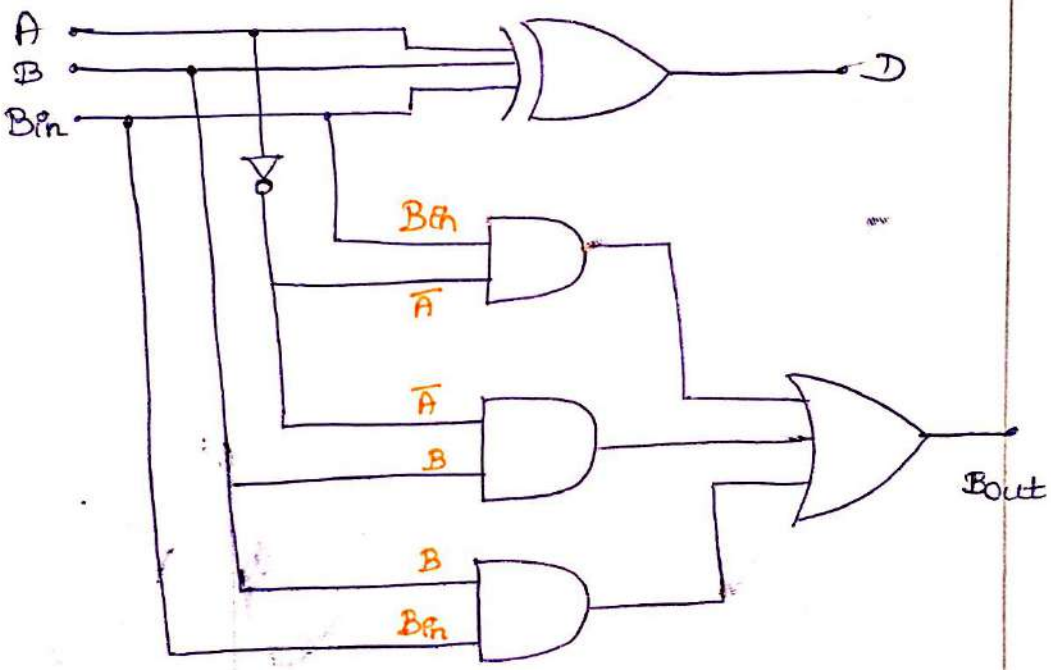
$$\boxed{D = B_{in} \oplus A \oplus B}$$

K-map for Bout :-

A \ BBin	00	01	11	10
0	0	1	1	1
1	0	0	1	0

$$B_{out} = \overline{A} B_{in} + \overline{A} B + B B_{in}.$$

Logic Diagram :-



A full Subtractor can also be implemented with two half-subtractors and one OR gate.

$$B_{out} = \overline{A} B_{in} + \overline{A} B + B B_{in}$$

$$= \overline{A} B_{in}(B + \overline{B}) + \overline{A} B + B B_{in}(A + \overline{A})$$

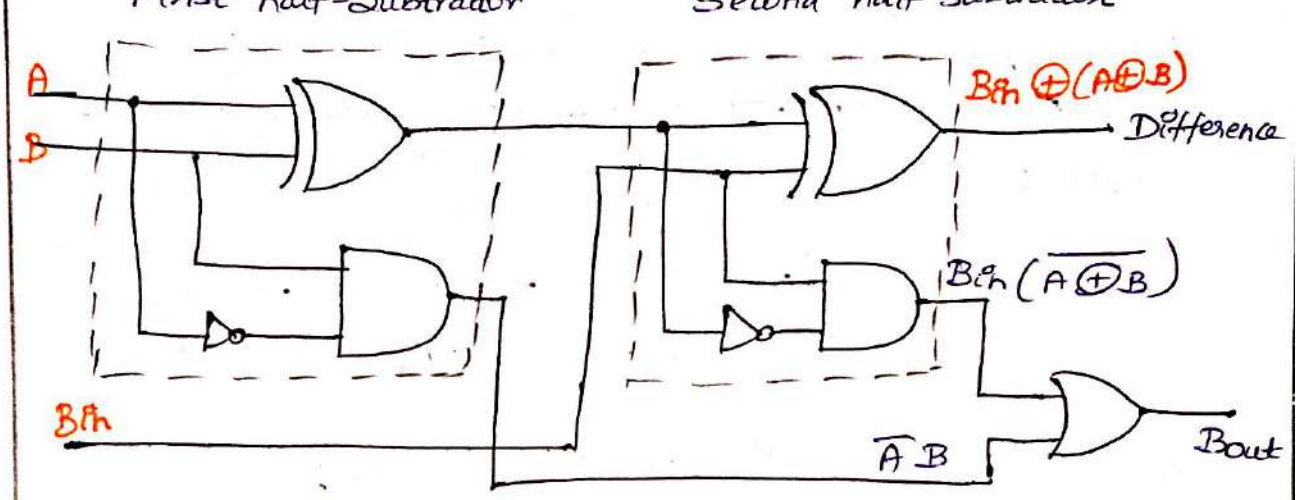
$$= \overline{A} B B_{in} + \overline{A}\,\overline{B} B_{in} + \overline{A} B + A B B_{in} + \overline{A} B B_{in}$$

$$= \overline{A} B (B_{in} + 1 + B_{in}) + \overline{A}\,\overline{B} B_{in} + A B B_{in}$$

$$= \overline{A} B + \overline{A}\,\overline{B} B_{in} + A B B_{in}$$

$$= \overline{A} B + B_{in}(\overline{A}\,\overline{B} + A B)$$

$$\boxed{B_{out} = \overline{A} B + B_{in}\,\overline{(A \oplus B)}}$$

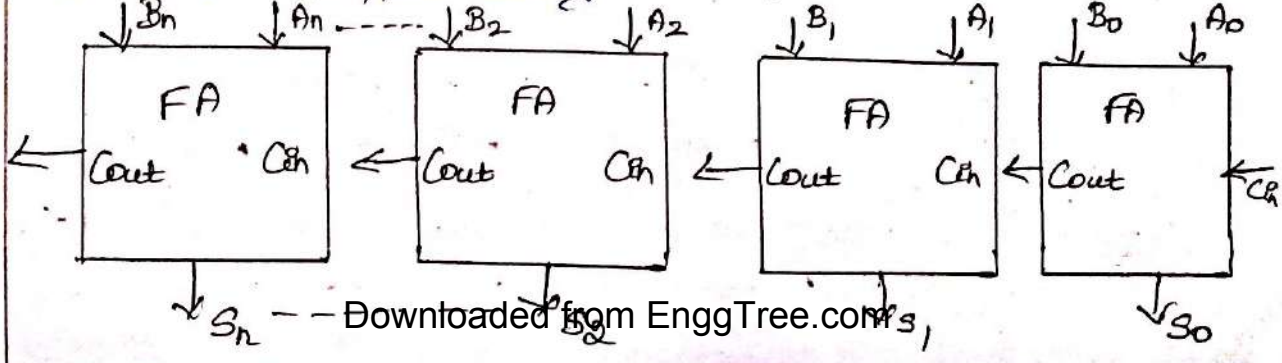


PARALLEL ADDER / RIPPLE CARRY ADDER

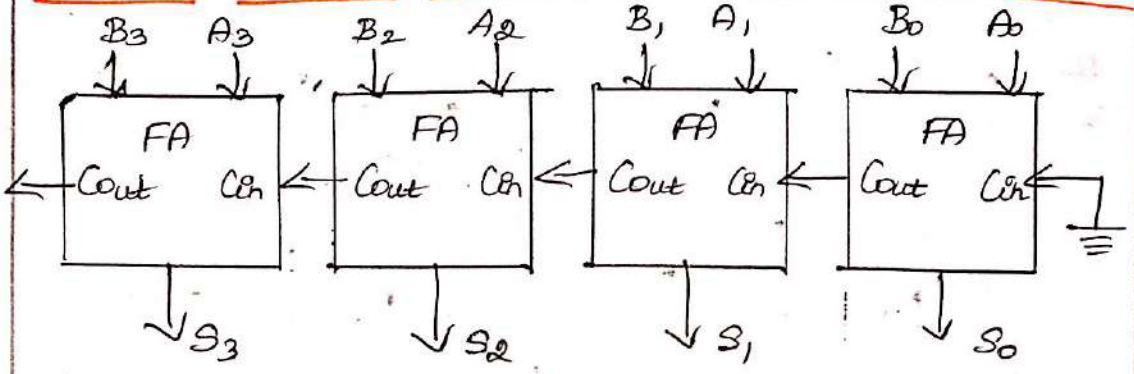
In Serial adder, a single full adder is capable of adding two one-bit numbers and an input carry. If the number of bit increases with bit by bit, addition is the time taking process. In order to increase the speed of operation we prefer parallel adder.

Construction and operation of n-bit parallel adder:

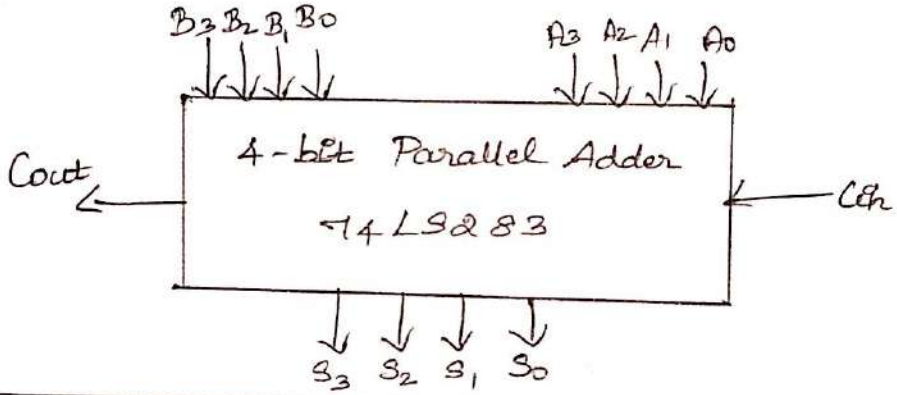
* If the number of bit increases in the binary number we can employ additional full adder.
* A 'n'-bit parallel adder can be constructed by using 'n' number of full adders connected in parallel.
* The carry o/p of each adder is connected to the carry i/p of the next higher order adder, hence the name "Ripple Carry Adder".



Design a 4-bit parallel Adder using full adders.



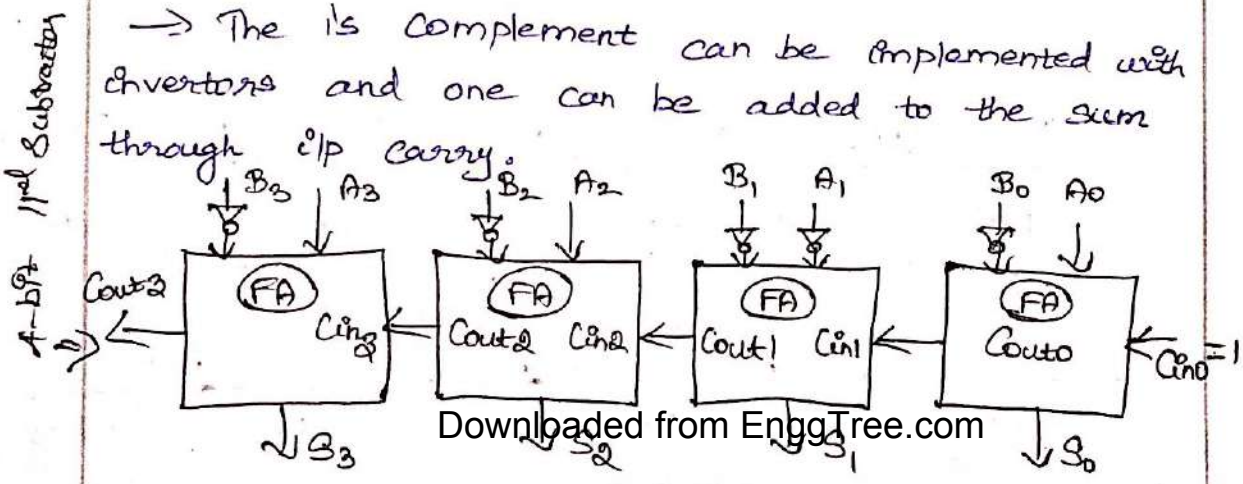
Binary Parallel Adder IC 74LS283



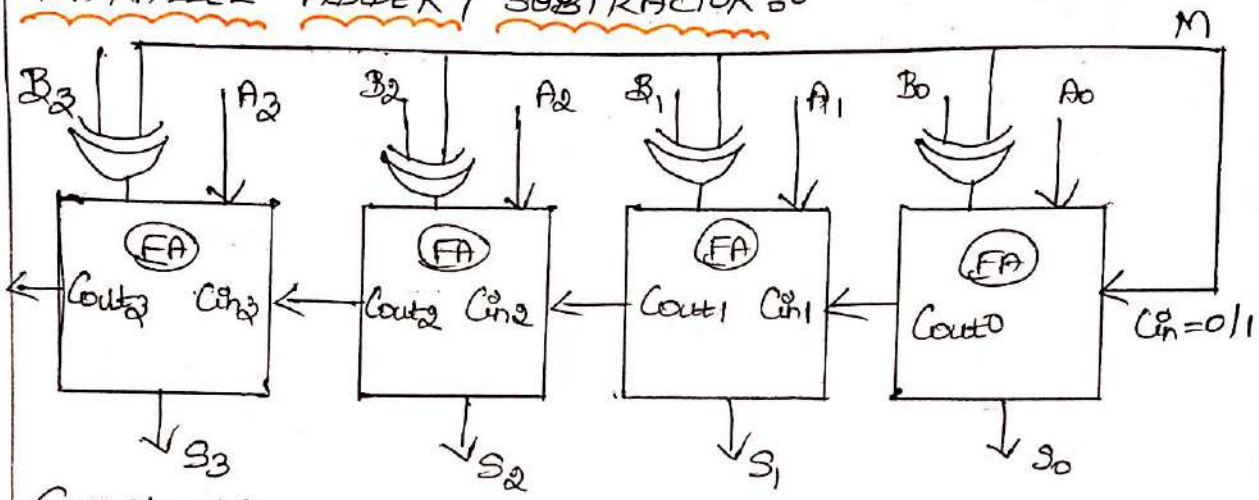
PARALLEL SUBTRACTOR

* The Subtraction of binary numbers can be done most conveniently by means of complements.
* Subtraction of A-B can be done by taking 2's complement of B and adding it to 'A'
* The 2's complement can be obtained by taking the 1's complement and adding 1 to the LS pair of bits.

→ The 1's complement can be implemented with invertors and one can be added to the sum through i/p carry.



PARALLEL ADDER / SUBTRACTOR :-



Construction :-

The addition and subtraction operations can be combined into one circuit with one common binary adder. This is done by including an EX OR gate with each full adder. It has mode selection switch 'M' to control the operation.

Operation :-

* The Mode 'M' i/p controls the operation of the ckt.
* M=0 → ckt adds as adder
  M=1 → ckt becomes subtractor.
* Each EXOR gate receives input M and one of the inputs of B.
* When M=0 ; we have $B \oplus 0 = B$. The full adders receives the value of B, the i/p carry is 0 and circuit performs A + B operation.
* When M=1 ; we have $B \oplus 1 = \overline{B}$ and Cin0 = 1. The B i/ps are all complemented and a 1 is added through the i/p carry. The circuit performs the operation A plus the 2's complement of B, (ie) A − B.

Look Ahead CARRY Adder

In parallel adder, the carry o/p of each full adder stage is connected to the carry i/p of the next higher order stage. Therefore the sum and carry outputs of any stage cannot be produced until the i/p carry occurs. This leads to a time delay in the addition process. This delay is known as "Carry Propagation delay."

Eg:
```
  0 1 0 1
  0 0 1 1
---------
  1 0 0 0
---------
```

Addition of LSB produces carry in to 2nd and 2nd to 3rd. From this the sum bit generated in the last position (MSB), depends on the carry that was generated by the addition of the previous position.

This means adder will not produce correct result until LSB carry has propagate through intermediate full adders.

So because of this time delay we can't add more number of bits.

One method of speeding up this process by eliminating inter stage carry delay is called look ahead carry addition.

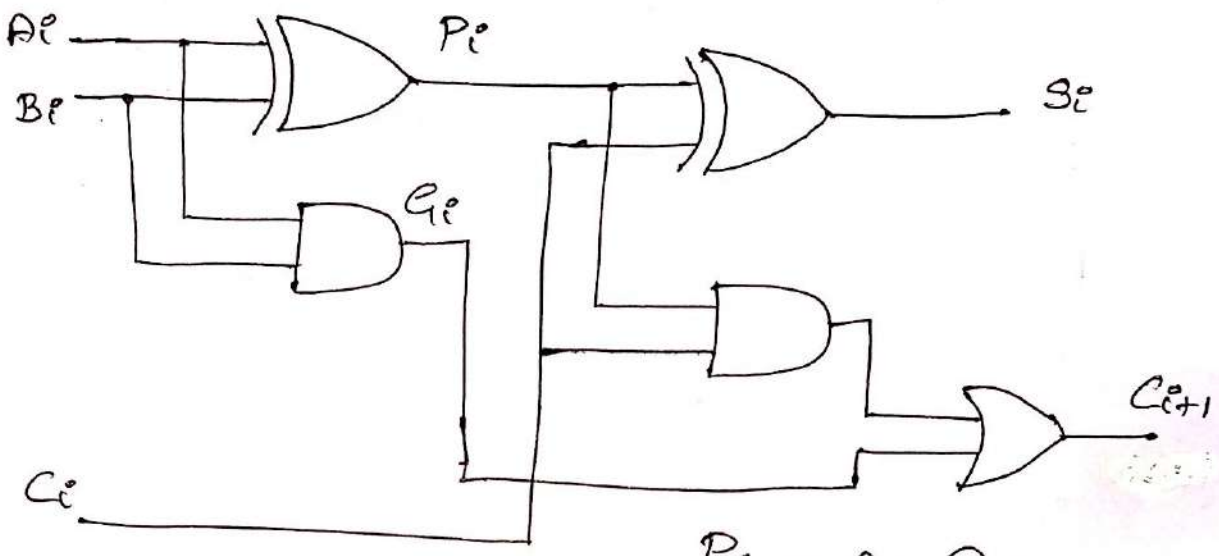
Implement full adder using 2 half adders :-

The output sum and carry can be expressed as,

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

G_i is called as carry generator and it produces a carry when both A_i and B_i are one.



$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

P_i is called carry propagate because it is term associated with the propagation of carry from C_i to C_{i+1}. Now the Boolean function for the carry o/p of each stage can be written as,

$i = 1 \Rightarrow$ $\boxed{C_2 = G_1 + P_1 C_1}$

$i = 2 \Rightarrow$ $C_3 = G_2 + P_2 C_2$

$= G_2 + P_2 (G_1 + P_1 C_1)$

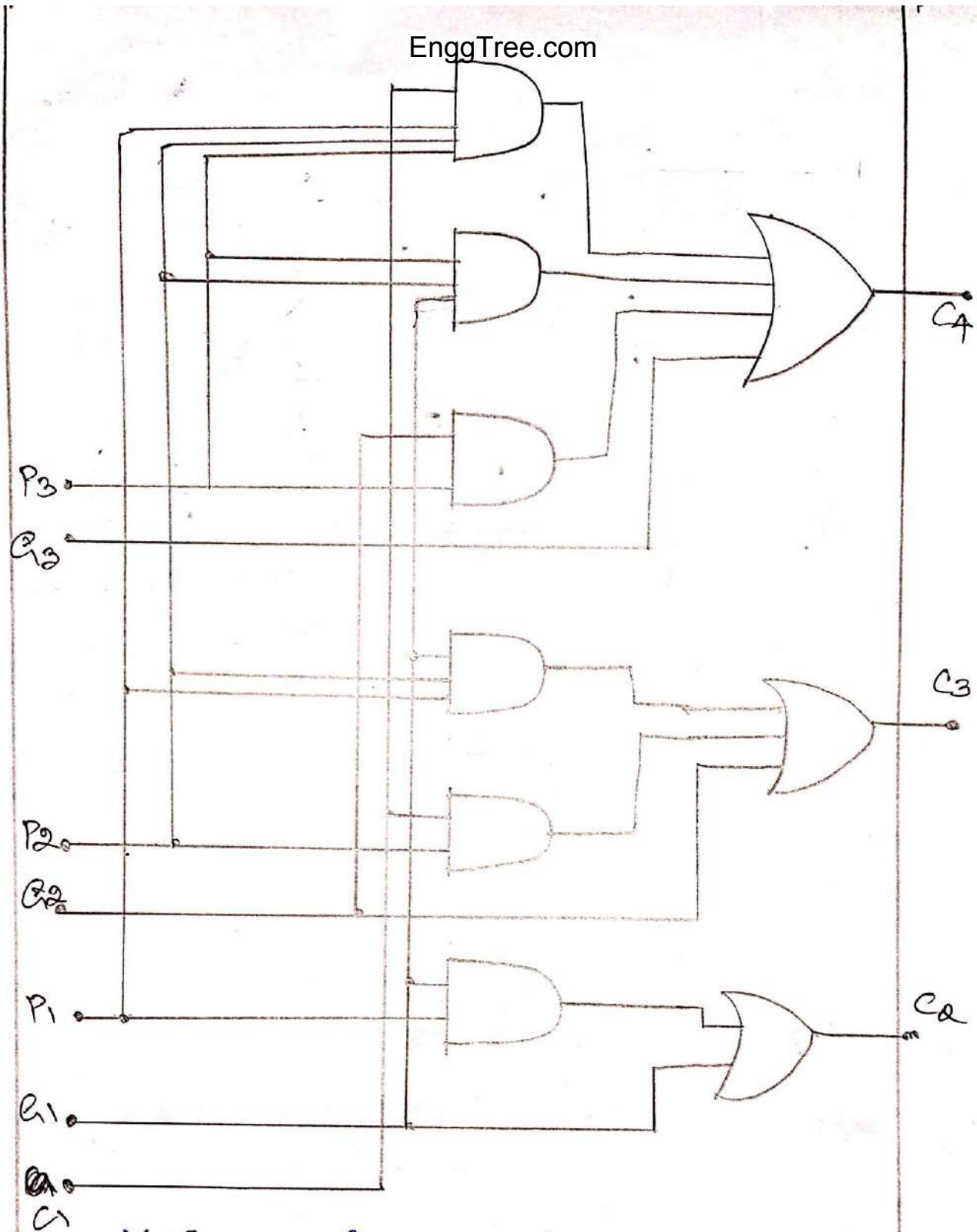
$\boxed{C_3 = G_2 + P_2 G_1 + P_1 P_2 C_1}$

$i = 3 \Rightarrow$ $C_4 = G_3 + P_3 C_3$

$= G_3 + P_3 [G_2 + P_2 G_1 + P_1 P_2 G_1]$

$\boxed{C_4 = G_3 + P_3 G_2 + P_2 P_3 G_1 + P_1 P_2 P_3 G_1}$

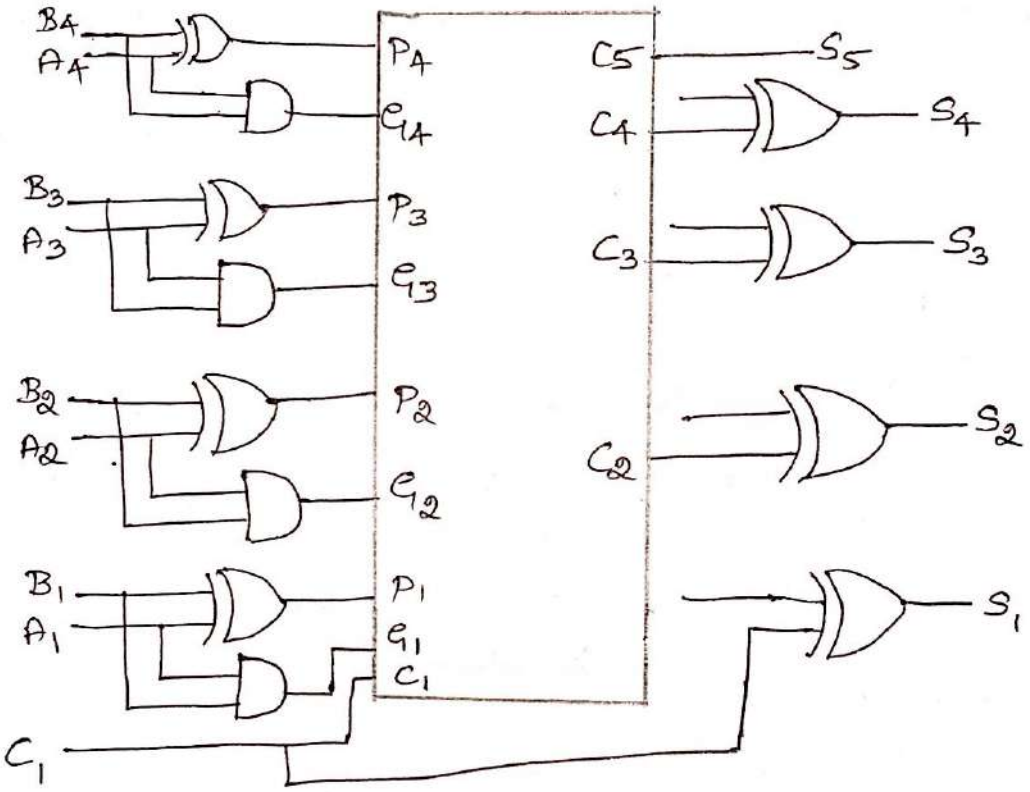
From the above boolean functions, for C_4 doesnot have to wait for C_3 and C_2 to propagate, infact C_4 is propagated at the same time as C_2 and C_3.



Using a look ahead carry generator we can easily construct a 4 bit parallel adder with a look ahead carry scheme. In this each sum o/p requires 2 EXOR gates.

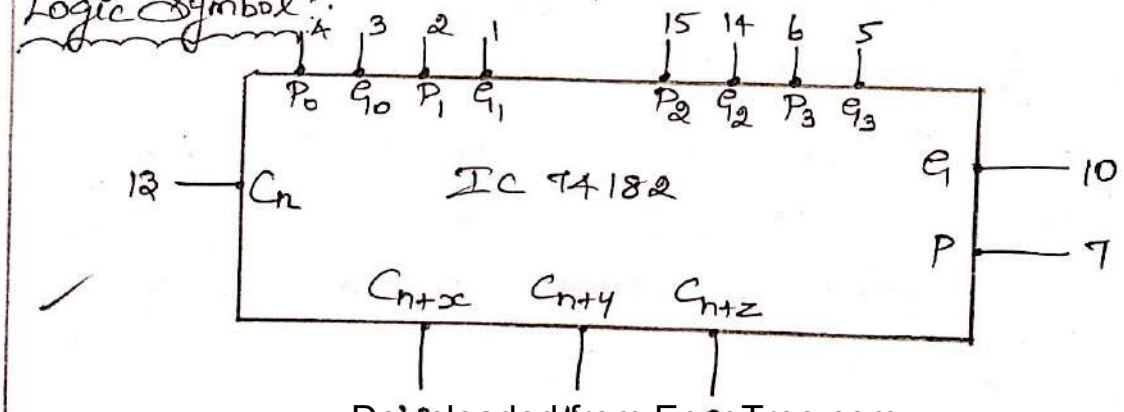
* The o/p of the 1st XOR gate generates P_i and the AND gate generates G_i.

* The carries are generated using look ahead carry generator and applied as i/p's to the second EXOR gate and the i/p to the EXOR gate is P_i.
* Thus second EXOR gates generates SUM o/p's.
* Each o/p generated after a delay of 2 levels of gates.
* Thus o/p's S_2 through S_4 have equal propagation delay time.

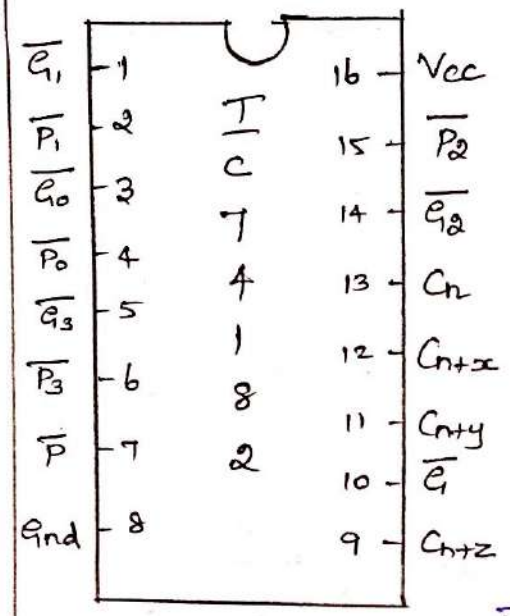


IC 74182 – Look ahead carry generator :-

Logic Symbol :-



Pin Diagram :-



$(\overline{P_0}, \overline{P_1}, \overline{P_2}, \overline{P_3})$ → four pairs of active low propagate

$(\overline{G_0}, \overline{G_1}, \overline{G_2}, \overline{G_3})$ → Carry generate Signals.

C_n → active high Carry i/p

C_{n+x} , C_{n+y} , C_{n+z} → anticipated active high Carrier

$\overline{P}$ → active low carry propagate

$\overline{G}$ → Carry generate o/p's.

BCD ADDER :-

The digital systems handle the decimal number in the form of binary coded decimal numbers (BCD).

A BCD adder is a circuit that adds 2 BCD digits and produces a sum digit also in BCD.

BCD numbers uses 10 digits, 0 to 9 which are represented in binary form 0000 to 1001, (i.e) each BCD digit is represented as a 4 bit binary number.

BCD addition Procedure :-

1.) Add 2 BCD numbers using ordinary binary addition.

2.) If 4-bit sum is equal to or less than 9, no correction is needed. The sum is in proper BCD form.

3.) If 4 bit sum is greater than 9 or if a carry is generated from a 4 bit sum, the sum is invalid.

4.) To correct the invalid sum, add $(0110)_2$ to the 4-bit sum. If a carry results from this addition, add it to the next higher order BCD digit.

To implement BCD adder we require:-

* 4 bit adder for initial addition.
* Logic circuit to detect sum greater than 9.
* One more 4 bit adder to add $(0110)_2$ in the sum if the sum is greater than 9 or carry is 1.

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of given truth table.

Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

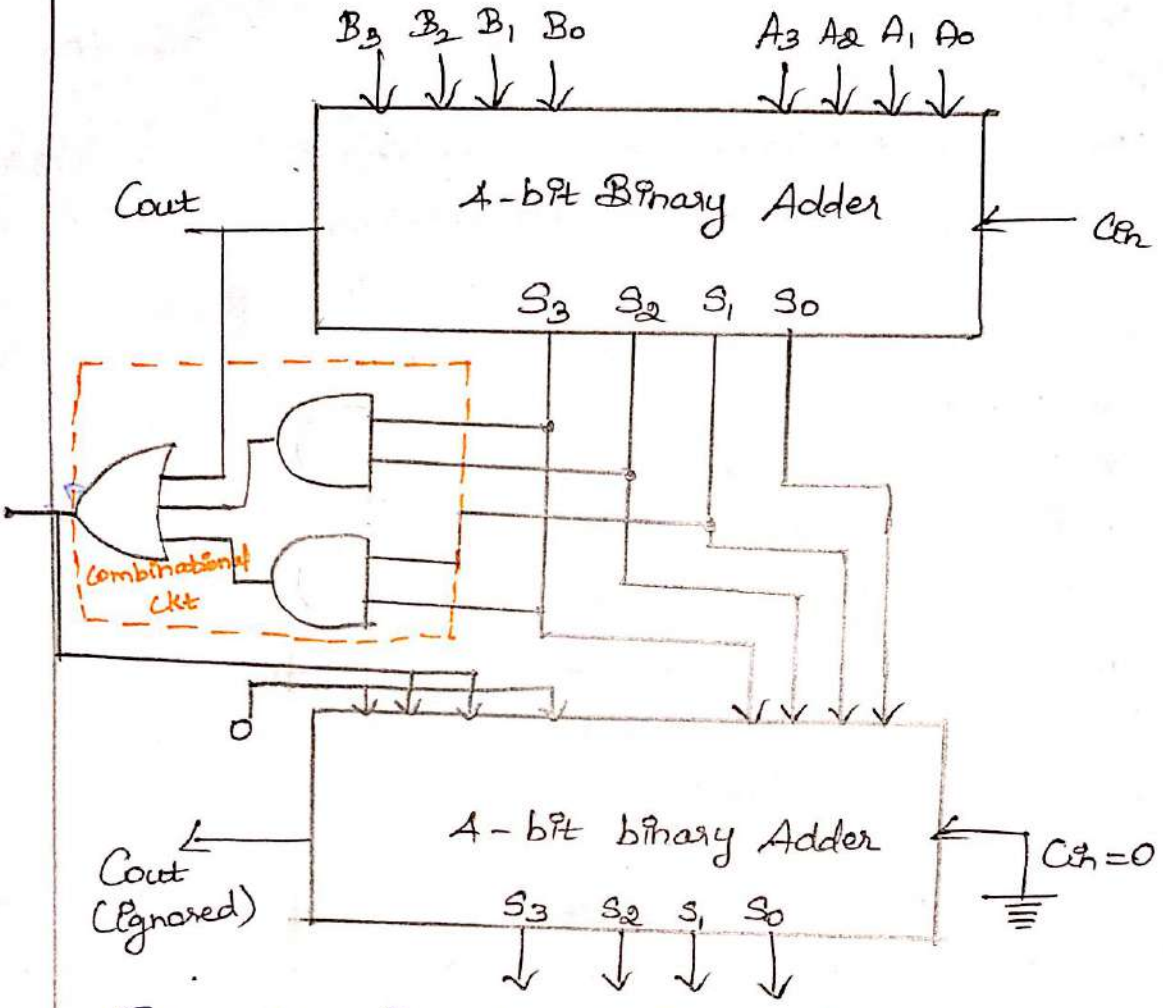
$Y = S_3S_2 + S_3S_1$

$Y=1$ → Indicates sum is greater than 9. We can put one more term Cout in the above expression.

$Y = S_3S_2 + S_3S_1 + Cout$ to check whether the carry is '1'.

If any 'one' condition is satisfied, we add 6 $(0110)_2$ in the sum.

Block diagram of BCD Adder



From the fig, the 2 BCD no's, together with i/p carry are first added in the top 4 bit binary adder to produce binary sum.

When the o/p carry is equal to zero, (ie, when sum ≤ 9, Cout = 0) nothing is added to binary sum. When it is equal to one (ie, when sum >9 or Cout = 1) binary 0110 is added to the binary sum through the bottom 4-bit binary adder.

The o/p carry generated from the bottom binary adder can be ignored, since it supplies information already available at the output-carry terminal.

MULTIPLEXERS :- (DATA SELECTORS)

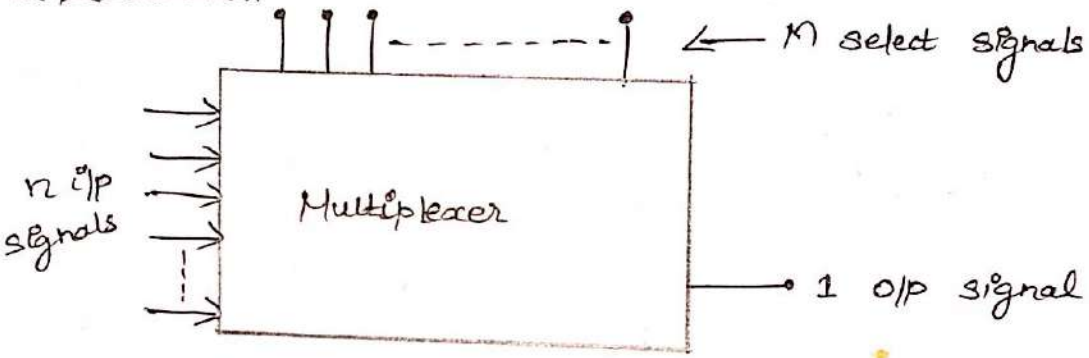
A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single o/p line.

A multiplexer is called as data selector.

The selection of a particular input line is controlled by set of selection lines.

Normally there are 2^n i/p lines and n selection lines.

Block diagram :-



* n input lines
* m select lines
* 1 o/p line
* $n = 2^m$
* To select 1 out of 4 i/p lines, 2 select lines are required.

Basic four input multiplexer :-

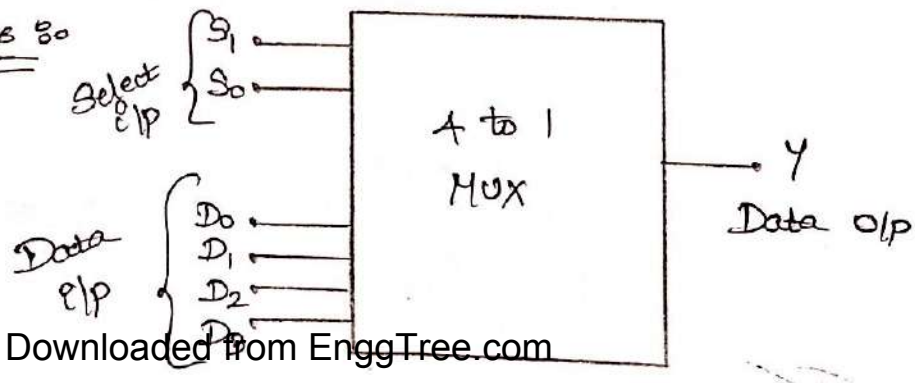
i/p is 4 $n = 4$ $\therefore m = 2$

4 i/p lines : D_0, D_1, D_2, D_3

2 select lines : S_0, S_1

1 o/p line : Y

Logic Symbols :-



Truth Table :-

Data Select I/p		Output
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Data Output

$Y = D_0$, if $S_1 = 0$ & $S_0 = 0$ $\quad Y = D_0 \overline{S_0}\,\overline{S_1} = D_0 \cdot 1 \cdot 1 = D_0$

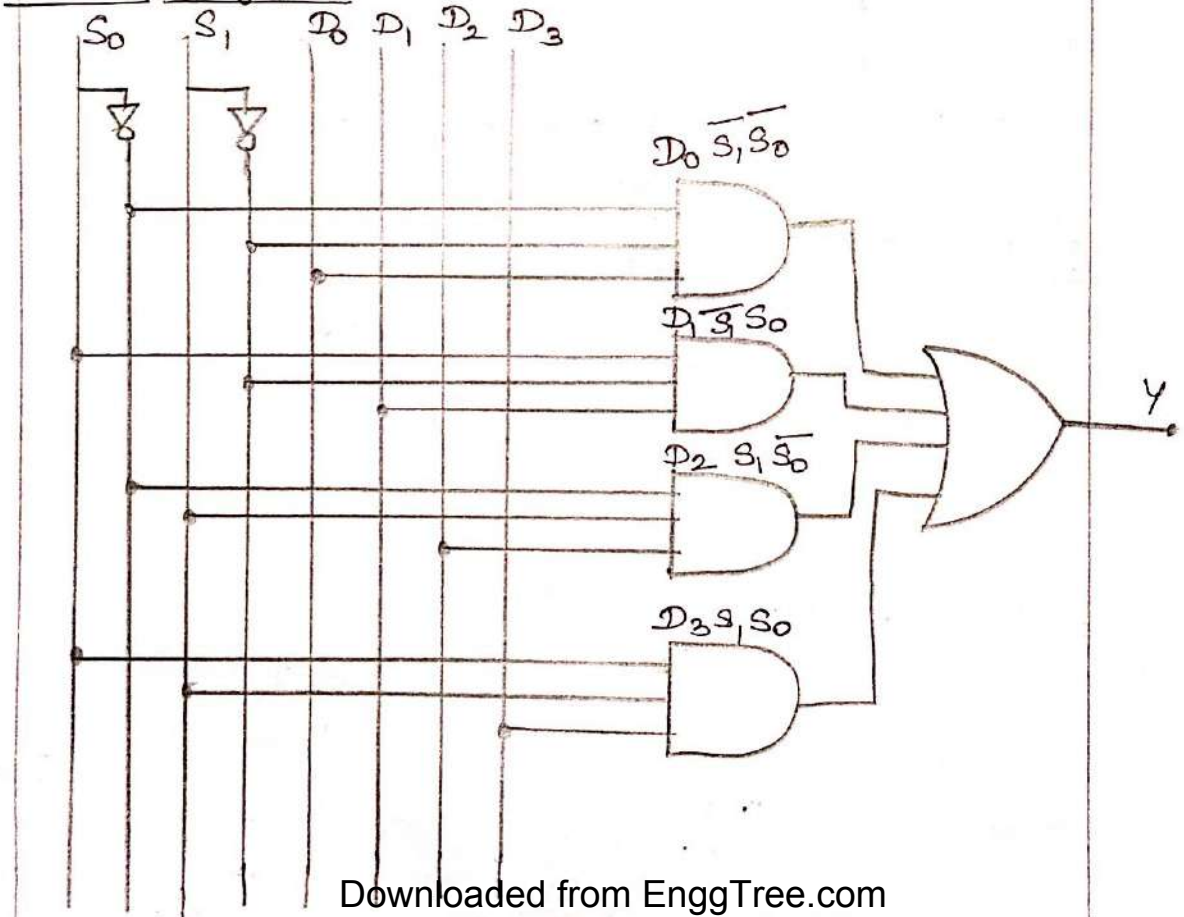
$Y = D_1$, if $S_1 = 0$ & $S_0 = 1$ $\quad Y = D_1 S_0 \overline{S_1} = D_1 \cdot 1 \cdot 1 = D_1$

$Y = D_2$, if $S_1 = 1$ & $S_0 = 0$ $\quad Y = D_2 \overline{S_0} S_1 = D_2 \cdot 1 \cdot 1 = D_2$

$Y = D_3$, if $S_1 = 1$ & $S_1 = 1$ $\quad Y = D_3 S_1 S_0 = D_3 \cdot 1 \cdot 1 = D_3$

$$Y = D_0 \overline{S_1}\,\overline{S_0} + D_1 S_0 \overline{S_1} + D_2 S_1 \overline{S_0} + D_3 S_1 S_0$$

Logic Diagram :-



Operation :-

* When $S_1 S_0 = 00$, AND gate associated with D_0 will have two of the inputs equal to '1' and 3rd input to D_0. The other 3 AND gates have '0' for atleast one of their inputs, which makes their o/p equal to '0', hence OR output will be equal to D_0. Thus selecting D_0 and the data on the input D_0 appears on data o/p line (Y).
* If $S_1 S_0 = 01$, Data on the i/p D_1 appears on the o/p.
* If $S_1 S_0 = 10$, Data on the i/p D_2 appears on the o/p.
* If $S_1 S_0 = 11$, Data on the i/p D_3 appears on the o/p.

8 : 1 Multiplexer :-

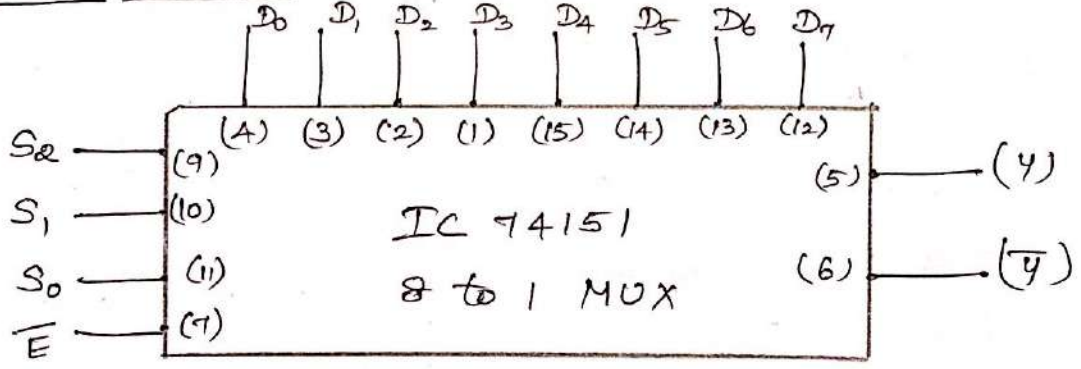
8 data i/ps ($D_0 - D_7$) lines

3 select i/ps ($S_2 - S_0$) lines $2^3 = 8$

1 o/p line (Y)

Enable i/p ($\overline{E}$), when $\overline{E} = 0$, the select i/p's $S_2 S_1 S_0$ will select one of the data i/p to pass through o/p Y. When $\overline{E} = 1$, MUX is disabled.

Logic Symbol :-



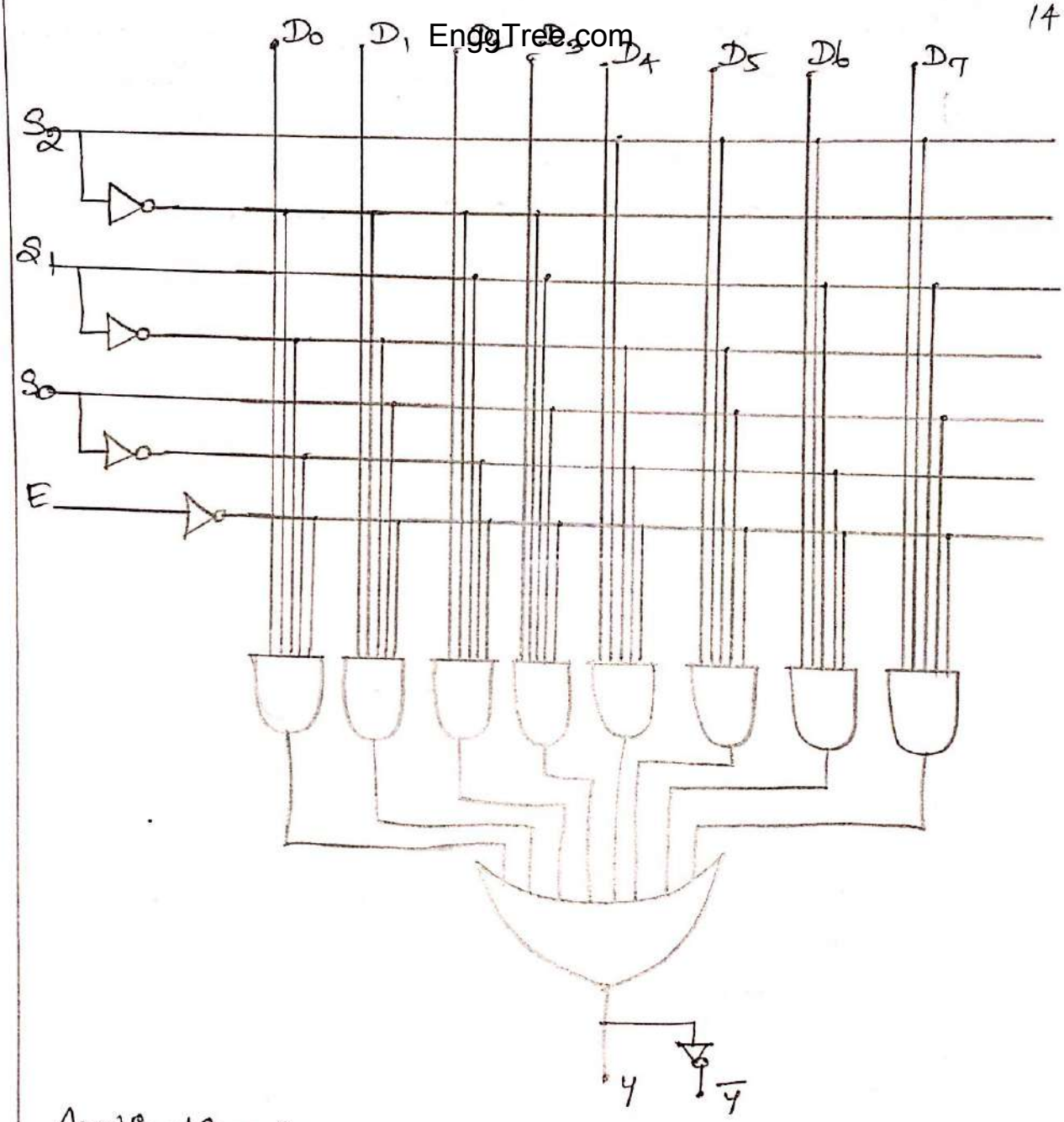
Truth Table :-

Inputs				Outputs	
$\overline{E}$	S_2	S_1	S_0	Y	$\overline{Y}$
1	X	X	X	1	0
0	0	0	0	D_0	$\overline{D_0}$
0	0	0	1	D_1	$\overline{D_1}$
0	0	1	0	D_2	$\overline{D_2}$
0	0	1	1	D_3	$\overline{D_3}$
0	1	0	0	D_4	$\overline{D_4}$
0	1	0	1	D_5	$\overline{D_5}$
0	1	1	0	D_6	$\overline{D_6}$
0	1	1	1	D_7	$\overline{D_7}$

$$Y = (D_0 \overline{S_2}\,\overline{S_1}\,\overline{S_0} + D_1 \overline{S_2}\,\overline{S_1}\,S_0 + D_2 \overline{S_2}\,S_1\,\overline{S_0} + D_3 \overline{S_2}\,S_1 S_0 + D_4 S_2 \overline{S_1}\,\overline{S_0} + D_5 S_2 \overline{S_1}\,S_0 + D_6 S_2 S_1 \overline{S_0} + D_7 S_2 S_1 S_0)\,\overline{E}.$$

Logic Diagram :-

Here each of the eight input D_0 through D_7 is applied to one input of an AND gate. Selection lines S_2, S_1 & S_0 are decoded to select particular AND gate. The o/p of AND gates are applied to a single gate (OR) that provides 1 line output.



Applications :-

→ Data Selection

→ Data routing

→ Operation Sequencing

→ Parallel to Serial Conversion

→ Waveform generation.

Implementation of Boolean function using Multiplexer :-

1.) Implement the following Boolean expression using suitable multiplexer,

$F(A,B,C,D) = \Sigma(0,1,3,4,8,9,15)$.

Soln :-

Since the given function is a four variable function, we need a multiplexer with 8 I/p lines and 3 select lines.

Apply variable B, C, D to the select lines.

Implementation Table :-

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	(0)	(1)	2	(3)	(4)	5	6	7
A	(8)	(9)	10	11	12	13	14	(15)
	1	1	0	$\bar{A}$	$\bar{A}$	0	0	A

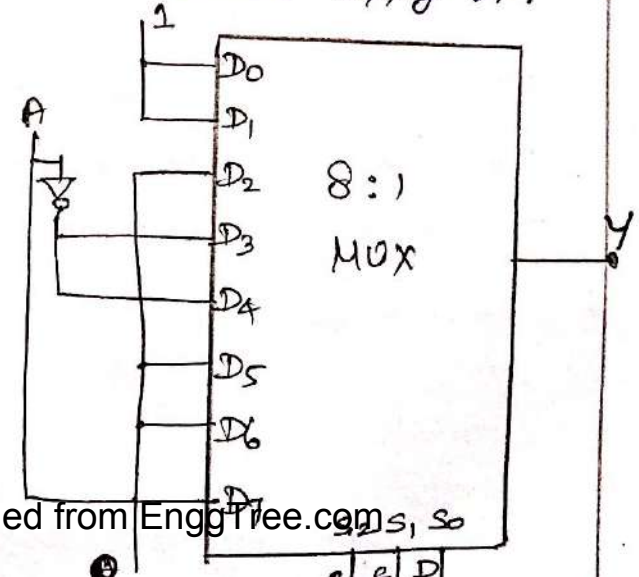
→ Circle the minterms of the function.

→ If both minterms in a column are not circled, apply '0' to the corresponding I/p.

→ If both minterms are circled, apply 1 to the corresponding I/p.

→ If top minterm alone is circled, apply $\bar{A}$ and if bottom minterm alone is circled apply A.

Draw a schematic for 8-to-1 MUX, apply logic 1 and logic 0 to the I/p according to the Implementation table :-



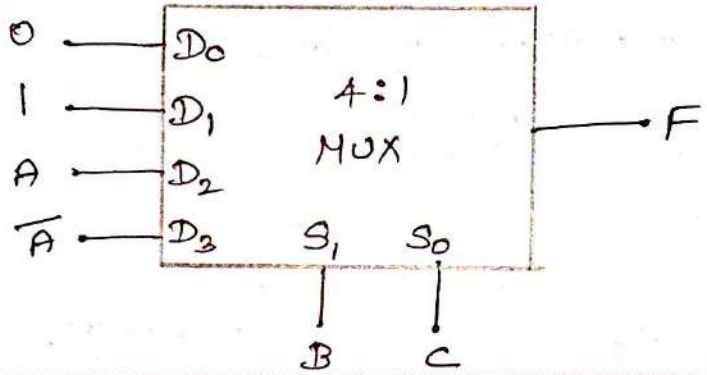
2.) Implement the following Boolean function using 4:1 MUX, $F(A,B,C) = \Sigma m(1,3,5,6)$

Soln :-

Implementation Table :-

	D_0	D_1	D_2	D_3
$\bar{A}$	0	(1)	2	(3)
A	4	(5)	(6)	7
	0	1	A	$\bar{A}$

MUX Implementation :-



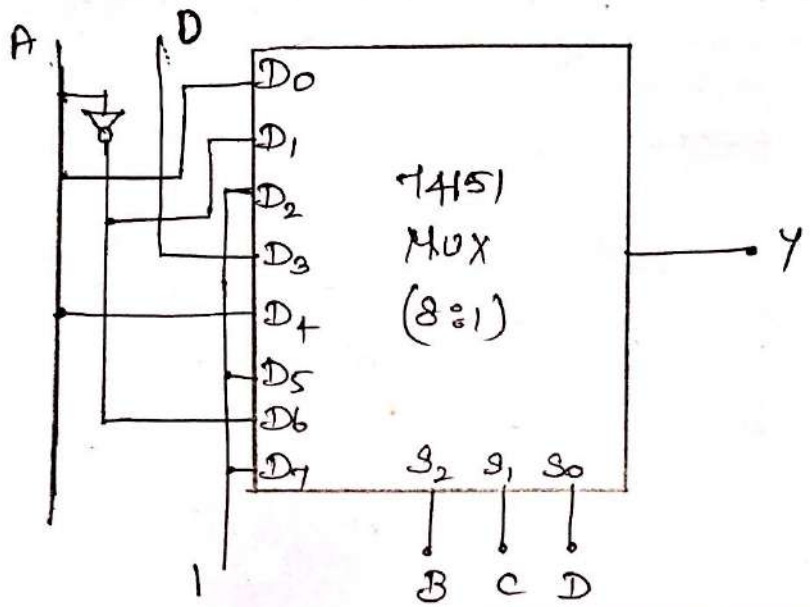
3.) Implement $F = \Sigma(1,2,5,6,7,8,10,12,13,15)$ using 74151 MUX.

Soln :-

Implementation Table :-

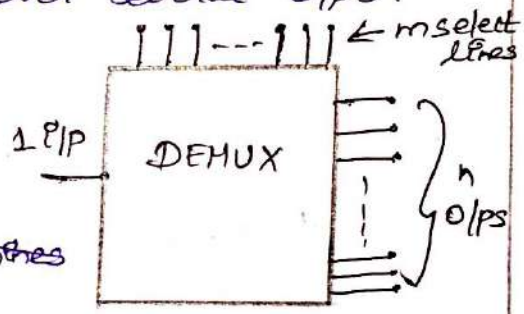
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	(1)	(2)	3	4	(5)	(6)	(7)
A	(8)	9	(10)	11	(12)	(13)	14	(15)
	A	$\bar{A}$	1	0	A	1	$\bar{A}$	1

Multiplexer Implementation



DEMULTIPLEXERS :- [Data Distributors]

* One to Many
* It takes information from one i/p and transmit the same over several o/ps.
* 1 input
* - o/p lines
* m - select lines
* ~~Select line determines to which o/p data~~



* At a time, only one o/p line is selected by the select lines and the i/p is transmitted to the selected o/p line.
* It is equivalent to single pole multiple way switch
* The enable i/p will enable the demux.
* Relation btw n o/p lines & m select lines is $n = 2^m$

1 to 4 Demultiplexer

Input – Single (D)

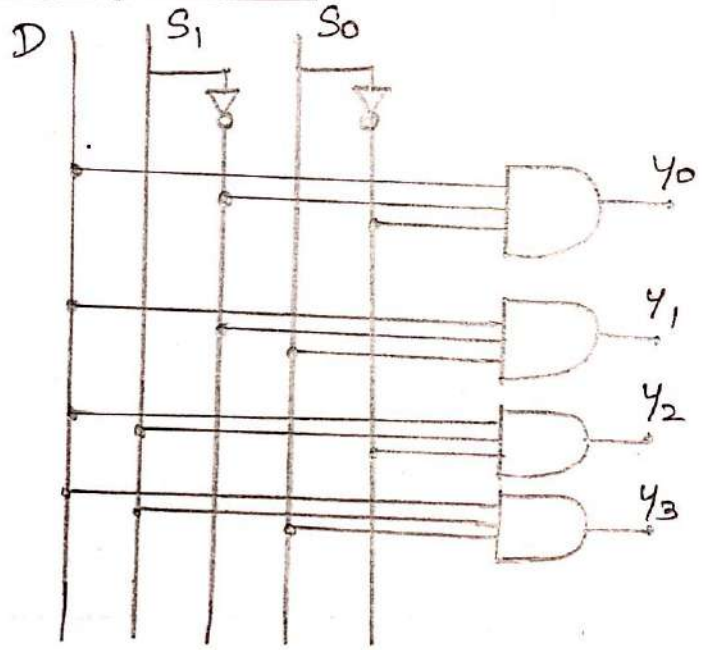
Output – 4 (Y_0, Y_1, Y_2, Y_3)

Select lines – 2 (S_0, S_1)

Truth Table :-

Data I/p	Select Input		Outputs				
D	S_1	S_0	Y_3	Y_2	Y_1	Y_0	
D	0	0	0	0	0	D	$Y_0 = D\bar{S_1}\bar{S_0}$
D	0	1	0	0	D	0	$Y_1 = D\bar{S_1}S_0$
D	1	0	0	D	0	0	$Y_2 = DS_1\bar{S_0}$
D	1	1	D	0	0	0	$Y_3 = DS_1S_0$

Logic Diagram :-



The two Select line enable one gate at a time and the data that appears on the I/p line passes through the selected gate to the O/p.

1 to 8 Demultiplexer

Consider a 1 to 8 demultiplexer which has a single i/p (D) and eight o/p (D_0 to D_7) and three select inputs (S_0, S_1 and S_2).

Inputs				Outputs							
S_2	S_1	S_0	D	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	D	D	0	0	0	0	0	0	0
0	0	1	D	0	D	0	0	0	0	0	0
0	1	0	D	0	0	D	0	0	0	0	0
0	1	1	D	0	0	0	D	0	0	0	0
1	0	0	D	0	0	0	0	D	0	0	0
1	0	1	D	0	0	0	0	0	D	0	0
1	1	0	D	0	0	0	0	0	0	D	0
1	1	1	D	0	0	0	0	0	0	0	D

From Truth Table

$D_0 = D\bar{S_2}\bar{S_1}\bar{S_0}$

$D_1 = D\bar{S_2}\bar{S_1}S_0$

$D_2 = D\bar{S_2}S_1\bar{S_0}$

$D_3 = D\bar{S_2}S_1S_0$

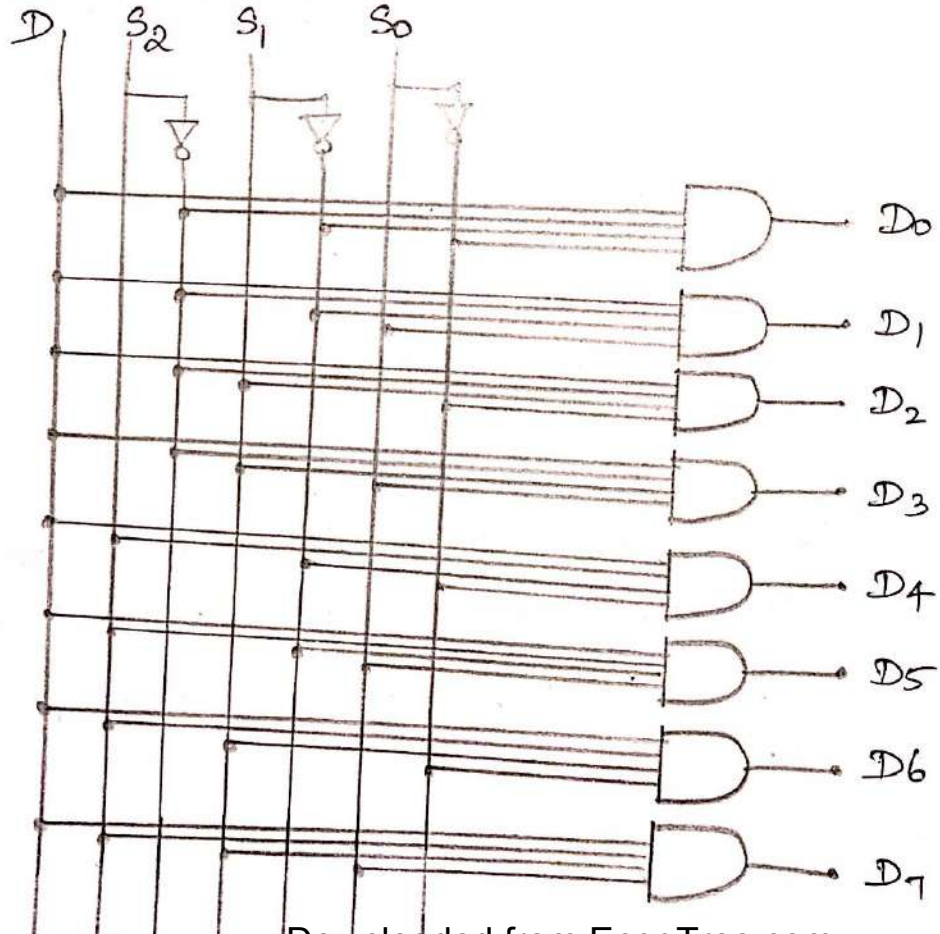
$D_4 = DS_2\bar{S_1}\bar{S_0}$

$D_5 = DS_2\bar{S_1}S_0$

$D_6 = DS_2S_1\bar{S_0}$

$D_7 = DS_2S_1S_0$

Logic Diagram :-



Applications of Demultiplexer :-

→ Used as decoder, data distributors.

→ Used in time division multiplexing at the receiving end as a data separator.

→ Used to implement Boolean expressions.

74154 — 1 : 16 Demultiplexer

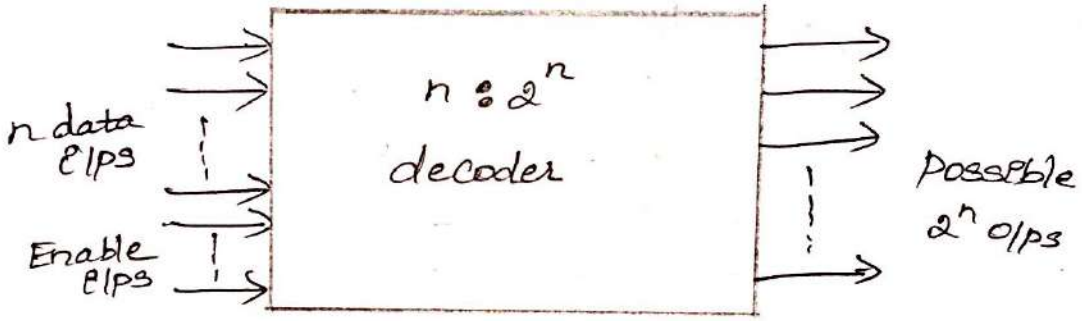
74155 — Dual 1 : 4 Demultiplexer.

DECODERS :-

A decoder is a multiple i/p – multiple o/p logic circuit which converts coded inputs into coded outputs, where i/p and o/p codes are different.

The i/p code generally has fewer bits than o/p code.

General Structure of decoder circuits :-

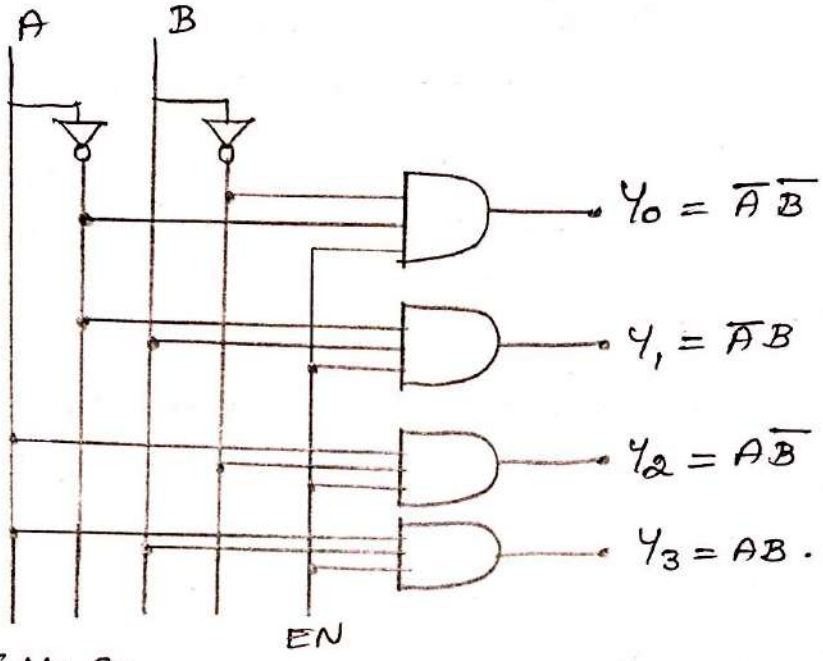


* The encoded information is presented as n i/ps producing 2^n possible o/ps.
* The 2^n o/p values are from 0 through $2^n - 1$.
* Sometimes an 'n'-bit binary code is truncated to represent fewer o/p values than 2^n.

Binary Decoder :-

A decoder which has an n-bit binary i/p code and one activated o/p, out of 2^n o/p code is called binary decoder.

A binary decoder is used when it is necessary to activate exactly one of the 2^n o/p based on an n-bit input value.



Truth Table :-

Inputs			Outputs			
EN	A	B	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Two i/ps are decoded into 4 o/ps, each o/p representing one of the minterms of 2 i/p variables.

The two invertors provide complement of the i/ps and each one of 4 AND gate generates one of the minterms.

Applications :- * Code converters
* Implementation of combinational ckts
* Address decoding
* BCD to 7-segment decoder.

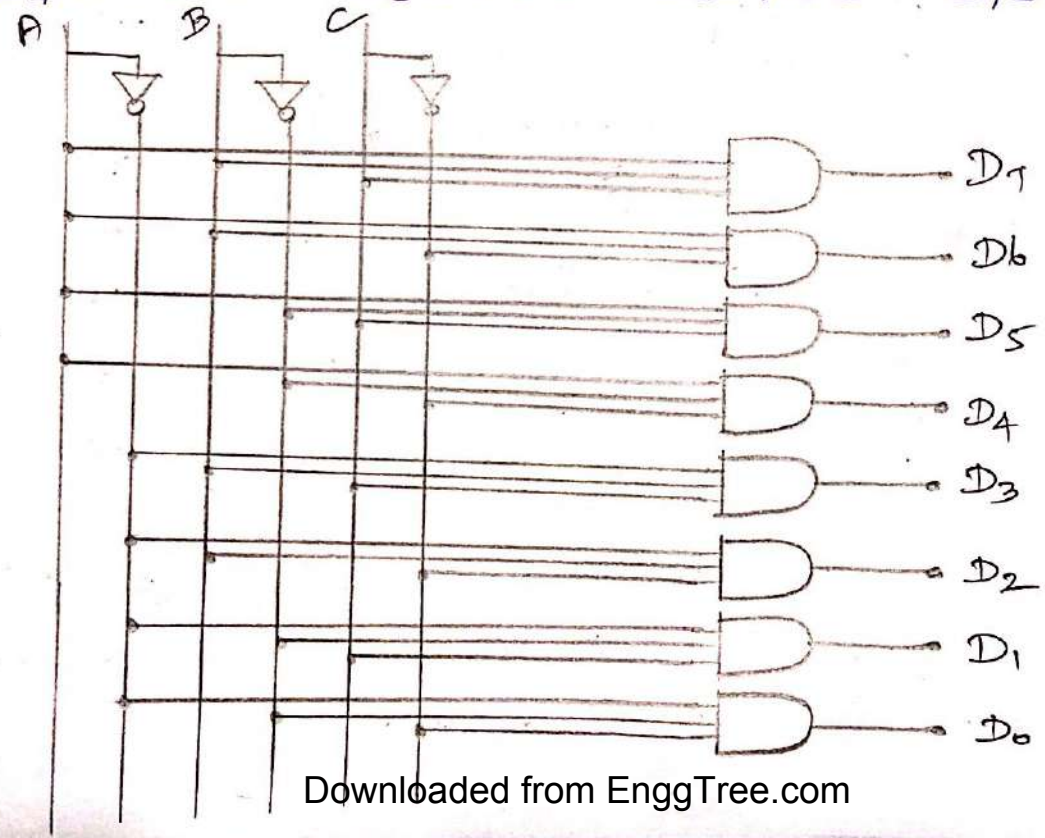
3 to 8 decoder

* 3 i/p (A, B, C)
* 8 o/p (D_0 to D_7)
* Only one of eight o/ps (D_0 to D_7) is selected based on 3 select i/ps.

Inputs			Outputs							
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

$D_0 = \overline{A}\,\overline{B}\,\overline{C}$ $D_1 = \overline{A}\,\overline{B}\,C$ $D_2 = \overline{A}\,B\,\overline{C}$ $D_3 = \overline{A}\,BC$

$D_4 = A\,\overline{B}\,\overline{C}$ $D_5 = A\,\overline{B}\,C$ $D_6 = AB\,\overline{C}$ $D_7 = ABC$.

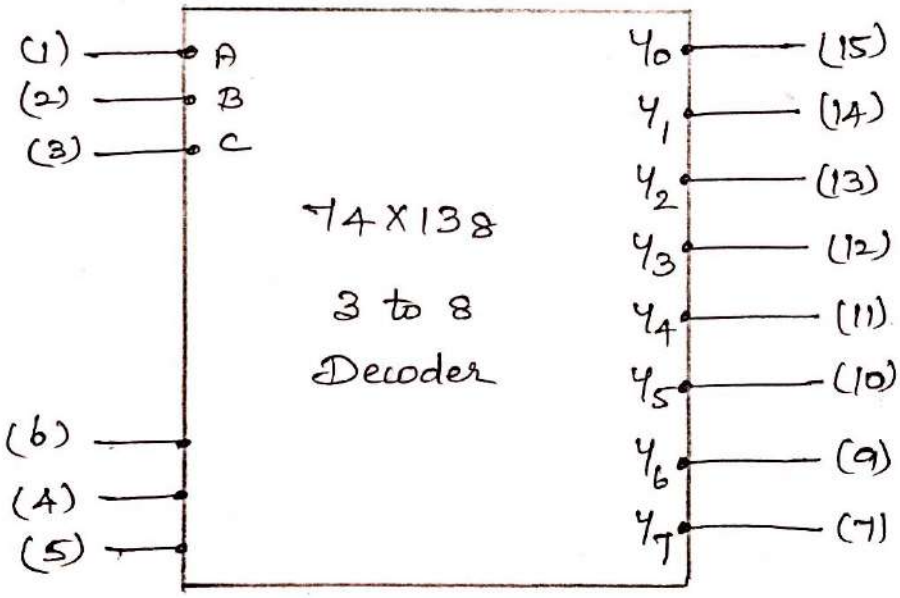


* It is also called as 1 of 8 decoder, since only one of eight o/p line is high.
* when ABC = 010, only AND gate 2 has all i/ps high and ∴ D_2 = HIGH.
* When ABC = 110, gate 6 has all i/p high ∴ D_6 = HIGH.
* It is also called as binary - octal decoder. Since the i/p represents 3 bit binary number and the o/p represent the eight digits in octal number system.

Enable i/ps :-

Some decoders have one or more enable i/ps to control the operation of the decoder, with enable line held HIGH, decoder functions normally and the i/p code A, B and C will determine which output is high.

IC 74X138 3 to 8 Decoder :-



74X138, it accepts 3 binary i/ps (ABC) and when enabled provides eight individual active low o/ps (Y_0 - Y_7).

The device has three enable i/ps :
two active low ($\overline{G_{2A}}$, $\overline{G_{2B}}$)
One active high (G_1)

IC 74139 - Dual 2 to 4 Decoder :-

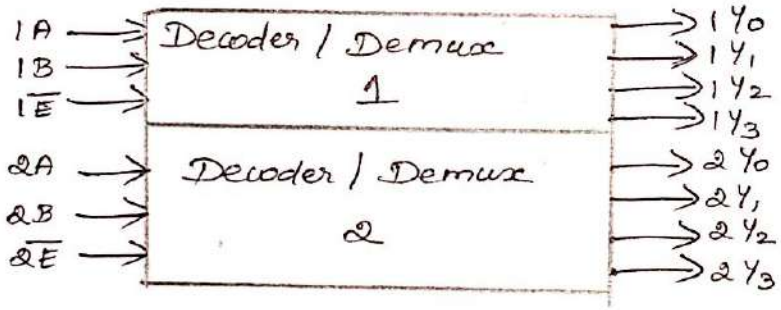
The 2 to 4 decoder with enable i/p can function as 1 to 4 line demultiplexer.

When $\overline{E}$ is taken as data i/p line and A & B are taken as selection i/ps.

Thus decoder with enable i/p is referred to as decoder / demultiplexer.

IC 74139 consists of two individual 2 to 4 decoder / demux in a single pack.

Each decoder has 2 - i/ps, 4 active low o/ps and 1 active low enable input. This active low enable i/p is used as data input in demux.



Realization of multiple output function using binary decoder :-

Decoder may have one of the two o/p states,
(i) active low
(ii) active high.

For active high output :-

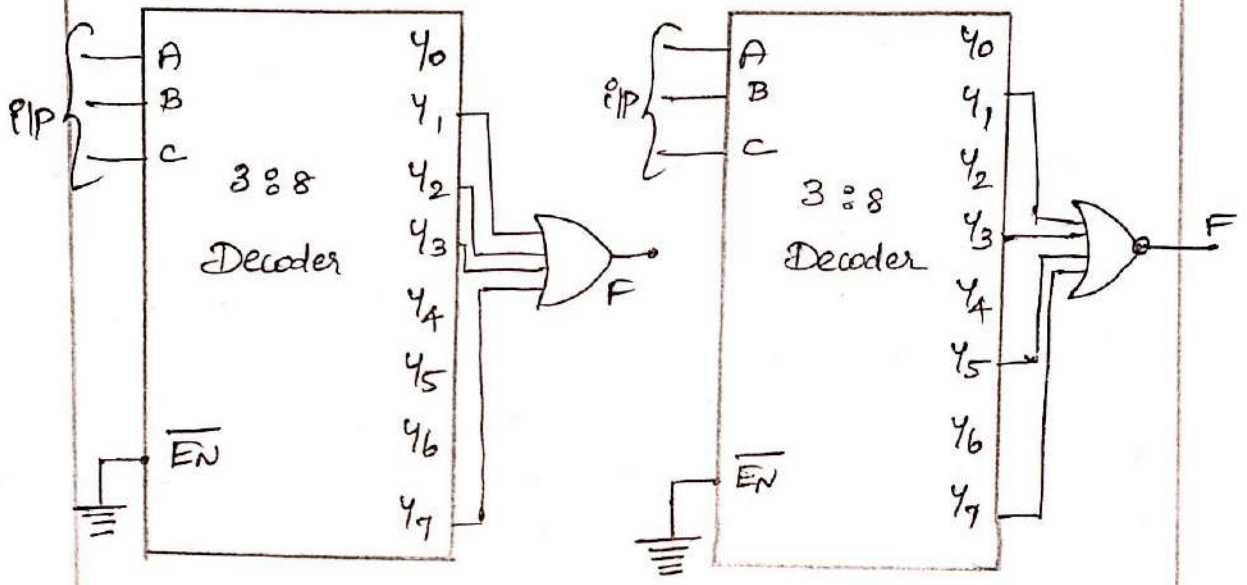
active high o/p + OR gate in the o/p = minterms of SOP
active high o/p + NOR gate in the o/p = maxterms of POS.

When decoder o/p is active high, it generates minterms for i/p variables (ie) it make selected o/p logic 1.

To implement SOP function one have to take sum of selected product terms generated by decoder, this is done by ORing the selected decoder o/p.

(Eg): $F = \Sigma m(1,2,3,7)$ using 3×8 decoder.

$F = \Sigma m(1,2,3,7)$ $\quad\quad$ $F = \pi M(1,3,5,7)$

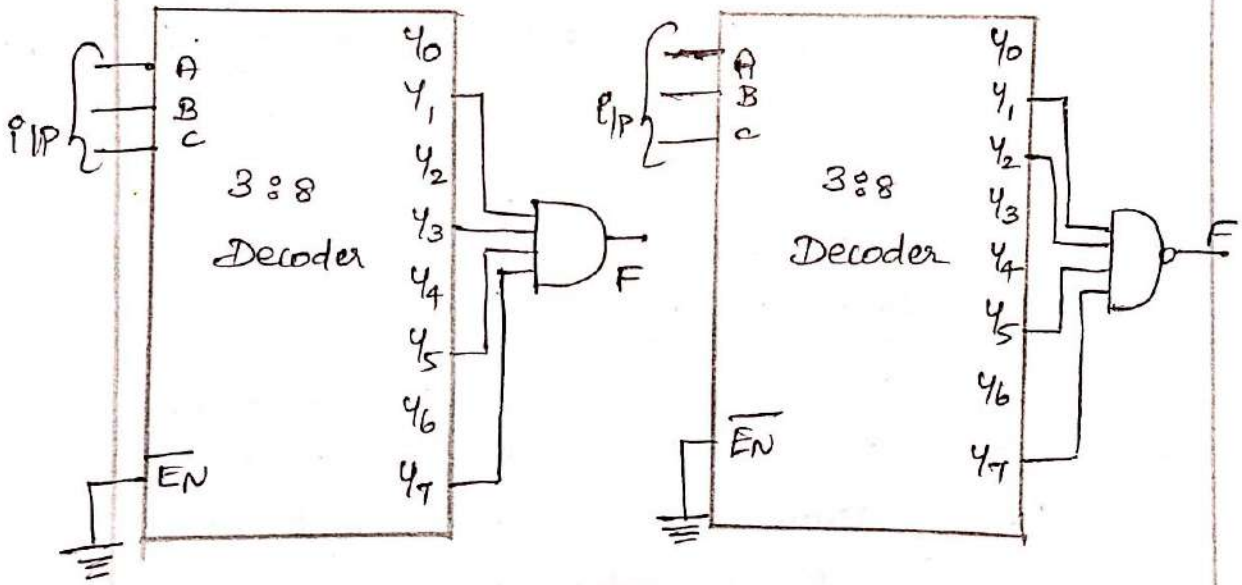


For active low o/p :-

active low o/p + and gate at o/p = Maxterms of pos

active low o/p + NAND gate at o/p = Minterms of SOP.

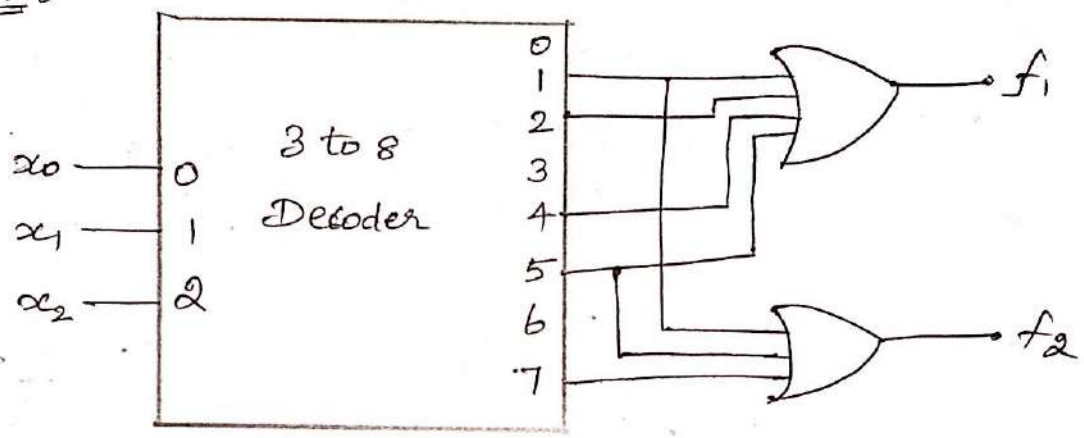
$F = \pi M(1,3,5,7)$ $\quad\quad$ $F = \Sigma m(1,2,5,7)$.



Implementation of Logic Boolean expression using decoder :-

1.) Implement the function $f_1(x_2, x_1, x_0) = \Sigma m(1,2,4,5)$ and $f_2 = \Sigma m(1,5,7)$ using 3 to 8 line decoder.

Soln :-



2.) Implement $f_1(x_2, x_1 x_0) = \Sigma m(0,1,3,4,5,6)$ & $f_2 = \Sigma m(1,2,3,4,6)$.

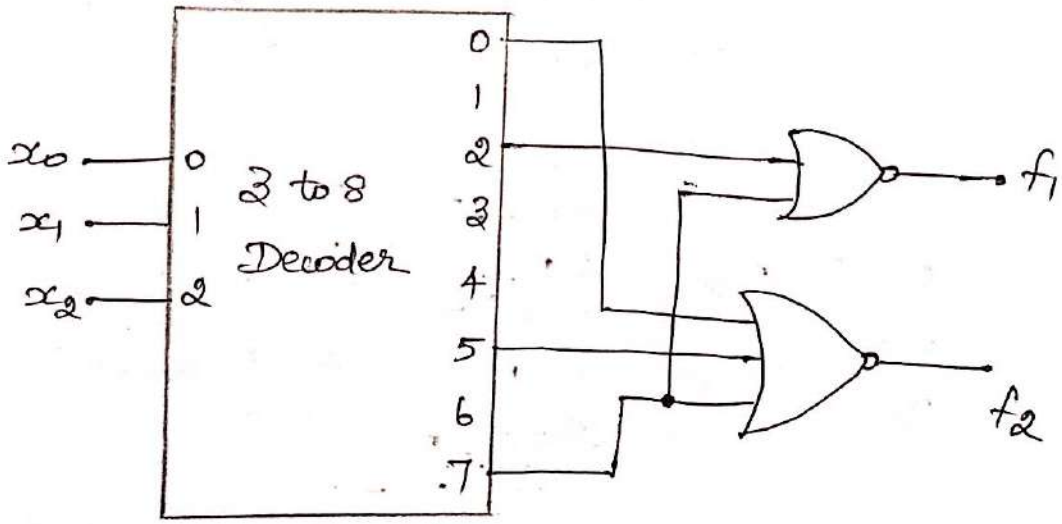
Soln :-

W.K.T, Complement of minterm canonical formula is the sum of those minterms not appearing in original formula

$f_1 = \Sigma m(0,1,3,4,5,6)$ $\quad$ $f_2 = \Sigma m(1,2,3,4,6)$

$\overline{f_1} = \Sigma m(2,7)$ $\quad$ $\overline{f_2} = \Sigma m(0,5,7)$

$\overline{\overline{f_1}} = \overline{\Sigma m(2,7)}$ $\quad$ $\overline{\overline{f_2}} = \overline{\Sigma m(0,5,7)}$



3.) Implement $f_1(x_2, x_1, x_0) = \Pi M(0,1,3,5)$ & $f_2(x_2, x_1, x_0) = \Pi M(1,3,6,7)$.

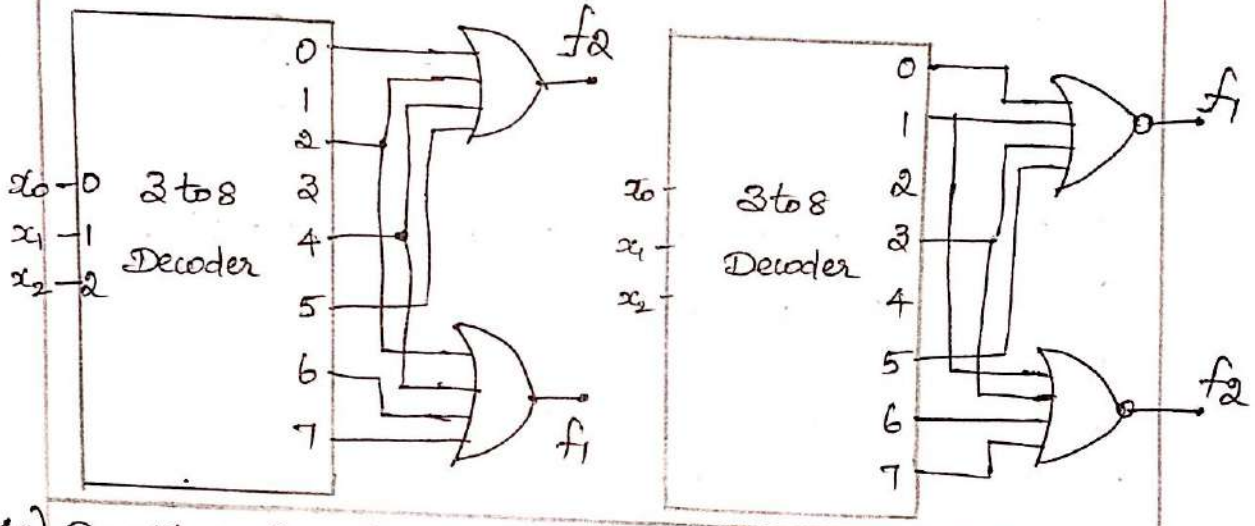
Maxterm canonical formula can be converted into an equivalent minterm canonical using leftout term.

$f_1(x_2, x_1, x_0) = \Pi M(0,1,3,5)$
$= \Sigma m(2,4,6,7)$

$f_2(x_2, x_1, x_0) = \Pi M(1,3,6,7)$
$= \Sigma m(0,2,4,5)$

$f_1(x_2, x_1, x_0) = \Pi M(0,1,3,5)$
$= \overline{\Sigma m(0,1,3,5)}$

$f_2(x_2, x_1, x_0) = \Pi M(1,3,6,7)$
$= \overline{\Sigma m(1,3,6,7)}$.



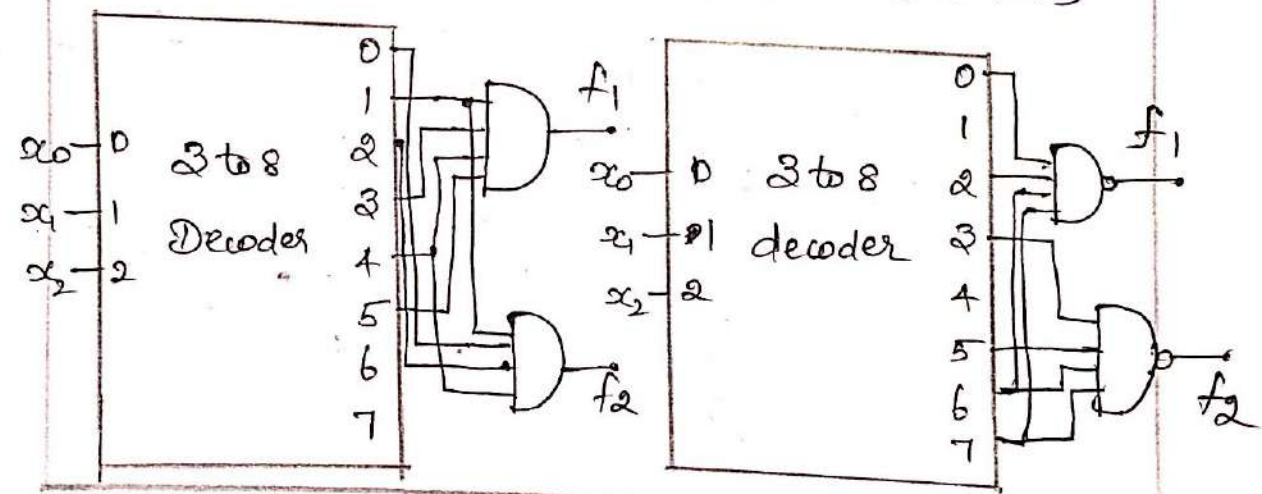
4.) Realize $f_1 = \Sigma m(0,2,6,7)$ & $f_2 = \Sigma m(3,5,6,7)$ using decoder with AND gates and NAND gates.

$f_1 = \Sigma m(0,2,6,7)$
$= \Pi M(1,3,4,5)$

$f_2 = \Sigma m(3,5,6,7)$
$= \Pi M(0,1,2,4)$

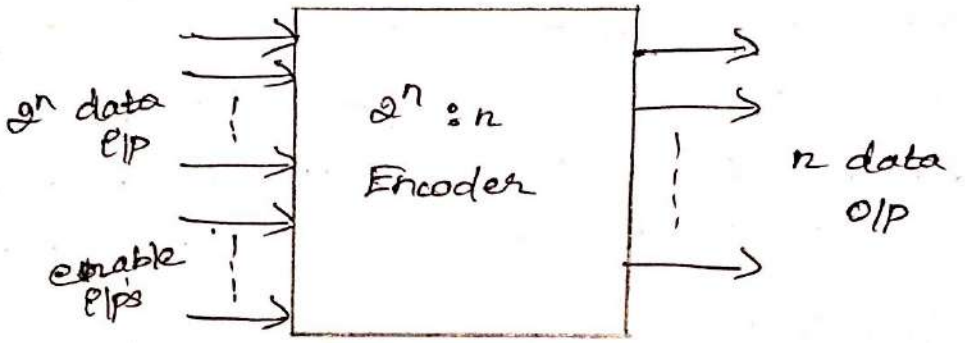
$f_1 = \Sigma m(0,2,6,7)$
$\overline{F_1} = \Pi M(0,2,6,7)$

$f_2 = \Sigma m(3,5,6,7)$
$\overline{f_2} = \Pi M(3,5,6,7)$



ENCODERS :-

* An encoder is a digital circuit that performs the inverse operation of a decoder.
* An encoder has 2^n (or fewer) i/p lines and n o/p lines.
* In encoder the o/p lines generate the binary code corresponding to the i/p value.



Priority Encoder :-

A priority Encoder is an encoder circuit that includes the priority function. In priority encoder, if 2 or more i/ps are equal to 1 at the same time, the i/p having the highest priority will take the precedence.

Inputs				Outputs.		
D_0	D_1	D_2	D_3	Y_1	Y_0	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

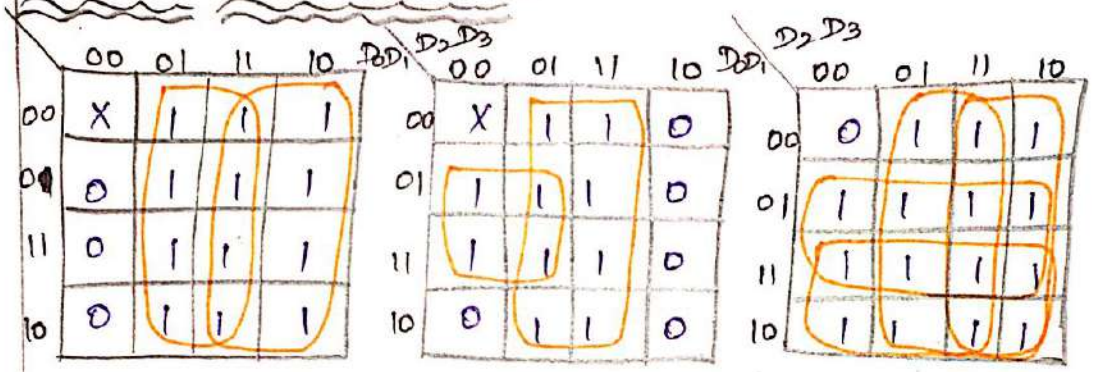
* D_3 has highest priority and D_0 has lowest priority when D_3 i/p is high regardless of other i/p's o/p is 11.
* D_2 has next priority. Thus $D_3 = 0$ & $D_2 = 1$, regardless of other two lowest priority i/p, o/p is 10.

→ The o/p for D_1 is generated only if higher priority i/ps are 0 and so on.

→ The o/p V indicates, one or more of the i/ps are equal to one.

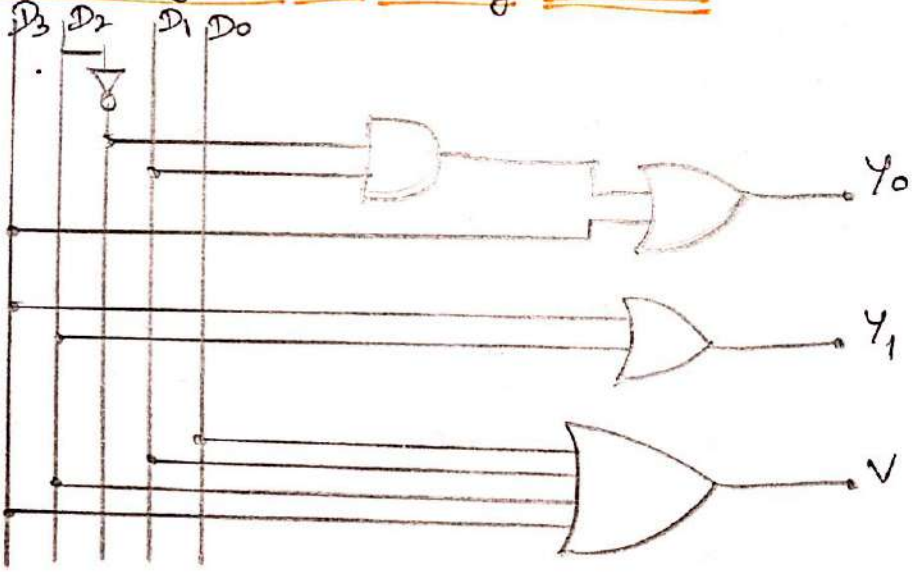
→ If all i/ps are 0, V is equal to 0, and the other 2 o/ps (Y_1 & Y_0) of the ckts are not used.

K-map Simplification :-

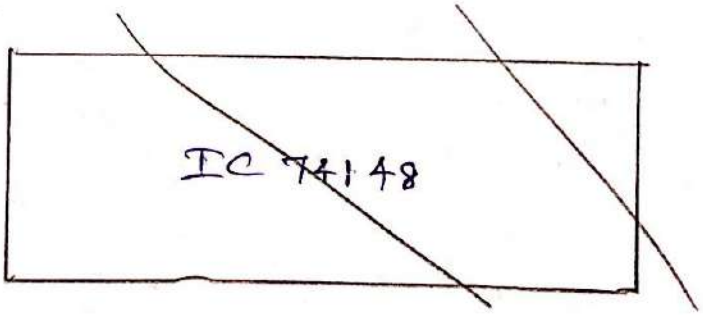


$Y_1 = D_2 + D_3$ $\quad Y_0 = D_3 + D_1\overline{D_2}$ $\quad Y = D_0 + D_1 + D_2 + D_3$

Logic Diagram for Priority Encoder :-



Priority Encoder IC 74XX148 :-



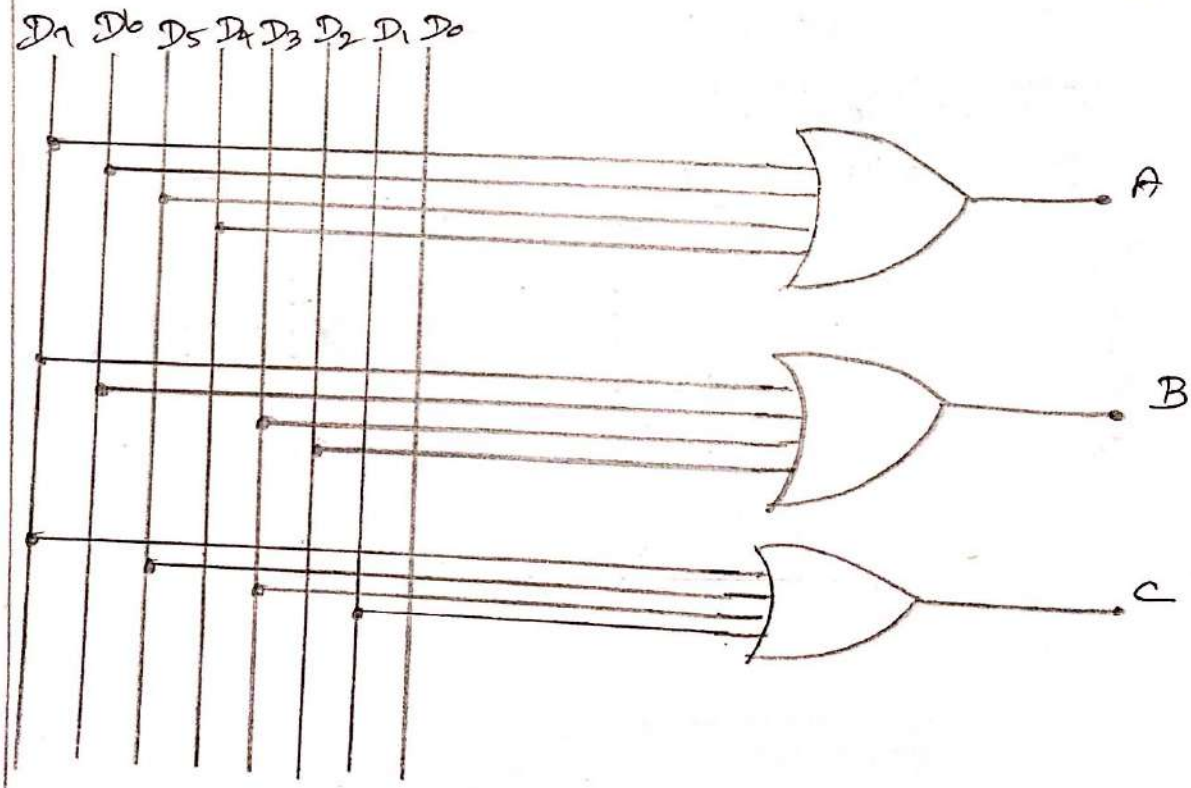
Octal to Binary Encoder

Octal to Binary Encoder has eight I/Ps and three O/Ps. It is assumed that only one I/P has a value 1 at any given time.

INPUT								OUTPUT		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

The Boolean Expression Is,

$A = D_4 + D_5 + D_6 + D_7$ $B = D_2 + D_3 + D_6 + D_7$ $C = D_1 + D_3 + D_5 + D_7$



DECIMAL TO BCD ENCODER:

Decimal to BCD encoder has got

⇒ 10 I/ps and 4 O/ps.

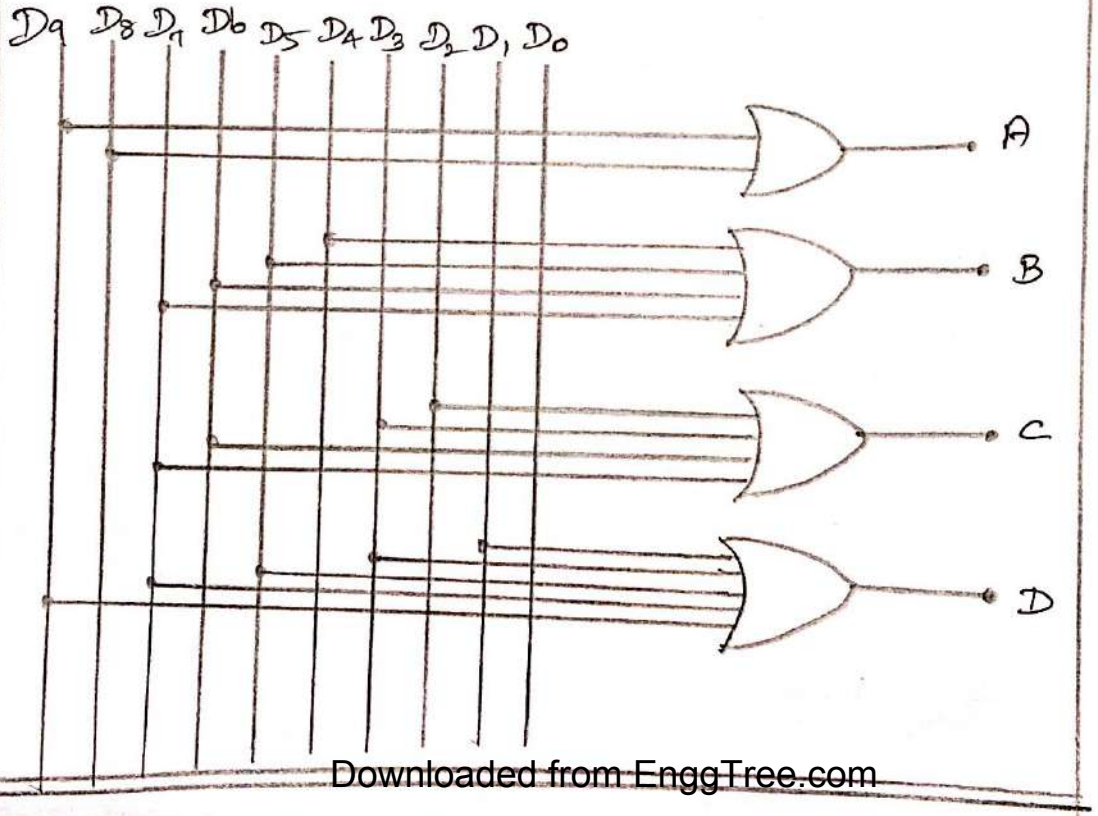
INPUTS										OUTPUTS			
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	A	B	C	D
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

Boolean Expressions for decimal to BCD encoder:

$A = D_8 + D_9$ $\quad B = D_4 + D_5 + D_6 + D_7$

$C = D_2 + D_3 + D_6 + D_7$ $\quad D = D_1 + D_3 + D_5 + D_7 + D_9$

Logic Diagram:



MAGNITUDE COMPARATOR

It is a combinational circuit, designed to compare the relative magnitude of two binary numbers (A & B) and generates one of the following o/p.

1.) $A = B$
2.) $A > B$
3.) $A < B$

Types :-
(1) 2-bit mag. Comparator
(2) 4-bit mag. Comparator.

(1.) 2-bit Magnitude Comparator :-

Let $A = A_1 A_0$; $B = B_1 B_0$

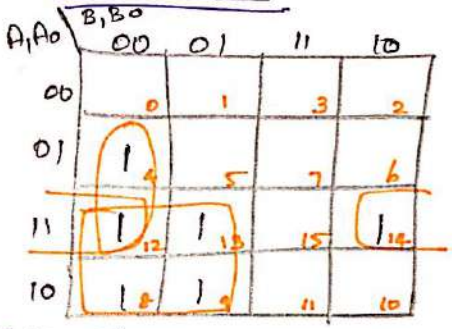
	A_1 A_0	B_1 B_0	$A>B$	$A=B$	$A<B$
0	0 0	0 0	0	1	0
1	0 0	0 1	0	0	1
2	0 0	1 0	0	0	1
3	0 0	1 1	0	0	1
4	0 1	0 0	1	0	0
5	0 1	0 1	0	1	0
6	0 1	1 0	0	0	1
7	0 1	1 1	0	0	1
8	1 0	0 0	1	0	0
9	1 0	0 1	1	0	0
10	1 0	1 0	0	1	0
11	1 0	1 1	0	0	1
12	1 1	0 0	1	0	0
13	1 1	0 1	1	0	0
14	1 1	1 0	1	0	0
15	1 1	1 1	0	1	0

$f(A>B) = \Sigma m(4, 8, 9, 12, 13, 14)$

$f(A=B) = \Sigma m(0, 5, 10, 15)$

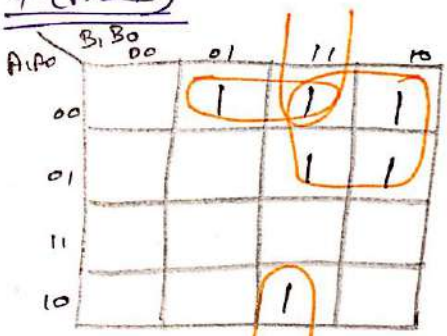
$f(A<B) = \Sigma m(1, 2, 3, 6, 7, 11)$

$f(A>B)$



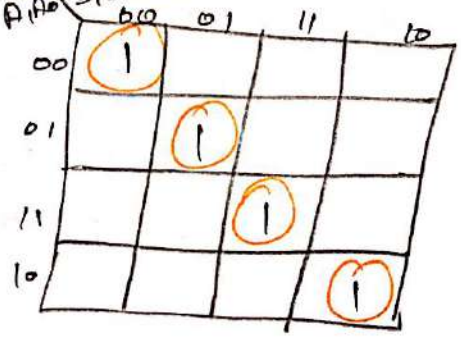
$f(A>B) = A_1 \overline{B_1} + A_1 A_0 \overline{B_0} + A_0 \overline{B_1}\,\overline{B_0}$

$f(A<B)$



$f(A<B) = \overline{A_1} B_1 + \overline{A_0} B_1 B_0 + \overline{A_1}\,\overline{A_0} B_0$

$f(A=B)$



$f(A=B) = \overline{A_1}\,\overline{A_0}\,\overline{B_1}\,\overline{B_0} + \overline{A_1} A_0 \overline{B_1} B_0 + A_1 A_0 B_1 B_0 + A_1 \overline{A_0} B_1 \overline{B_0}$

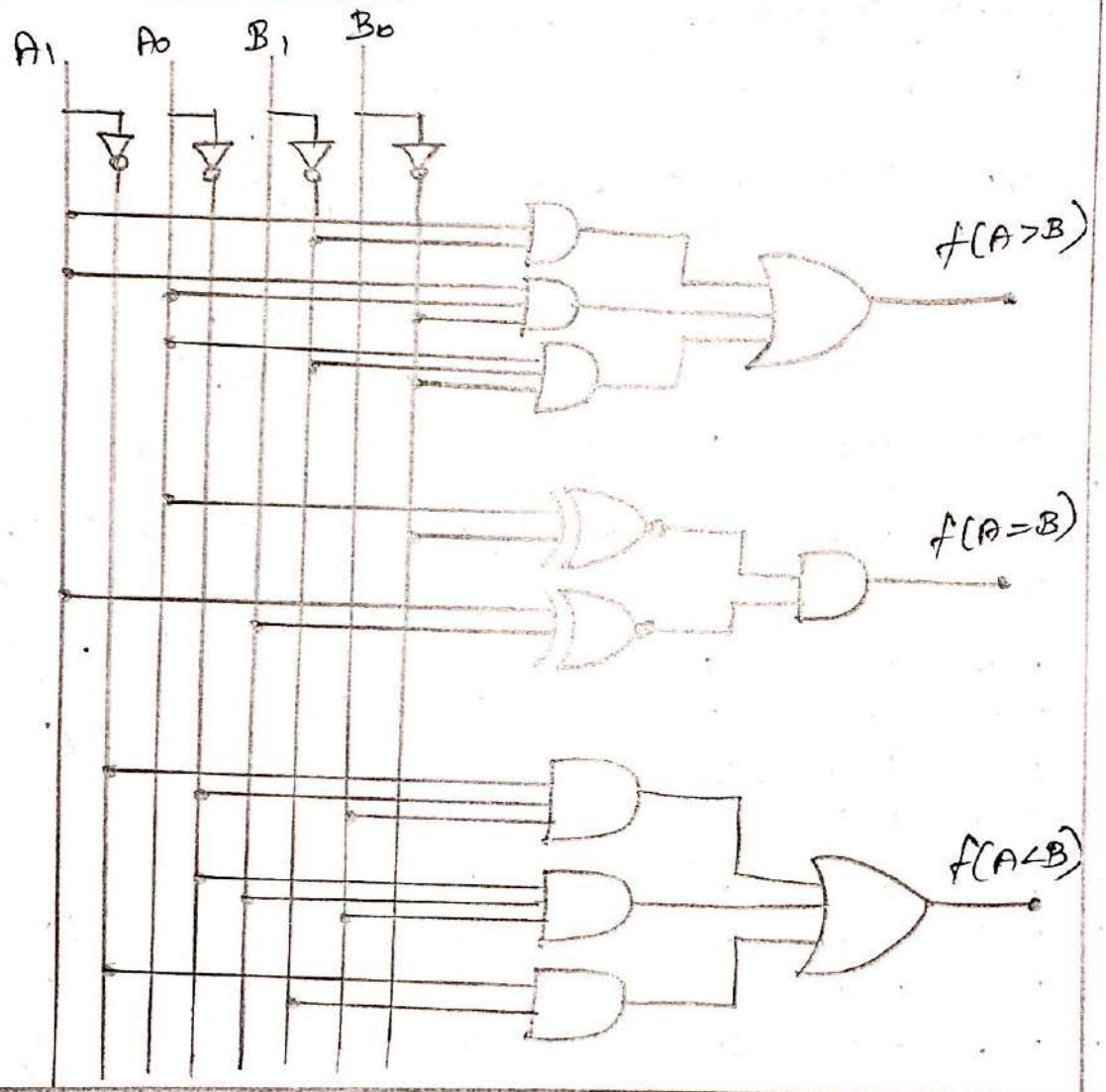
$$= \overline{A_1}\,\overline{B_1}\,(\overline{A_0}\,\overline{B_0} + A_0 B_0) + A_1 B_1 (A_0 B_0 + \overline{A_0}\,\overline{B_0})$$

$$= \overline{A_1}\,\overline{B_1}\,(A_0 \odot B_0) + A_1 B_1 (A_0 \odot B_0)$$

$$= (A_0 \odot B_0)(\overline{A_1}\,\overline{B_1} + A_1 B_1)$$

$$f(A=B) = (A_0 \odot B_0) \cdot (A_1 \odot B_1)$$

Logic Diagram :-



(2.) 4-bit Magnitude Comparator :-

To compare two 4-bit numbers, Consider

$A = A_3 A_2 A_1 A_0$

$B = B_3 B_2 B_1 B_0$.

To find if A=B :

Two 4-bit numbers A and B are said to be equal, if $A_3 = B_3$ & $A_2 = B_2$ & $A_1 = B_1$ & $A_0 = B_0$.

$E_3 = A_3 \odot B_3$

$E_2 = A_2 \odot B_2$

$E_1 = A_1 \odot B_1$

$E_0 = A_0 \odot B_0$

$\therefore f(A=B) = E_3 \cdot E_2 \cdot E_1 \cdot E_0$

To find if A>B :

i) If $A_3 > B_3$ ($A_3 = 1, B_3 = 0$) then A>B. It is represented as $A_3 \overline{B_3}$.

ii) If $A_3 = B_3$ & $A_2 > B_2$ ($A_2 = 1, B_2 = 0$), then A>B. It is represented as $E_3 A_2 \overline{B_2}$.

iii) If $A_3 = B_3$ & $A_2 = B_2$ & $A_1 > B_1$, ($A_1 = 1, B_1 = 0$), then A>B. It is represented as $E_3 E_2 A_1 \overline{B_1}$.

iv) If $A_3 = B_3$ & $A_2 = B_2$ & $A_1 = B_1$ & $A_0 > B_0$ ($A_0 = 1, B_0 = 0$), then A>B. It is represented as $E_3 E_2 E_1 A_0 \overline{B_0}$.

$\therefore f(A>B) = A_3 \overline{B_3} + [E_3 \cdot (A_2 \overline{B_2})] + [E_3 \cdot E_2 \cdot (A_1 \overline{B_1})] + [E_3 \cdot E_2 \cdot E_1 \cdot (A_0 \overline{B_0})]$

To find if A<B :

i) If $A_3 < B_3$ ($A_3 = 0$; $B_3 = 1$) ; then A<B. It is represented as $\overline{A_3} B_3$.

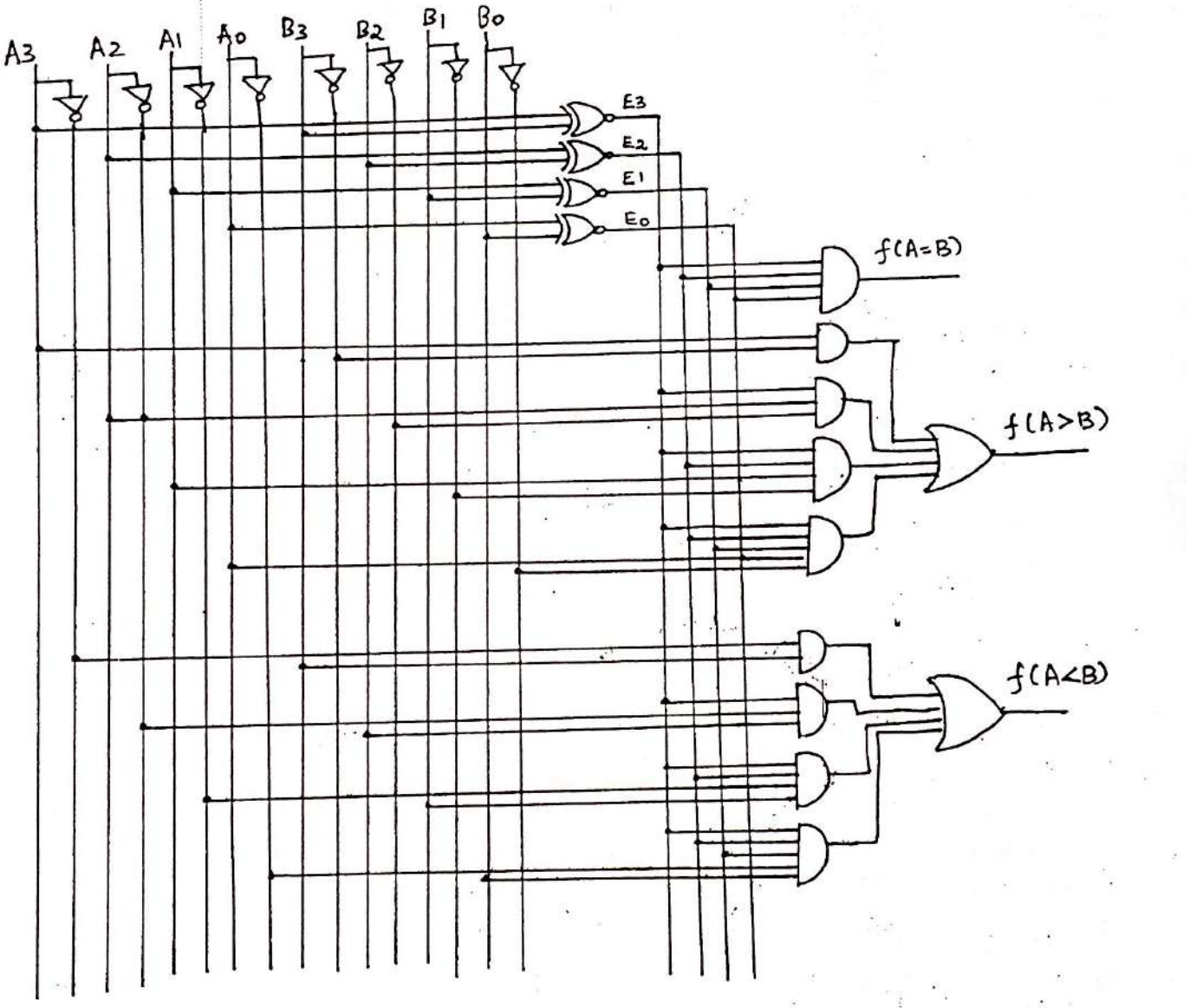
ii) If $A_3 = B_3$ & $A_2 < B_2$ ($A_2 = 0$; $B_2 = 1$) then A<B. It is represented as $E_3 \overline{A_2} B_2$.

iii) If $A_3 = B_3$ & $A_2 = B_2$ & $A_1 < B_1$, ($A_1 = 0$; $B_1 = 1$), then A<B. It is represented as $E_3 E_2 \overline{A_1} B_1$.

iv) If $A_3 = B_3$ & $A_2 = B_2$ & $A_1 = B_1$ & $A_0 < B_0$ ($A_0 = 0$; $B_0 = 1$) then A<B, It is represented as $E_3 E_2 E_1 \overline{A_0} B_0$.

$\therefore f(A<B) = \overline{A_3} B_3 + [E_3 \cdot (\overline{A_2} B_2)] + [E_3 \cdot E_2 \cdot (\overline{A_1} B_1)] + [E_3 \cdot E_2 \cdot E_1 \cdot (\overline{A_0} B_0)]$

Logic diagram for 4-bit Magnitude Comparator:



GATE LEVEL MINIMIZATION

* Gate level minimization is the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit

Simplification of Boolean Functions Using Karnaugh Map.

* The Karnaugh Map (K-Map) is a diagram made up of squares. Each square represents one minterm of the function that is to be minimized.

* The Map method is first proposed by Veitch and modified by Karnaugh.

* In this technique, the information contained in a truth table or the information contained in Pos/Sop form is represented on the map.

* This method is regarded as a pictorial form of a truth table.

Advantage of Map Method

⇒ Simple and straight forward procedure for minimizing the Boolean functions.

In K-Map,

→ Group of two cells is called a pair
→ Group of four cells is called a quad
→ Group of eight cells is called a octet.

⇒ 2- Variable K-map (contains $2^2 = 4$ squares)
⇒ 3- Variable K-map (contains $2^3 = 8$ squares)
⇒ 4- Variable K-Map (contains $2^4 = 16$ squares)
⇒ 5- Variable K-map (contains $2^5 = 32$ squares)

Conditions for Grouping:

* In choosing the adjacent squares in a map, we must ensure that

1. all the minterms/maxterms of the function are covered when we combine the squares (i.e all 1s/0s are grouped).
2. The number of terms in the expression should be minimized.
3. There should not be redundant terms
4. Grouping should not be done diagonally.
5. Odd number of squares should not be grouped.
6. Grouping of squares should be made in the power of 2 (2^n)

TWO - VARIABLE K-Map.

* In a two-variable K-Map, there are 4 squares (2^2), one for each minterms.

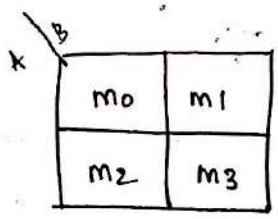
* In a 2-variable K-Map,

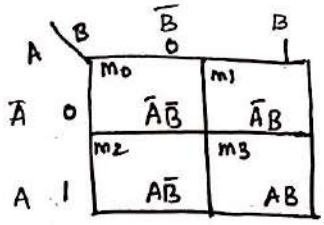
→ one square represents a minterm, giving a term with 2 literals

→ Two adjacent squares represent a term with 1 literal

→ Four adjacent squares represent a function that is always equal to 1.

Two Variable K-Map Representation.





① Map the expression $\bar{A}\bar{B} + \bar{A}B + AB$ using k-map and reduce the function

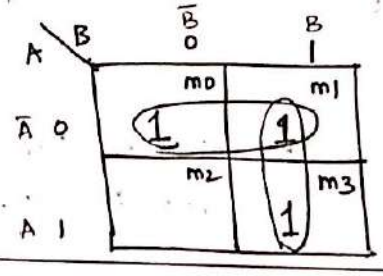
Solution:

*This expression can be expressed as sum of minterms as follows.

$F = \bar{A}\bar{B} + \bar{A}B + AB$

$= 00 + 01 + 11$

$F = \Sigma m(0, 1, 3)$. — Sum of minterms



$A = 1$
$\bar{A} = 0$

$F = \bar{A} + B$ Ans

② Map the expression $x'y + xy' + xy$ using k-Map and reduce the function.

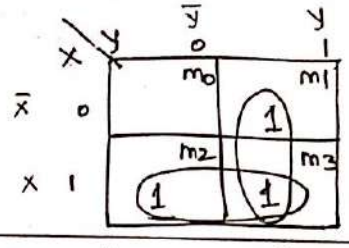
Solution:

*This expression can be expressed as sum of minterms as follows

$F = x'y + xy' + xy$

$= 01 + 10 + 11$

$F = \Sigma m(1, 2, 3)$. — Sum of minterms



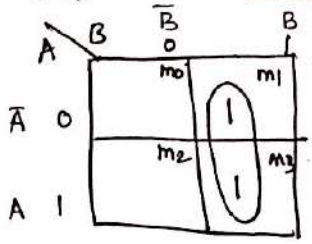
$X = 1$
$\bar{X} = 0$

$F = X + Y$ Ans.

③ Simplify the boolean function $f = \Sigma m(1, 3)$ using k-Map

Solution

Given: $\Sigma m(1, 3)$ — Sum of minterms



$A = 1$
$\bar{A} = 0$

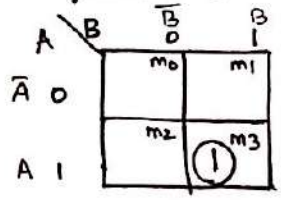
$F = B$ Ans

(4) Using K-map obtain Minimal SOP for $f = \Sigma m(3)$.

Solution:

Given $f = \Sigma m(3)$ — Sum of minterms

$A = 1$, $\bar{A} = 0$



$F = AB$ Ans.

∴ Minimal SOP = AB

(5) Using K-Map obtain minimal SOP for the given truth table.

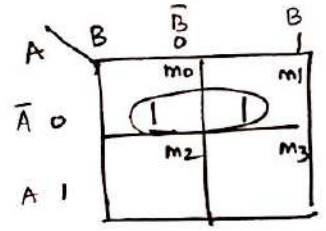
A	B	F
0	0	1
0	1	1
1	0	0
1	1	0

Solution:

* From the truth table, the minterms are identified as

$\Sigma m(0, 1)$

$A = 1$, $\bar{A} = 0$



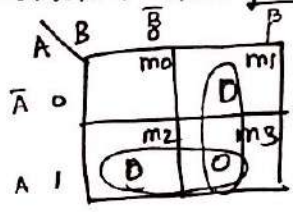
$F = \bar{A}$ Ans

∴ Minimal SOP = $\bar{A}$

(6) Obtain minimal POS for $f = \Pi(1,2,3)$ using k-map.

Solution:

Given the maxterms, $\Pi(1, 2, 3)$. The maxterms are plotted as 0 in the k-map



$F = \bar{A} \cdot \bar{B}$ Ans.

(POS form) $\bar{A} = 1$, $A = 0$

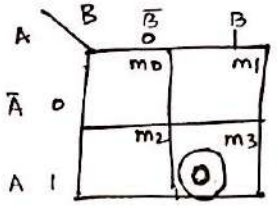
∴ Minimal POS = $\bar{A} \cdot \bar{B}$

(7) Obtain minimal POS for $f = \Pi(3)$ using k-map.

Solution:

Given the maxterms, $\Pi(3)$. The maxterms are represented as '0' in the k-map.

$\bar{A} = 1$, $A = 0$



$F = (\bar{A} + \bar{B})$

∴ Minimal POS = $\bar{A} + \bar{B}$

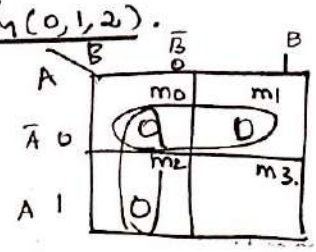
(8) Using K-Map obtain minimal POS for the given truth table.

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Solution

* From the truth table, the maxterms are identified as

$\Pi_M(0, 1, 2)$.



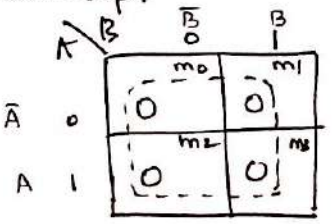
$F = B \cdot A$ (POS form)

Minimal POS = $A \cdot B$

(9) Using k-map obtain the minimal POS for $\Pi_M(0,1,2,3)$.

Solution: (Prod of maxterms)

Given POS $\Pi_M(0,1,2,3)$. The given maxterms are represented as '0' in the K-Map.



$A = 0$
$\bar{A} = 1$

Minimal POS = 1

These maxterms can be represented as $A\bar{B} + A\bar{B} + \bar{A}B + \bar{A}\bar{B}$

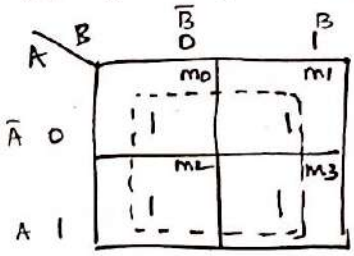
$= A(B+\bar{B}) + \bar{A}(B+\bar{B})$

$= A + \bar{A} = 1$

(10) Using k-Map obtain the minimal SOP for $\Sigma_m(0,1,2,3)$.

Solution: (Sum of minterms)

Given SOP $\Sigma_m(0,1,2,3)$. The given minterms are represented as '1' in the K-Map.



$A = 1$
$\bar{A} = 0$

Minimal SOP = 1

These minterms can be represented as $\bar{A}\bar{B} + \bar{A}B + A\bar{B} + AB$

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

$= \bar{A} + A = 1$

THREE-VARIABLE K-Map

* In a three-variable K-Map, there are 8 squares (2^3), one for each minterm.

* In a 3-variable K-Map,
  - → One square represents a minterm with three literals.
  - → Two adjacent squares represent a term with two literals.
  - → Four adjacent squares represent a term with one literal.
  - → Eight adjacent squares produce a function that is always equal to 1

* Any two adjacent squares in the Map differ by only one variable. Ie k-map uses a graycode sequence

Three Variable K-Map Representation.

A \ BC	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}C$ 01	BC 11	$B\bar{C}$ 10
$\bar{A}$ 0	m_0 $\bar{A}\bar{B}\bar{C}$	m_1 $\bar{A}\bar{B}C$	m_3 $\bar{A}BC$	m_2 $\bar{A}B\bar{C}$
A 1	m_4 $A\bar{B}\bar{C}$	m_5 $A\bar{B}C$	m_7 ABC	m_6 $AB\bar{C}$

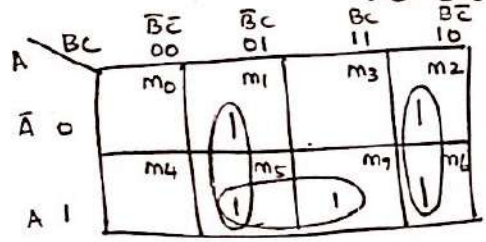
① Simplify $F = \bar{A}\bar{B}C + A\bar{B}C + \bar{A}B\bar{C} + AB\bar{C} + ABC$ using k-Map.

Solution:

$F = \bar{A}\bar{B}C + A\bar{B}C + \bar{A}B\bar{C} + AB\bar{C} + ABC$

$= 001 + 101 + 010 + 110 + 111$

$= \Sigma m(1,2,6,7,5)$ ⇒ Sum of minterms



$A = 1$, $\bar{A} = 0$

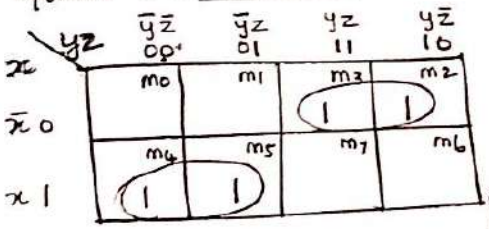
$G_1(1,5) = \bar{B}C$
$G_2(2,6) = B\bar{C}$
$G_3(5,7) = AC$

Simplified $F = \bar{B}C + B\bar{C} + AC$

② Simplify the Boolean function $F(x,y,z) = \Sigma(2,3,4,5)$ using k-map

Solution:

Given the minterms $\Sigma(2,3,4,5)$



$x = 1$, $\bar{x} = 0$

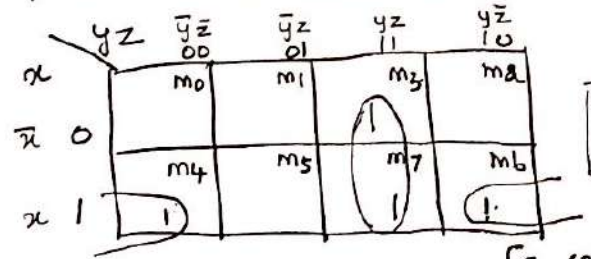
$G_1(4,5) = x\bar{y}$
$G_2(2,3) = \bar{x}y$

Simplified $F = x\bar{y} + \bar{x}y$

③ Simplify the Boolean function $F(x,y,z) = \Sigma(3,4,6,7)$ using k-Map.

Solution:

Given the minterms $\Sigma(3,4,6,7)$



$x = 1$, $\bar{x} = 0$

$G_1(3,7) = yz$
$G_2(4,6) = x\bar{z}$

Simplified $F = yz + x\bar{z}$

④ For the boolean function $F = \bar{A}C + \bar{A}B + A\bar{B}C + BC$

i) Express F as sum of minterms

ii) Find minimal SOP expression.

Solution:

i) Sum of minterms of F

$= \bar{A}C + \bar{A}B + A\bar{B}C + BC$

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

$= \bar{A}BC + \bar{A}\bar{B}C + \bar{A}BC + \bar{A}B\bar{C} + A\bar{B}C + ABC + \bar{A}BC$

$= \bar{A}BC + \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}C + ABC$

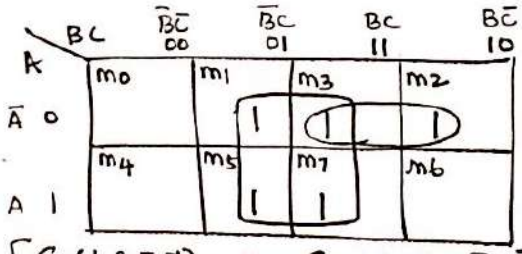
$= 011 + 001 + 010 + 101 + 111$

$= m_3 + m_1 + m_2 + m_5 + m_7$

$F = \Sigma(1,2,3,5,7)$ ⇒ sum of minterms

ii) Finding Minimal SOP expression

*Using 3-variable k-map, the minterms are represented as '1'.



$A = 1$, $\bar{A} = 1$

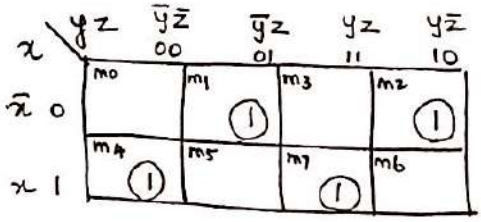
$[G_1(1,3,5,7) = C,\ G_2(2,3) = \bar{A}B]$

Minimal SOP = $C + \bar{A}B$

(5) Simplify $S(x,y,z) = \Sigma(1,2,4,7)$ using k-map.

Solution:

Given minterms = $\Sigma(1,2,4,7)$

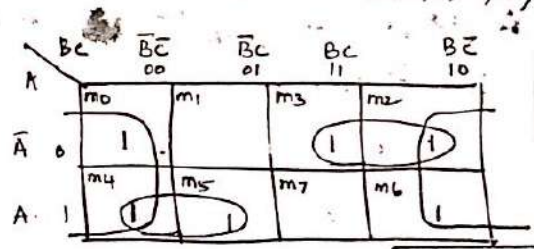


Ans $S = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$

(6) Simplify $F = \Sigma m(0,2,3,4,5,6)$ using k-map.

Solution:

Given minterms = $\Sigma(0,2,3,4,5,6)$



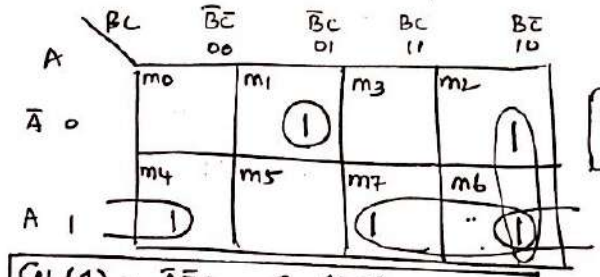
$G_1(0,2,4,6) = \bar{C}$
$G_2(4,5) = A\bar{B}$
$G_3(2,3) = \bar{A}B$

Ans: $F = \bar{C} + A\bar{B} + \bar{A}B$

(7) Obtain minimal SOP for $F = \Sigma(1,2,4,6,7)$ using K-Map

Solution:

Given minterms = $\Sigma(1,2,4,6,7)$



$G_1(1) = \bar{A}\bar{B}C$, $G_2(6,7) = AB$,
$G_3(2,6) = B\bar{C}$, $G_4(4,6) = A\bar{C}$

Ans: $F = \bar{A}\bar{B}C + A\bar{C} + AB + B\bar{C}$

(8) Minimize the expression $Y = A\bar{B}C + \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + \bar{A}B\bar{C}$

Solution:

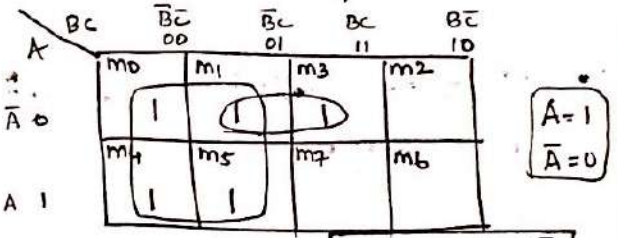
$Y = A\bar{B}C + \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + \bar{A}B\bar{C}$

Minterms are:

$Y = 101 + 001 + 011 + 100 + 000$

$= m_5 + m_1 + m_3 + m_4 + m_0$

$\Sigma m(0,1,3,4,5)$



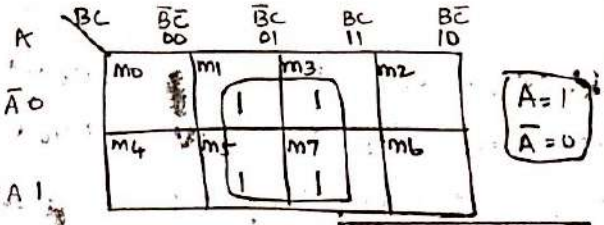
$G_1(0,1,4,5) = \bar{B}$
$G_2(1,3) = \bar{A}C$

Ans: $Y = \bar{B} + \bar{A}C$

(9) Minimize $Y(A,B,C) = \Sigma_m(1,3,5,7)$ using k-map.

Solution:

Given minterms = $\Sigma(1,3,5,7)$



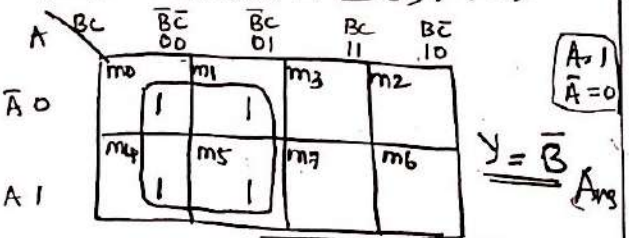
$G_1(1,3,5,7) = C$

Ans: $Y = C$

(10) Minimize $Y(A,B,C) = \Sigma_m(0,1,4,5)$ using k-map.

Solution:

Given minterms = $\Sigma(0,1,4,5)$



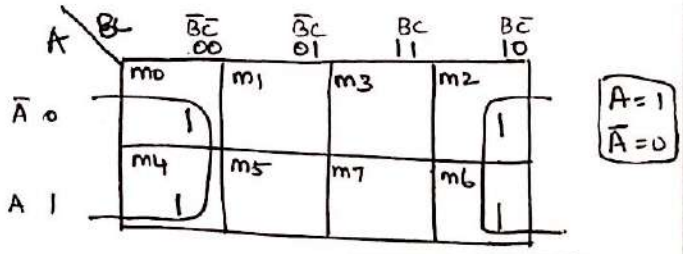
$G_1(0,1,4,5) = \bar{B}$

$Y = \bar{B}$ Ans

(11) Minimize $Y(A,B,C) = \Sigma m(0,3,4,6)$ using k-map.

Solution:

Given minterms = $\Sigma m(0,2,4,6)$



A=1
Ā=0

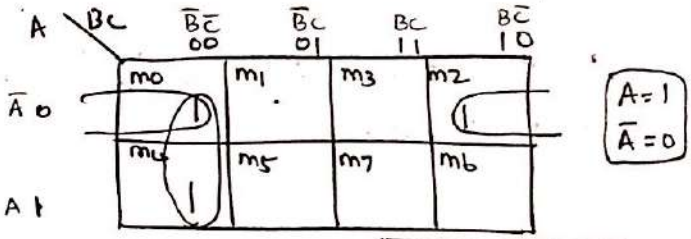
Ans $Y = \bar{C}$

$G_1(0,2,4,6) = \bar{C}$

(12) Minimize $Y(A,B,C) = \Sigma m(0,2,4)$ using k-map.

Solution:

Given minterms = $\Sigma m(0,2,4)$



A=1
Ā=0

Ans: $\bar{B}\bar{C} + \bar{A}\bar{C}$

$G_1(0,4) = \bar{B}\bar{C}$
$G_2(0,2) = \bar{A}\bar{C}$

(13) Minimize the logic function specified by the truth table using k-map. I/p:- A,B,C O/p:-Y

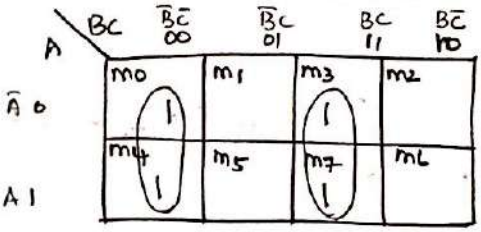
A	B	C	Y	
0	0	0	1	→ m0
0	0	1	0	→ m1
0	1	0	0	→ m2
0	1	1	1	→ m3
1	0	0	1	→ m4
1	0	1	0	→ m5
1	1	0	0	→ m6
1	1	1	1	→ m7

Solution:

Given the truth table. The minterms are identified as those terms the produces '1' for the function Y.

∴ Minterms are m0, m3, m4, m7

(ie) $Y = \Sigma m(0,3,4,7)$



A=1
Ā=0

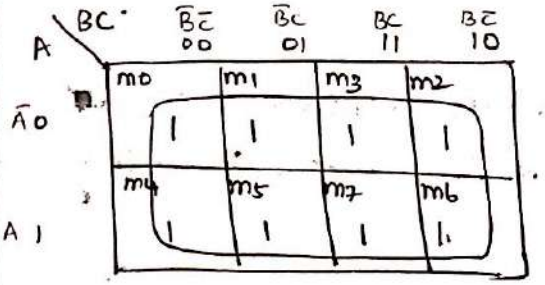
∴ Ans: $Y = \bar{B}\bar{C} + BC$

$G_1(0,4) = \bar{B}\bar{C}$
$G_2(3,7) = BC$

(14) Minimize $\Sigma m(0,1,2,3,4,5,6,7)$ using k-map.

Solution:

Given minterms = {0,1,2,3,4,5,6,7}



A=1
Ā=0

Ans F = 1

Justification:

$\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC + AB\bar{C}$

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

$= \bar{A}\bar{B} + \bar{A}B + A\bar{B} + AB$

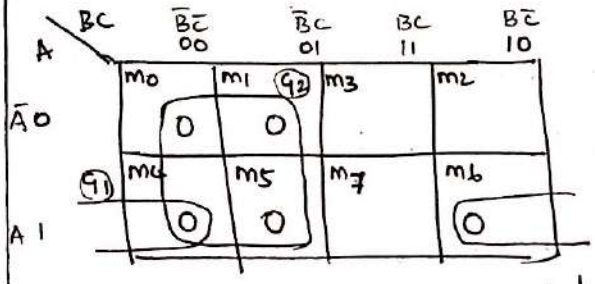
$= \bar{A}(B+\bar{B}) + A(B+\bar{B})$

$= \bar{A} + A = 1$

(16) Simplify the expression $Y = \Pi_M(0,1,4,5,6)$ using k-map.

Solution:

Given maxterms = $\Pi_M(0,1,4,5,6)$



(X) A=0, Ā=1

Notes: Maxterms are represented as '0' on the k-map.

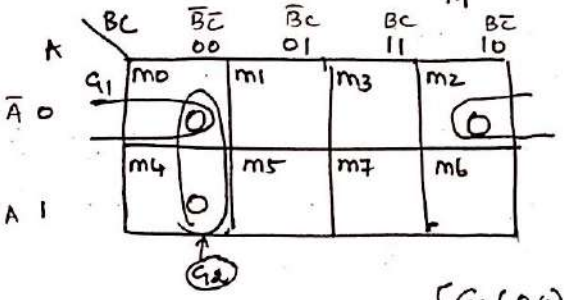
[G1(0,1,4,5) = B̄
G2(4,6) = Ā+C]

Ans $F = (\bar{A}+C)\cdot B$

(15) Simplify the expression $Y = \Pi_M(0,2,4)$ using k-map.

Solution:

Given maxterms = $\Pi_M(0,2,4)$



(X) A=0, Ā=1

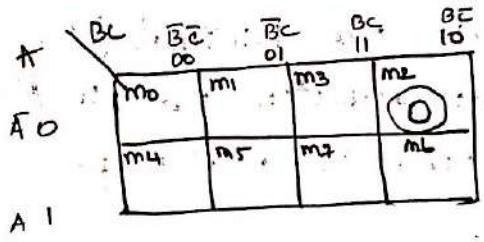
[G1(0,4) = B+C
G2(0,2) = A+C]

Ans: $(A+C)\cdot(B+C)$

(17) Simplify the expression $Y = \Pi_M(2)$ using k-map.

Solution:

Given maxterms = $\Pi_M(2)$



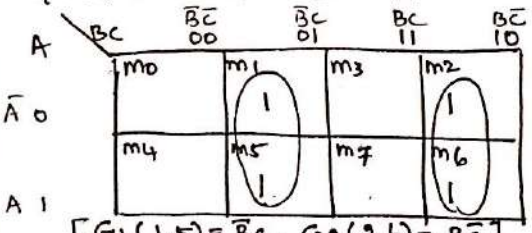
Ans: $Y = A+\bar{B}+C$.

(18) Obtain a) Minimal SOP b) Minimal POS for $F = \Sigma_m(1,2,5,6)$ using k-map.

Solution

a) Finding Minimal SOP:

Given Minterms = $\Sigma_m(1,2,5,6)$



(X) A=1, Ā=0

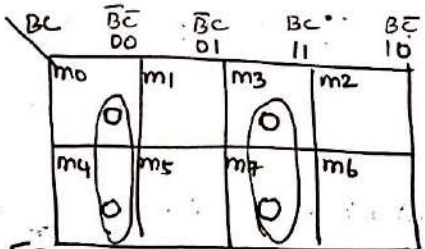
[G1(1,5) = B̄C, G2(2,6) = BC̄]

Minimal SOP = $\bar{B}C + B\bar{C}$ Ans

b) Finding minimal POS:

Given minterms = $\Sigma_m(1,2,5,6)$.

The maxterms are identified as $\Pi_M(0,3,4,7)$



(X) A=0, Ā=1

[G1(0,4) = B+C, G2(3,7) = B̄+C̄]

Minimal POS = $(B+C)\cdot(\bar{B}+\bar{C})$

(19) Obtain the minimal POS for the logic function given in the truth table using k-map.

Solution:

A	B	C	Y	
0	0	0	0	m_0
0	0	1	0	m_1
0	1	0	0	m_2
0	1	1	1	m_3
1	0	0	0	m_4
1	0	1	1	m_5
1	1	0	0	m_6
1	1	1	1	m_7

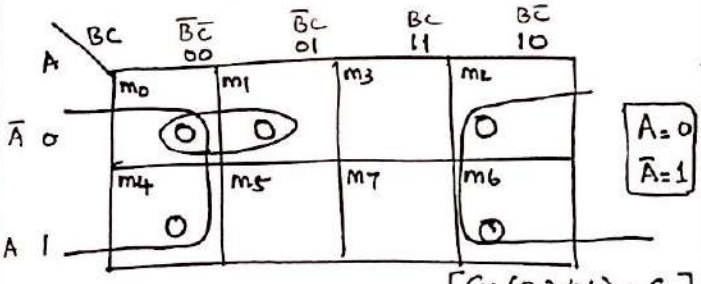
Maxterms

Minterms

*To obtain minimal POS, identify the maxterms from the truth table.

* Maxterms are identified as

$Y = \prod_M (0,1,2,4,6)$.



$Y = C \cdot (A+B)$

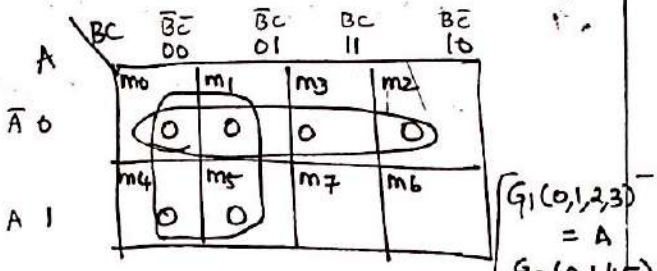
$[G_1(0,2,4,6) = C$
$G_2(0,1) = A+B]$

Simplified function $Y = C \cdot (A+B)$

(20) Minimize $F(A,B,C) = \prod_M (0,1,2,3,4,5)$ using k-map.

Solution:

Maxterms are given as (0,1,2,3,4,5)
They are plotted as '0' on the k-map



$[G_1(0,1,2,3) = A$
$G_2(0,1,4,5) = B]$

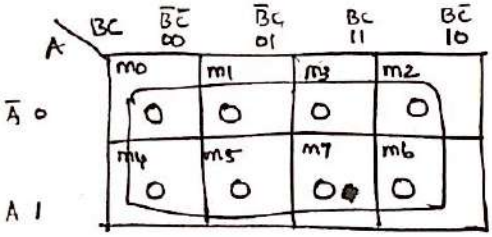
$F = B \cdot A$

∴ Minimized function $F = B \cdot A$

(21) Minimize $F(A,B,C) = \prod(0,1,2,3,4,5,6,7)$ using k-map.

Solution:

Given Maxterms = $\prod(0,1,2,3,4,5,6,7)$

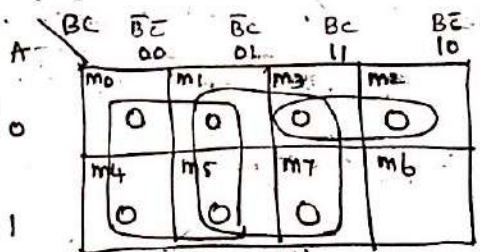


Ans: $F = 1$

(22) Minimize $F(A,B,C) = \prod(0,1,2,3,4,5,7)$ using k-map.

Solution

Given maxterms = $\prod(0,1,2,3,4,5,7)$



$[G_1(0,1,4,5) = \bar{B}$, $G_2(1,3,5,7) = \bar{C}$,
$G_3(2,3) = A+\bar{B}]$

Minimized Function = $B \cdot \bar{C} \cdot (A+\bar{B})$

FOUR-VARIABLE K-MAP.

* In a four-variable k-map, there are 16 squares (2^4), one for each minterms.

* In a 4-variable k-map,

→ One square represents one minterm, giving a term with four literals.

→ Two adjacent squares represent a term with three literals.

→ Four adjacent squares represent a term with two literals.

→ Eight adjacent squares represent a term with one literal.

→ Sixteen adjacent squares represent a function that is always 1.

* Four variable k-map Representation *

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

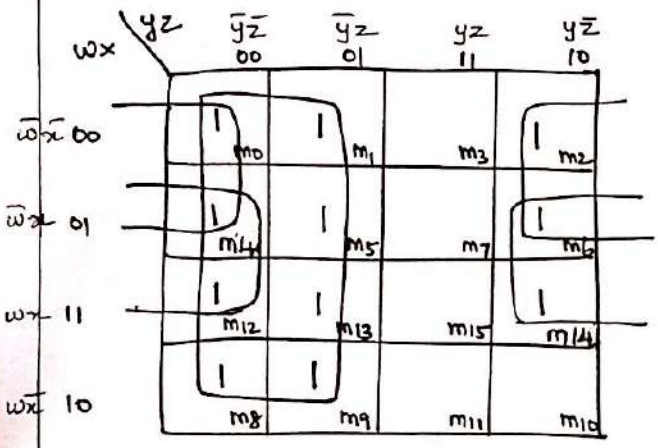
AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	m_0 $\bar{A}\bar{B}\bar{C}\bar{D}$	m_1 $\bar{A}\bar{B}\bar{C}D$	m_3 $\bar{A}\bar{B}CD$	m_2 $\bar{A}\bar{B}C\bar{D}$
$\bar{A}B$ 01	m_4 $\bar{A}B\bar{C}\bar{D}$	m_5 $\bar{A}B\bar{C}D$	m_7 $\bar{A}BCD$	m_6 $\bar{A}BC\bar{D}$
AB 11	m_{12} $AB\bar{C}\bar{D}$	m_{13} $AB\bar{C}D$	m_{15} $ABCD$	m_{14} $ABC\bar{D}$
$A\bar{B}$ 10	m_8 $A\bar{B}\bar{C}\bar{D}$	m_9 $A\bar{B}\bar{C}D$	m_{11} $A\bar{B}CD$	m_{10} $A\bar{B}C\bar{D}$

① Simplify the Boolean function

$F(w,x,y,z) = \Sigma\{0,1,2,4,5,6,8,9,12,13,14\}$ using k-map.

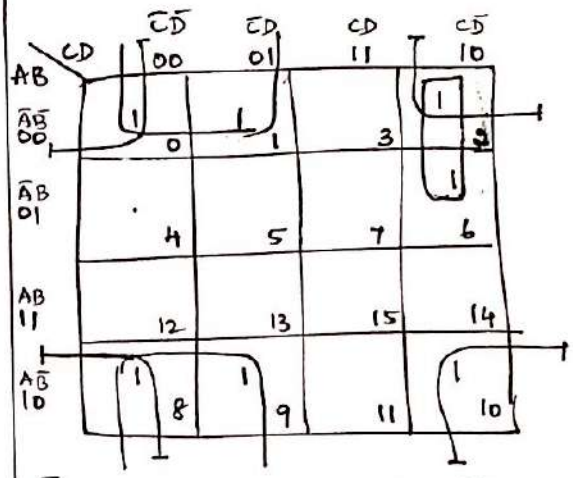
Solution :

Since the function has four variables, a four-variable map must be used. The minterms listed in the sum are marked by 1's in the map.



Group 1 $(0,1,4,5,12,13,8,9) = \bar{y}$
Group 2 $(0,4,2,6) = \bar{w}\bar{z}$
Group 3 $(4,12,6,14) = x\bar{z}$

∴ Simplified function $F = \bar{y} + \bar{w}\bar{z} + x\bar{z}$



Group 1 $(0,1,8,9) = \bar{B}\bar{C}$
Group 2 $(2,6) = \bar{A}C\bar{D}$
Group 3 $(0,2,8,10) = \bar{B}\bar{D}$

∴ Simplified function $F = \bar{B}\bar{C} + \bar{A}C\bar{D} + \bar{B}\bar{D}$

② Simplify the function $F = \bar{A}\bar{B}\bar{C} + \bar{B}C\bar{D} + \bar{A}BC\bar{D} + A\bar{B}\bar{C}$ using k-map.

Solution:

* The given function F should be expressed as sum of minterms.

$$F = \bar{A}\bar{B}\bar{C} + \bar{B}C\bar{D} + \bar{A}BC\bar{D} + A\bar{B}\bar{C}$$

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

$$= \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + \bar{A}\bar{B}CD + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D + A\bar{B}\bar{C}\bar{D}$$

$$= 0001 + 0000 + 1010 + 0010 + 0110 + 1001 + 1000$$

$$= m_1 + m_0 + m_{10} + m_2 + m_6 + m_9 + m_8$$

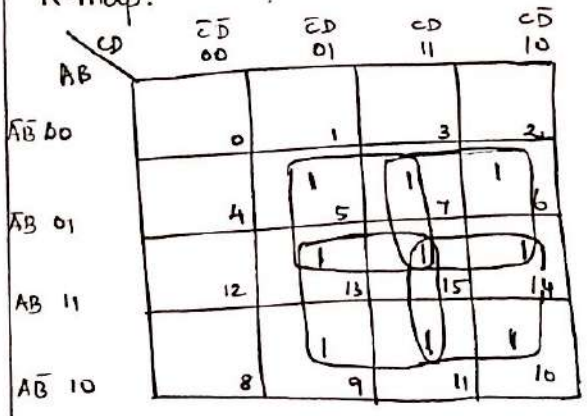
$$F = \Sigma(0,1,2,6,8,9,10)$$ (Ans)

* The minterms listed in the sum are marked by 1's in the map.

③ Find minimal SOP for $F = \Sigma m(5,6,7,9,10,11,13,14,15)$ using K-Map.

Solution:

Given the sum of minterms. These minterms are expressed as 1's in the K-map.



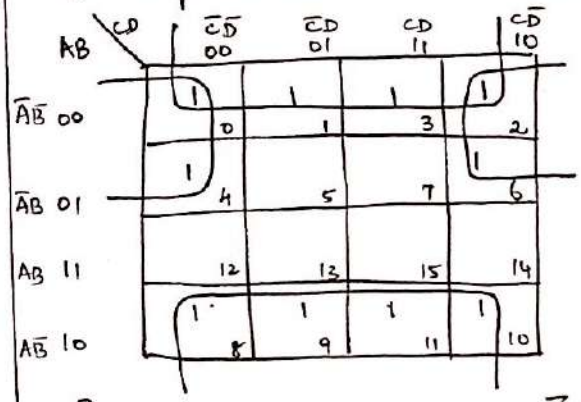
$G_1(5,7,13,15) = BD$, $G_2(6,7,14,15) = BC$
$G_3(8,9,12,13) = A\bar{D}$, $G_4(10,11,14,15) = A\bar{C}$

∴ Simplified $F = BD + BC + A\bar{C} + A\bar{D}$

④ Find minimal SOP for $F = \Sigma m(0,1,2,3,4,6,8,9,10,11)$ using K-map.

Solution:

Given the sum of minterms. These are represented as 1 in the map.



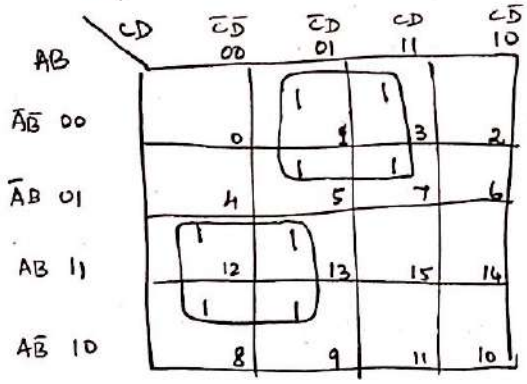
[Group 1 $(0,1,2,3,8,9,10,11) = \bar{B}$
Group 2 $(0,2,4,6) = \bar{A}\bar{D}$]

∴ Minimal SOP = $\bar{B} + \bar{A}\bar{D}$.

⑤ $Y = m_1 + m_3 + m_5 + m_7 + m_8 + m_9 + m_{12} + m_{13}$. Simplify the expression.

Solution:

Given the minterms $\Sigma(1,3,5,7,8,9,12,13)$. These minterms are represented as 1 in the map.



[Group 1 $(1,3,5,7) = \bar{A}D$, Group 2 $(8,9,12,13) = A\bar{C}$]

∴ Simplified function $F = \bar{A}D + A\bar{C}$

⑥ Plot the logical expression $ABCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}C + AB$ on a 4-variable map and reduce it.

Solution:

Find the sum of minterms for the logical expression

$Y = ABCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}C + AB$

$= ABCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}C(D+\bar{D}) + AB(C+\bar{C})(D+\bar{D})$

$= ABCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD + A\bar{B}C\bar{D} + ABC(D+\bar{D}) + AB\bar{C}(D+\bar{D})$

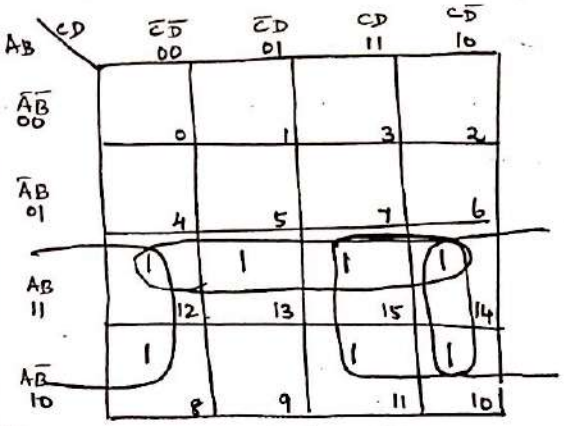
$= ABCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD + A\bar{B}C\bar{D} + ABCD + ABC\bar{D} + AB\bar{C}D + AB\bar{C}\bar{D}$.

$= ABCD + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD + A\bar{B}C\bar{D} + ABC\bar{D} + AB\bar{C}D + AB\bar{C}\bar{D}$

$= 1111 + 1000 + 1011 + 1010 + 1110 + 1101 + 1100$

$= m_{15} + m_8 + m_{11} + m_{10} + m_{14} + m_{13} + m_{12}$

Sum of minterms $= \Sigma(8,10,11,12,13,14,15)$



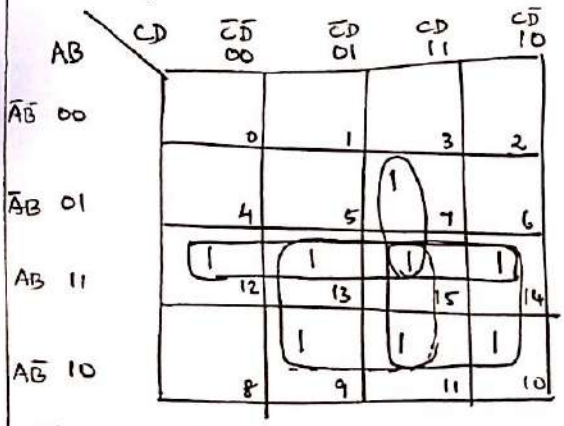
[Group 1 $(12,13,14,15) = AB$
Group 2 $(10,11,14,15) = AC$
Group 3 $(8,10,12,14) = A\bar{D}$]

∴ The reduced expression $= AB + AC + A\bar{D}$

⑦ Simplify $Y = \sum_m(7,9,10,11,12,13,14,15)$ using k-map.

Solution:

Given the minterms. These minterms are represented as 1 on the k-map.



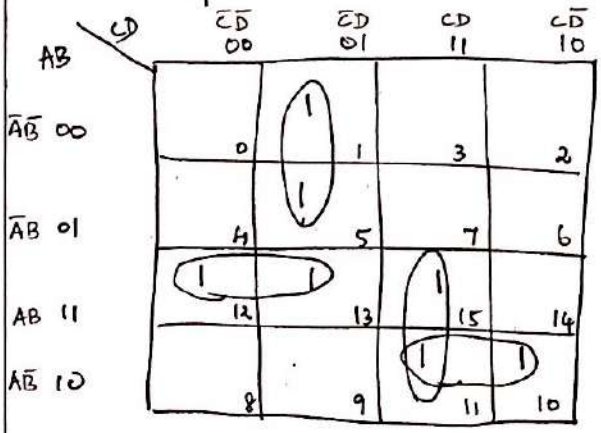
Group1 (12,13,14,15) = AB
Group2 (9,11,13,15) = AD
Group3 (10,11,14,15) = AC
Group4 (7,15) = BCD

∴ Simplified function F = $AB + AD + AC + BCD$

⑧ Simplify $Y = m_1 + m_5 + m_{10} + m_{11} + m_{12} + m_{13} + m_{15}$.

Solution

Given the minterms $\sum(1,5,10,11,12,13,15)$ These minterms are represented as 1 on the k-map.



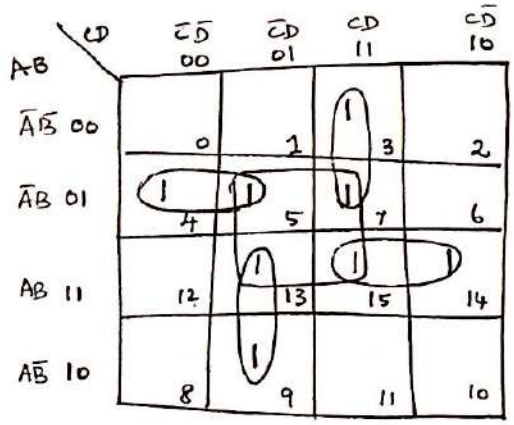
Group1 (1,5) = $\bar{A}\bar{C}D$
Group2 (12,13) = $AB\bar{C}$
Group3 (11,15) = ACD
Group4 (10,11) = $A\bar{B}C$

∴ Simplified $F = \bar{A}\bar{C}D + AB\bar{C} + ACD + A\bar{B}C$.

⑨ Simplify $Y = \sum_m(3,4,5,7,9,13,14,15)$ using k-map.

Solution:

Given the minterms $\sum(3,4,5,7,9,13,14,15)$. These minterms are represented as 1 on the k-map.



G1 (5,7,13,15) = BD
G2 (3,7) = $\bar{A}CD$
G3 (4,5) = $\bar{A}B\bar{C}$
G4 (9,13) = $A\bar{C}D$
G5 (14,15) = ABC

* Here Group 1 is redundant because all the minterms in this group belongs to another group. The redundant group has to be ignored

$Y = \bar{A}CD + \bar{A}B\bar{C} + A\bar{C}D + ABC$

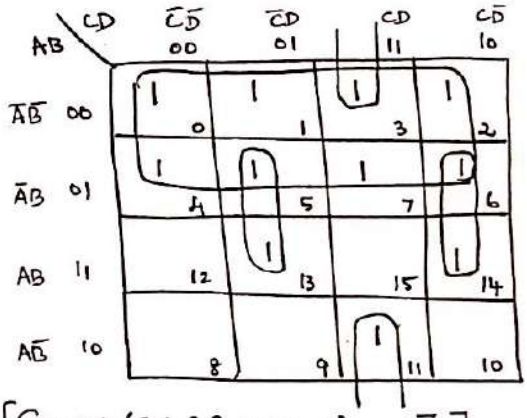
(10) Minimize

$Y(A,B,C,D) = \Sigma(0,1,2,3,4,5,6,7,11,13,14)$ **using k-map.**

Solution

Given minterms $\Sigma(0,1,2,3,4,5,6,7,11,13,14)$.

These minterms are represented as 1's in the k-map.



Group1 $(0,1,2,3,4,5,6,7) = \bar{A}$
Group2 $(3,11) = \bar{B}CD$
Group3 $(5,13) = B\bar{C}D$
Group4 $(6,14) = BC\bar{D}$

∴ Simplified function $Y =$

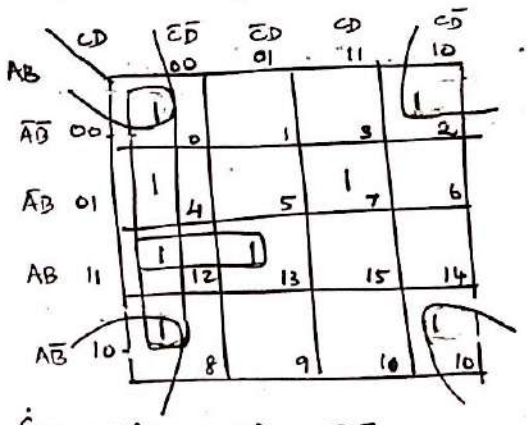
$$\bar{A} + B\bar{C}D + BC\bar{D} + \bar{B}CD$$

(11) Minimize $Y(A,B,C,D) = \Sigma(0,2,4,7,8,10,12,13)$ using k-map.

Solution:

Given minterms $\Sigma(0,2,4,7,8,10,12,13)$.

These minterms are represented as 1's in the k-map.



Group1 $(0,2,8,12) = \bar{C}\bar{D}$
Group2 $(12,13) = AB\bar{C}$
Group3 $(0,2,8,10) = \bar{B}\bar{D}$

∴ The Minimized function

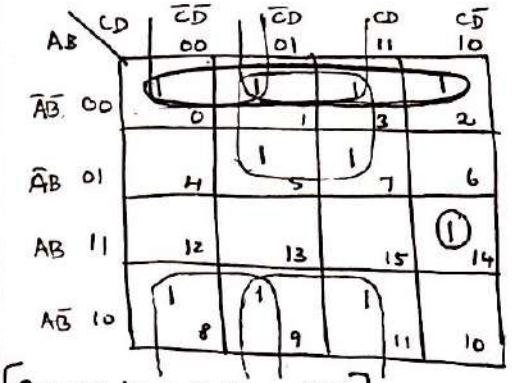
$$Y = \bar{C}\bar{D} + AB\bar{C} + \bar{B}\bar{D}.$$

(12) Minimize the following logic function using k-map.

$Y = \Sigma\{0,1,2,3,5,7,8,9,11,14\}$

Solution:

Given minterms. These minterms are represented as 1's in the k-map.



Group1 $(0,1,2,3) = \bar{A}\bar{B}$
Group2 $(1,3,5,7) = \bar{A}D$
Group3 $(0,1,8,9) = \bar{B}\bar{C}$
Group4 $(1,3,9,11) = \bar{B}D$
Group5 $(14) = ABC\bar{D}$

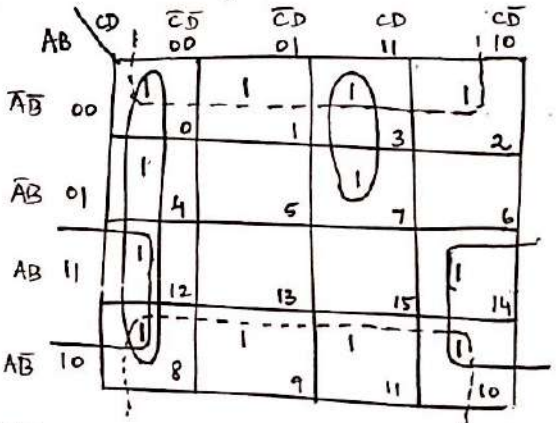
∴ Minimized function $Y =$

$$\bar{A}\bar{B} + \bar{A}D + \bar{B}\bar{C} + \bar{B}D + ABC\bar{D}.$$

(13) Simplify the Boolean expression $Y = \Sigma(0,1,2,3,4,7,8,9,10,11,12,14)$ using k-map.

Solution:

Given the minterms of the function. These minterms are represented as 1's on the k-map.



Group 1 $(0,1,2,3,8,9,10,11) = \bar{B}$
Group 2 $(0,4,8,12) = \bar{C}\bar{D}$
Group 3 $(8,10,12,14) = A\bar{D}$.
Group 4 $(3,7) = \bar{A}CD$

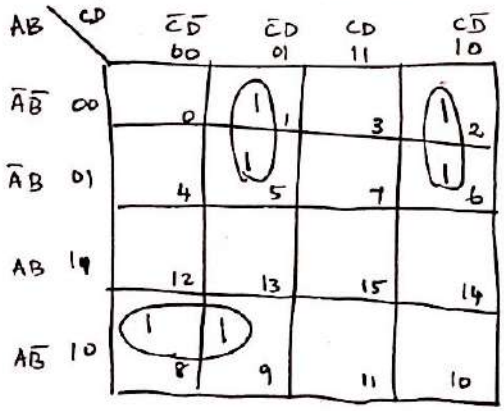
∴ Simplified function $Y =$

$$\bar{B} + \bar{C}\bar{D} + A\bar{D} + \bar{A}CD$$

(14) Simplify the Boolean expression $Y = \Sigma(1,2,5,6,8,9)$ using k-map.

Solution:

Given the minterms. These minterms are represented as 1's on the k-map.



Group1 $(1,5) = \bar{A}\bar{C}D$
Group2 $(2,6) = \bar{A}C\bar{D}$
Group3 $(8,9) = A\bar{B}\bar{C}$

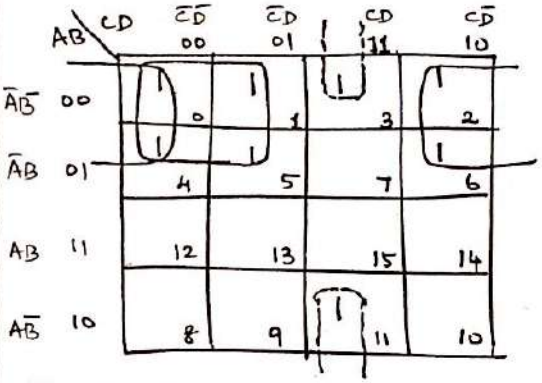
∴ The simplified Boolean expression

$$Y = \bar{A}\bar{C}D + A\bar{B}\bar{C} + \bar{A}C\bar{D}$$

(15) Simplify the Boolean expression $Y = \Sigma(0,1,2,3,4,5,6,11)$ using k-map.

Solution:

Given the minterms. These minterms are represented as 1's on the k-map



Group1 $(0,1,4,5) = \bar{A}\bar{C}$
Group 2 $(0,4,2,6) = \bar{A}\bar{D}$
Group 3 $(3,11) = \bar{B}CD$

∴ Simplified Boolean expression

$$Y = \bar{A}\bar{C} + \bar{A}\bar{D} + \bar{B}CD.$$

(15) Minimize the following Boolean expression using k-map $Y(A,B,C,D) = AB\bar{C} + BCD + BC\bar{D}$

Solution:

The given Boolean expression should be written as sum of minterms

$Y = AB\bar{C} + BCD + BC\bar{D}$

$= AB\bar{C}(D+\bar{D}) + BCD(A+\bar{A}) + BC\bar{D}(A+\bar{A})$

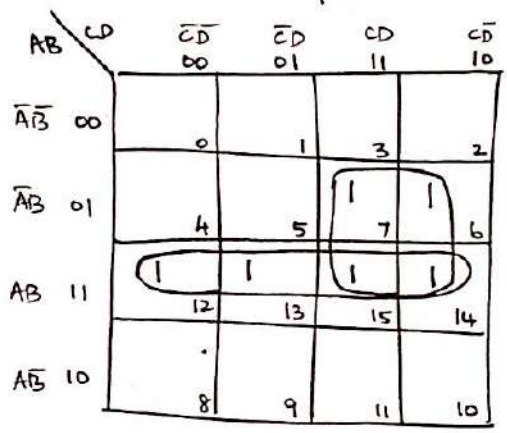
$= AB\bar{C}D + AB\bar{C}\bar{D} + ABCD + \bar{A}BCD + ABC\bar{D} + \bar{A}BC\bar{D}$

$= 1101 + 1100 + 1111 + 0111 + 1110 + 0110$

$= m_{13} + m_{12} + m_{15} + m_7 + m_{14} + m_6$

$\therefore Y = \Sigma(6,7,12,13,14,15)$ (*)

Now, these minterms are represented as 1's on the k-map.



[Group1 (6,7,14,15) = BC
Group2 (12,13,14,15) = AB]

$\therefore$ Simplified function $Y = AB + BC$

(16) Minimize the following Boolean expression using k-map

$Y = AB\bar{C} + BCD + B\bar{D}$

Solution:

The given Boolean expression should be written as the sum of minterms.

$Y = AB\bar{C} + BCD + B\bar{D}$

$= AB\bar{C}(D+\bar{D}) + BCD(A+\bar{A}) + B\bar{D}(A+\bar{A})(C+\bar{C})$

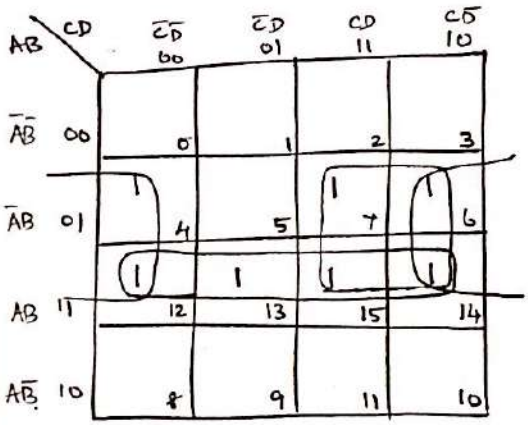
$= AB\bar{C}D + AB\bar{C}\bar{D} + ABCD + \bar{A}BCD + ABC\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} + \bar{A}B\bar{C}\bar{D}$

$= 1101 + 1100 + 1111 + 0111 + 1110 + 0110 + 0100$

$= m_{13} + m_{12} + m_{15} + m_7 + m_{14} + m_6 + m_4$

$\therefore Y = \Sigma(4,6,7,12,13,14,15)$ (*)

Now, these minterms are represented as 1's on the k-map.



[Group1 (12,13,14,15) = AB
Group2 (4,6,12,14) = $B\bar{D}$
Group3 (6,7,14,15) = BC]

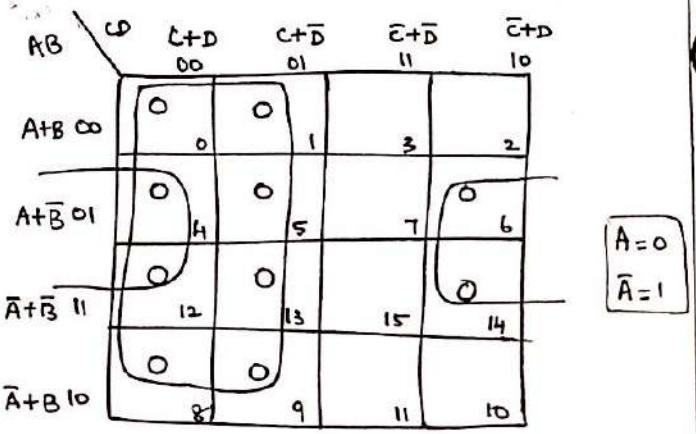
$\therefore$ Simplified function Y

$= AB + B\bar{D} + BC$

(17) Simplify the Boolean expression $Y = \Pi_M(0,1,4,5,6,8,9,12,13,14)$ using k-map method.

Solution:

Given the maxterms. The maxterms are represented as '0' in the k-map



Group1 (0,1,4,5,8,9,12,13) = C
Group2 (4,6,12,14) = $\bar{B}+D$

∴ The simplified function

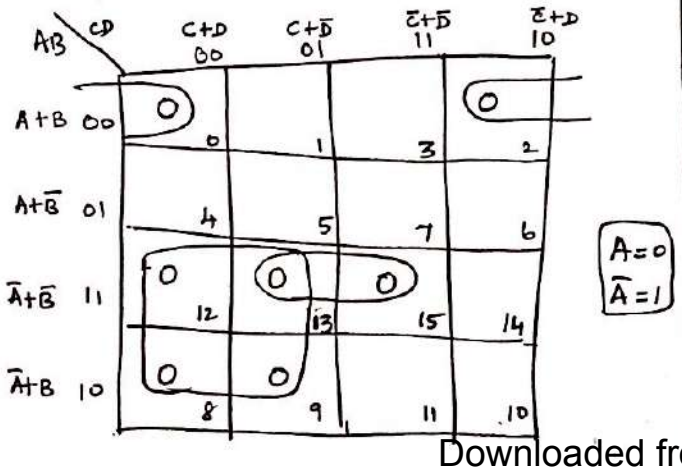
$Y = C \cdot (\bar{B}+D)$.

(18) Reduce the following expression using k-map technique.

$Y = \Pi_M(0,2,8,9,12,13,15)$

Solution:

Given the maxterms. The maxterms are represented as '0' on the k-map.



Group1 (8,9,12,13) = $\bar{A}+C$
Group2 (13,15) = $\bar{A}+\bar{B}+\bar{D}$
Group3 (0,2) = $A+B+D$

∴ The simplified function

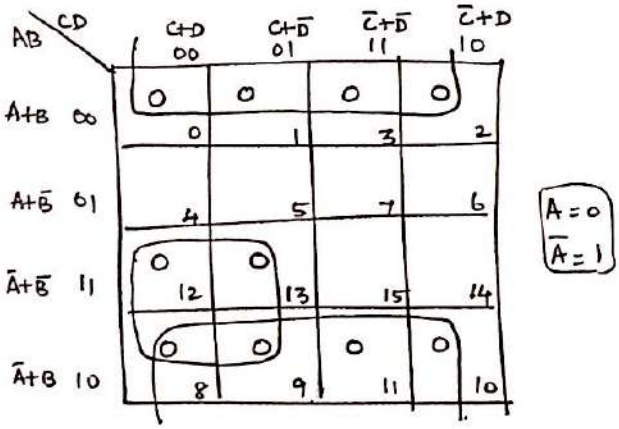
$Y = (\bar{A}+C)\cdot(\bar{A}+\bar{B}+\bar{D})\cdot(A+B+D)$

(19) Reduce the function using k-map technique.

$Y = \Pi_M(0,1,2,3,8,9,10,11,12,13)$

Solution:

Given the maxterms. The maxterms are represented as '0' on the k-map



Group1 (0,1,2,3,8,9,10,11) = B
Group2 (8,9,12,13) = $\bar{A}+C$

∴ The simplified function

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

* DONT CARE CONDITIONS *

INCOMPLETELY SPECIFIED FUNCTIONS:

* Functions that have unspecified outputs for some input combinations are called incompletely specified functions.

DONT CARE CONDITIONS:

* A don't-care term for a function is an input-sequence for which the function output does not matter. Its indicated d, Φ, X

* These dont care conditions can be used on a map to provide further simplification of the Boolean expression.

(X) ⇒ To obtain Minimal SOP, Value '1' can be assigned to the selected dont care combination.

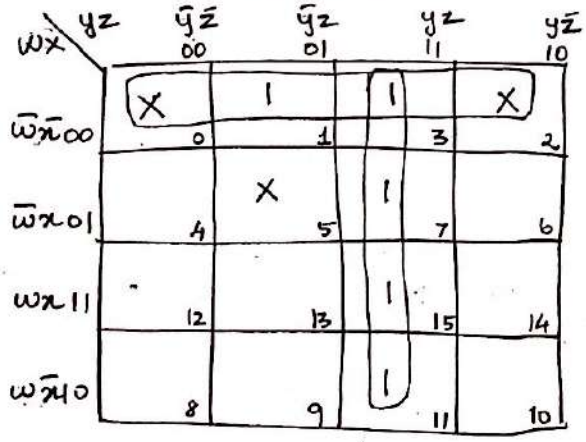
(X) ⇒ To obtain Minimal POS, Value '0' can be assigned to the selected dont care combination.

① Simplify the Boolean Function $F(w,x,y,z) = \Sigma(1,3,7,11,15)$. which has the dont care conditions $d(w,x,y,z) = \Sigma(0,2,5)$.

Solution:

Given: $\Sigma(1,3,7,11,15) + d(0,2,5)$.

The minterms are represented as 1 and dont care terms are represented as X. X is assigned a value '1' if necessary



Note: All 1's should be grouped but all 'X' need not be grouped. It is assigned a value only when further simplification is possible.

Group 1 (0, 1, 3, 2) = $\bar{w}\bar{x}$

Group 2 (3, 7, 11, 15) = yz

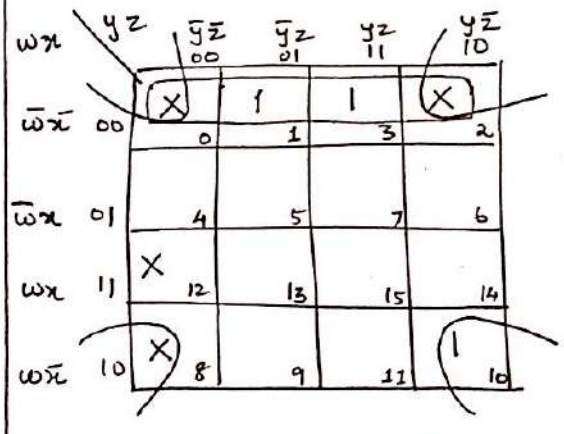
∴ The simplified function

$$F = \bar{w}\bar{x} + yz$$

(2) Simplify the Boolean Function $F(w,x,y,z) = \Sigma(1,3,10) + \Sigma_d(0,2,8,12)$ using k-map.

Solution:

* Given the minterms $\Sigma(1,3,10)$ and dont care terms $\Sigma(0,2,8,12)$.
* Minterms are marked as 1 on the k-map.
* The dont care terms are marked as 'x' on the k-map. and assigned a value 1 (if necessary), because the Function F is going to be simplified to obtain minimal SOP.



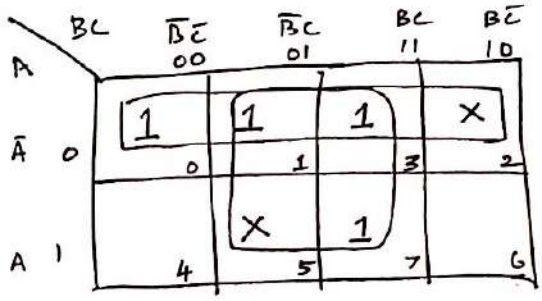
[Group 1 $(0,1,2,3) = \bar{w}\bar{x}$
Group 2 $(0,2,8,10) = \bar{x}\bar{z}$]

∴ Minimal SOP = $\bar{w}\bar{x} + \bar{x}\bar{z}$

(3) Simplify $F(A,B,C) = \Sigma_m(0,1,3,7) + \Sigma_d(2,5)$.

Solution:

* Given the minterms. These are marked as 1 on the k-map
* Given the dont care terms. These terms are marked as 'x'.



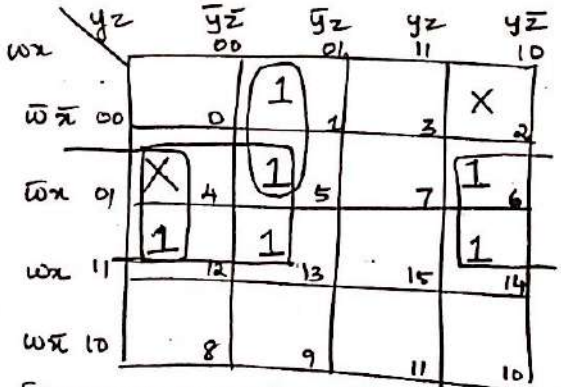
Group 1 $(0,1,2,3) = \bar{A}$
Group 2 $(1,3,5,7) = C$

∴ Minimal SOP $F = \bar{A} + C$.

(4) Find Minimal SOP for $F(w,x,y,z) = \Sigma_m(1,5,6,12,13,14) + \Sigma_d(2,4)$.

Solution:

* Given the minterms. These are marked as 1 on the k-map.
* Given the dont-care terms. These terms are marked as 'x'.
* Since, we are asked to find minimal SOP, X is assigned a value 1 if necessary, else it can be ignored.



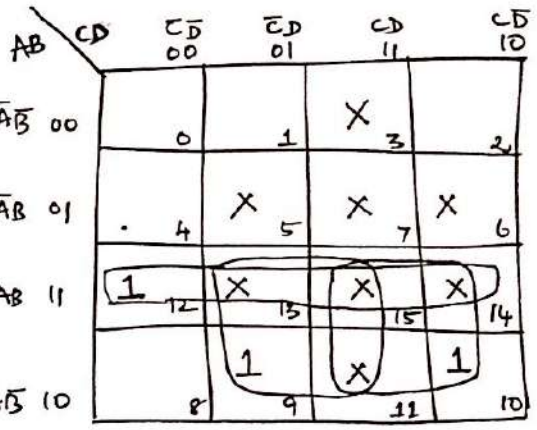
[Group 1 $(4,5,12,13) = x\bar{y}$, Group 2 $(1,5) = \bar{w}\bar{y}z$
Group 3 $(4,6,12,14) = x\bar{z}$]

∴ Minimal SOP = $x\bar{y} + \bar{w}\bar{y}z + x\bar{z}$

⑤ Find minimal SOP for $F = \Sigma_m(9,10,12) + \Sigma_d(3,5,6,7,11,13,14,15)$ using k-map.

Solution:

* Given the minterms. These minterms are represented as 1 on the k-map
* Given the dont care terms. These terms are represented as 'X' on the k-map.
* X is assigned a value 1 if necessary, else it can be ignored.



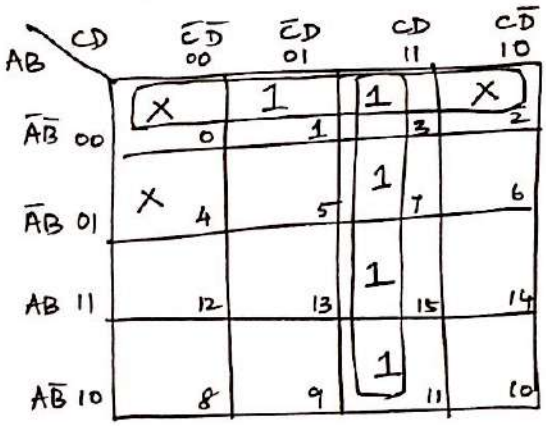
[Group1 (12, 13, 14, 15) = AB
Group2 (9, 11, 13, 15) = AD
Group3 (10, 11, 14, 15) = AC]

∴ The Minimal SOP F = AB + AD + AC

⑥ Find minimal SOP of $f(A,B,C,D) = \Sigma_m(1,3,7,11,15) + \Sigma_d(0,2,4)$ using k-map.

Solution:

* Given the minterms. These minterms are represented as 1 on the k-map
* Given the dont care terms. These terms are represented as 'X' on the k-map.
* X is assigned a value 1 if necessary, else it can be ignored.



[Group1 (3, 7, 11, 15) = CD
Group2 (0, 1, 2, 3) = ĀB̄]

∴ Minimal SOP = $\bar{A}\bar{B} + CD$.

⑦ Using k-map, simplify the Boolean expression & obtain a) Minimal SOP and b) Minimal POS.

$Y = \Sigma_m(0,2,3,6,7) + \Sigma_d(8,10,11,15)$

Solution:

Given

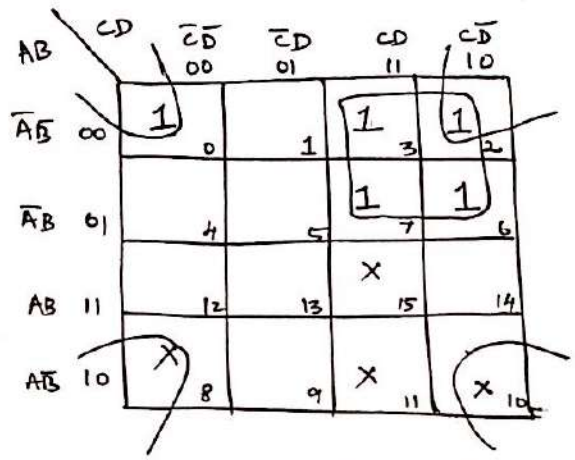
* The minterms and the dont care terms.

a) Finding Minimal SOP:

From the Question,

$\Sigma_m(0,2,3,6,7)$ & $\Sigma_d(8,10,11,15)$

*The minterms in the list are marked as 1 on k-map.
*The dont care terms(Σd) is marked as X. X is assigned a value 1 if needed.



For SOP
A = 1
Ā = 0

[Group1 (2,3,6,7) = $\bar{A}C$
Group2 (0,2,8,10) = $\bar{B}\bar{D}$]

∴ The Minimal SOP $Y = \bar{A}C + \bar{B}\bar{D}$

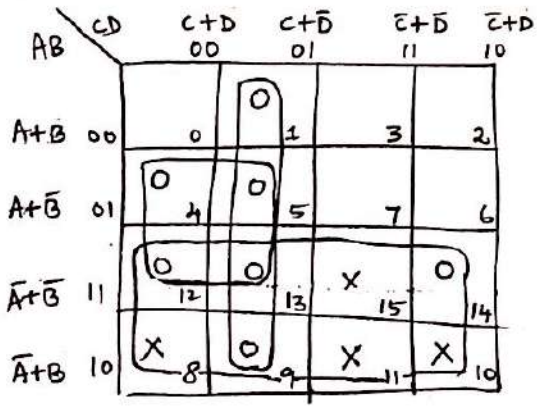
b) Finding Minimal Pos:

From the Question, minterms are $\Sigma_m(0,2,3,6,7)$ & $\Sigma_d(8,10,11,15)$.

* The maxterms are identified as $\Pi_M(1,4,5,9,12,13,14)$.

* The maxterms in the list are marked as 0 on k-map.

* The dont care terms (Σd) is marked as X. Here, X is assigned a 'value 0' if needed, as we have to find minimal Pos.



For Pos
A = 0
Ā = 1

[Group1 (1,5,9,13) = $C + \bar{D}$
Group2 (4,5,12,13) = $\bar{B} + C$
Group3 (8,9,10,11,12,13,14,15) = $\bar{A}$]

∴ The Minimal Pos

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

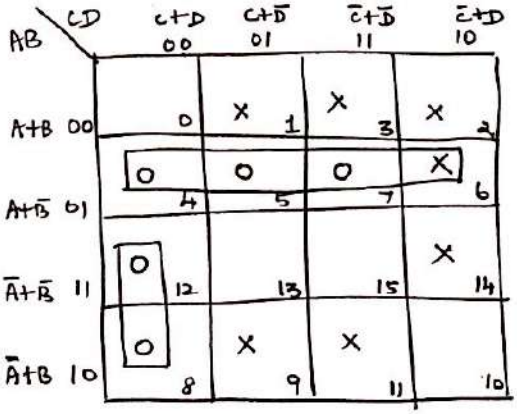
(8) Find Minimal POS for the Boolean expression $Y = \Pi_M(4,5,7,8,12) \cdot d(1,2,3,9,6,11,14)$ using k-map.

Solution

Given the maxterms and dont care terms.

* The maxterms in the given list are marked as 0 on the k-map

* The dont care terms (Σd) is marked as X. Here, X is assigned a 'value 0' if needed, as we have to find minimal Pos.



For Pos
A = 0
Ā = 1

Group1 (4,5,6,7) = $A + \bar{B}$
Group2 (8,12) = $\bar{A} + C + D$

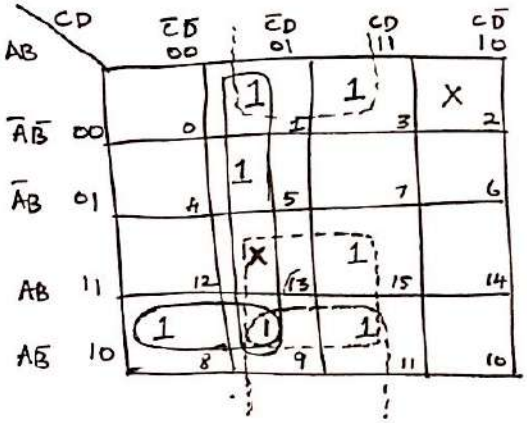
∴ The minimal Pos = $(A+\bar{B}).(\bar{A}+C+D)$

(9) Minimize the following Boolean functions using k map.

i) $Y = \Sigma_m(1,3,5,8,9,11,15) + d(2,13)$

Solution:

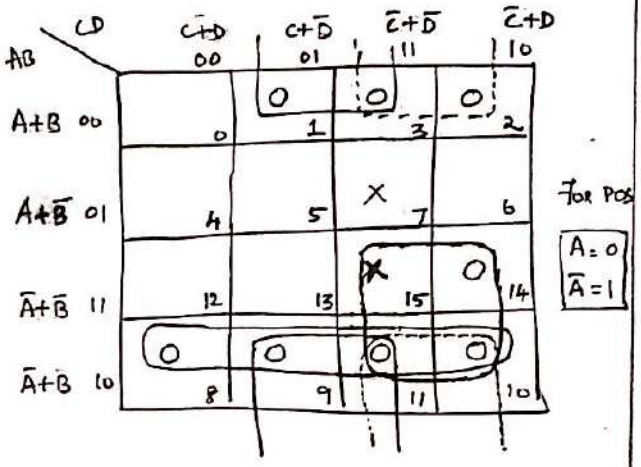
* Given the minterms and the dont care terms.
* The minterms are marked as 1 on the k-map.
* The dont care terms are marked as X. X is assigned a value 1 if needed because we are finding minimal SOP.



[Group 1 (1,5,9,13) = $\bar{C}D$
Group 2 (9,11,13,15) = AD
Group 3 (1,3,9,11) = $\bar{B}D$
Group 4 (8,9) = $A\bar{B}\bar{C}$]

∴ The Minimal SOP = $\bar{C}D + AD + \bar{B}D + A\bar{B}\bar{C}$

* The dont care terms are marked as X. X is assigned a value 0 if needed, because we are finding minimal POS.



[Group 1 (8,9,10,11) = $\bar{A}+B$
Group 2 (10,11,14,15) = $\bar{A}+\bar{C}$
Group 3 (1,3,9,11) = $B+\bar{D}$
Group 4 (2,3,10,11) = $B+\bar{C}$]

∴ The Minimal POS =

$(\bar{A}+B)\cdot(\bar{A}+\bar{C})\cdot(B+\bar{D})\cdot(B+\bar{C})$.

(10) Minimize the following Boolean function using k-map.

$Y = \overline{\Pi}_M(1,2,3,8,9,10,11,14), d(7,15)$.

Solution:

* Given the maxterms and the dont care terms.
* The maxterms are marked as 0 on the k-map.

* PRIME IMPLICANTS *

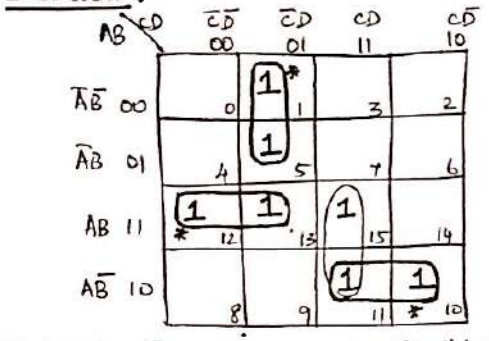
* A prime implicant is a product term that can be obtained from the map by combining all possible maximum number of adjacent squares.

* The prime implicant is called as essential prime implicant if it is the "only prime implicant" that covers the minterm.

(Ie) An **essential** prime implicant is the largest possible group of 1, which has atleast a single '1' which cannot be combined in any other way.

Example 1 : $F = \Sigma(1,5,10,11,12,13,15)$

Solution :



[$G1(1,5) = *\bar{A}\bar{C}D$, $G2(12,13) = *AB\bar{C}$
$G3(11,15) = ACD$, $G4(10,11) = *A\bar{B}C$]

Here, prime implicants are: $\bar{A}\bar{C}D$, $AB\bar{C}$, ACD, $A\bar{B}C$

Essential prime implicants are: $\bar{A}\bar{C}D$, $AB\bar{C}$ and $A\bar{B}C$

Explanation:

1) Consider $G1(1,5)$. In this group, minterm 5 can be combined with 13. But, minterm 1 cannot be combined in any other way. So, it is essential prime implicant

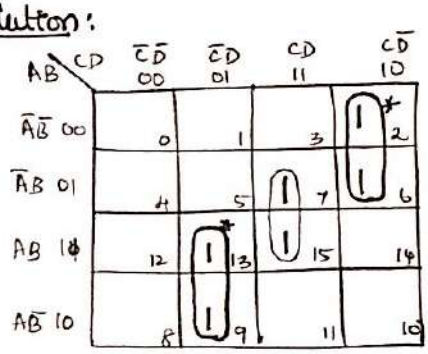
2) Consider $G2(12,13)$. In this group, minterm 13 can also be combined with 15. But, minterm 1 cannot be combined in any other way. So, it is essential prime implicant.

3) Consider $G3(11,15)$. In this group, minterm 11 can be combined with 10 & minterm 15 can also be combined with 13. Both minterms of the group can be combined in other ways. So, it is not essential

4) Consider $G4(10,11)$. In this group, minterm 11 can also be combined with 15. But, minterm 10 cannot be combined in any way. So, it is essential prime implicant

Example 2: $F = \Sigma(2,6,7,9,13,15)$

Solution:



[$G1(2,6) = \bar{A}C\bar{D}$, $G2(7,15) = BCD$
$G3(9,13) = A\bar{C}D$]

Prime implicants are: $\bar{A}C\bar{D}$, BCD, $A\bar{C}D$

Essential prime implicants are: $\bar{A}C\bar{D}$, $A\bar{C}D$

Explanation:

1) Consider $G1(2,6)$. In this group, minterm 6 can also be grouped with 7. But, minterm 2 cannot be combined in any other way. So, it is essential prime implicant.

2) Consider $G2(7,15)$. In this group, minterm 7 can also be grouped with 6 & minterm 15 can also be grouped with 13. Both minterms of the group can be combined in other ways. So, it is not an essential prime implicant.

3) Consider $G3(9,13)$. In this group, minterm 13 can also be grouped with 15. But, minterm 9 cannot be combined in any other way. So, it is an essential prime implicant.

FIVE - VARIABLE K-MAP

* A five- variable K-map contains (2^5) 32 squares, each square contains one minterm.

* A five- variable k-map consists of 2 four- variable maps with variables A, B, C, D, E.

* The left-hand 4-variable map represents the 16 squares containing minterms 0 to 15, in which A=0.

* The right-hand 4-variable map represents the 16 squares containing minterms 16 to 32, in which A=1.

* Each square in the A=0 map is adjacent to the corresponding square in the A=1 map.

A FIVE VARIABLE MAP

A = 0

BC \ DE	$\bar{D}\bar{E}$ 00	$\bar{D}E$ 01	DE 11	$D\bar{E}$ 10
$\bar{B}\bar{C}$ 00	m_0	m_1	m_3	m_2
$\bar{B}C$ 01	m_4	m_5	m_7	m_6
BC 11	m_{12}	m_{13}	m_{15}	m_{14}
$B\bar{C}$ 10	m_8	m_9	m_{11}	m_{10}

A = 1

BC \ DE	00	01	11	10
00	m_{16}	m_{17}	m_{19}	m_{18}
01	m_{20}	m_{21}	m_{23}	m_{22}
11	m_{28}	m_{29}	m_{31}	m_{30}
10	m_{24}	m_{25}	m_{27}	m_{26}

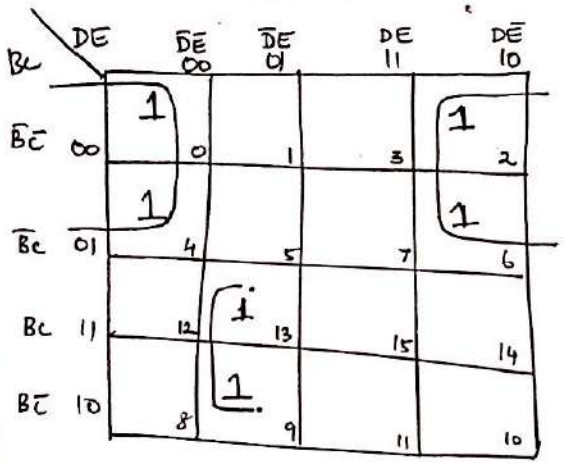
* One square represents a minterm, giving a term with 5 literals
* Two adjacent squares represent a term with 4 literals.
* Four adjacent squares represent a term with 3 literals.
* Eight adjacent squares represent a term with 2 literals
* Sixteen adjacent squares represent a term with 1 literal
* 32 adjacent squares represent a term which is always 1

① Simplify the Boolean function

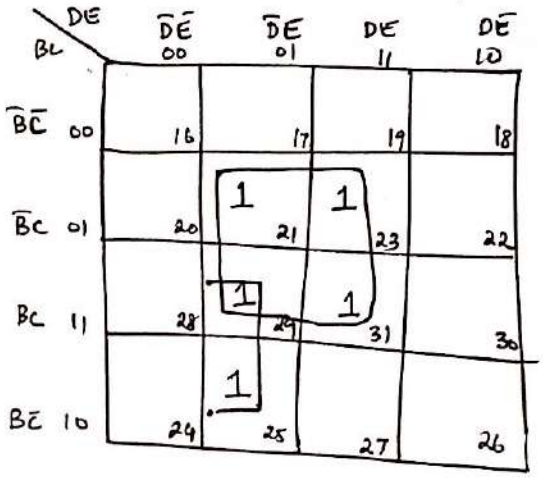
$F(A,B,C,D,E) = \Sigma(0,2,4,6,9,13,21,23,25,29,31)$

Solution.

A=0



A = 1



Group1 (21,23,29,31) = $AC\bar{E}$

Group2 (9,13,25,29) = $B\bar{D}\bar{E}$

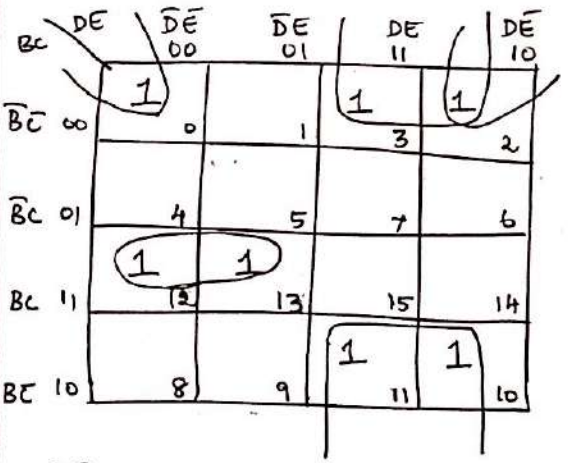
Group3 (0,2,4,6) = $\bar{A}\bar{B}\bar{E}$

∴ Simplified Function $F = AC\bar{E} + B\bar{D}\bar{E} + \bar{A}\bar{B}\bar{E}$

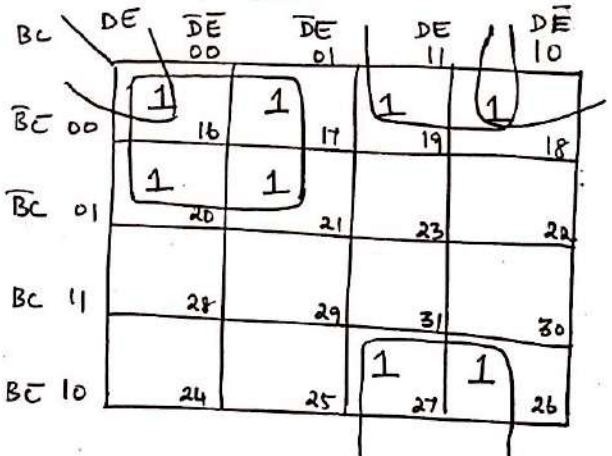
② Simplify $F = \Sigma(0,2,3,10,11,12,13,16,17,18,19,20,21,26,27)$

Solution.

A=0



A=1



Group1 (2,3,10,11,18,19,26,27) = $\bar{C}D$, Group2 (0,2,16,18) = $\bar{B}\bar{C}\bar{D}$

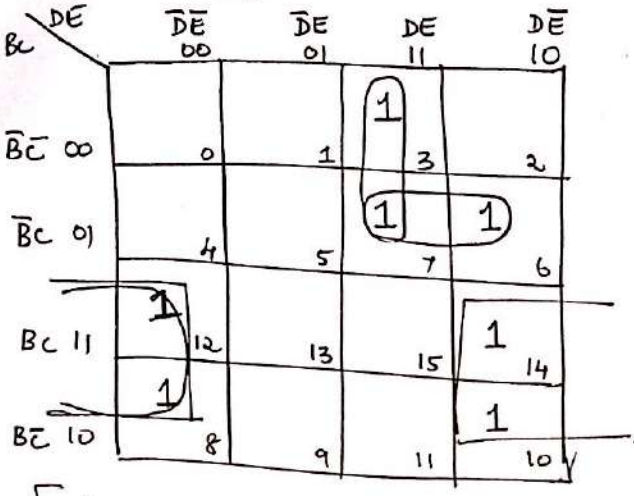
Group3 (16,17,20,21) = $A\bar{B}\bar{D}$, Group4 (12,13) = $\bar{A}BC\bar{D}$

∴ Simplified Function $F = \bar{C}D + A\bar{B}\bar{D} + \bar{A}BC\bar{D} + \bar{B}\bar{C}\bar{E}$.

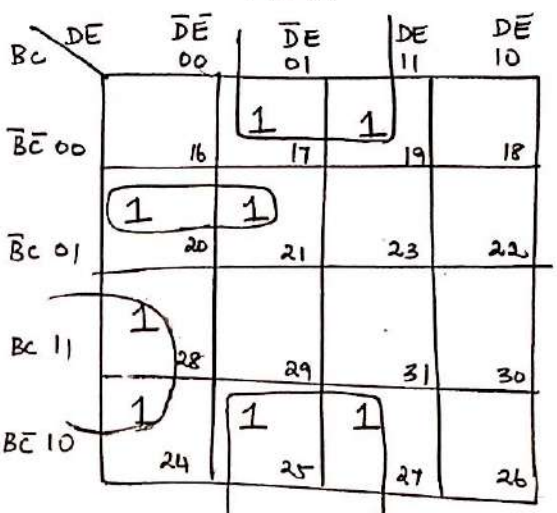
(3) Simplify $Y = \sum(3,6,7,8,10,12,14,17,19,20,21,24,25,27,28)$

Solution

A=0



A=1



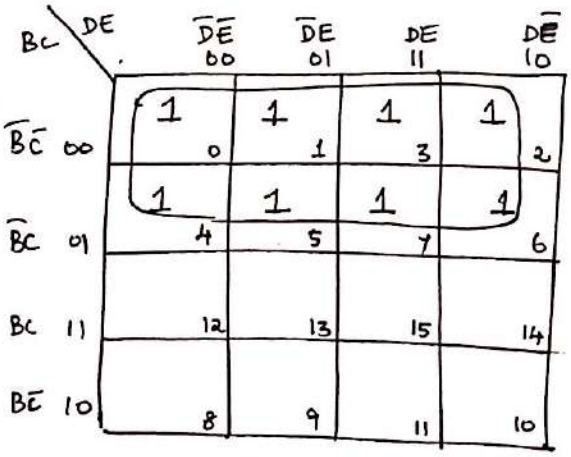
Group1 $(8,10,12,14) = \bar{A}B\bar{E}$, Group2 $(17,19,25,27) = A\bar{C}E$
Group3 $(8,12,24,28) = B\bar{D}\bar{E}$, Group4 $(20,21) = A\bar{B}\bar{D}$
Group5 $(3,7) = \bar{A}\bar{B}DE$, Group6 $(6,7) = \bar{A}\bar{B}CD$

∴ Simplified function $F = \bar{A}B\bar{E} + A\bar{C}E + B\bar{D}\bar{E} + A\bar{B}\bar{D} + \bar{A}\bar{B}DE + \bar{A}\bar{B}CD$

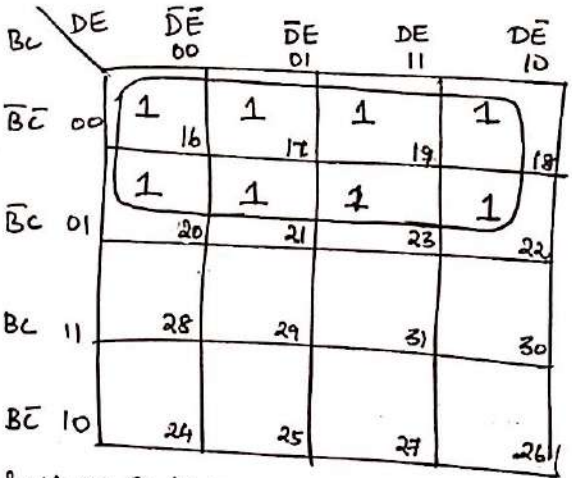
(4) Minimize $Y(A,B,C,D,E) = \sum_m(0,1,2,3,4,5,6,7,16,17,18,19,20,21,22,23)$.

Solution:

A=0



A=1

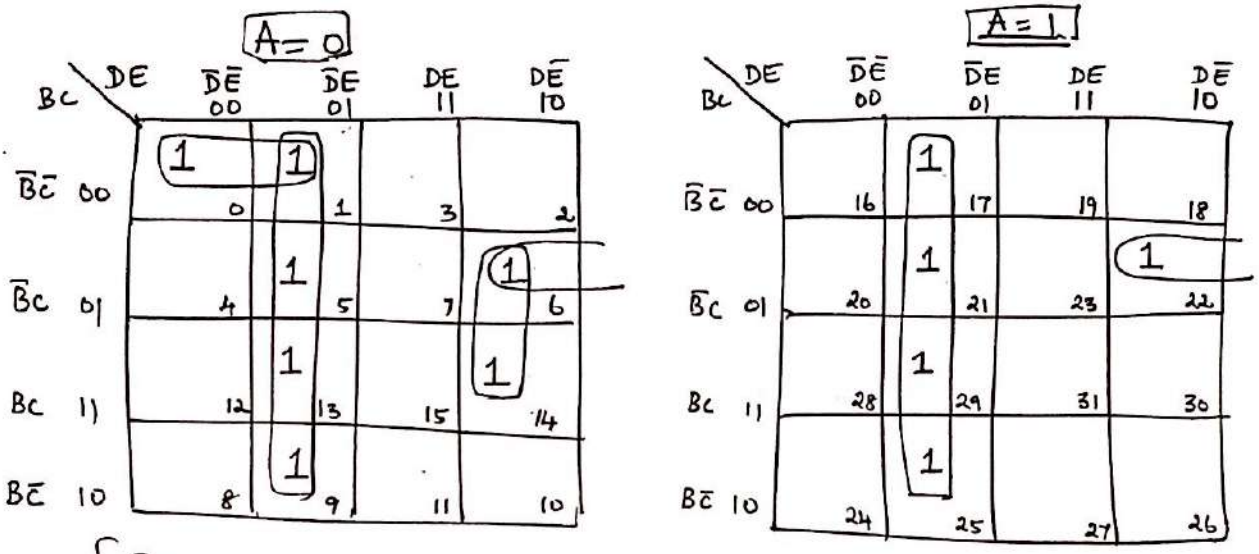


Group1 $(0,1,2,3,4,5,6,7$ & $16,17,18,19,20,21,22,23) = \bar{B}$

∴ Simplified Boolean function $Y = \bar{B}$.

⑤ Simplify $Y(A,B,C,D,E) = \sum m(0,1,5,6,9,13,14,17,21,22,25,29)$

Solution:

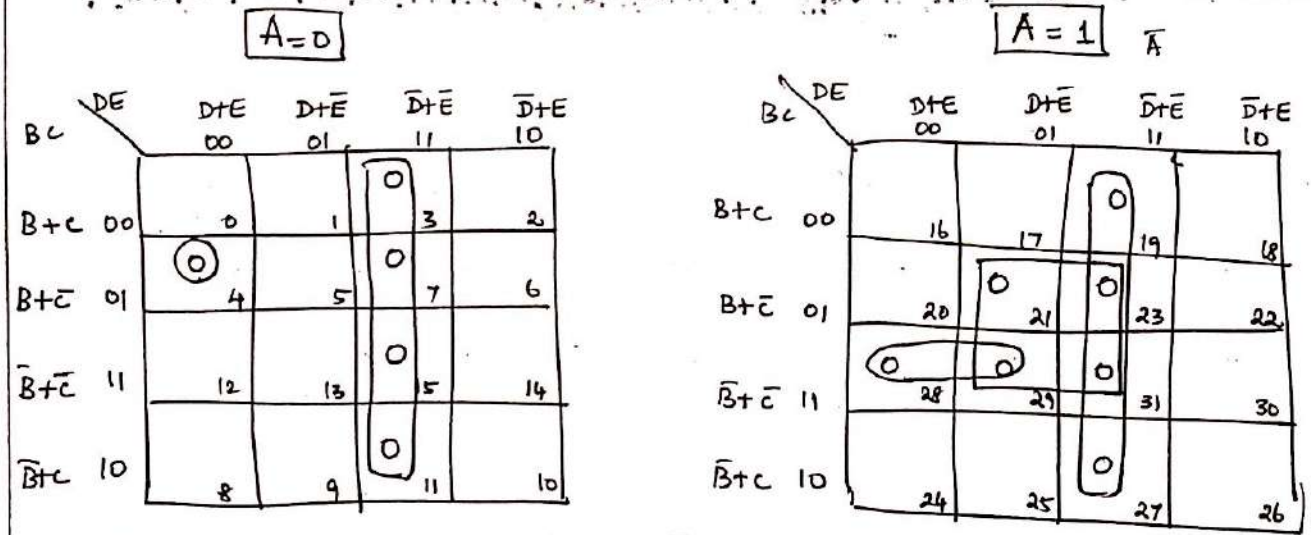


[Group1 (1,5,9,13,17,21,25,29) = $\bar{D}E$, Group2 (0,1) = $\bar{A}\bar{B}\bar{C}\bar{D}$
Group3 (6,14) = $\bar{A}CD\bar{E}$, Group4 (6,22) = $\bar{B}CD\bar{E}$]

∴ Simplified function $Y = \bar{D}E + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}CD\bar{E} + \bar{B}CD\bar{E}$

⑥ Simplify $F(A,B,C,D,E) = \prod_M(3,4,7,11,15,19,21,23,27,28,29,31)$

Solution:



Group1 (3,7,11,15,19,23,27,31) = $\bar{D}+\bar{E}$
Group2 (21,23,29,31) = $\bar{A}+\bar{C}+\bar{E}$
Group3 (28,29) = $\bar{A}+\bar{B}+\bar{C}+D$
Group4 (4) = $A+B+\bar{C}+D+E$.

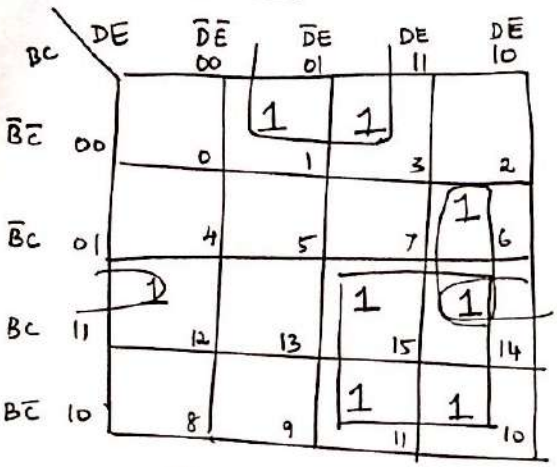
∴ Simplified POS = $(\bar{D}+\bar{E}).(\bar{A}+\bar{C}+\bar{E}).(\bar{A}+\bar{B}+\bar{C}+D).(A+B+\bar{C}+D+E)$.

⑦ Simplify the following switching function

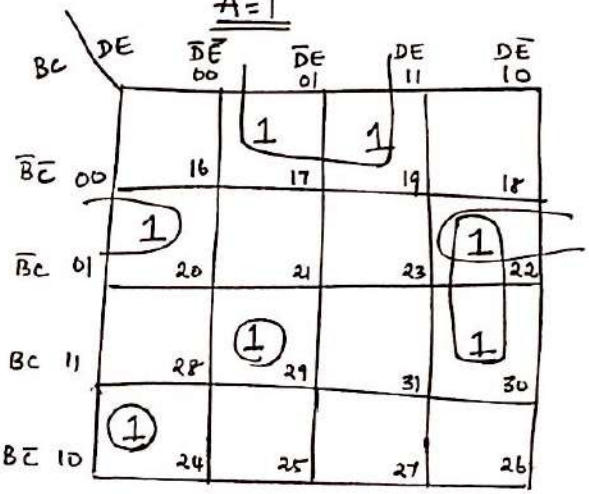
$f(A,B,C,D,E) = \Sigma_m(1,3,6,10,11,12,14,15,17,19,20,22,24,29,30)$

Solution:

A = 0



A = 1



Group1 (10,11,14,15) = $\bar{A}BD$, Group2 (1,3,17,19) = $\bar{B}\bar{C}E$
Group3 (6,14,22,30) = $CD\bar{E}$, Group4 (12,14) = $\bar{A}BC\bar{E}$
Group5 (20,22) = $A\bar{B}C\bar{E}$, Group6 (29) = $ABC\bar{D}E$
Group7 (24) = $AB\bar{C}\bar{D}\bar{E}$

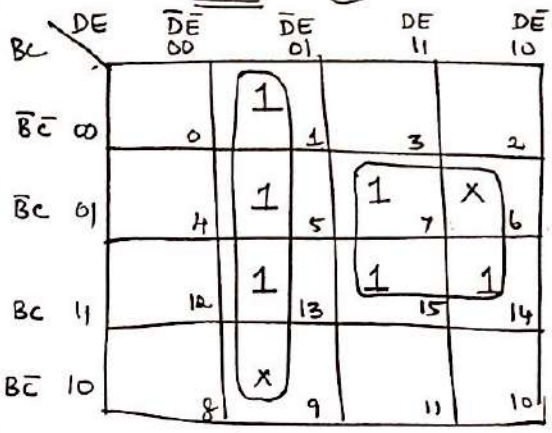
∴ Simplified SOP = $\bar{A}BD + \bar{B}\bar{C}E + CD\bar{E} + \bar{A}BC\bar{E} + A\bar{B}C\bar{E} + ABC\bar{D}E + AB\bar{C}\bar{D}\bar{E}$

⑧ Obtain the Minimal SOP for the function

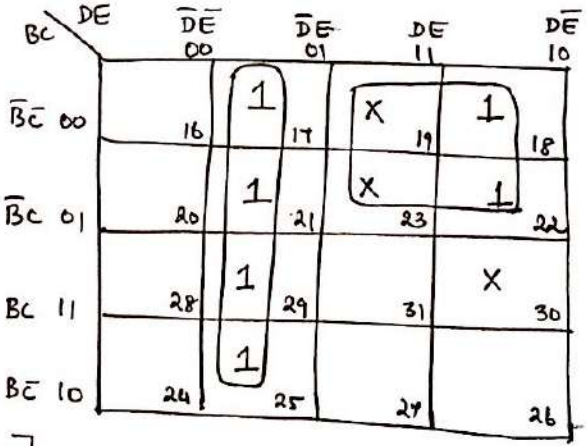
$Y = \Sigma m(1,5,7,13,14,15,17,18,21,22,25,29) + \Sigma d(6,9,19,23,30)$

Solution:

A = 0 (Ā)



A = 1 (A)



Group1 (1,5,9,13,17,21,25,29) = $\bar{D}E$
Group2 (6,7,14,15) = $\bar{A}CD$
Group3 (18,19,22,23) = $A\bar{B}D$.

∴ The Minimal SOP = $\bar{D}E + \bar{A}CD + A\bar{B}D$.

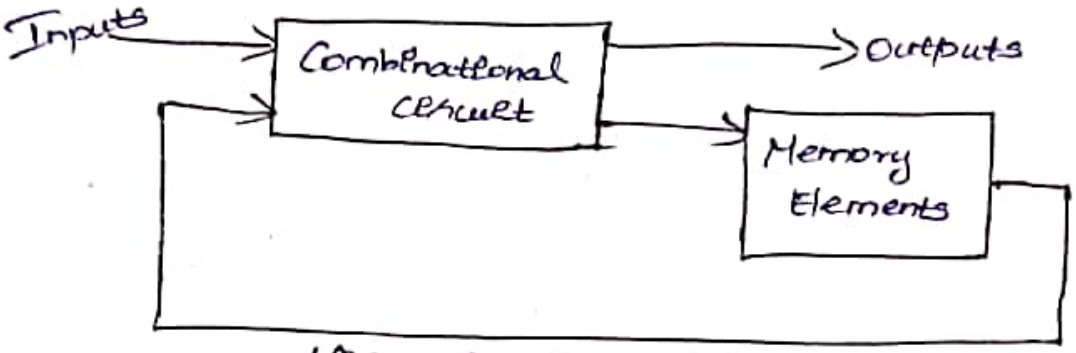
UNIT - II

SYNCHRONOUS SEQUENTIAL CIRCUITS.

Sequential Circuit :-

* A sequential circuit consists of a combinational circuit to which storage elements are connected to form a feedback path.

* The output of a sequential circuit is a function of both the i/ps and the present state of the storage elements.



(Fig): A sequential circuit.

→ The information stored in the memory elements at any time defines the present state of the sequential ckt.

→ The present state and external input determine the outputs and the next state and outputs of the sequential circuit.

2m

Sequential Circuit

Synchronous Sequential ckt	Asynchronous sequential ckt.
(1) Changes in i/p signals can affect memory elements only at discrete intervals of time (i.e) upon activation of clock signal	(1) Changes in i/p signals can affect memory elements at any instance of time.
(2) Memory elements are clocked flipflops	(2) Memory elements are either unclocked flipflops or time delay elements.
(3) The maximum operating speed of clock depends on the time delays involved	(3) Because of absence of clock, the circuit can operate faster than synchronous sequential circuits.
(4) Easier to design	(4) More difficult to design.

(X) 2m

Difference between Combinational ckt & Sequential ckt...

Combinational circuit	Sequential Circuit.
(1) The o/p depends only on the combination of i/p variables	(1) The o/p depends on the present i/p variables and also on the previous history of o/p variables.
(2) Memory element is not required	(2) Memory element is required to store the past history.
(3) Faster in speed	(3) Slower in speed.
(4) Easy to design	(4) Comparatively harder to design.

STORAGE ELEMENTS :

→ A storage element is a digital circuit that can maintain a binary state indefinitely until directed by an i/p signal to switch states.

→ The major difference among various types of storage elements are in the number of i/ps they possess and in the manner in which the inputs affect the binary state.

Latches :-

→ It is the basic storage element.

→ As the name indicates, it latches '0' or '1'.

→ The o/p of the latch changes immediately when its i/p changes.

Flip Flops :-

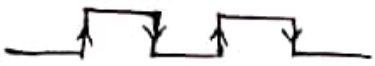
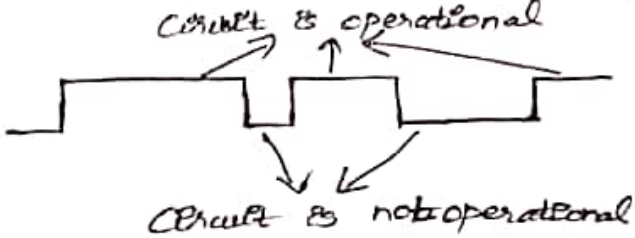
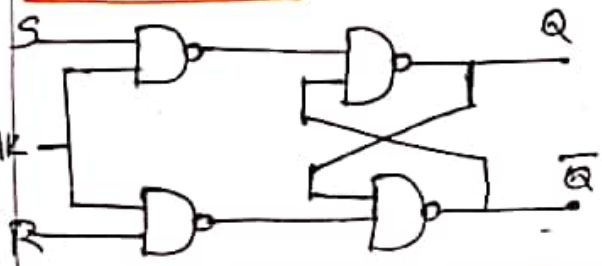
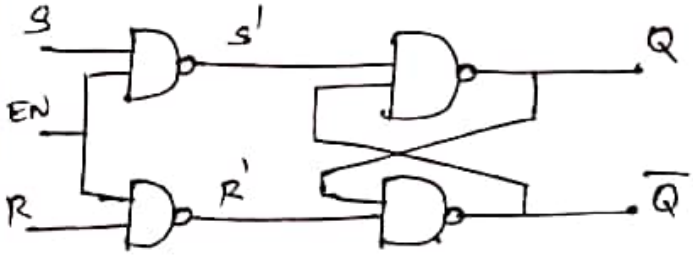
→ It stores 1 bit information.

→ It has 2 states (0 or 1)

→ The o/p of the flipflop changes only when its i/p changes when its clock is active.

→ The changes in i/p does not affect output when the clock is not activated.

(7.) Difference between Flip flops and Latches.

FLIP FLOPS	LATCHES.
(1.) Flipflops are storage elements controlled by clock transition.	(1.) Latches are storage elements that operate with signal levels (i.e) controlled by Enable I/p.
(2.) Edge-sensitive devices	(2.) Level-sensitive devices.
(3.) Consider the clock signal,  When the clock signal goes from low to high or from high to low, the circuit is operational. This is called as edge triggering.	(3.) Consider the enable signal,  This is called level triggering.
(4.) SR Flipflop. 	(4.) SR Latch with control I/p:. 

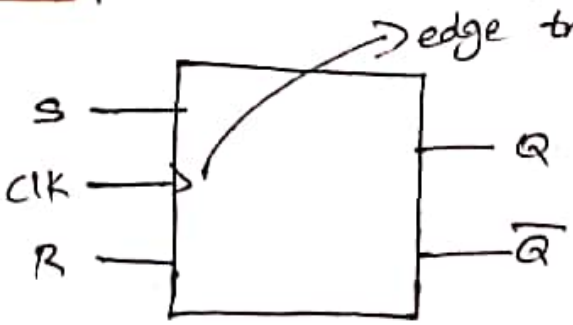
FLIP FLOPS

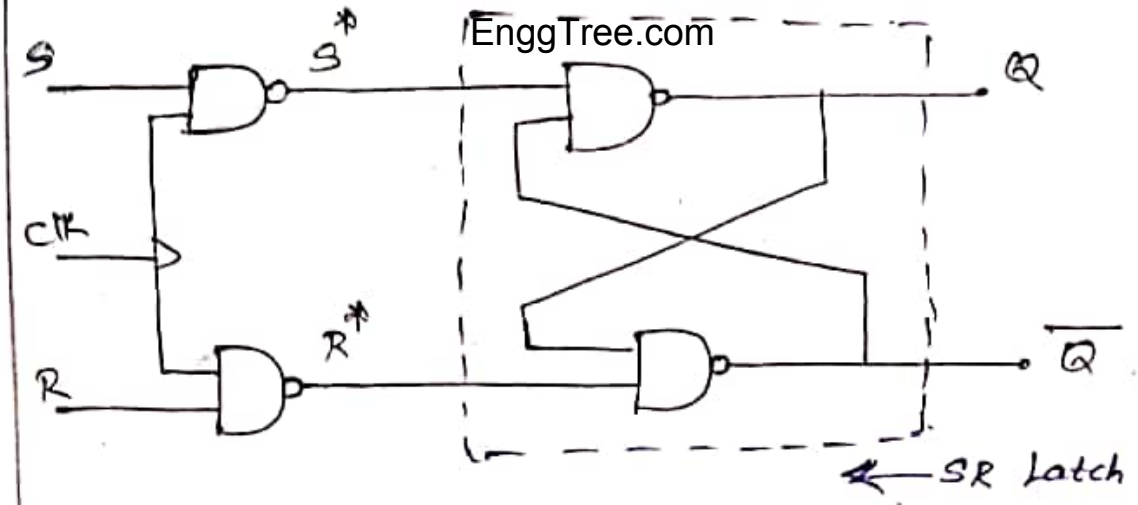
→ The state of a latch or a flipflop is switched by a change in the control I/p. This momentary change is called a trigger.

→ The signal transition it causes (0 to 1, 1 to 0) is said to trigger the flipflop.

SR FLIPFLOP [SR - Set Reset] :.

Symbol :-





$$S^* = \overline{S \cdot clk} = \overline{S} + \overline{clk}$$

$$R^* = \overline{R \cdot clk} = \overline{R} + \overline{clk}.$$

① Consider the truth table for SR latch:

S^*	R^*	Q	$\overline{Q}$
0	0	Invalid	
0	1	1	0
1	0	0	1
1	1	Memory.	

$S^* = 1$, $R^* = 1$

$S^* = 1$, $R^* = 0$

$S^* = 0$, $R^* = 1$

$S^* = 0$, $R^* = 0$

Truth Table for SR flipflop :-

CLK	S	R	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	Invalid.

Q_n → Present state

Q_{n+1} → Next state.

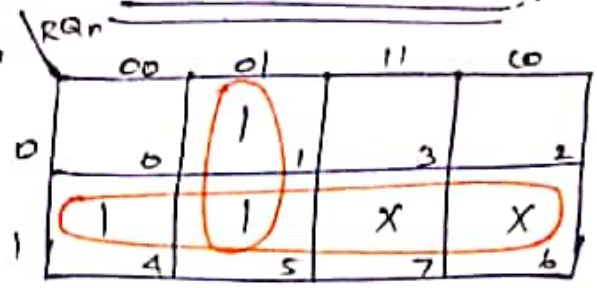
② Characteristic Table :-

→ The characteristic table is used to find out the value of Q_{n+1} (next state) which is dependent on the input and the 'previous state.

→ It is derived from the truth table of SR flipflop.

	S	R	Q_n	Q_{n+1}
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	x
7	1	1	1	x

To find Q_{n+1} :



Next state $Q_{n+1} = S + \bar{R} Q_n$

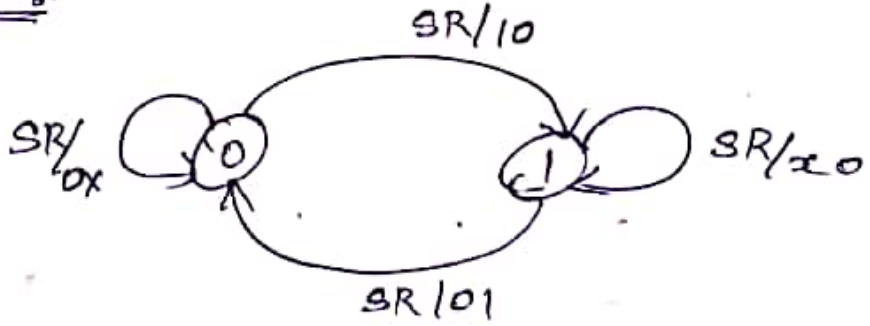
(3) Excitation Table :-

→ Excitation table is derived from characteristic table, Two inputs = Q_n, Q_{n+1}

Two outputs = S, R.

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

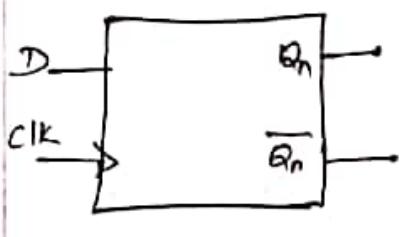
(4) State Diagram :-



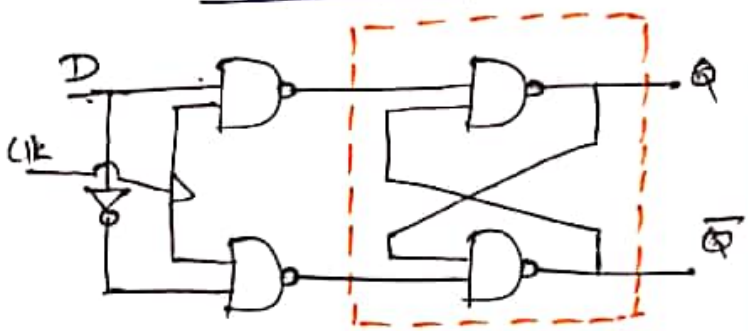
D- Flipflop :- [D - Data / Delay].

→ The D- flipflop is constructed from SR flipflop.
→ Used when we have to store only the data.
→ From the truth table of SR flipflop, it is observed that, to produce a output (0 or 1), S and R values are complement of each other.
→ So, instead of using two individual inputs, (S, R), a single input 'D' is used.

Symbol :-



D- Flipflop :-



① Truth Table for D- Flipflop :-

CLK	D	Q_{n+1}
0	X	Q_n (memory)
1	0	0
1	1	1

② Characteristic Table :-

→ The Char. Table is used to find out the value of Q_{n+1} which is dependent on input and previous state.

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Next State,
$Q_{n+1} = D$

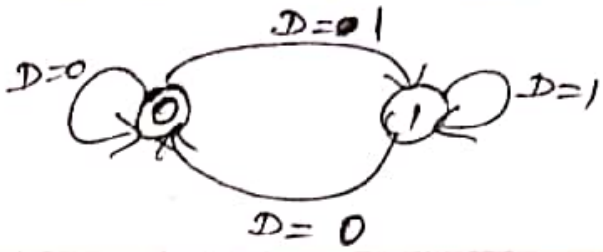
(3) **Excitation Table :-**

→ The excitation table is derived from characteristic table. Two inputs => Q_n, Q_{n+1}

One o/p => D

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

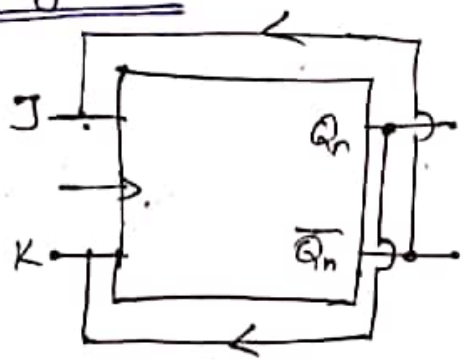
(4.) **State Diagram :-**



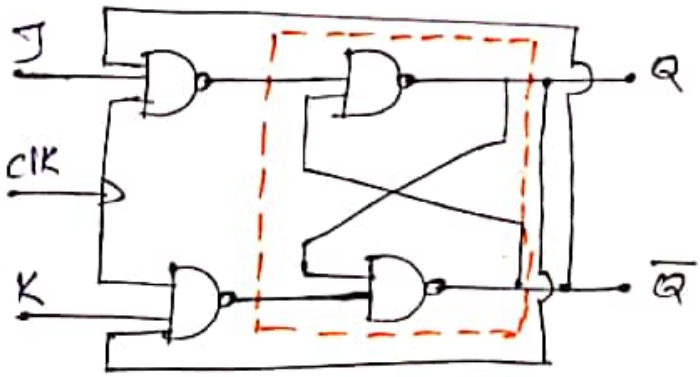
JK FLIPFLOP :-

→ The JK flipflop is used to overcome the invalid state of SR flipflop (i.e) S=1, R=1) by some useful action (toggling).

Symbol :-



JK-FF :-



(1.) **Truth Table for JK-FF :-**

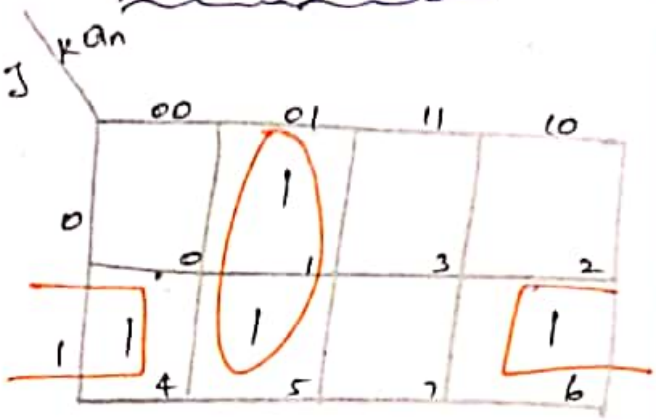
CLK	J	K	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	$\overline{Q_n}$

(2) Characteristic Table :-

→ The char. Table is used to find out Q_{n+1} which is dependent on input and previous state.

	J	K	Q_n	Q_{n+1}	
0	0	0	0	0	
1	0	0	1	1	
2	0	1	0	0	
3	0	1	1	0	
4	1	0	0	1	
5	1	0	1	1	
6	1	1	0	1	toggle action
7	1	1	1	0	

To find next state (Q_{n+1}):-



Next State,

$$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$$

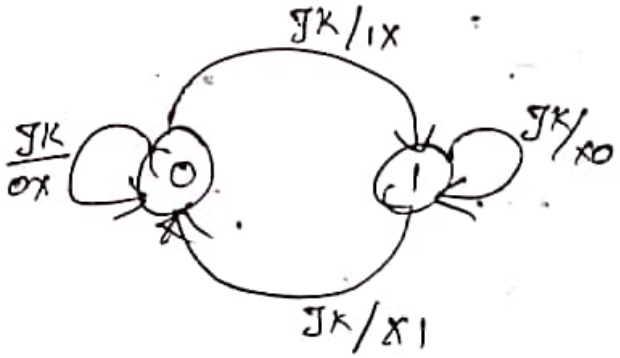
(3) Excitation Table :-

Two i/ps : Q_n, Q_{n+1}

Two o/ps : J, K

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(4) State diagram:-

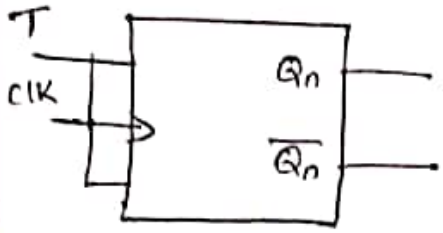


T- FLIPFLOP :- [T- Toggle] :-

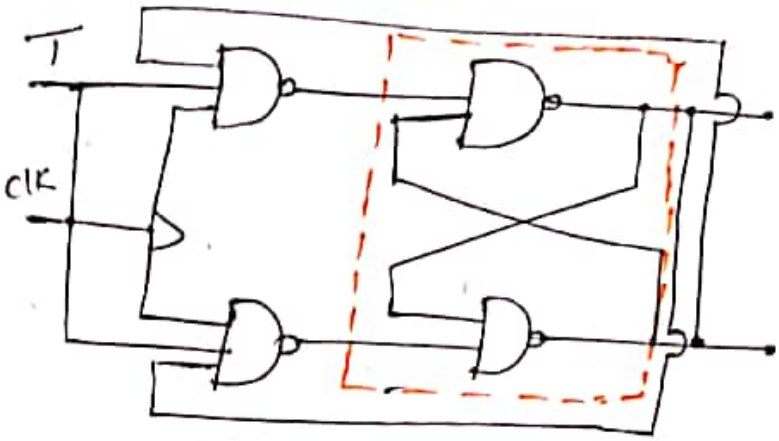
→ T- flipflop is used when we want only toggling action.

→ Jk flipflop is used to obtain T- flipflop.

Symbol :-



T - flipflop :-



① Truth Table for T-FF:-

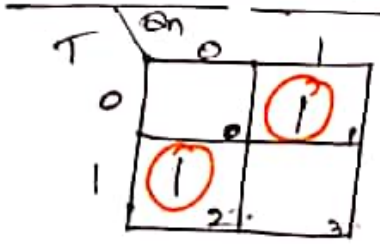
CLK	T	Q_{n+1}	
0	X	Q_n	Memory
1	0	Q_n	
1	1	$\overline{Q_n}$	toggle state.

② Characteristic Table :-

→ Used to find out next state (Q_{n+1}) which is dependent on input and previous state.

T	Q_n	Q_{n+1}	
0	0	0	(Q_n)
0	1	1	(Q_n)
1	0	1	(Q_n)
1	1	0	($\overline{Q_n}$).

To find next state (Q_{n+1}):-



$= \overline{T}Q_n + T\overline{Q_n} = T \oplus Q_n$.

Next state $\boxed{Q_{n+1} = T \oplus Q_n}$.

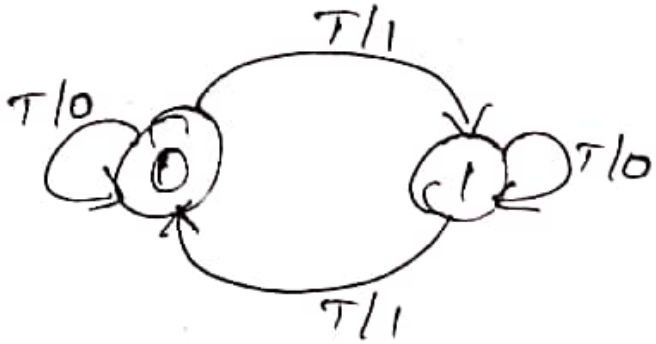
③ Excitation Table :-

Two i/ps : Q_n, Q_{n+1}

One o/p = T

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

④ State Diagram :-



Master Slave JK flip flop :-

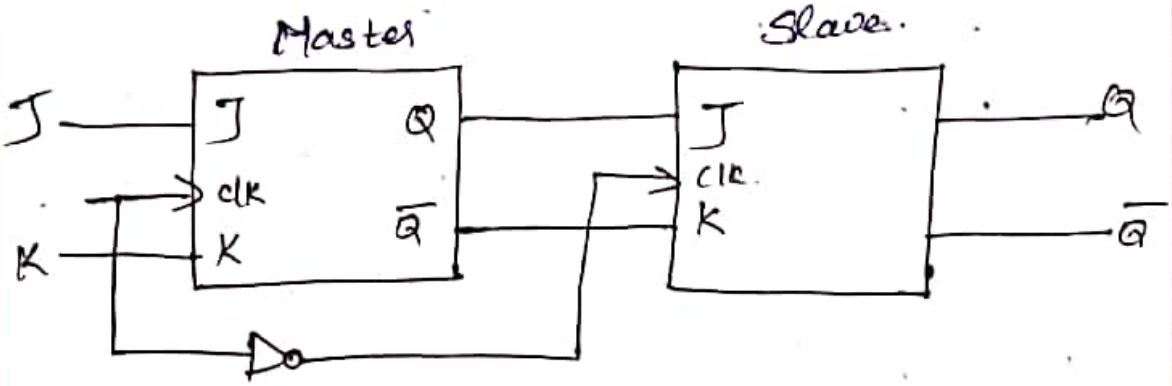
Race Around Condition :-

→ Before the development of edge triggered flipflop, the timing problem in level triggered FF was often handled by master slave FF.

→ In JK FF, consider the inputs $J=1$, $K=1$ and o/p $Q=0$; when clock pulse is applied, the o/p will change from 0 to 1 after time interval Δt.

→ For $J=1$ and $K=1$ and $Q=1$, the o/p $\bar{Q}=0$, after time Δt. Output will oscillate back and forth between 1 and 0 for every Δt time period.

→ At the end of the clock pulse, value of Q is ambiguous. This is called race around condition. This can be avoided by master-slave JK FF.



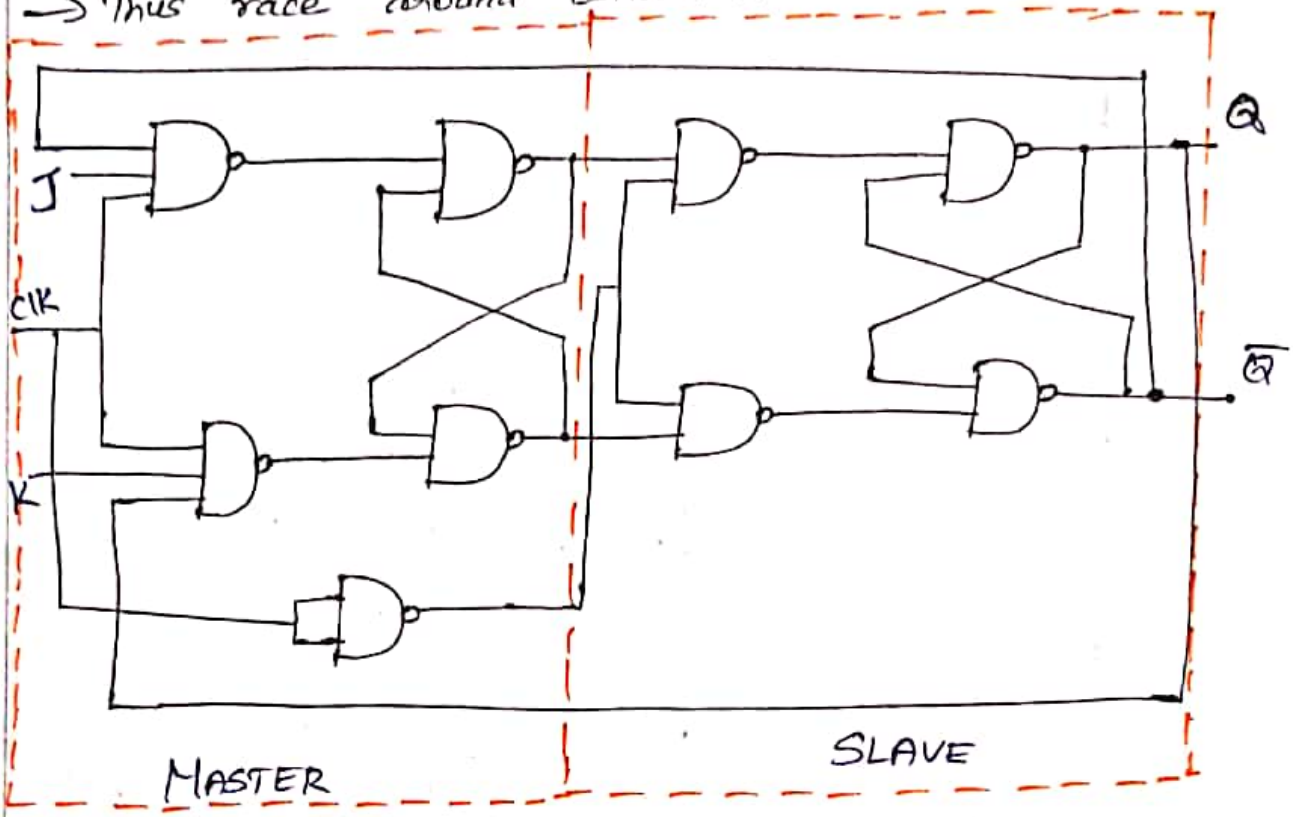
→ The clock signal is directly connected to the master flipflop and is connected through inverter to the slave flipflop.

→ When clk i/p has +ve edge, master FF acts according to the JK i/p but slave will not respond.

→ When clk i/p has -ve edge, slave FF will copy the master o/p.

→ Master doesnot respond to the feedback from Q and $\overline{Q}$ since it requires a +ve edge.

→ Thus race around condition is avoided here.



* If J=1, K=0, (+ve) clk, the o/p of master Q=1, which drive the input J of slave FF when (-ve) clk edge slave also set. Slave copies the action of master.
* When J=0, K=1 master resets the $\overline{Q}$=1 of master which makes K i/p of slave set. Slave FF resets on the arrival of trailing edge of clk. Thus copying action of master FF.
* If J=K=1; master toggles on +ve edge of clock and slave FF toggles on -ve edge of clk.
* If J=K=0, it doesnot produce any change.

Triggering of Flipflops :-

The state of FF is switched by a momentary change in the input signal which is called as trigger. Clocked FFs are triggered by pulses.

Based on the specific interval or point in the clock during or at which triggering of FF takes place, it is classified into,
(i) Level triggering (ii) Edge triggering.

Level triggering :-

FFs are enabled high on Loco. The flipflop action is dependent on the entire period of the pulse.

Positive level triggering :- Flipflop changes its state when clock is positive.

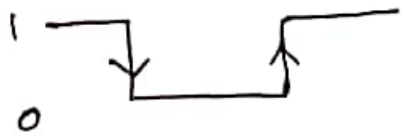
Negative level triggering :- Flipflop changes its state when clock is negative.

Edge triggering :-

A clock pulse may be either positive or negative. A positive clk source remains at '0' during the interval between pulses and goes to 1 during the occurence of a pulse. The pulse goes through two signal transitions: from 0 to 1 and 1 to 0.

Positive transition is defined as positive edge and negative transition is defined as negative edge.

positive pulse



Negative pulse.

Simplified characteristic Table for all FF:

SR or JK	For SR FF Q(t+1)	For JK FF Q(t+1)
0 0	Q(t)	Q(t)
0 1	0	0
1 0	1	1
1 1	X	$\overline{Q(t)}$

T or D	For T FF Q(t+1)	For D FF Q(t+1)
0	Q(t)	0
1	$\overline{Q(t)}$	1

Excitation Table :-

Q(t)	Q(t+1)	S	R	J	K	T	D
0	0	0	X	0	X	0	0
0	1	1	0	1	X	1	1
1	0	0	1	X	1	1	0
1	1	X	0	X	0	0	1

Characteristic (or) Next state eqn for all FF:

For SR F.F → $Q(t+1) = S + \overline{R}Q$

For JK FF → $Q(t+1) = J\overline{Q} + \overline{K}Q$

For T FF → $Q(t+1) = T \oplus Q$

For D FF → $Q(t+1) = D$.

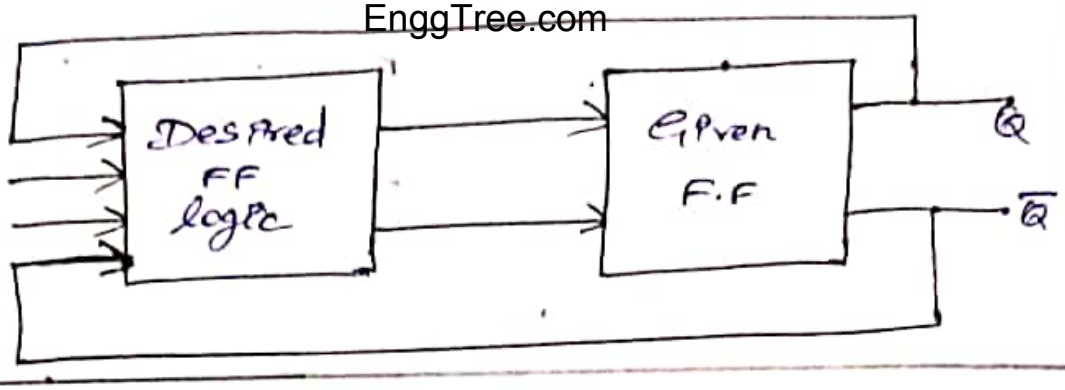
FLIP FLOP CONVERSIONS :-

PROCEDURE :-

1.) Write the characteristic Table for the desired FF.

2.) From the characteristic table, using Q(t) & Q(t+1), write the excitation table for the given FF.

3.) Write the simplified eqn using K-map as a function of present state & desired ff i/ps and Draw the equivalent diagram.



1.) Conversion of SR FF (Given FF) into JK FF (Desired FF) :

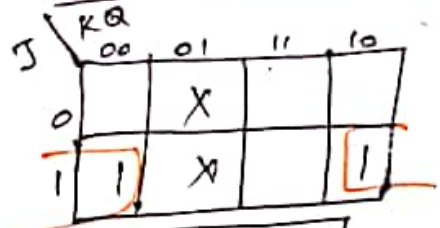
Soln :-

Char. table of Desired FF:

J	K	Q(t)	Q(t+1)	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

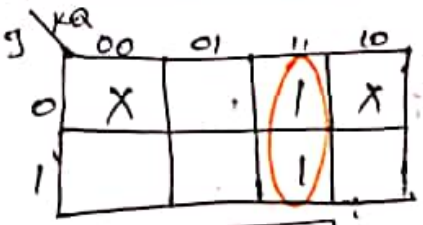
Excitation table of given FF

K-map for S :

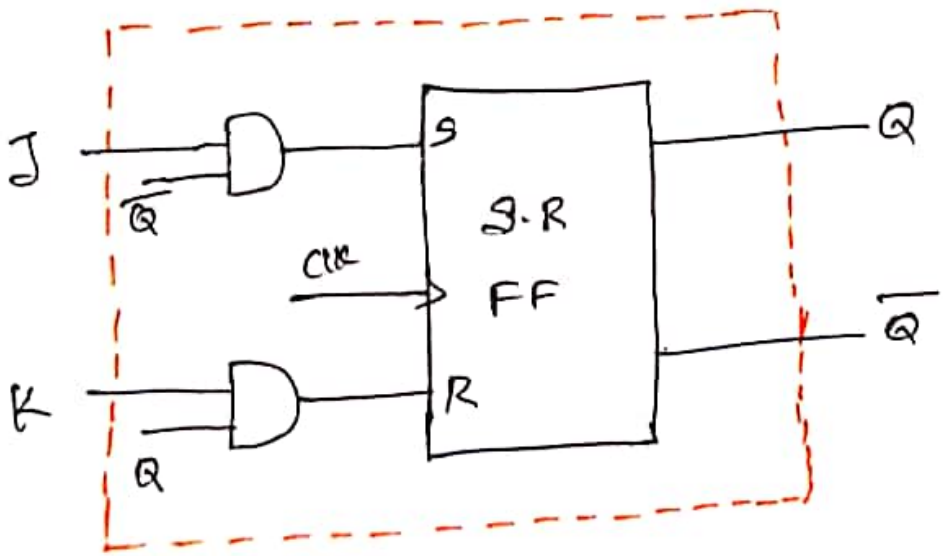


$S = J\overline{Q}$

K-map for R :



$R = KQ$



2.) Derive D FF from SR FF :.

D FF → Desired

SR FF → Given.

Soln:-

D	Q(t)	Q(t+1)	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

Char. table of desired FF

Excitation table of Given FF

K-map for S::

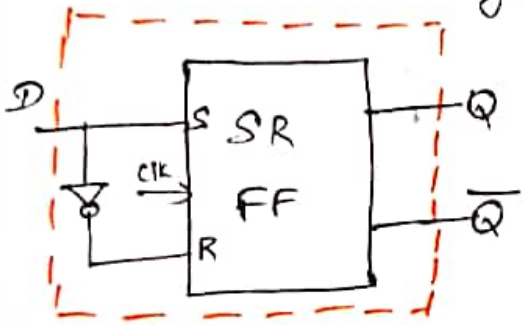
D \ Q	0	1
0		
1	1	X

$S = D$

K-map for R::

D \ Q	0	1
0	X	1
1		

$R = \bar{D}$



3.) Derive T from Delay FF:.

Soln:- ↓ Desired ↓ Given.

T	Q(t)	Q(t+1)	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Char. table of Desired FF

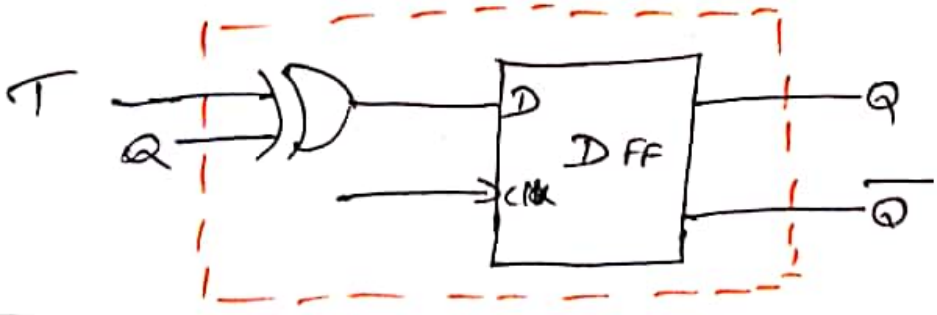
Excitation table of Given FF

K-map for D

T \ Q	0	1
0	0	1
1	1	0

$D = \bar{T}Q + T\bar{Q}$

$D = T \oplus Q$



4) Derive Delay from T f.f :
(Desired, Given)

Soln:

D	Q(t)	Q(t+1)	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

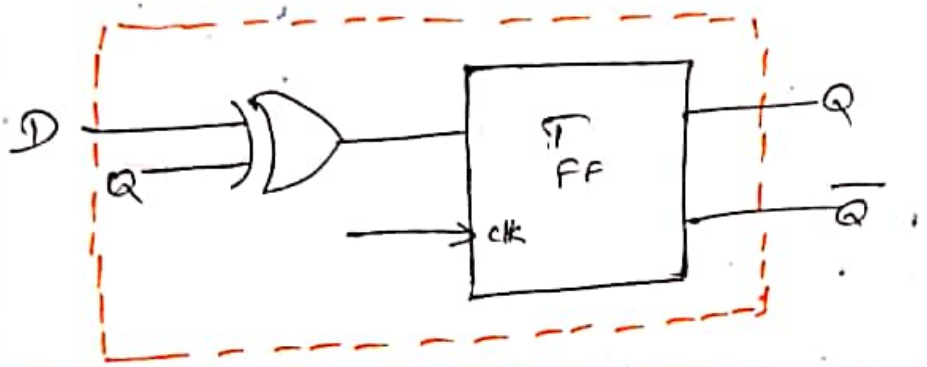
char table of desired FF

Excitation table of gn FF

K-map for T:

D \ Q	0	1
0		1
1	1	

$T = D \oplus Q$



ANALYSIS OF CLOCKED SEQUENTIAL CIRCUITS:

→ The analysis consists of obtaining state table or state diagram for the given time sequence of inputs, outputs & internal states.

→ A state table or state diagram are then presented to describe the behaviour of the sequential circuit.

State Equation :-

The state equation specifies the next state as a function of the present state and inputs.

Finite State Machines :-

→ All the state variables in sequential circuit are binary in nature. If the number of state variables is 'n', then the sequential

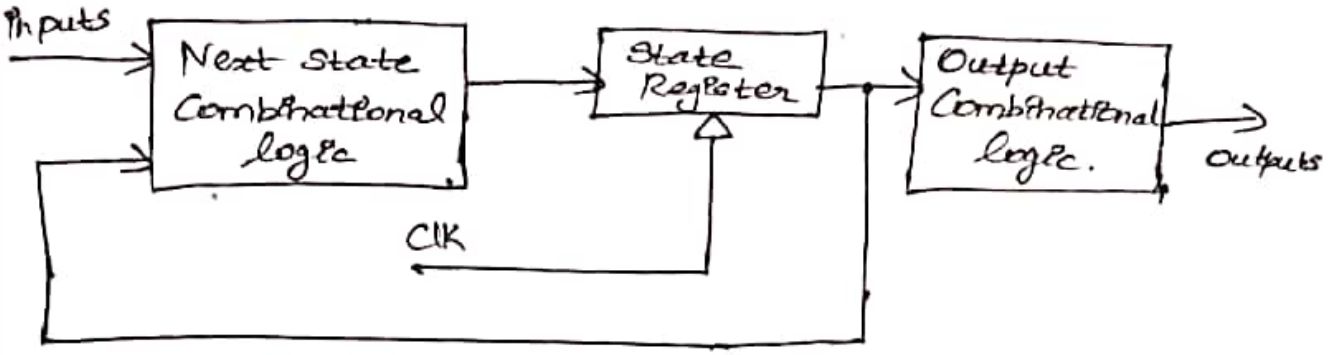
circuit has 2^n possible states which are finite.

→ Therefore, Sequential circuits are referred as Finite State Machines.

Two Models of clocked Sequential circuit :-

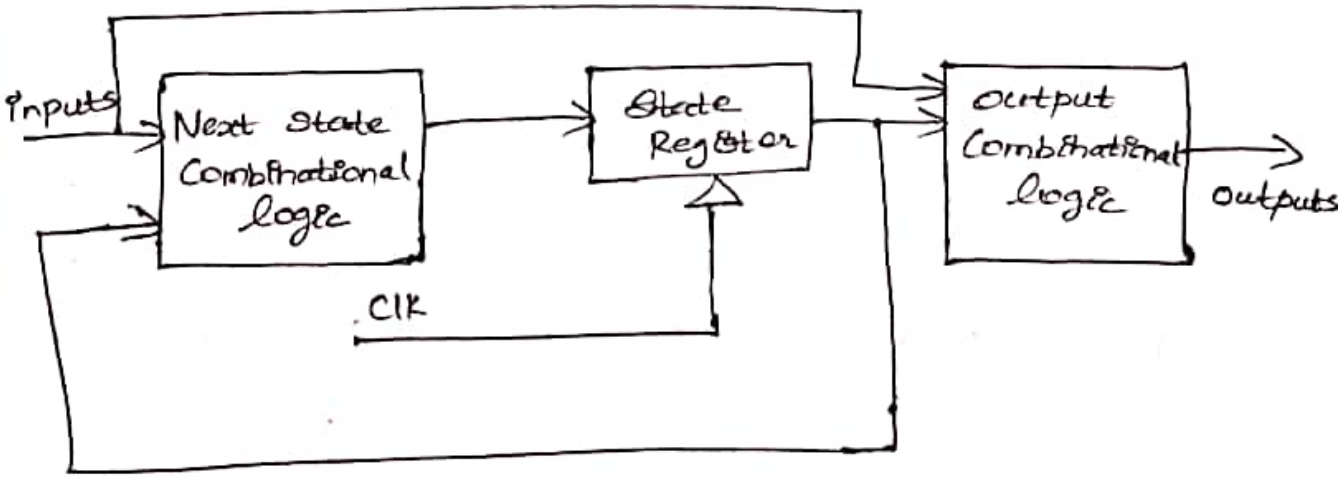
(1.) Moore Model :-

The output is the function of only the present state.



(2.) Mealy Model :-

The output is the function of both the present state and input.



PROBLEMS :- [PROCEDURE].

1.) From the given logic diagram, find i) FF i/ps ii) External i/ps & present state iii) o/p iv) Next state eqn.

2.) Draw the state table,

Ex. i/p & P.S	FF i/ps	NS	O/p

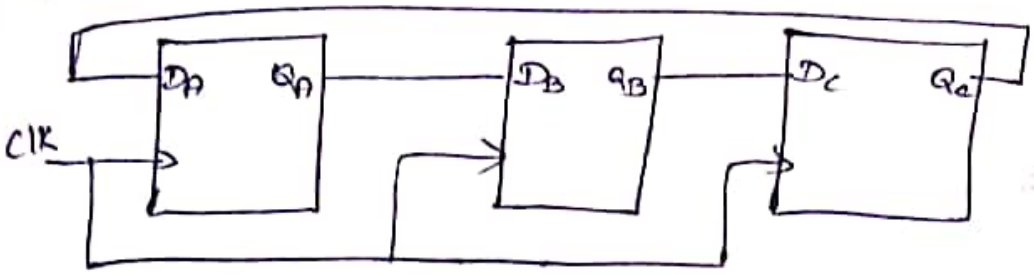
⇒ Second form of S.T

PS	NS	O/P

3.) Draw the state diagram.

PROBLEMS :-

1-) Draw the state table, state diagram for the seq. circuit shown below,



Soln :-

Step 1 : Present state → Q_A, Q_B, Q_C

F.F i/ps → $D_A = Q_C$
$D_B = Q_A$
$D_C = Q_B$

no ext. i/p &
no o/p here

Step 2 :

State Table :-

PS			FF i/ps			NS		
Q_A	Q_B	Q_C	D_A	D_B	D_C	$Q_A(t+1)$	$Q_B(t+1)$	$Q_C(t+1)$
0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	0	0
0	1	0	0	0	1	0	0	1
0	1	1	1	0	1	1	0	1
1	0	0	0	1	0	0	1	0
1	0	1	1	1	0	1	1	0
1	1	0	0	1	1	0	1	1
1	1	1	1	1	1	1	1	1

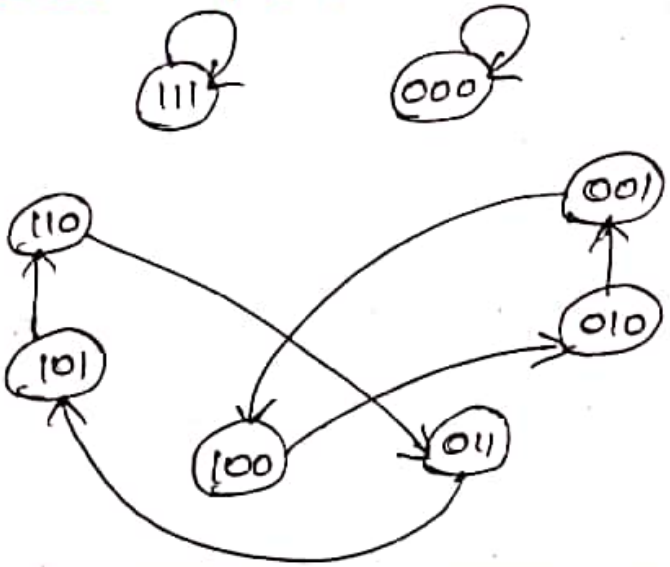
Next state equation :-

$$Q_A(t+1) = D_A$$
$$Q_B(t+1) = D_B$$
$$Q_C(t+1) = D_C$$

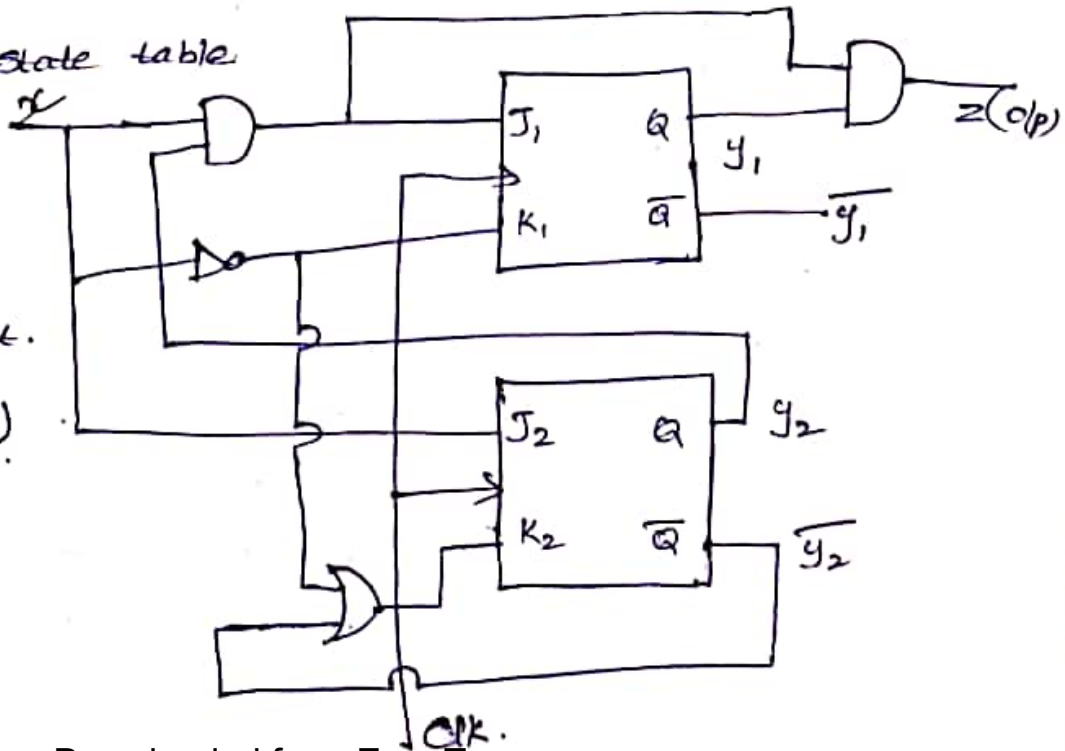
Second form of State Table :-

PS			NS		
Q_A	Q_B	Q_C	$Q_{A(t+1)}$	$Q_{B(t+1)}$	$Q_{C(t+1)}$
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	1	1	1	0
1	1	0	0	1	1
1	1	1	1	1	1

Step 3 :- State Diagram :-



2.) Draw the state table and state diagram from the given circuit. (Mealy Model)



Soln :

Note: o/p depends on p.s & ex. i/p ⇒ Healy Model

* Step 1:.

FF. i/ps ⇒

$J_1 = x \cdot y_2$

$K_1 = \overline{x}$

$J_2 = x$

$K_2 = \overline{y_2} + \overline{x}$.

P.S → $y_1 y_2$

External i/p → x

o/p, $z = x \cdot y_2 \cdot y_1$

$Q(t+1) = J\overline{Q} + \overline{K}Q$.

N.S → can be found from N.S eqn, (or) from the Char. table.

$y_1(t+1) = J_1 \overline{y_1} + \overline{K_1} y_1$

$y_2(t+1) = J_2 \overline{y_2} + \overline{K_2} y_2$.

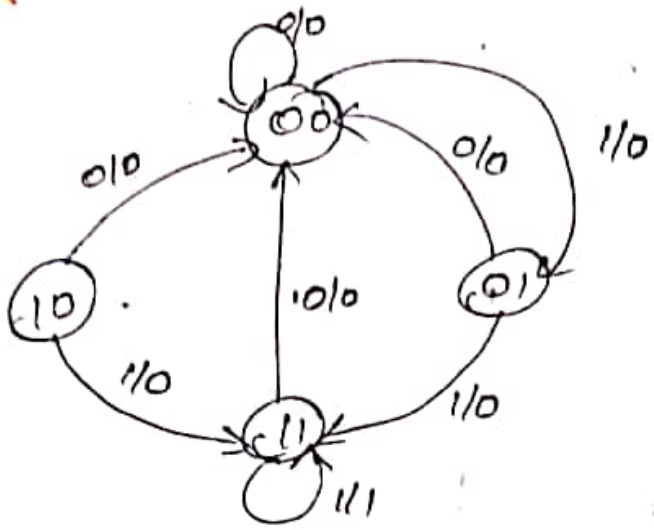
* Step 2 :. STATE TABLE.

Ex. i/p + PS			FF i/ps				N.S		o/p.
x	y_1	y_2	J_1	K_1	J_2	K_2	$y_1(t+1)$	$y_2(t+1)$	z
0	0	0	0	1	0	1	0	0	0
0	0	1	0	1	0	1	0	0	0
0	1	0	0	1	0	1	0	0	0
0	1	1	0	1	0	1	0	0	0
1	0	0	0	0	1	1	0	1	0
1	0	1	1	0	1	0	1	1	0
1	1	0	0	0	1	1	1	1	0
1	1	1	1	0	1	0	1	1	1

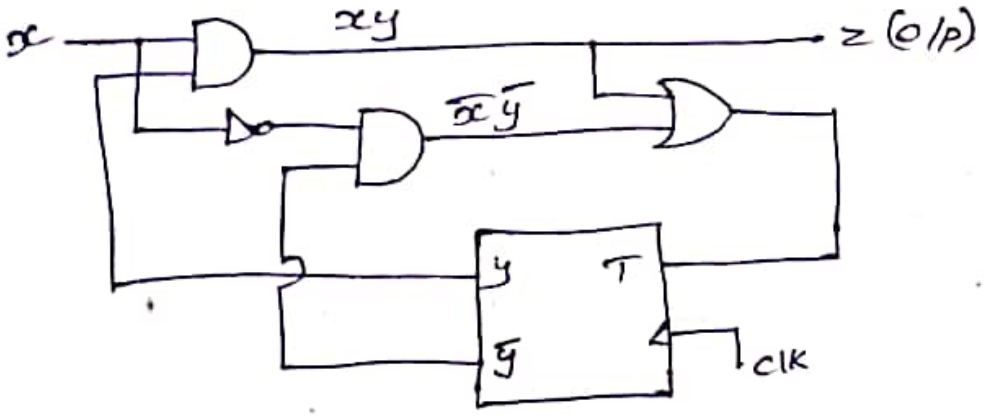
Second form of State table:.

y_1 y_2	$x=0$ $y_1 y_2$	$x=1$ $y_1 y_2$	$x=0$ z	$x=1$ z
PS	N.S		o/p	
0 0	0 0	0 1	0	0
0 1	0 0	1 1	0	0
1 0	0 0	1 1	0	0
1 1	0 0	1 1	0	1

Step 3 :- State Diagram :-



3.) Derive the state Table and state Diagram.



Soln :-

Step 1 :- P.S $\Rightarrow y$, FF i/ps $\Rightarrow T = xy + \bar{x}\bar{y} = \overline{x \oplus y}$.

o/p , $Z = xy$, Ext. i/p $\Rightarrow x$.

Next state eqn , $Q(t+1) = T \oplus Q$

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

Step 2 :- STATE TABLE :-

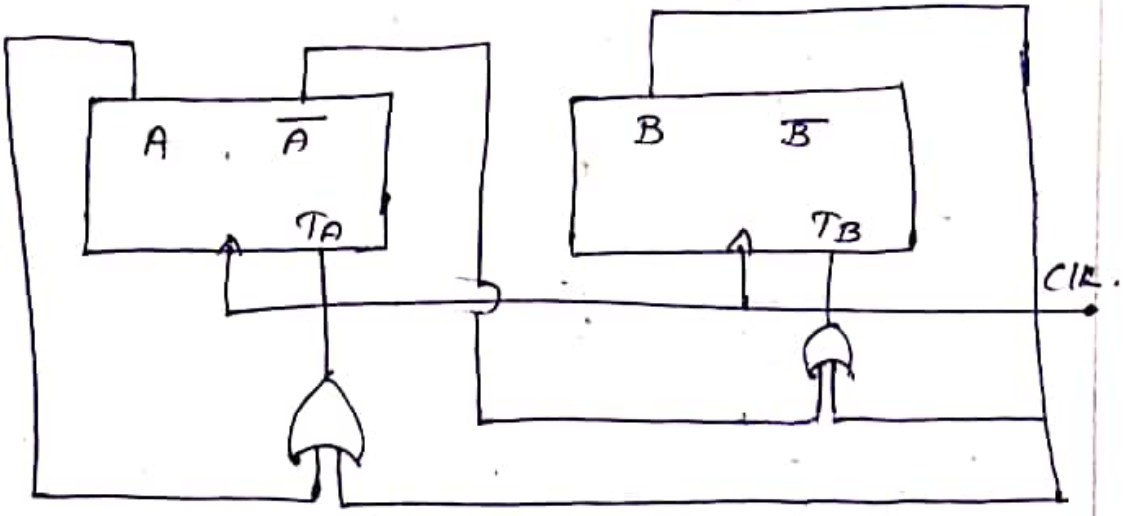
Ext. i/p & PS		FF i/p	N.S	o/p
x	y	T	y(t+1)	(z)
0	0	1	1	0
0	1	0	1	0
1	0	0	0	0
1	1	1	0	1

Second form of State Table :-

P.S	N.S		O/P	
y	x=0 y(t+1)	x=1 y(t+1)	x=0 z	x=1 z
0	1	0	0	0
1	1	0	0	1

Step 3: STATE DIAGRAM:

4.) Derive State Table & State diagram of the ckt.



Soln :-

Step 1 :- P.S → A, B

F.F i/ps → $T_A = A + B$

$T_B = \overline{A} + B$

Next state eqn, $Q(t+1) = T \oplus Q$ ⇒

$A(t+1) = T_A \oplus A$

$B(t+1) = T_B \oplus B$.

Step 2:-

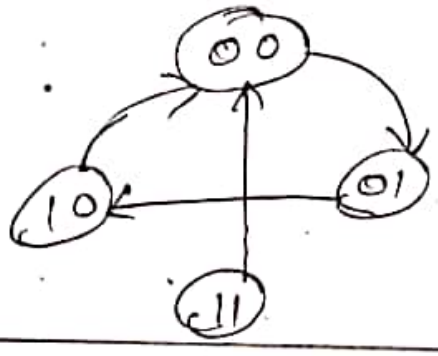
STATE TABLE

P.S		FF i/ps		N.S	
A	B	T_A	T_B	A(t+1)	B(t+1)
0	0	0	1	0	1
0	1	1	1	1	0
1	0	1	0	0	0
1	1	1	1	0	0

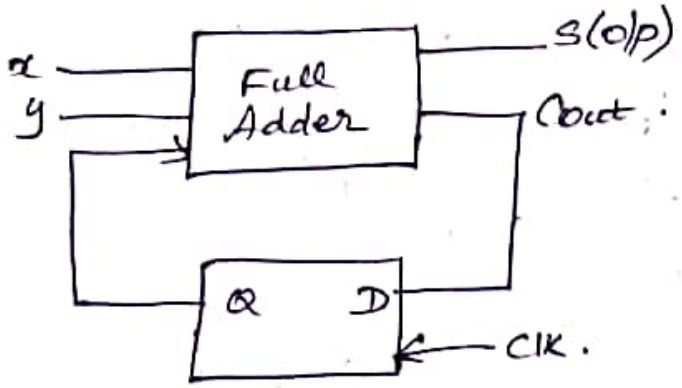
2nd form of State Table

P.S		N.S	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	0	0
1	1	0	0

Step 3:- STATE DIAGRAM:-



5.) A seq. circuit has one FF, two i/ps x, y and one o/p s., It consists of a full adder connected to D FF as shown below. Derive the State Table & State dgm.



Soln:-

Step 1:- External i/ps = x, y

P.S → Q = Qn

F.F i/p, D = Cout

Next state eqn, Q(t+1) = D.

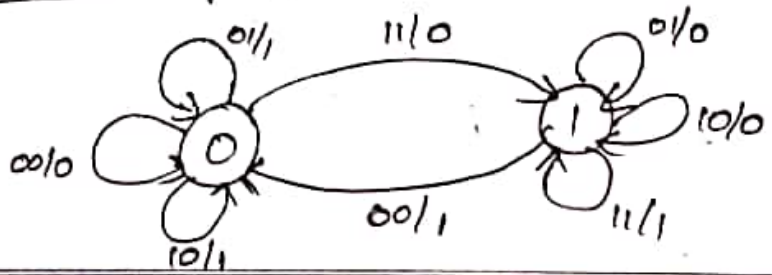
Step 2 : STATE TABLE:

Ext i/p & Ps			FF i/p	NS	O/p, S
x	y	Q	D (or) Cout	Q(t+1)	Sum
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1

Second form of State Table :

Ps	NS Q(t+1)				O/p, S			
Q	xy=00	xy=01	xy=10	xy=11	xy=00	xy=01	xy=10	xy=11
0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	0	0	1

Step 3 : STATE DIAGRAM:



Moore Vs Healy ckt Models :

1)

MOORE CKTS	HEALY CKTS
① It's o/p is a fn of present state only	① It's o/p is a fn of present state as well as present i/p.
② I/p changes doesnot affect the o/p.	② I/p changes may affect the o/p of the circuit.
③ This ckt requires more no. of states for implementing the same function.	③ It requires less no. of states for implementing same function.

Examples for Sequential circuit :-

1.) A seq. ckt with Delay FF A and B has two i/ps x & y and one o/p, z is specified by the following next state eqn and o/p eqn. Draw the state diagram and logic diagram.

$$A(t+1) = \bar{x}y + xa$$
$$B(t+1) = \bar{x}b + xa$$
$$z = b.$$

Soln :-

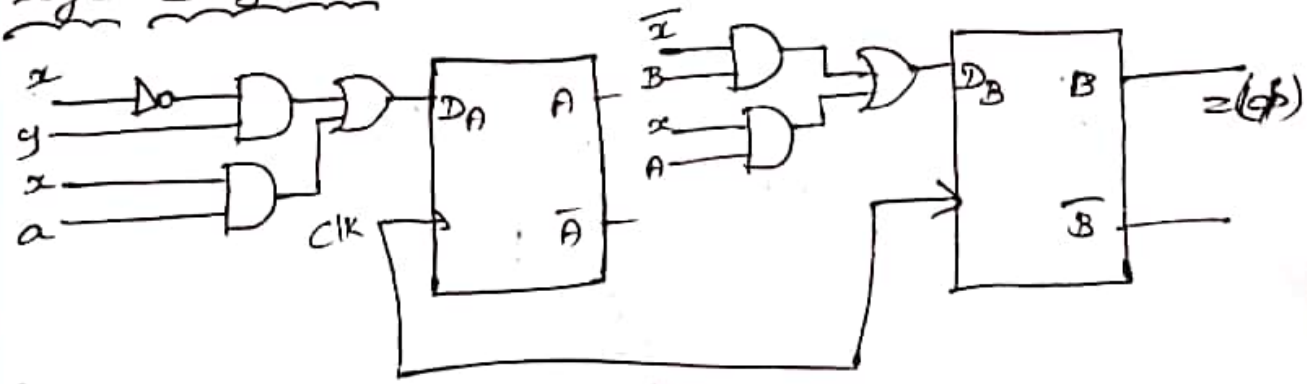
For Delay F.F, $Q(t+1) = D$

$$\therefore \boxed{\begin{aligned} D_A &= \bar{x}y + xa \\ D_B &= \bar{x}b + xa \end{aligned}} \qquad \text{o/p, } \boxed{z = b}$$

(o/p depends only on present state) → Moore ckt,

P.S ⇒ A, B ; External i/p ⇒ x, y.

Logic Diagram :-



STATE TABLE :-

Ext i/p & PS				FF i/ps		NS		o/p
x	y	A	B	D_A	D_B	A(t+1)	B(t+1)	z
0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	1	1
0	0	1	0	0	0	0	0	0
0	0	1	1	0	1	0	1	1
0	1	0	0	1	0	1	0	0
0	1	0	1	1	1	1	1	1
0	1	1	0	1	0	1	0	0
0	1	1	1	1	1	1	1	1

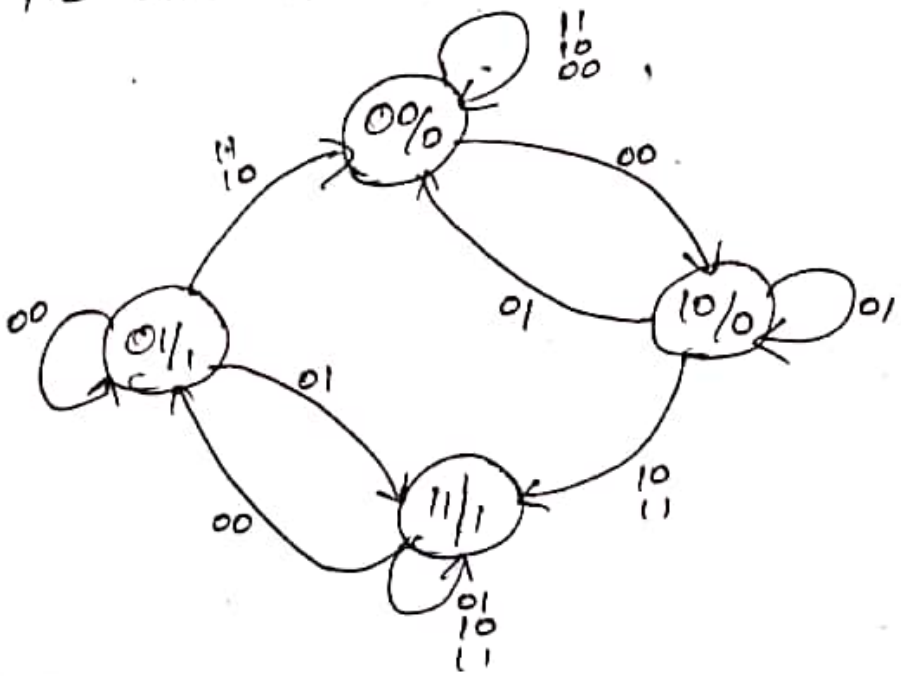
x	y	A	B	D_A	D_B	A(t+1)	B(t+1)	z
1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1
1	0	1	0	1	1	1	1	0
1	0	1	1	1	1	1	1	1
1	1	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	1
1	1	1	0	1	1	1	1	0
1	1	1	1	1	1	1	1	1

Second form of State Table:-

PS		NS								O/P, z
		xy=00		xy=01		xy=10		xy=11		z=9
A	B	A	B	A	B	A	B	A	B	
0	0	0	0	1	0	0	0	0	0	0
0	1	0	1	1	1	0	0	0	0	1
1	0	0	0	1	0	1	1	1	1	0
1	1	0	1	1	1	1	1	1	1	1

State Diagram :-

[As O/P depends only on present state, P.S and O/P should be within a single circle]



2.) A Synchronous seq. ckt has 3 delay FF, one i/p x. It is described by the following i/p function,

$D_A = (B\bar{C} + \bar{B}C)x + (BC + \bar{B}\bar{C})\bar{x}$ (Mealy Model).

$D_B = A$; $D_C = B$

i) Derive the next state eqn

ii) Draw the logic diagram

iii) Draw the state diagram.

Soln :-

$$D_A = (B\bar{C} + \bar{B}C)x + (BC + \bar{B}\bar{C})\bar{x}$$

$$= (B \oplus C)x + \overline{(B \oplus C)} \cdot \bar{x} \quad \because [y = (B \oplus C)].$$

$$= y \cdot x + \bar{y}\bar{x}$$

$$= \overline{y \oplus x} = \overline{B \oplus C \oplus x}$$

$$\boxed{D_A = \overline{B \oplus C \oplus x}}$$

Step 1 :

Present State → A, B, C ; Ext. i/p → x

F.F i/ps → $D_A = \overline{B \oplus C \oplus x}$; $D_B = A$; $D_C = B$.

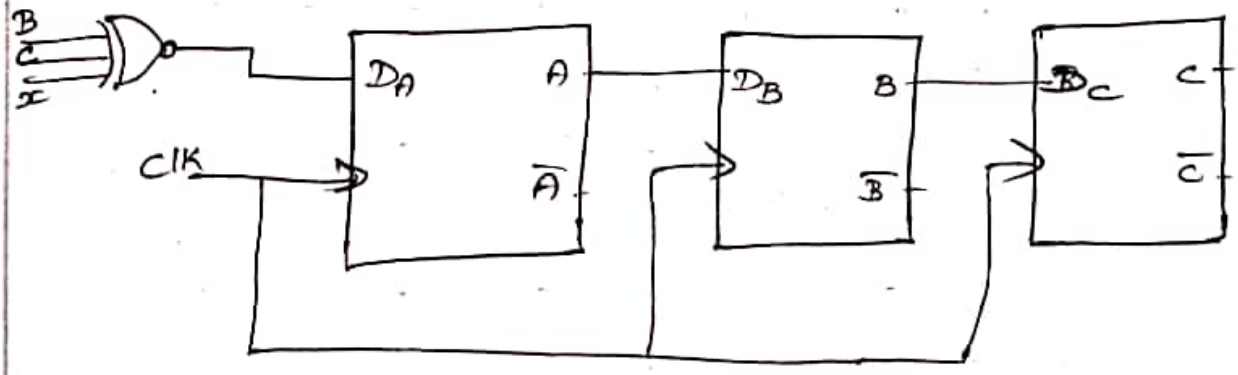
i) Next state eqn for Delay F.F :-

$Q(t+1) = D$ ∴ $A(t+1) = D_A$

$B(t+1) = D_B$

$C(t+1) = D_C$.

ii) Logic Diagram :-



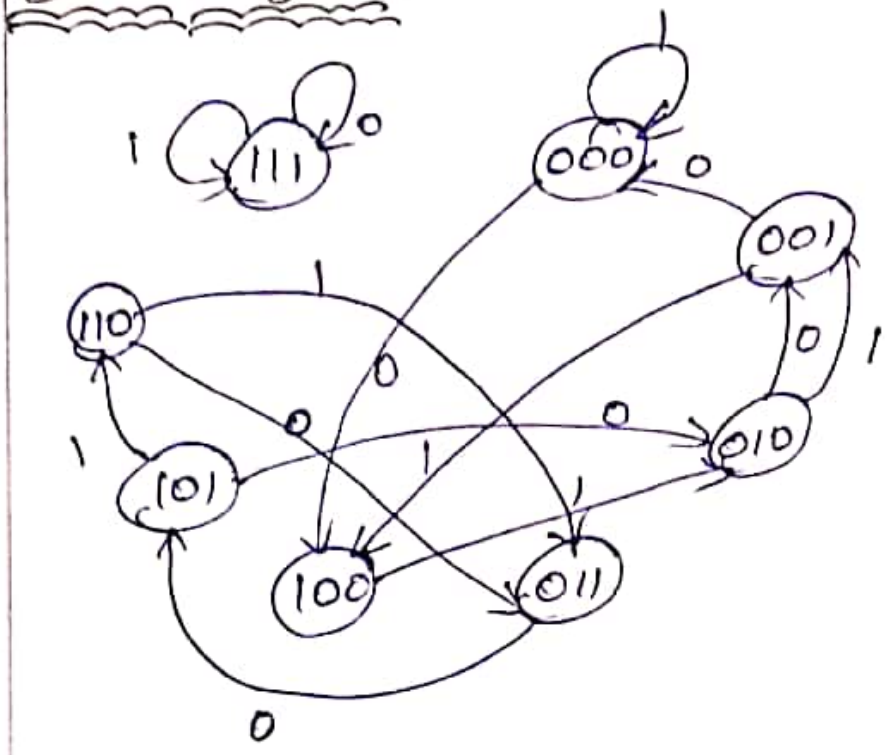
Step 2 :- State Table :-

Ext i/p & PS				FF i/ps			NS		
x	A	B	C	D_A	D_B	D_C	A(t+1)	B(t+1)	C(t+1)
0	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	1
0	0	1	1	1	0	1	1	0	1
0	1	0	0	1	1	0	1	1	0
0	1	0	1	0	1	0	0	1	0
0	1	1	0	0	1	1	0	1	1
0	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	1	0	0
1	0	1	0	0	0	1	0	0	1
1	0	1	1	0	0	1	0	0	1
1	1	0	0	0	1	0	0	1	0
1	1	0	1	1	1	0	1	1	0
1	1	1	0	0	1	1	0	1	1
1	1	1	1	0	1	1	0	1	1

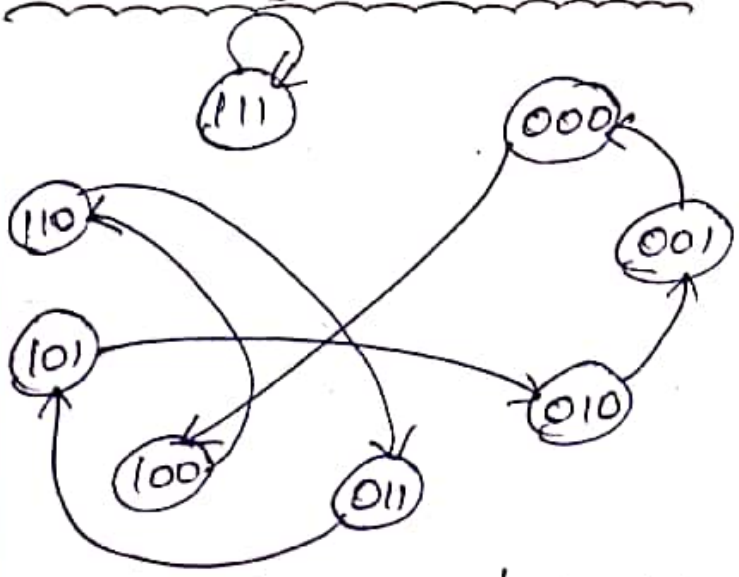
Second form of State table :-

PS			NS					
			x=0			x=1		
A	B	C	A	B	C	A	B	C
0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	0	1
0	1	1	1	0	1	1	0	1
1	0	0	1	1	0	0	1	0
1	0	1	0	1	0	1	1	0
1	1	0	0	1	1	0	1	1
1	1	1	1	1	1	1	1	1

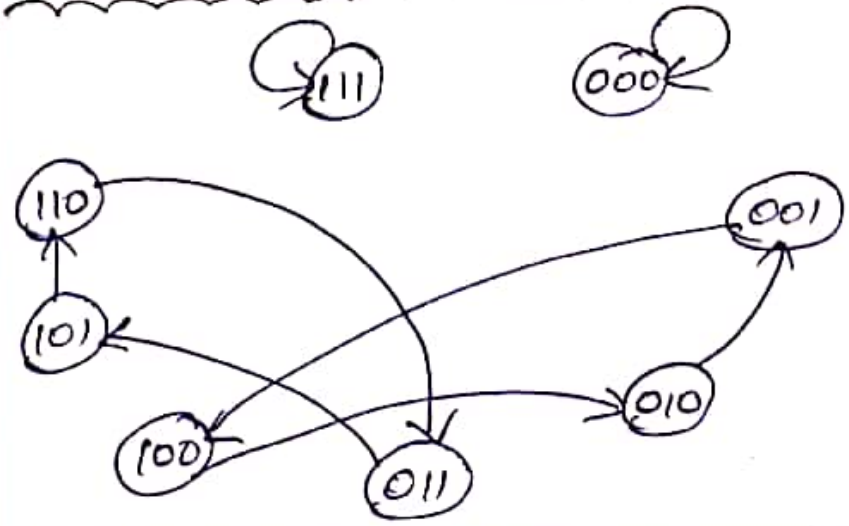
State Diagram :-



State Diagram, when x=0 :-



State Diagram, when x=1 :-



STATE REDUCTION :-

→ Design process, Consider the process of minimizing the cost of internal ckt.

→ The two most reductions are reduction in FFs and reduction in no. of gates.

→ In state reduction, reducing the no. of states in a sequential circuit without altering the i/p and o/p relation.

→ The two states are said to be equivalent, if for each set of i/ps, they give exactly the same o/p and send the ckt either to the same state or an equivalent state.

→ When two states are equivalent, one of these can be removed without altering the i/p & o/p relation.

→ The states marked inside the circles are denoted by letter instead of binary value.

Steps to Reduce :-

1.) Find which states are equal along with the o/p.

2.) If two are equal, replace one by the other.

For(eg) If $a \equiv b$, while we remove b, Sub b=a.

Prob:1.) Reduce no. of state in the following state table and draw the reduced state dgm.

PS	NS		O/P	
	x=0	x=1	x=0	x=1
a	f	b	0	0
b	d	c	0	0
c	f	e	0	0
d	g	a	1	0
e	d	c	0	0
f	f	b	1	1
g	g	h	0	1
h	g	a	1	0

Soln :-

PS	NS		O/P	
	x=0	x=1	x=0	x=1
→ a	f	b	0	0
→ b	d	~~c~~ a	0	0
X → c	f	~~e~~ b	0	0
→ d	g	a	1	0
X → e	d	~~e~~ a	0	0
f	f	b	1	1
g	g	~~h~~ d	0	1
X → h	g	a	1	0

1.) h ≡ d

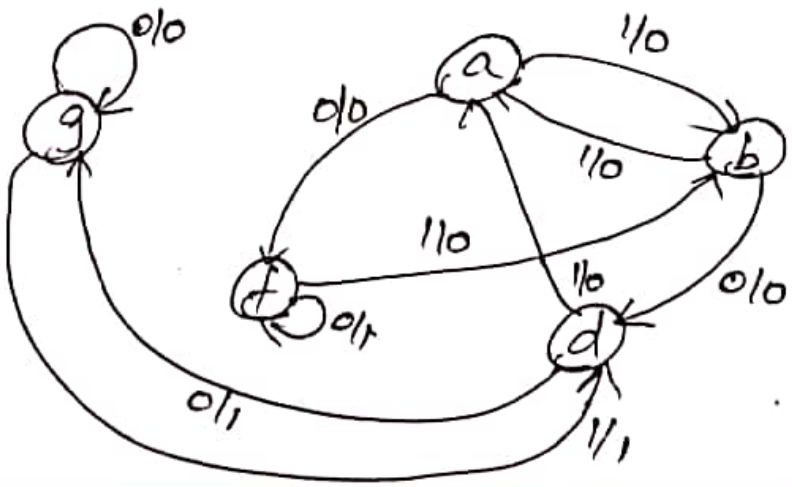
2.) b ≡ e

3.) c ≡ a

Reduced State Table :-

PS	NS		O/P	
	x=0	x=1	x=0	x=1
a	f	b	0	0
b	d	a	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

State Diagram :-



2.) Minimize the state table shown below:-

PS	NS, z (O/P)	
	x=0	x=1
A	B, 0	C, 0
B	B, 0	D, 0
C	B, 0	C, 0
D	E, 1	C, 0
E	B, 0	D, 0

Soln:-

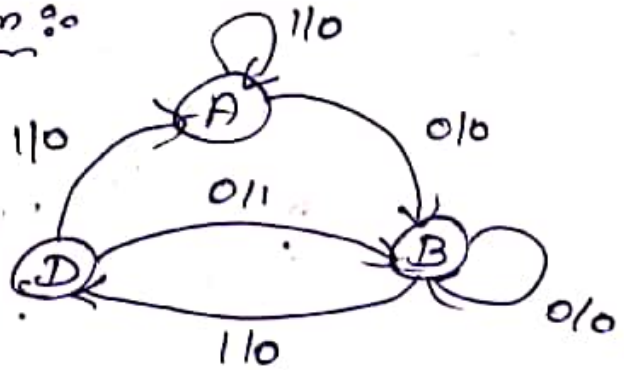
PS	NS		O/P, z	
	x=0	x=1	x=0	x=1
→ A	B	~~C~~ A	0	0
→ B	B	D	0	0
X → C	B	C	0	0
D	~~E~~ B	~~C~~ A	1	0
X → E	B	D	0	0

i.) $E \equiv B$ 2.) $C \equiv A$

Reduced state table:-

PS	N.S		O/P, z	
	x=0	x=1	x=0	x=1
A	B	A	0	0
B	B	D	0	0
D	B	A	1	0

State diagram:-



3.) Reduce the no. of states in the following state Table and (a) tabulate the reduced state table. (b) Starting from state a and I/p sequence 0 1 1 1 0 0 1 0 0 1 1 determine the o/p sequence for the given reduced table.

Soln:-

	PS	NS		O/P	
		x=0	x=1	x=0	x=1
→	a	f	b	0	0
→	b	d	a	0	0
X →	c	f	b	0	0
→	d	g	a	1	0
X →	e	d	c	0	0
	f	f	b	1	1
	g	g	d	0	1
X →	h	g	a	1	0

① $d \equiv h$
② $b \equiv e$
③ $a \equiv c$

Reduced state table / Transition table :-

PS	NS		O/P, Z	
	x=0	x=1	x=0	x=1
a	f	b	0	0
b	d	a	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

ii)

I/p sequence	State transition from a	O/p sequence.
0	a → f	0
1	f → b	1
1	b → a	0
1	a → b	0
0	b → d	0
0	d → g	1
1	g → d	1
0	d → g	1
0	g → g	0
1	g → d	1
1	d → a	0

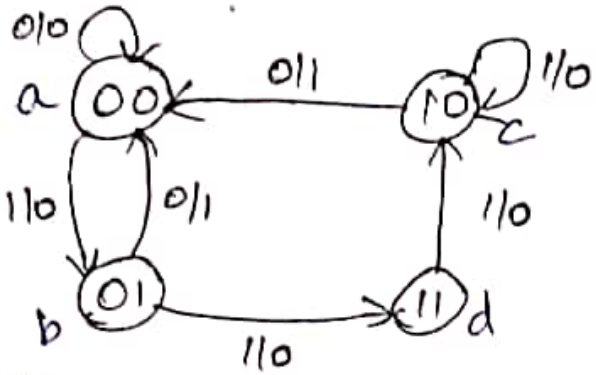
STATE ASSIGNMENT :-

Three possible binary state assignments, [for three bits, possible states → 8]:

State	Assignment 1 Binary	Ass-2 Gray code	Ass-3 One-Hot code.
a	000	000	00001
b	001	001	00010
c	010	011	00100
d	011	010	01000
e	100	010	10000

* only for five states.

1.) Reduce the following State diagram and draw the reduced one.



Soln:- Let us assign State for the given binary values.

a → 00 ; b → 01 ; c → 10 ; d → 11 .

State table :-

PS	NS		O/P	
	x=0	x=1	x=0	x=1
a	a	b	0	0
→ b	a	d ≠ b	1	0
X → c	a	c	1	0
X → d	a	c	1	0

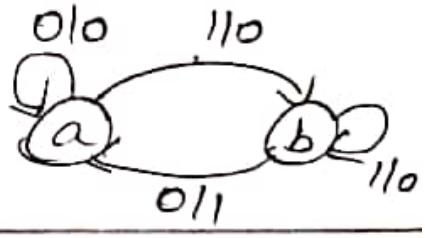
(1.) d ≡ c

(2.) b ≡ c

Reduced State table :-

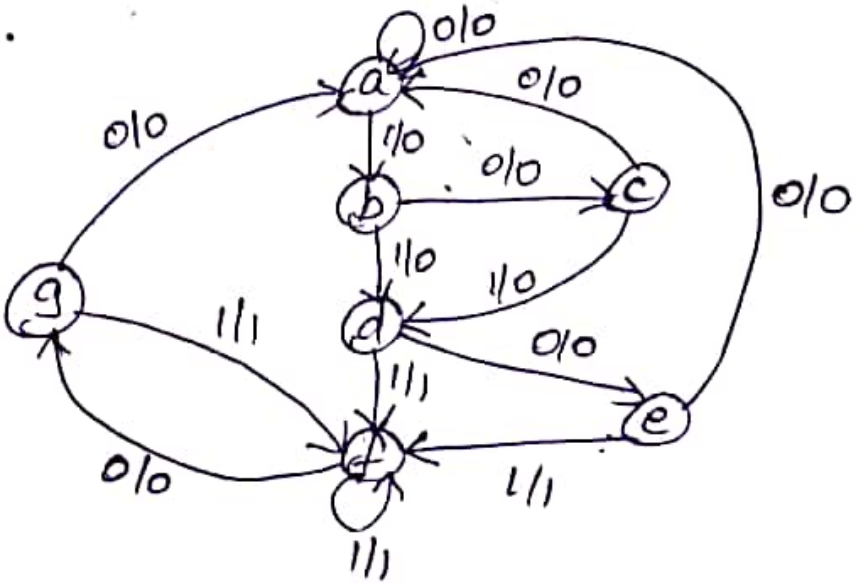
PS	NS		O/P	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	a	b	1	0

State diagram :-



U.B

2.) Reduce the following State diagram & draw the reduced one.



Soln :- STATE TABLE :-

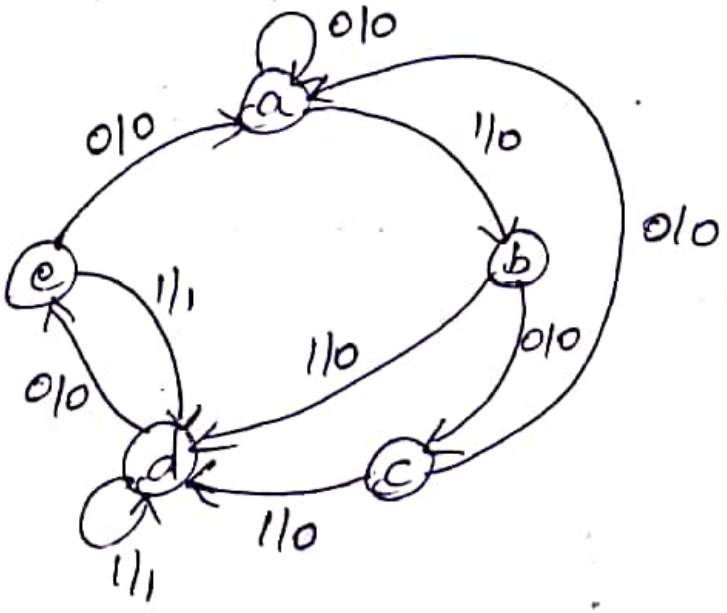
PS	NS		O/P	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
→ d	e	fd	0	1
→ e	a	fd	0	1
X → f	ge	f	0	1
X → g	a	f	0	1

① g≡e
② d≡f

Reduced State Table :-

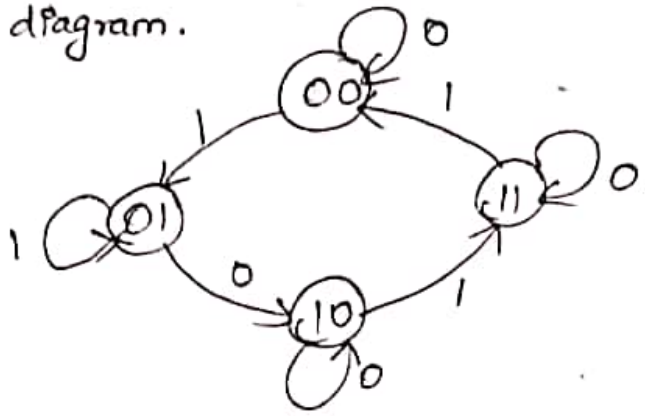
PS	NS		O/P	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

STATE DIAGRAM :-



DESIGN OF SYNCHRONOUS SEQUENTIAL CIRCUITS :-

1.) Design a Syn. seq. ckt using JK FF for the following State diagram.



Soln :-

Step 1 :- State Table :-

Let us take P.S → A, B
↑
2 F.F
Ext. i/p → x
FF i/ps → $J_A K_A$, $J_B K_B$.

PS		NS			
		x=0		x=1	
A	B	A	B	A	B
0	0	0	0	0	1
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	0

State Table :-

Ext. i/p & PS			N.S		FF i/ps			
x	A	B	A(t+1)	B(t+1)	J_A	K_A	J_B	K_B
0	0	0	0	0	0	X	0	X
0	0	1	1	0	1	X	X	1
0	1	0	1	0	X	0	0	X
0	1	1	1	1	X	0	X	0
1	0	0	0	1	0	X	1	X
1	0	1	0	1	0	X	X	0
1	1	0	1	1	X	0	1	X
1	1	1	0	0	X	1	X	1

K-map for J_A :-

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

$$J_A = \bar{x} B$$

K-map for K_A :

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

$K_A = x\,B$

K-map for J_B :

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

$J_B = x$.

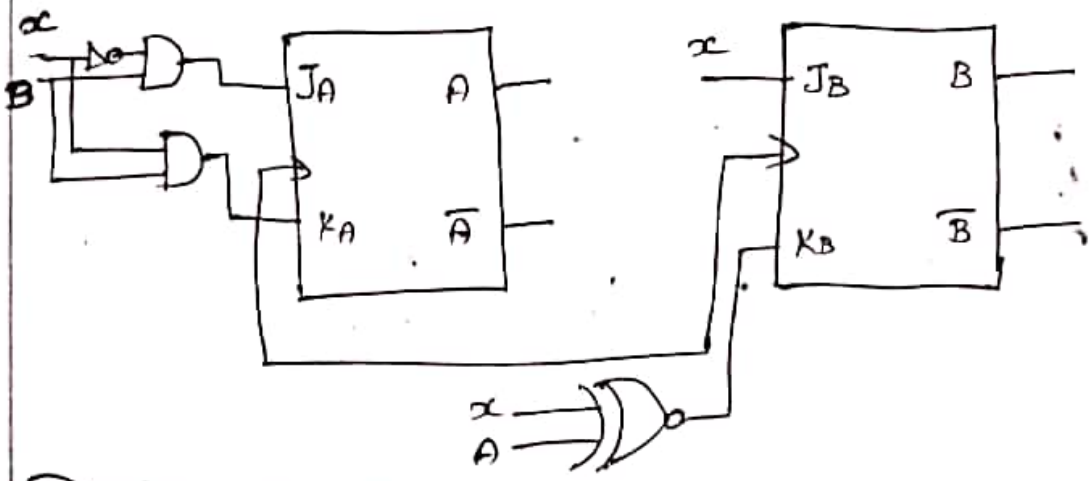
K-map for K_B :

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

$K_B = \bar{x}\,\bar{A} + x\,A$

$K_B = \overline{x \oplus A}$.

Implementation :-

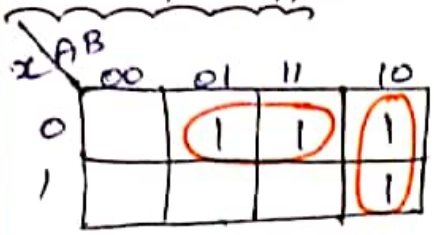


Design using delay FF :-

Ext. i/p & PS			NS		FF i/Ps	
x	A	B	A(t+1)	B(t+1)	D_A	D_B
0	0	0	0	0	0	0
0	0	1	1	0	1	0
0	1	0	1	0	1	0
0	1	1	1	1	1	1
1	0	0	0	1	0	1
1	0	1	0	1	0	1
1	1	0	1	1	1	1
1	1	1	0	0	0	0

$D_A = A(t+1)$
$D_B = B(t+1)$

K-map for D_A :-



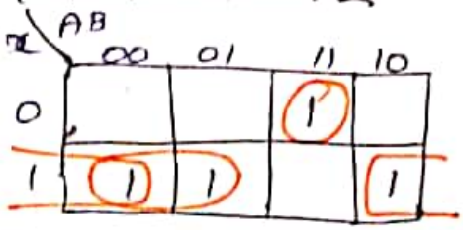
$$D_A = \bar{x}B + A\bar{B}$$

[$y = AB$

$\bar{y} = \overline{AB}$

$= \bar{A} + \bar{B}$]

K-map for D_B :-



$$D_B = \bar{x}AB + x\bar{A} + x\bar{B}$$

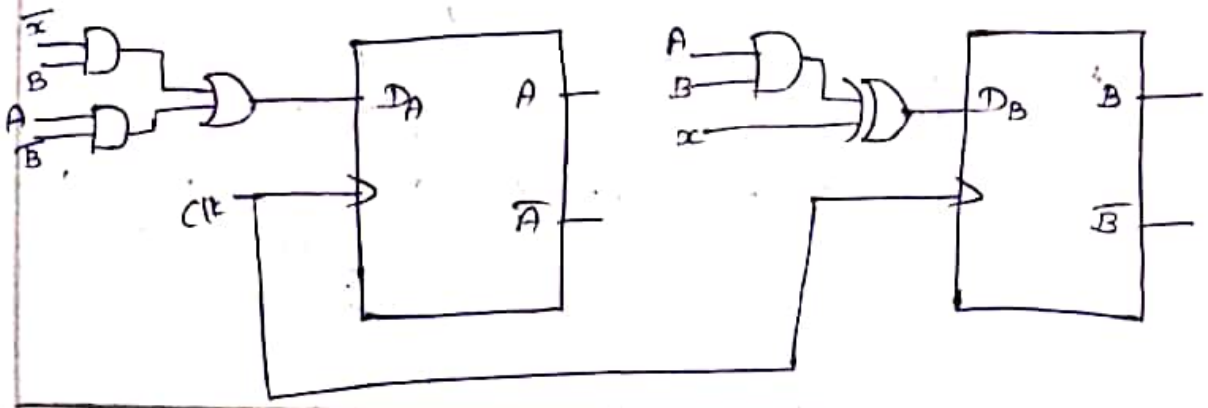
$$= \bar{x}AB + x(\bar{A} + \bar{B})$$

$$= \bar{x}y + x\bar{y}$$

$$= x \oplus y$$

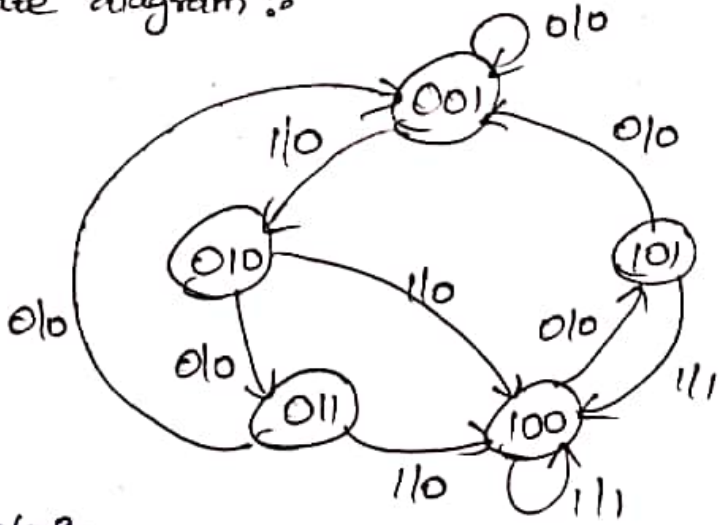
$$D_B = x \oplus AB$$

Implementation :-



Design with Unused States :-

(1.) Design a Syn. Seq. circuit using JK FF for the following state diagram :-



Soln :-

As there are three binary values, we need 3 FF, Let it be A, B, C.

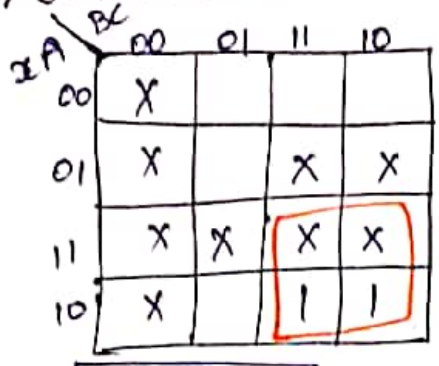
Unused States are 101, 110, 111 → don't care for the next state & FF i/p.

STATE TABLE:

PS			N.S										O/p	
			x=0			x=1								
A	B	C	A	B	C	A	B	C					x=0	x=1
0	0	0	X	X	X	X	X	X					X	X
0	0	1	0	0	1	0	1	0					0	0
0	1	0	0	1	1	1	0	0					0	0
0	1	1	0	0	1	1	0	0					0	0
1	0	0	1	0	1	1	0	0					0	1
1	0	1	0	0	1	1	0	0					0	1
1	1	0	X	X	X	X	X	X					X	X
1	1	1	X	X	X	X	X	X					X	X

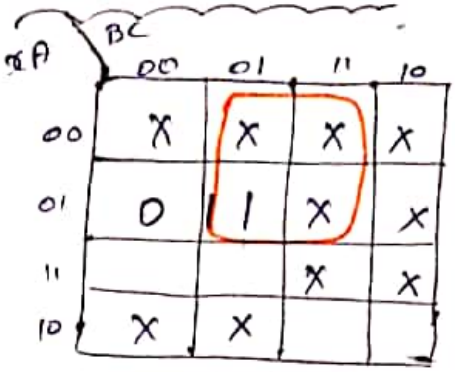
Ext i/p & PS				NS			FF i/ps						O/p
x	A	B	C	A(t+1)	B(t+1)	C(t+1)	S_A	R_A	S_B	R_B	S_C	R_C	Y
0	0	0	0	X	X	X	X	X	X	X	X	X	X
0	0	0	1	0	0	1	0	X	0	X	X	0	0
0	0	1	0	0	1	1	0	X	X	0	1	0	0
0	0	1	1	0	0	1	0	X	0	1	X	0	0
0	1	0	0	1	0	1	X	0	0	X	1	0	0
0	1	0	1	0	0	1	0	1	0	X	X	0	0
0	1	1	0	X	X	X	X	X	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X	X	X	X	X
1	0	0	0	X	X	X	X	X	X	X	X	X	X
1	0	0	1	0	1	0	0	X	1	0	0	1	0
1	0	1	0	1	0	0	1	0	0	1	0	X	0
1	0	1	1	1	0	0	1	0	0	1	0	1	0
1	1	0	0	1	0	0	X	0	0	X	0	X	1
1	1	0	1	1	0	0	X	0	0	X	0	1	1
1	1	1	0	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X

K-map for S_A :-



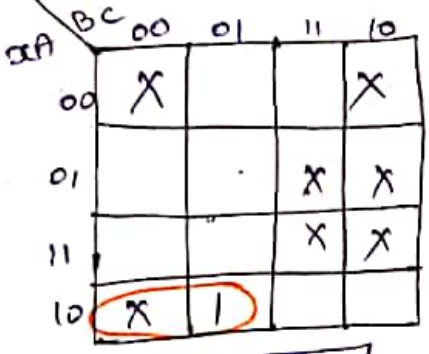
$S_A = xB$

K-map for R_A :-



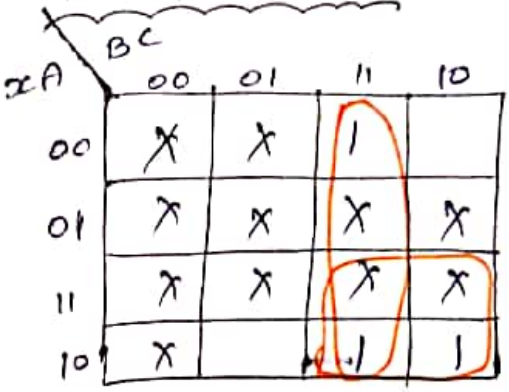
$R_A = \bar{x}C$

K-map for S_B :-



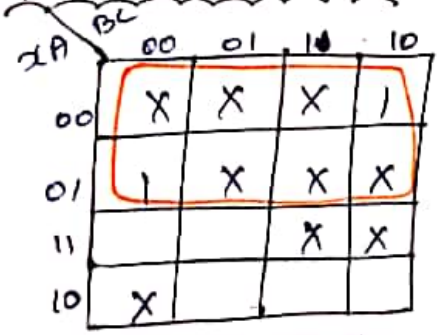
$S_B = x\bar{A}\bar{B}$

K-map for R_B :-



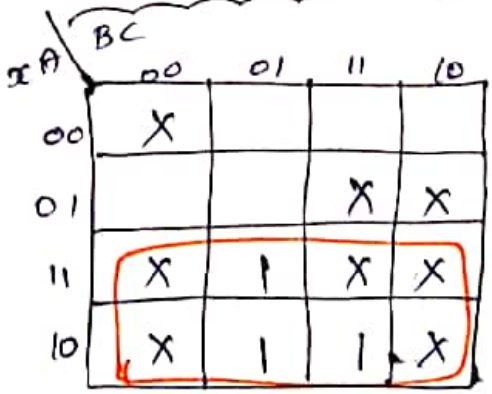
$R_B = BC + xB$

K-map for S_C :-



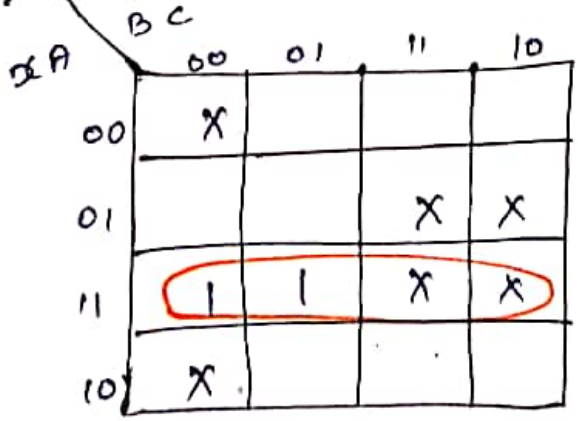
$S_C = \bar{x}$

K-map for R_C :-

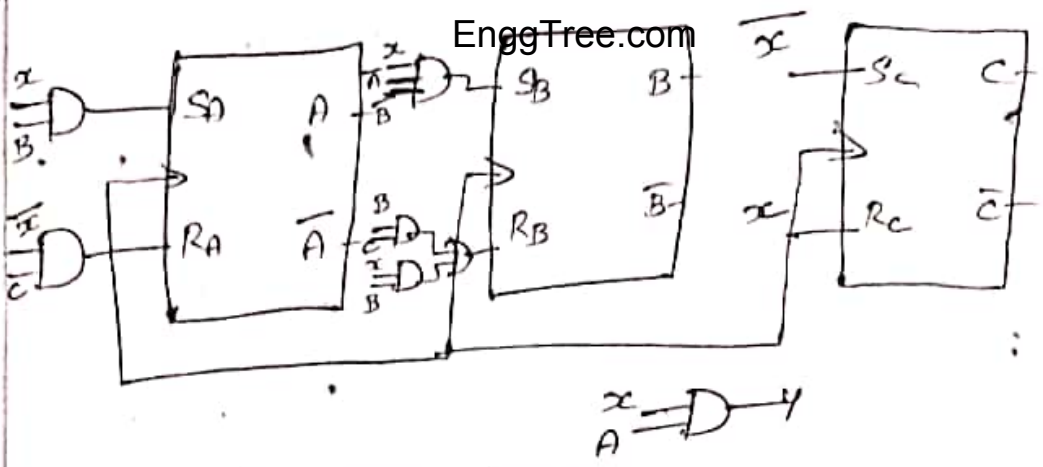


$R_C = x$

K-map for y :-



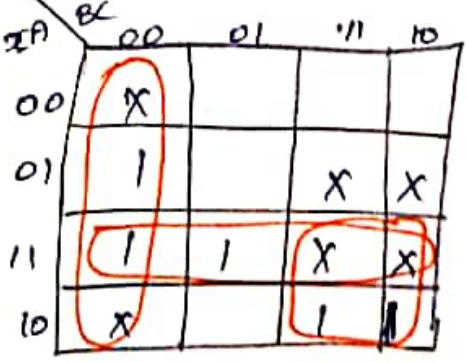
$y = xA$



Design with Delay FF :-

Ext. i/p & PS				NS			FF i/Ps.			O/P
x	A	B	C	A(t+1)	B(t+1)	C(t+1)	D_A	D_B	D_C	y
0	0	0	0	X	X	X	X	X	X	X
0	0	0	1	0	0	1	0	0	1	0
0	0	1	0	0	1	1	0	1	1	0
0	0	1	1	0	0	1	0	0	1	0
0	1	0	0	1	0	1	1	0	1	0
0	1	0	1	0	0	1	0	0	1	0
0	1	1	0	X	X	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X	X
1	0	0	0	X	X	X	X	X	X	X
1	0	0	1	0	1	0	0	1	0	0
1	0	1	0	1	0	0	1	0	0	0
1	0	1	1	1	0	0	1	0	0	0
1	1	0	0	1	0	0	1	0	0	1
1	1	0	1	1	0	0	1	0	0	1
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X

K-map for D_A :-



$D_A = \bar{B}\bar{C} + xA + xB$

$\boxed{D_A = \bar{B}\bar{C} + x(A+B)}$

K-map for D_B :-

xA \ BC	00	01	11	10
00	X			1
01			X	X
11			X	X
10	X	1		

$$D_B = x\bar{A}\bar{B} + \bar{x}B\bar{C}$$

K-map for D_C :-

xA \ BC	00	01	11	10
00	X	1	1	1
01	1	1	X	X
11			X	X
10	X			

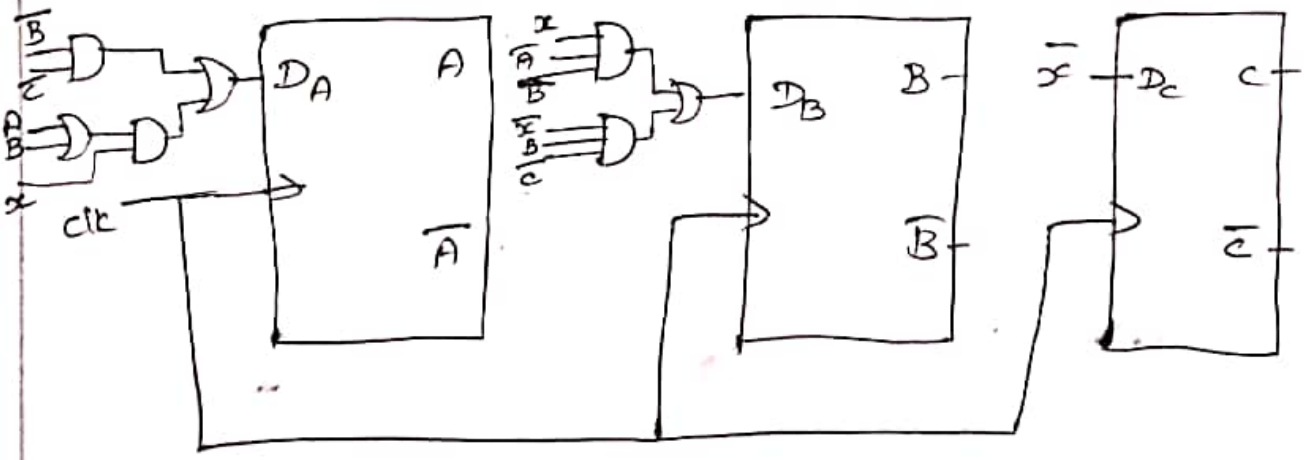
$$D_C = \bar{x}$$

K-map for y :-

xA \ BC	00	01	11	10
00	X			
01			X	X
11	1	1	X	X
10	X			

$$y = xA$$

Implementation :-



COUNTERS :-

A sequential circuit that goes through a prescribed sequence of states upon the application of input is a Counter.

→ It is also used to count the number of clock pulses.

→ Counters are used for counting the occurence of events.

They consists of a series of F.F which undergo a sequence of states due to the application of clock pulses.

Types of Counters :-

Counters → Series / Serial / Asynchronous / Ripple.
→ Synchronous / Parallel.

Asynchronous Counter	Synchronous Counter.
(1.) Clock pulses are not given simultaneously for each flipflops.	(1.) Clock pulses are given simultaneously to each flipflop.
(2.) Clock i/p for one FF is the o/p of the previous FF (may be Q or $\bar{Q}$).	(2.) Common external clock for all the flipflops.

→ Counter may be up or down counter

→ Each of the count of counter is called the state of the counter.

Modulus of the Counter :-

The no. of states through which the counter passes before returning to the steady state is called Modulus of the counter.

(Mod) modulus of the counter → No. of distinct counts including zero.

n-bit counter will have n F.F and 2^n states, and divides the i/p frequency by 2^n. Hence it is a divide by 2^n counter.

Eg) :. 2-bit Counter → 2 F.F → $2^2 = 4$ States.
It is a MOD-4 Counter.

→ No. of F.F's required to construct a MOD-N Counter is given by $N \leq 2^n$ n → no. of FF's.

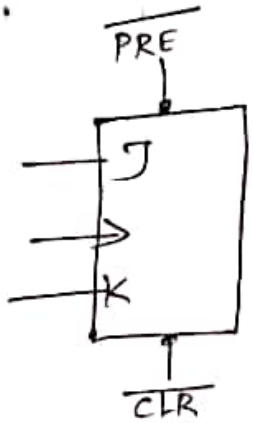
Counting of a counter → Sequential Counting (up/down)
→ Non-Sequential Counting.

* Next State of a Counter is purely depends on the present state only.

Asynchronous Inputs :-

Asynchronous i/p affect the F.F o/p independently of the synchronous i/ps and clock i/p.

$\overline{PRE}$, $\overline{CLR}$ → These i/ps can be used to set ($\overline{PRE}$) FF to 1 (or) reset ($\overline{CLR}$) to 0 regardless of the i/p and clock pulses.



$\overline{PRE}$	$\overline{CLR}$	F.F Response.
0	0	Not used
0	1	Q = 1
1	0	Q = 0
1	1	Clocked operation.

Asynchronous Counter uses T F.F to perform Counting fn. The actual hardware used is J-K FF connected in Toggle mode [J = K = 1].

Delay F.F may also be used.

Up-Counter → 0-1-2 ---- 0

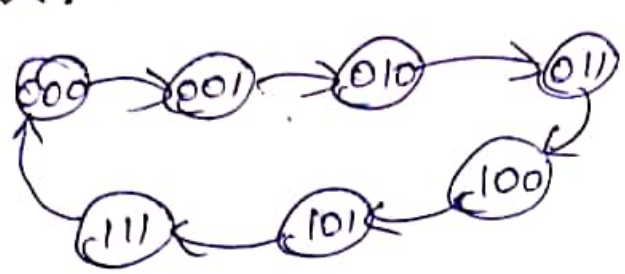
Down-Counter → 0-7-6-5 --- 0

Ripple Counter :-

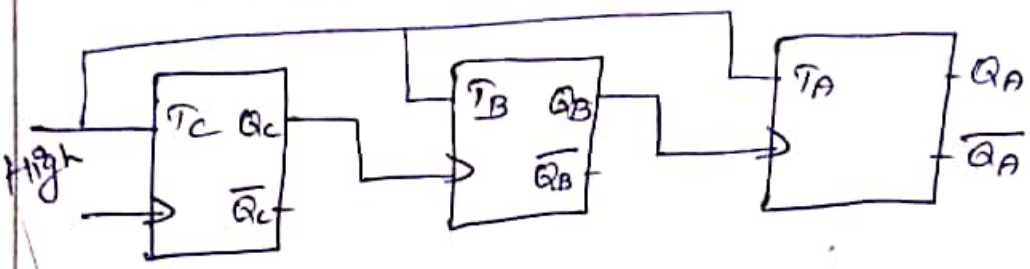
Output of one F.F is given to i/p of another F.F

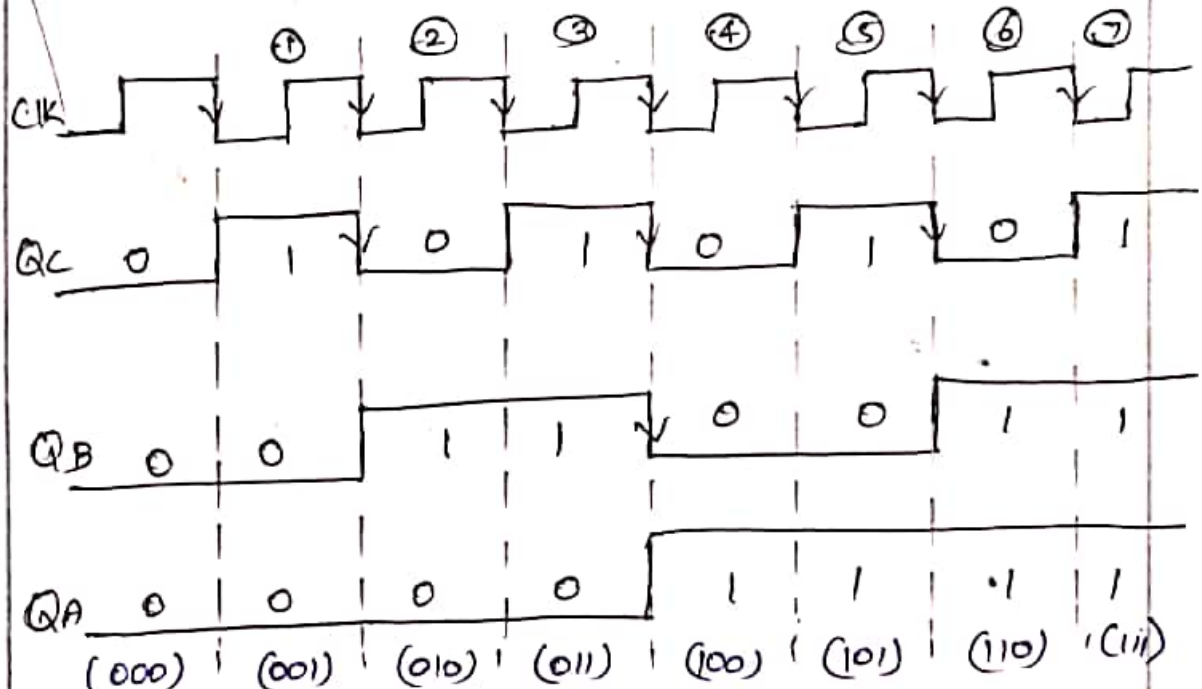
Negative Edge triggering :-

3-bit Ripple up-counter [using T-F.F]



Let → ABC (MSB)





(0-1-2-3-4-5-6-7-0) → 8 clock pulses are needed

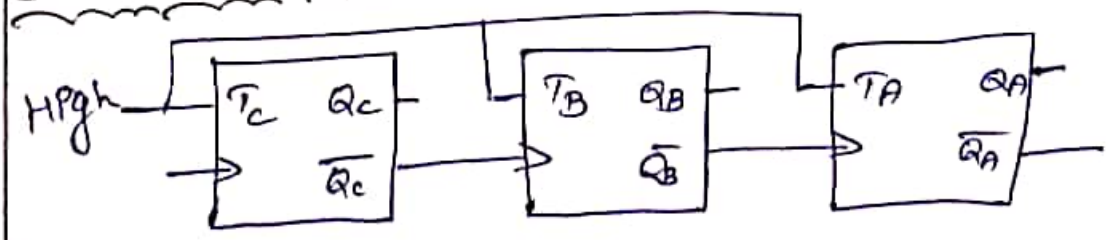
0 → 7 → 0

7 clock pulse needed to count from (0-7).

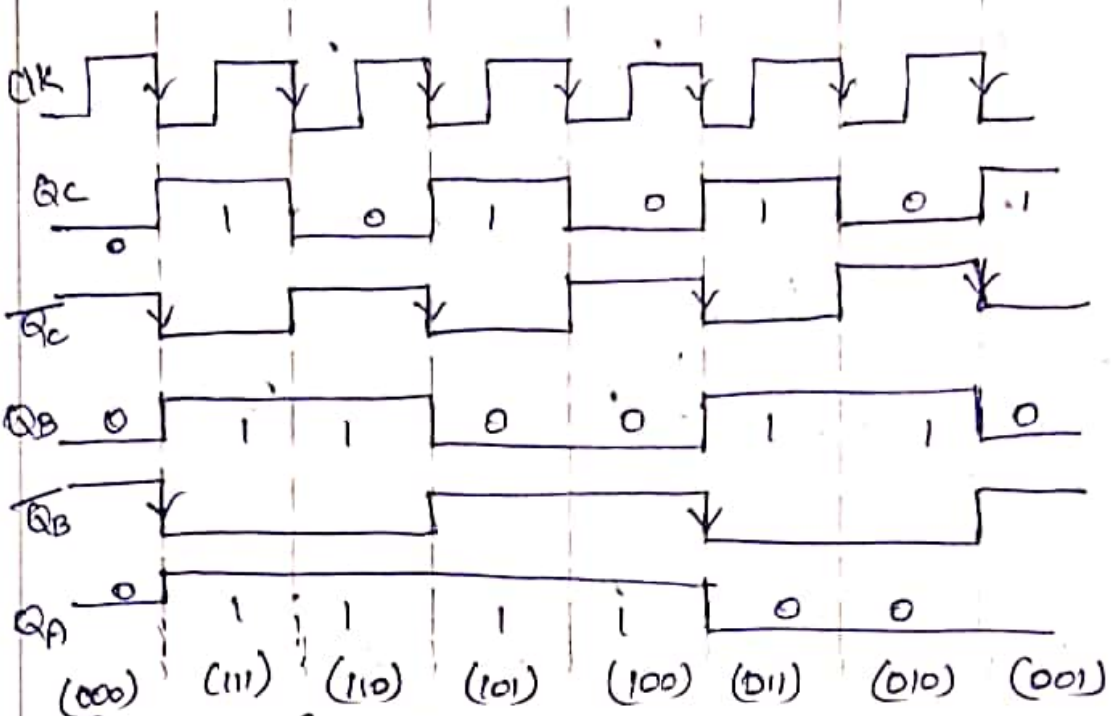
$Q_A Q_B Q_C$

MSB ↑ LSB

3-bit Ripple Down Counter :-



O/p is taken from (Q_A Q_B Q_C) only.

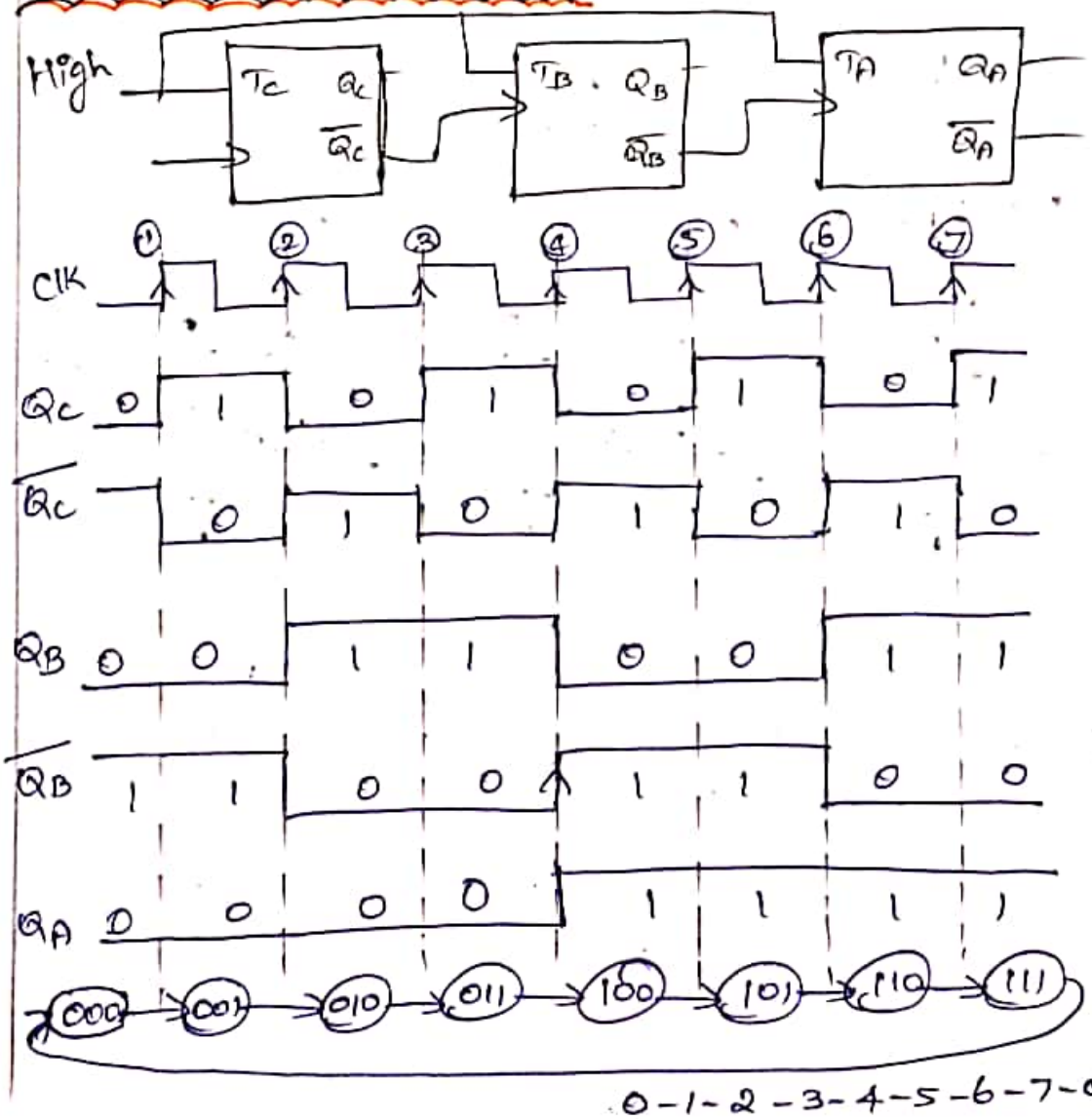


(0-7-6-5-4-3-2-1-0)

Q_A Q_B Q_C
↑ MSB ↑ LSB.

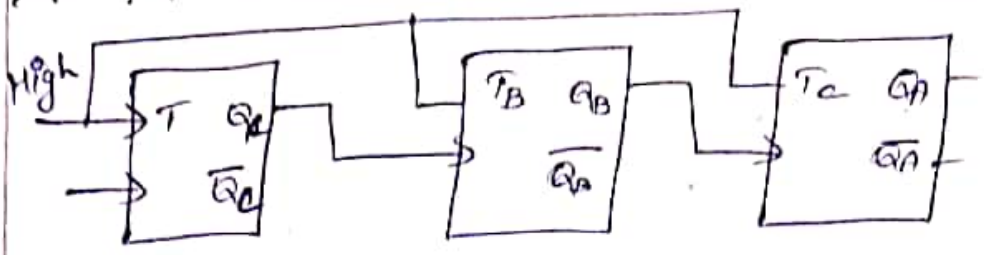
POSITIVE EDGE TRIGGERING:-

3-bit Ripple Up Counter:-

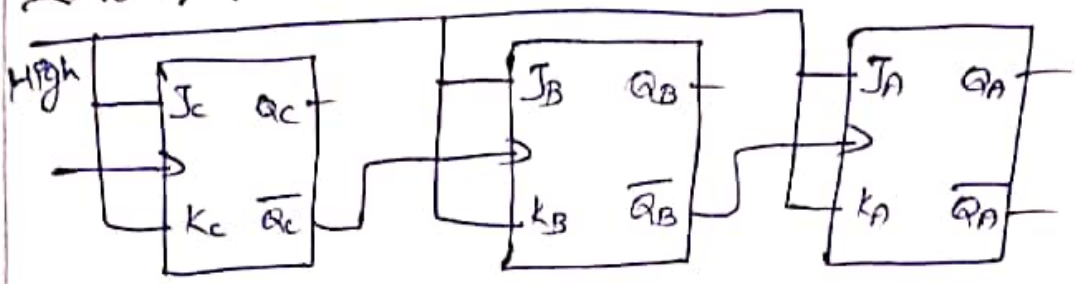


0-1-2-3-4-5-6-7-0.

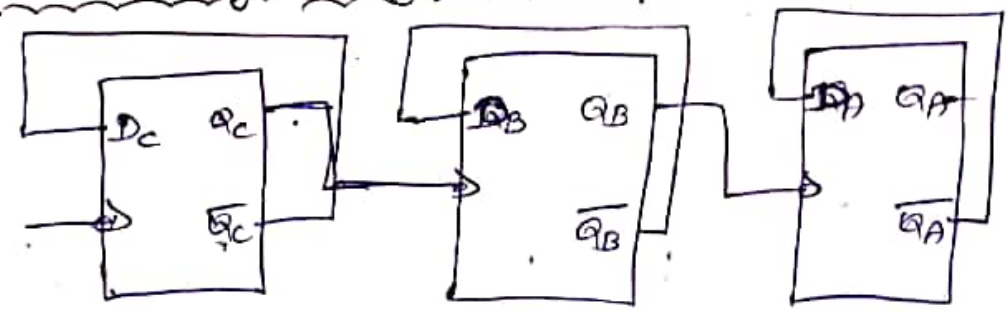
For Ripple down counting [using D FF].



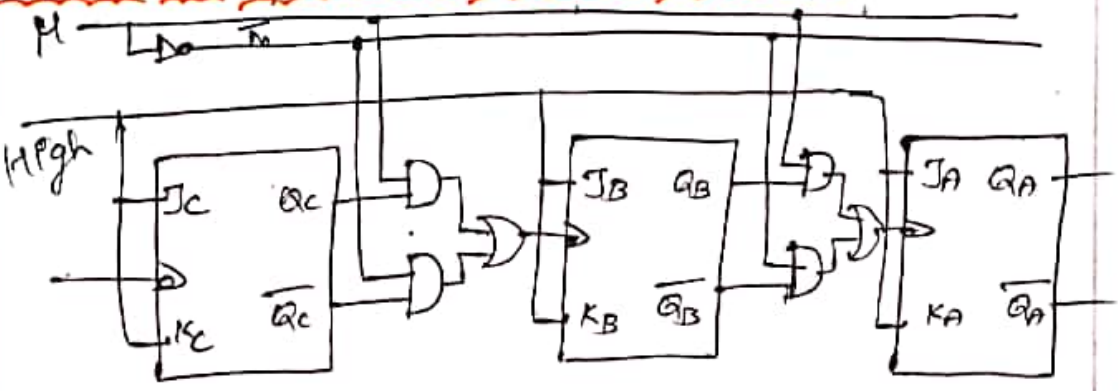
using (J-K) Flipflop:



Up-Counting using D FF (3-bit):



Design of UP/DOWN RIPPLE COUNTER:



M = 0 → up counting

M = 1 → down counting.

Logic 1 on M → enables Gates 1 & 2 → up Counting.
disables Gates 3 & 4

Logic 0 on M → Enables AND gates 3 & 4 → Down Counting.
Disables AND gates 1 & 2

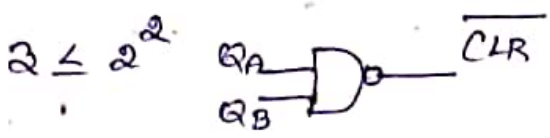
DESIGN OF ASYNCHRONOUS COUNTER :-

1.) Design MOD-3 ripple counter :

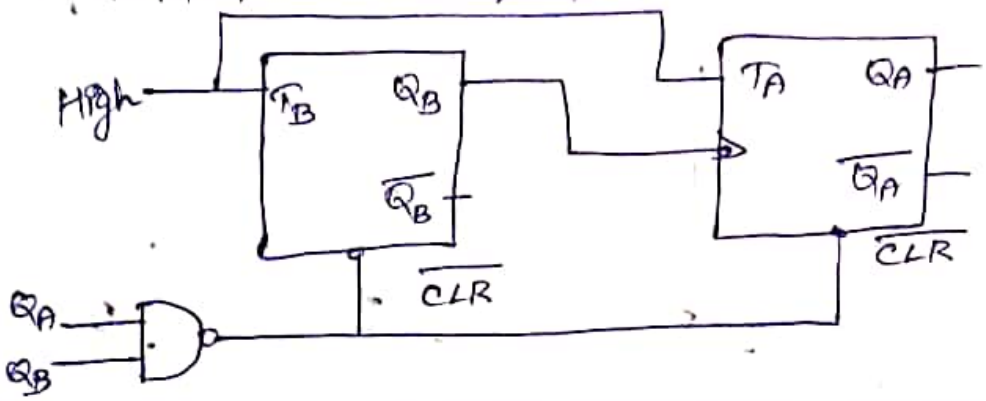
Soln :- Sequence 0-1-2-0

Q_A	Q_B
0	0
0	1
1	0
1	1

$2 \leq 2^2$



2 flipflops are required.



2.) Design MOD-10 / BCD / Decade ripple counter :-

i) using T FF ii) D-FF iii) JK-FF.

Soln :- Mod-N $N \leq 2^n$

$10 \leq 2^4$ $\Rightarrow$ 4 F.Fs are required.

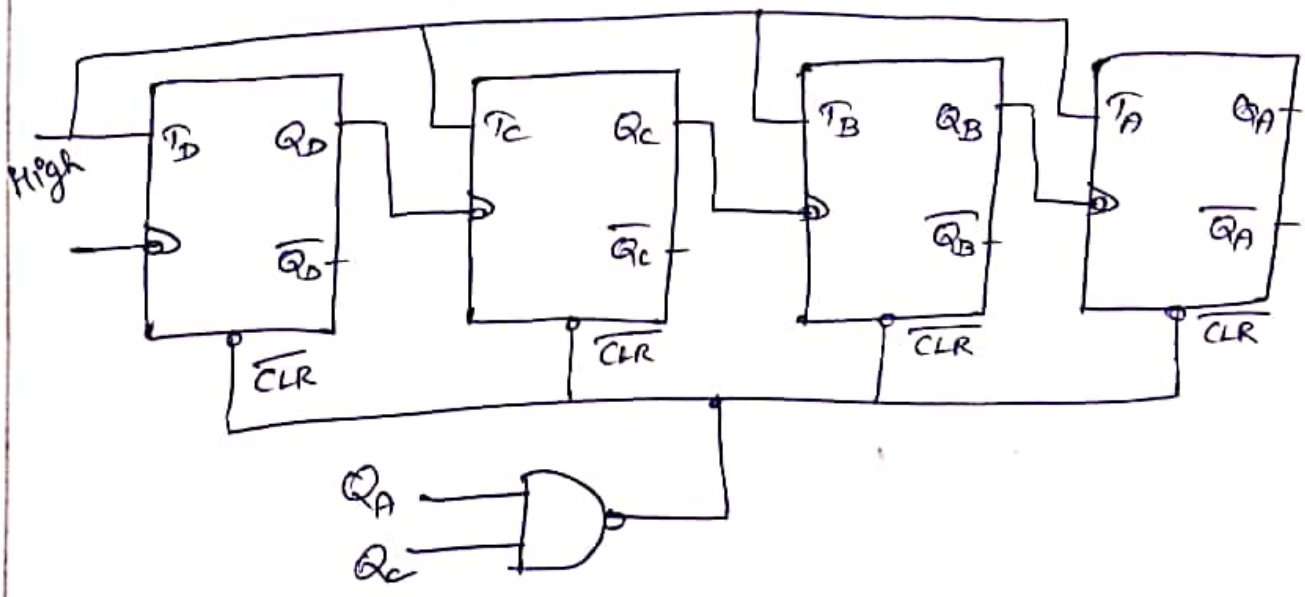
Using T Flipflop:

Sequence 0-1-2-3-4-5-6-7-8-9-0

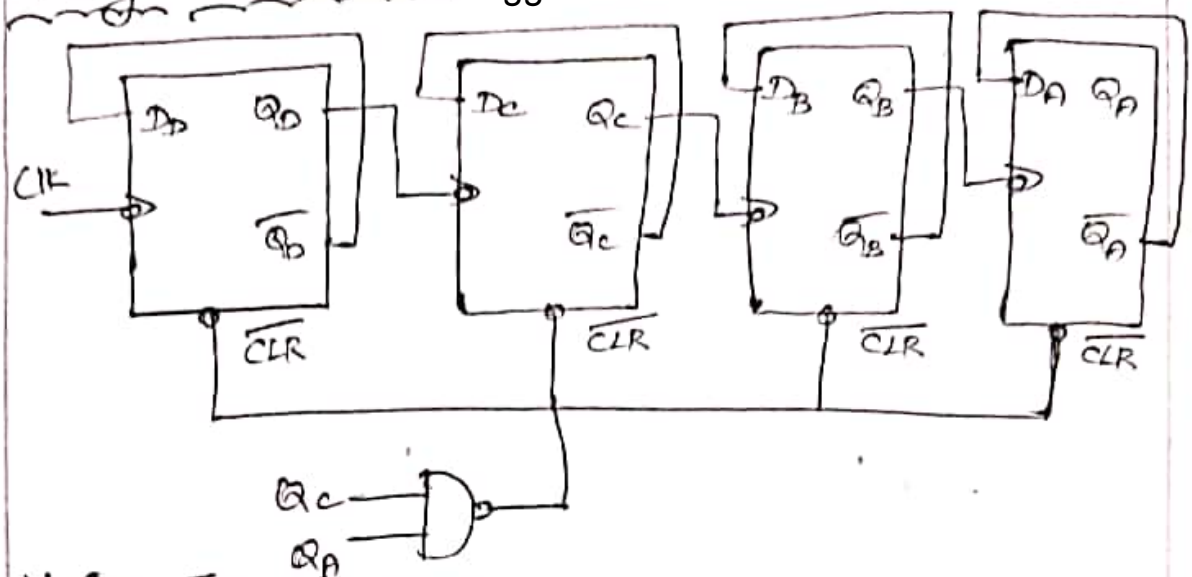
when 10 comes, it has to be reset to 0.

$Q_A\ Q_B\ Q_C\ Q_D$

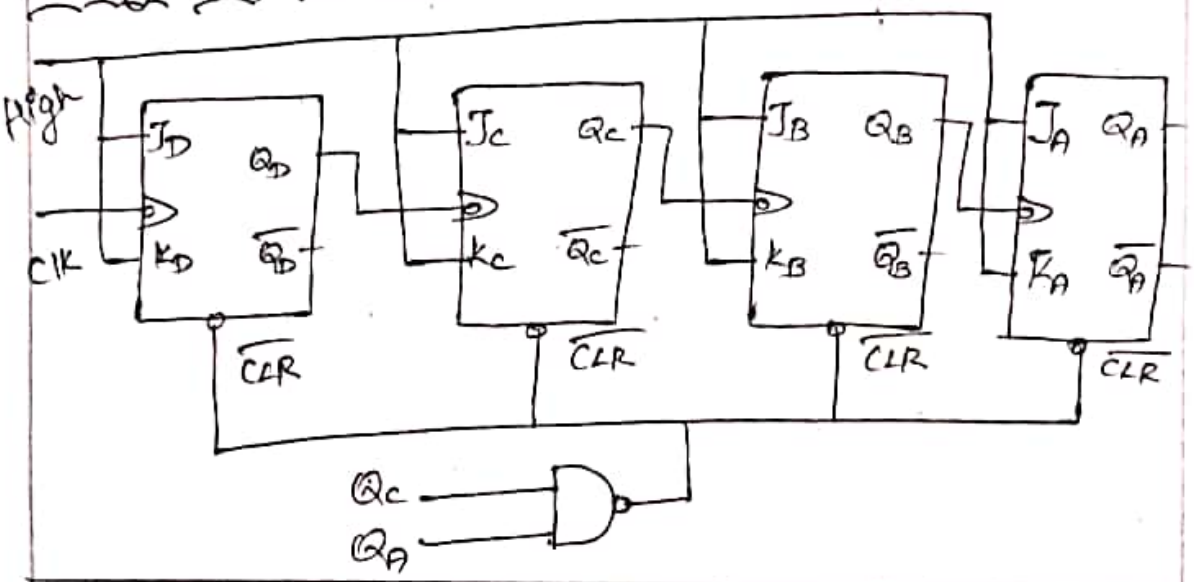
$10 \rightarrow 1\ 0\ 1\ 0$



Using D Flipflop :-



Using JK Flipflop :-



CASCADING RIPPLE COUNTER :-

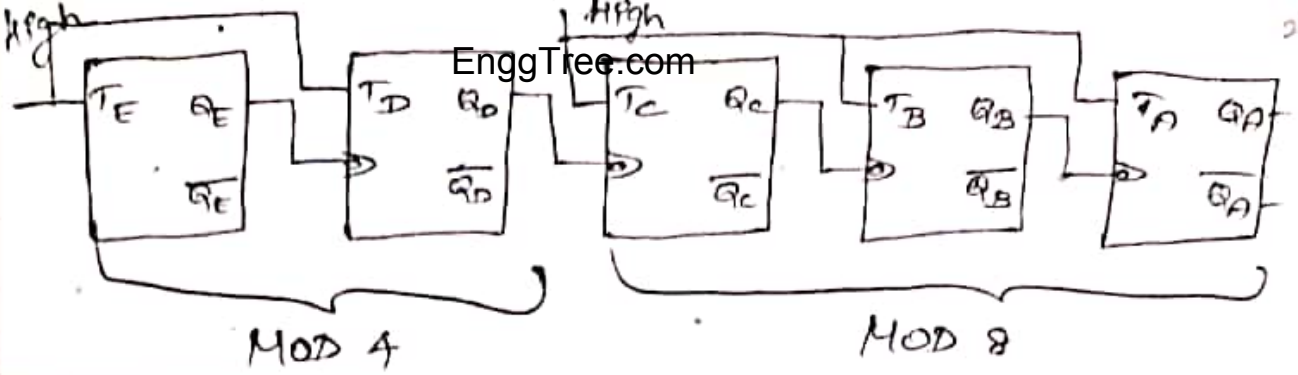
Ripple counters can be cascaded to increase the modulus of the counter.

A MOD M & MOD N counter is cascaded to give a MOD MN counter. While cascading, MSB of the first counter is connected to the clock of the second counter.

MOD 32 $\Rightarrow$ MOD 4 X MOD 8

A B C D E

mod 4 mod 8



DESIGN OF SYNCHRONOUS COUNTERS:-

(1) Sequential Counting → Up / Down

(2) Non-Sequential Counting (check-lock out condition).

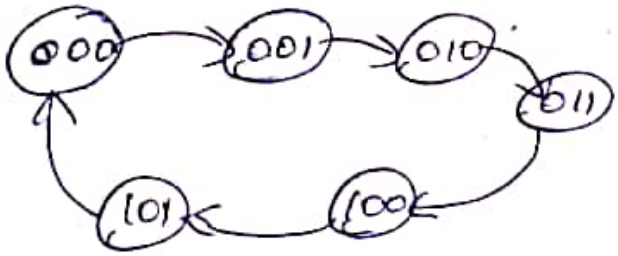
(1) Design Divide by 6 (or) Mod-6 Counter using
i) T FF ii) JK FF

Soln:- MOD-N $N \leq 2^n$

$6 \leq 2^3$ $n = 3$ 3FFs are required.

Let P.S be → A B C

Sequence → 0-1-2-3-4-5-0



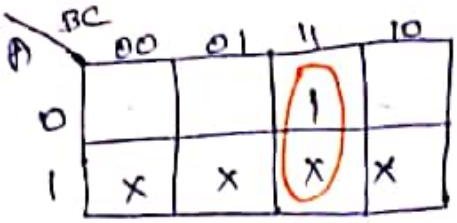
Unused states are → 6, 7
(i.e) 110, 111

STATE TABLE:-

PS			NS			FF i/ps						For T-FF		
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C	T_A	T_B	T_C
0	0	0	0	0	1	0	X	0	X	1	X	0	0	1
0	0	1	0	1	0	0	X	1	X	X	1	0	1	1
0	1	0	0	1	1	0	X	X	0	1	X	0	0	1
0	1	1	1	0	0	1	X	X	1	X	1	1	1	1
1	0	0	1	0	1	X	0	0	X	1	X	0	0	1
1	0	1	0	0	0	X	1	0	X	X	1	1	0	1
1	1	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X	X	X	X

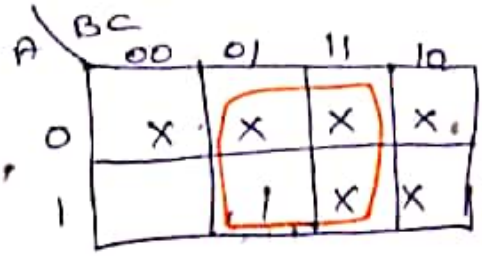
JK FF — T F-F

K-map for J_A:



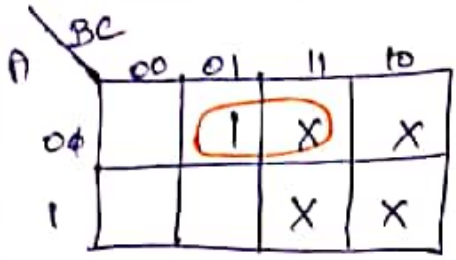
$J_A = BC$

K_A:



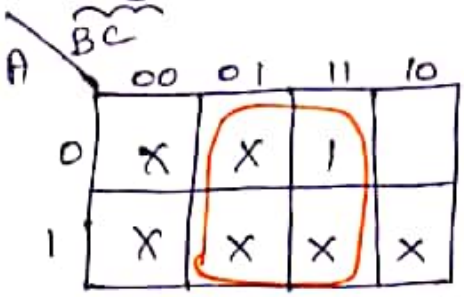
$K_A = C$

K-map for J_B:



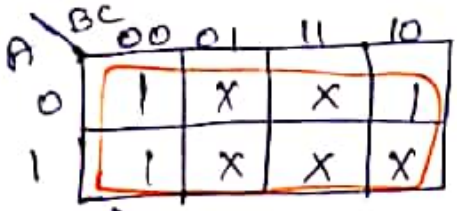
$J_B = \bar{A}C$

K_B:



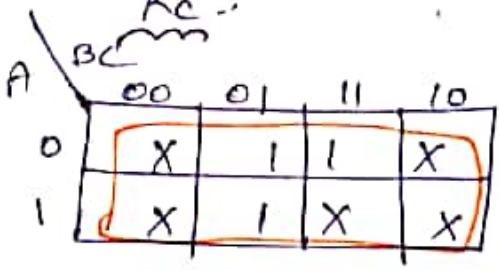
$K_B = C$

J_C:



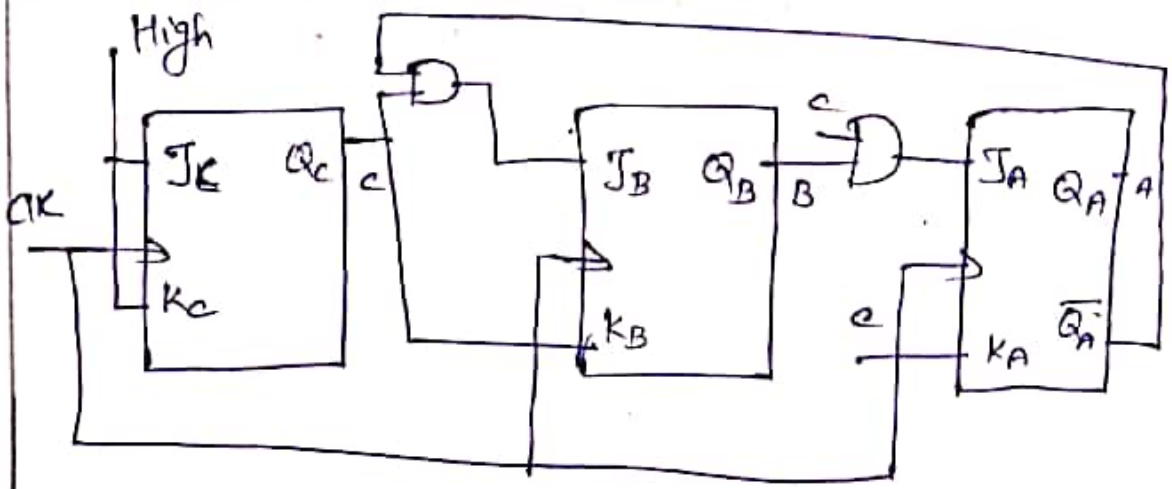
$J_C = 1$

K_C:



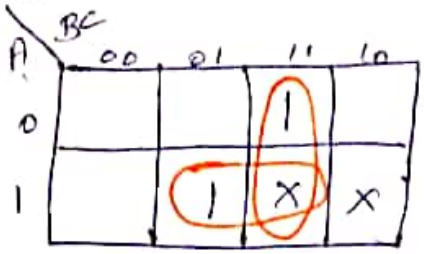
$K_C = 1$

c) Using JK Flipflop:-



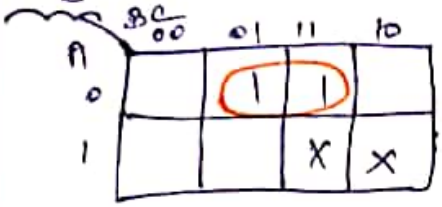
Using T-Flip flop:

K-map for T_A :-



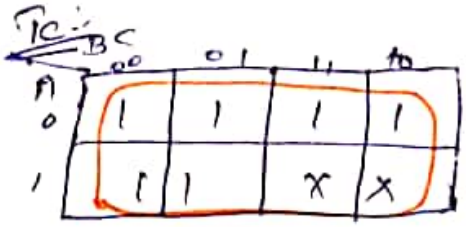
$$T_A = AC + \bar{B}C$$

T_B :-

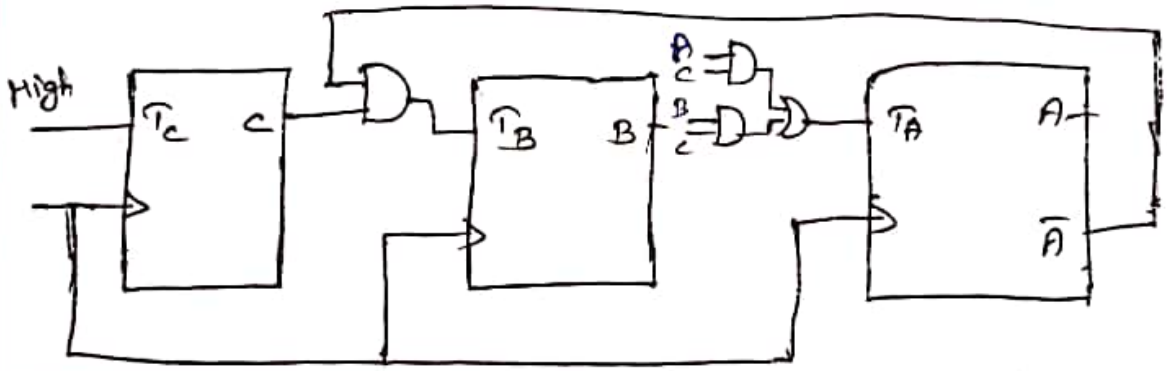


$$T_B = \bar{A}C$$

T_C :-



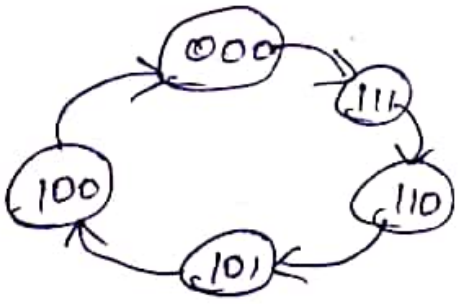
$$T_C = 1$$



2.) Design mod-5 down counter (or) Divide by 5 counter using T F.F:

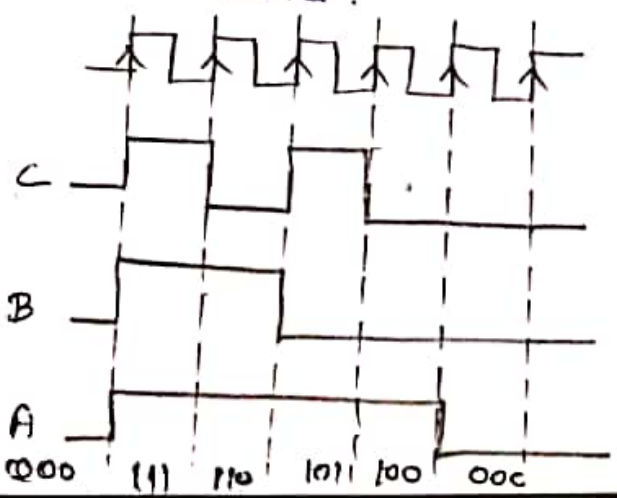
Soln:- $5 \leq 2^3$; $n = 3$ F.F are required (A B C)

Sequence → 0-7-6-5-4-0



Unused States:

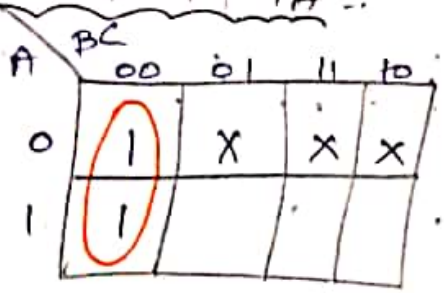
1, 2, 3.



State Table :-

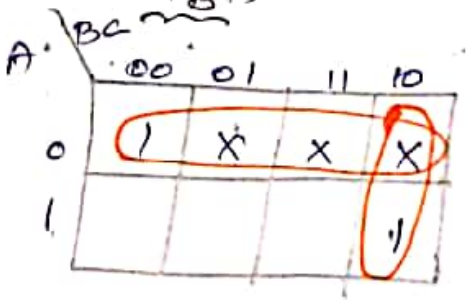
PS			NS			FF i/ps		
A	B	C	A	B	C	T_A	T_B	T_C
0	0	0	1	1	1	1	1	1
0	0	1	X	X	X	X	X	X
0	1	0	X	X	X	X	X	X
0	1	1	X	X	X	X	X	X
1	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1
1	1	0	1	0	1	0	1	1
1	1	1	1	1	0	0	0	1

K-map for T_A :-



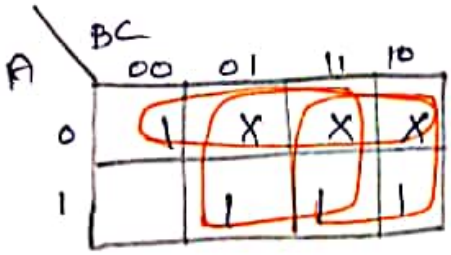
$T_A = \bar{B}\bar{C}$

T_B :-

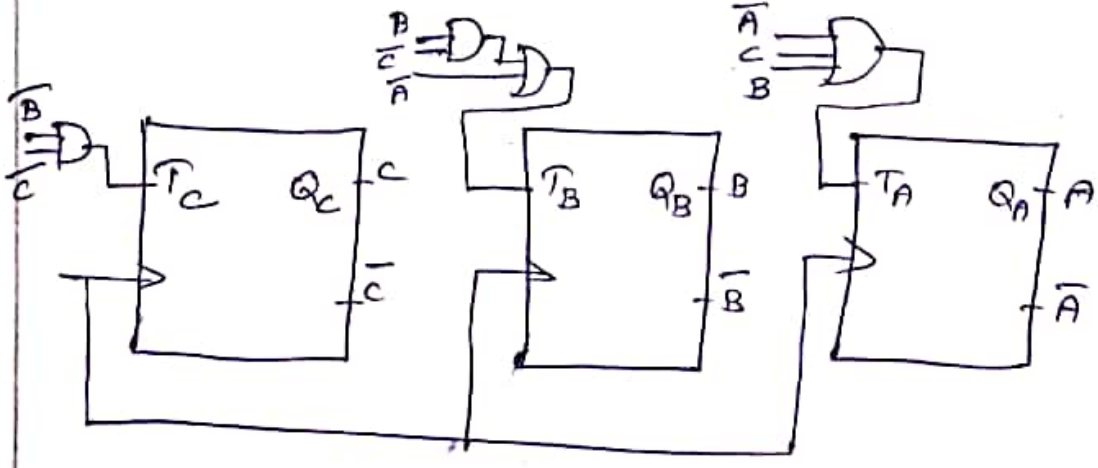


$T_B = \bar{A} + B\bar{C}$

T_C :-



$T_C = \bar{A} + C + B$



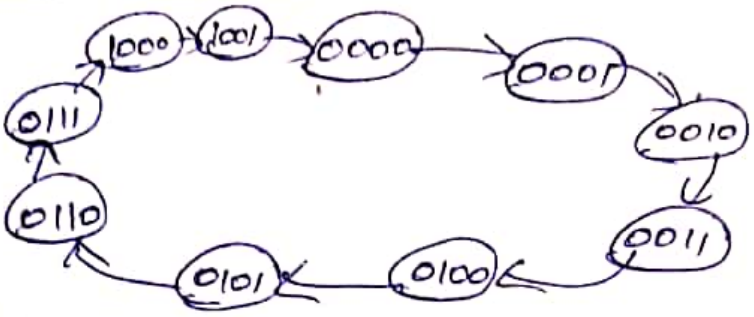
3.) Design of BCD/DECADE/MOD-10 Counter using JK FF.

Soln:

Sequence 0-1-2-3-4-5-6-7-8-9-0.

$N = 10$; $10 \leq 2^4$ $\quad$ P.S $\Rightarrow$ 4 (A, B, C, D).

State diagram:



State Table:

PS				NS				FF I/Ps							
A	B	C	D	A	B	C	D	J_A	K_A	J_B	K_B	J_C	K_C	J_D	K_D
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	0	0	0	0	X	1	0	X	0	X	X	1
1	0	1	0	X	X	X	X	X	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X

Kmap for J_A :-

AB \ CD	00	01	11	10
00				
01			1	
11	X	X	X	X
10	X	X	X	X

$J_A = BCD$

K_A :-

AB \ CD	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	X	X	X	X
10		1	X	X

$K_A = D$

Kmap for J_B :-

AB \ CD	00	01	11	10
00			1	
01	X	X	X	X
11	X	X	X	X
10			X	X

$J_B = CD$

K_B :-

AB \ CD	00	01	11	10
00	X	X	X	X
01			1	
11	X	X	X	X
10	X	X	X	X

$K_B = CD$

Kmap for J_C :-

AB \ CD	00	01	11	10
00		1	X	X
01		1	X	X
11	X	X	X	X
10			X	X

$J_C = \bar{A}D$

K_C :-

AB \ CD	00	01	11	10
00	X	X	1	
01	X	X	1	
11	X	X	X	X
10	X	X	X	X

$K_C = D$

Kmap for J_D :-

AB \ CD	00	01	11	10
00	1	X	X	1
01	1	X	X	1
11	X	X	X	X
10	1	X	X	X

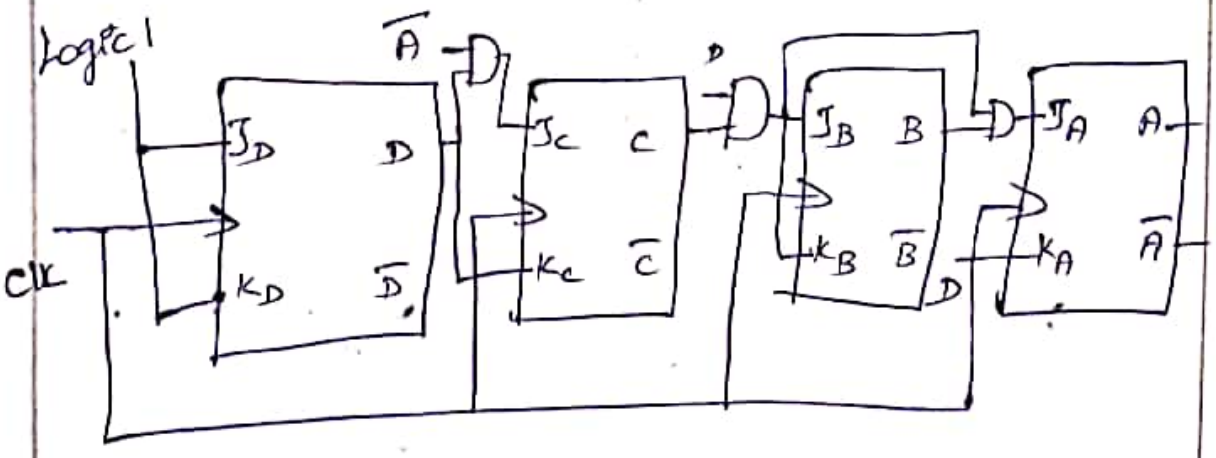
$J_D = 1$

K_D :-

AB \ CD	00	01	11	10
00	X	1	1	X
01	X	1	1	X
11	X	X	X	X
10	X	1	X	X

$K_D = 1$

Logic 1



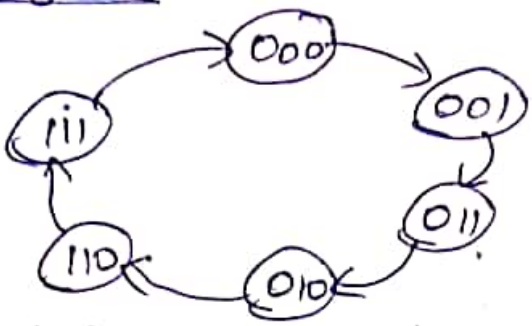
4.) Design of MOD-6 UNIT DISTANCE CODE / GRAY CODE: 9A
Seq. ckt using JK Flipflop.

Soln:- Unit distance code → Gray code.

∴ $N = 6$; $6 \le 2^3$ 3 F.F's are required

Let P.S be (A, B, C).

State Diagram:-



State Table:-

PS			NS			F.F Inputs					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	1	0	X	1	X	X	0
0	1	0	0	1	0	0	X	X	0	X	1
0	1	1	1	1	0	1	X	X	0	0	X
1	0	0	1	1	1	X	0	X	0	1	X
1	0	1	0	0	0	X	1	X	1	X	1
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X

Kmap for J_A:-

A \ BC	00	01	11	10
0				1
1	X	X	X	X

$J_A = B\bar{C}$

K_A:-

A \ BC	00	01	11	10
0	X	X	X	X
1	X	X	1	

$K_A = C$

J_B :-

A \ BC	00	01	11	10
0		1	X	X
1	X	X	X	X

$J_B = C$

K_B :-

A \ BC	00	01	11	10
0	X	X		
1	X	X	1	

$K_B = AC$

J_C:

A \ BC	00	01	11	10
0	1	X	X	
1	X	X	X	1

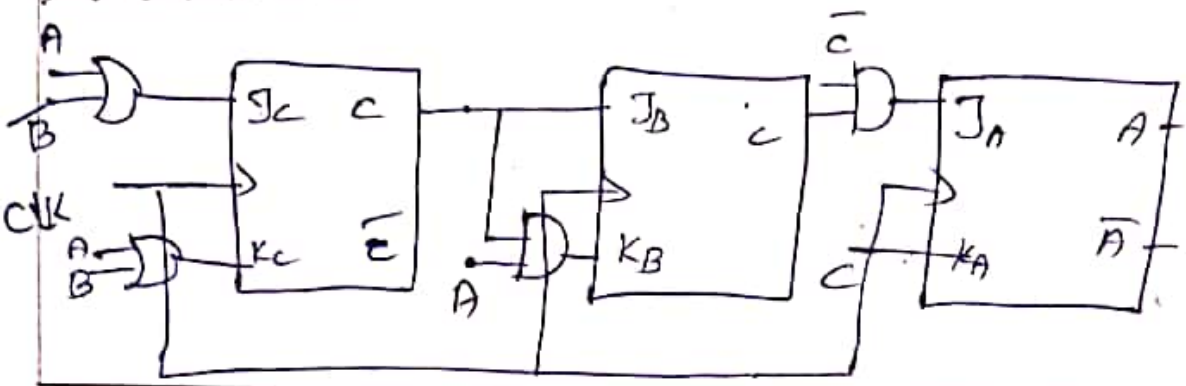
$J_C = A + \bar{B}$

K_C:

A \ BC	00	01	11	10
0	X		1	X
1	X	1	X	X

$K_C = A + B$

Implementation:



5.) Design of MOD-8 Bidirectional counter (or) Updown counter using JK flipflop.

Soln :- $N \leq 2^n$; $8 \leq 2^3$ 3 FF's are required.

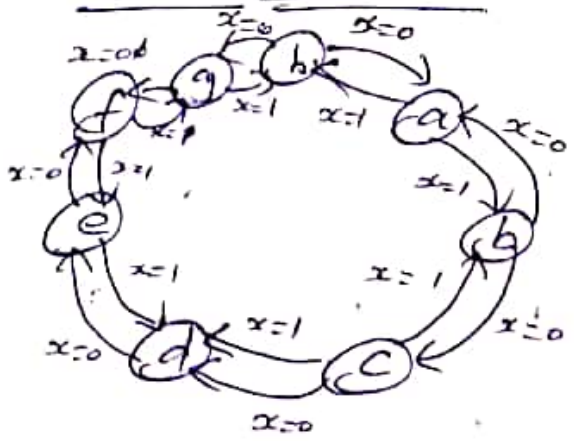
Let us assign $x=0 \rightarrow$ up counting

$x=1 \rightarrow$ down counting.

Let P.S be $\rightarrow A, B, C$.

Let a → 000
b → 001
c → 010
d → 011
e → 100
f → 101
g → 110
h → 111

State Diagram :-



F·F i/ps → J_A K_A
J_B K_B
J_C K_C

STATE TABLE:-

PS				NS			FF i/ps					
x	A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	0	1	0	X	0	X	1	X
0	0	0	1	0	1	0	0	X	1	X	X	1
0	0	1	0	0	1	1	0	X	X	0	1	X
0	0	1	1	1	0	0	1	X	X	1	X	1
0	1	0	0	1	0	1	X	0	0	X	1	X
0	1	0	1	1	1	0	X	0	1	X	X	1
0	1	1	0	1	1	1	X	0	X	0	1	X
0	1	1	1	0	0	0	X	1	X	1	X	1
1	0	0	0	1	1	1	1	X	1	X	1	X
1	0	0	1	1	1	0	1	X	1	X	X	1
1	0	1	0	1	0	1	1	X	X	1	1	X
1	0	1	1	1	0	0	1	X	X	1	X	1
1	1	0	0	0	1	1	X	1	1	X	1	X
1	1	0	1	0	1	0	X	1	1	X	X	1
1	1	1	0	0	0	1	X	1	X	1	1	X
1	1	1	1	0	0	0	X	1	X	1	X	1

Kmap for J_A :-

xA \ BC	00	01	11	10
00			1	
01	X	X	X	X
11	X	X	X	X
10	1	1	1	1

$J_A = x$

K_A :-

xA \ BC	00	01	11	10
00	X	X	X	X
01			1	
11	1	1	1	1
10	X	X	X	X

$K_A = A + Bc$

J_B :-

xA \ BC	00	01	11	10
00		1	X	X
01		1	X	X
11	1	1	X	X
10	1	1	X	X

$J_B = x + C$

K_B :-

xA \ BC	00	01	11	10
00	X	X	1	
01	X	X	1	
11	X	X	1	1
10	X	X	1	1

$K_B = x + C$

J_C

J_A \ BC	00	01	11	10
00	1	X	X	1
01	1	X	X	1
11	1	X	X	1
10	1	X	X	1

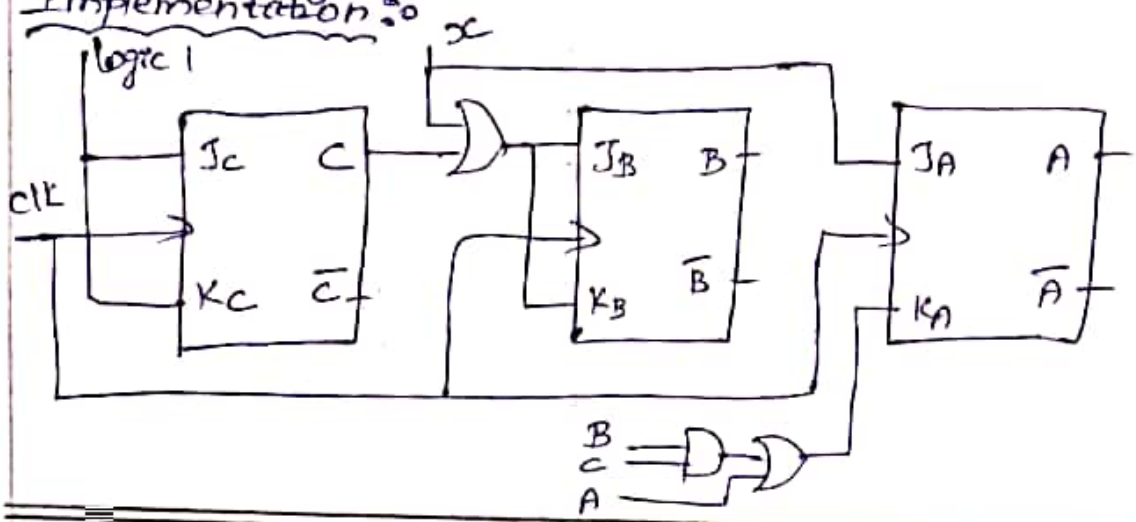
$J_C = 1$

K_C :

K_A \ BC	00	01	11	10
00	X	1	1	X
01	X	1	1	X
11	X	1	1	X
10	X	1	1	X

$K_C = 1$

Implementation :-



NON - SEQUENTIAL COUNTING :-

LOCK-OUT CONDITION :-

In a counter, if the next state of some unused state is again some unused state, it may happen that the counter remains in unused states never to arrive at a used state. Such a condition is called a lock-out condition.

To Avoid :-

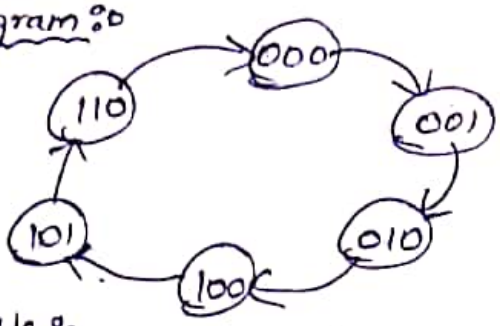
The counter should be provided with an additional logic circuitry. This will force all unused states into valid next state.

1.) Design a Synchronous Counter that counts the sequence 0,1,2,3,4,5,6,0 using JK flipflop.

Soln :- Max. Count = 6 ; $6 \leq 2^3$

∴ 3 FFs are required Let PS be A, B, C

State Diagram :-

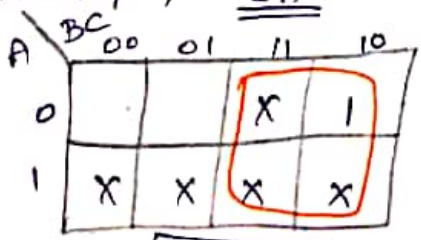


State Table :-

PS			NS			FF i/ps					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
0	1	1	X	X	X	X	X	X	X	X	X
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X
1	1	1	X	X	X	X	X	X	X	X	X

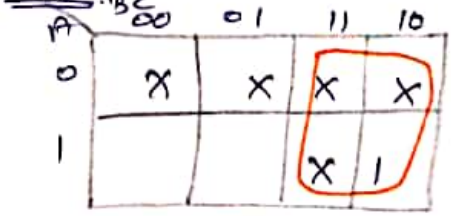
unused state → 3, 7.

K-map for J_A



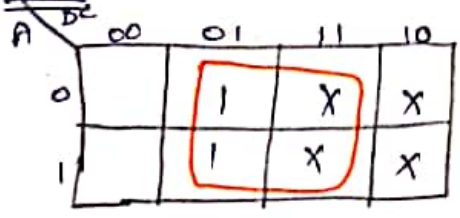
$J_A = B$

K_A :



$K_A = B$

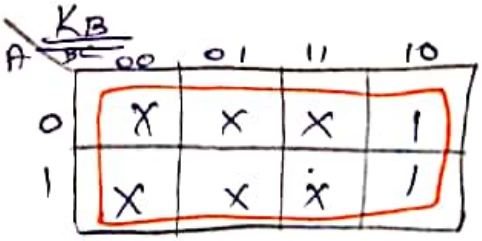
J_B



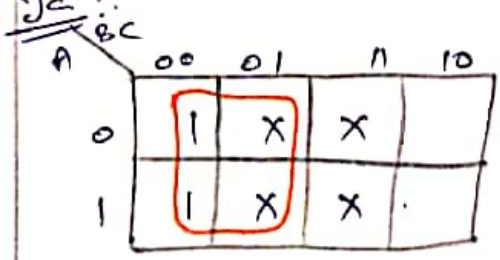
$J_B = C$

$K_B = 1$

K_B



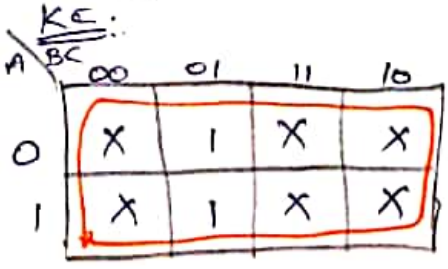
J_C :



$J_C = \overline{B}$

$K_C = 1$

K_C :

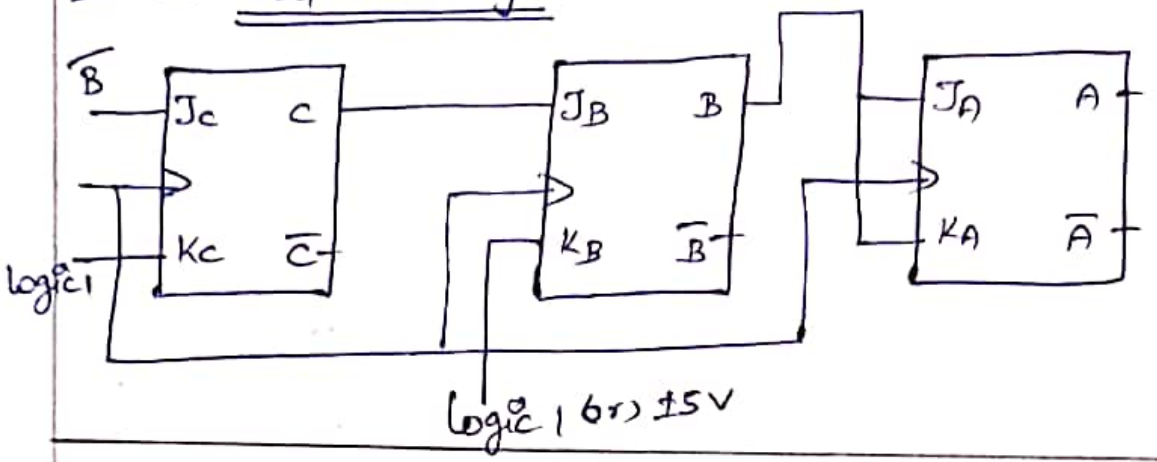


Checking self-starting / lock out condition:-

PS			FF i/ps						NS		
A	B	C	J_A	K_A	J_B	K_B	J_C	K_C	A	B	C
0	1	1	1	1	1	1	0	1	1	0	0 → 4
1	1	1	1	1	1	1	0	1	0	0	0 → 0

3 → 4 7 → 0

As the unused states enter into valid states. It is Self-Starting.



2.) Design a counter for the following sequence. Is the counter self-starting. 0-7-5-6-1-2-0.

Soln:- Max. counting, 7 $N \leq 2^n$

$7 \leq 2^3$; 3 F.F are required.

PS			NS			FF i/ps.					
A	B	C	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	1	1	1	1	X	1	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	0	0	0	X	X	1	0	X
0	1	1	X	X	X	X	X	X	X	X	X
1	0	0	X	X	X	X	X	X	X	X	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	1	X	1	X	1	1	X
1	1	1	1	0	1	X	0	X	1	X	0

K-map for J_A :

A \ BC	00	01	11	10
0	1		X	
1	X	X	X	X

$J_A = \bar{B}\bar{C}$

$K_A = \bar{C}$

K_A :

A \ BC	00	01	11	10
0	X	X	X	X
1	X			1

J_B :

A \ BC	00	01	11	10
0	1	1	X	X
1	X	1	X	X

$J_B = 1$

$K_B = 1$

K_B :

A \ BC	00	01	11	10
0	X	X	X	1
1	X	X	1	1

J_C :

A \ BC	00	01	11	10
0	1	X	X	
1	X	X	X	1

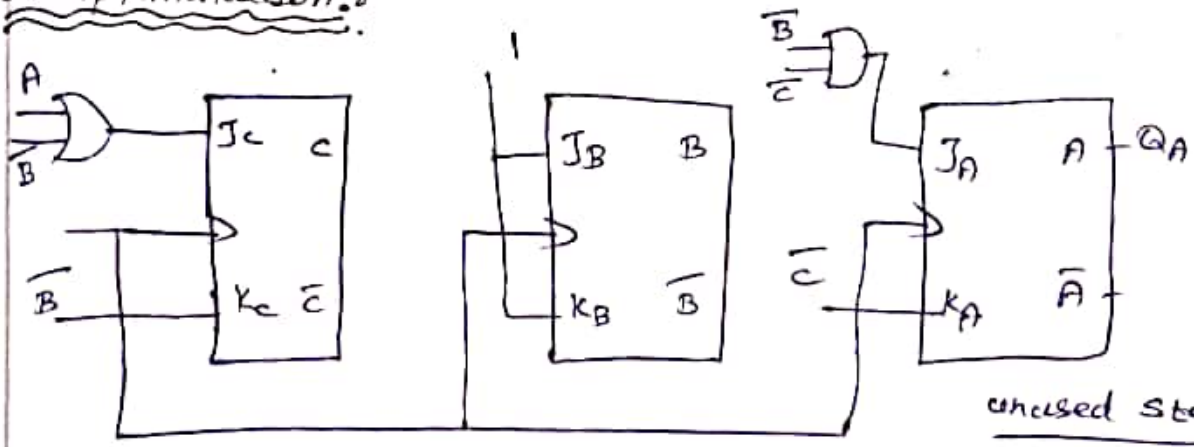
$J_C = A + \bar{B}$

$K_C = \bar{B}$

K_C :

A \ BC	00	01	11	10
0	X	1	X	X
1	X	1		X

Implementation :-



unused states

011 → 3

100 → 4

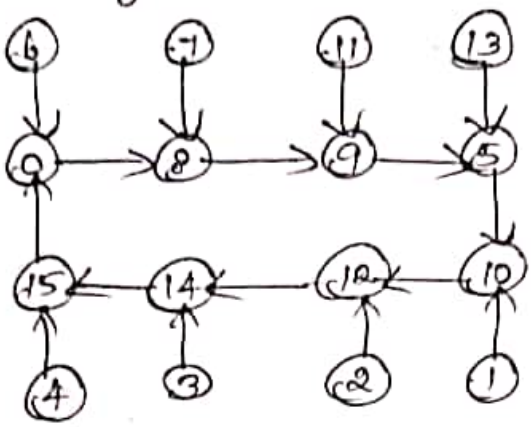
To check layout condition / self starting :-

Unused P.S			FF i/ps						NS		
A	B	C	J_A	K_A	J_B	K_B	J_C	K_C	A	B	C
0	1	1	0	0	1	1	0	0	0	0	1
1	0	0	1	1	1	1	1	1	0	1	1

3 → 1 4 → 3

So it is self starting.

3.) Design a counter which counts the following sequence using D-FF.



Soln:-

STATE TABLE:-

PS				NS				F.F			
A	B	C	D	A	B	C	D	D_A	D_B	D_C	D_D
0	0	0	0	1	0	0	0	1	0	0	0
0	0	0	1	1	0	1	0	1	0	1	0
0	0	1	0	1	1	0	0	1	1	0	0
0	0	1	1	1	1	1	0	1	1	1	0
0	1	0	0	1	1	1	1	1	1	1	1
0	1	0	1	1	0	1	0	1	0	1	0
0	1	1	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1	1	0	0	1
1	0	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	1	0	0	1	1	0	0
1	0	1	1	1	0	0	1	1	0	0	1
1	1	0	0	1	1	1	0	1	1	1	0
1	1	0	1	0	1	0	1	0	1	0	1
1	1	1	0	1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0

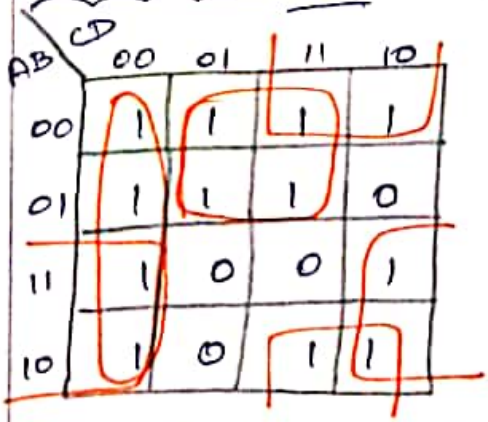
$D_A = A(t+1)$

$D_B = B(t+1)$

$D_C = C(t+1)$

$D_D = D(t+1)$

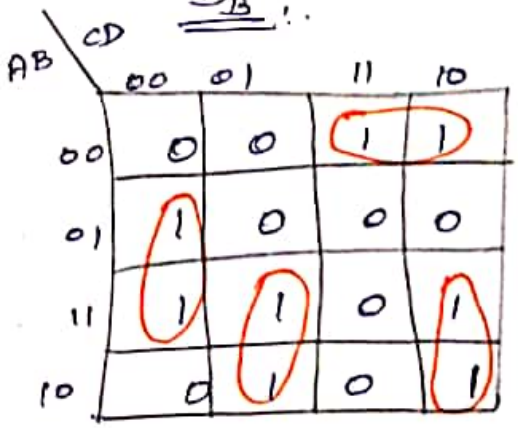
K-map for D_A:-



$D_A = \bar{C}\bar{D} + \bar{A}D + A\bar{D} + \bar{B}C$

$D_A = \bar{C}\bar{D} + \bar{B}C + (A \oplus D)$

D_B :-



$D_B = \bar{A}\bar{B}C + B\bar{C}\bar{D} + A\bar{C}D + AC\bar{D}$

$D_B = \bar{A}\bar{B}C + B\bar{C}\bar{D} + A(C \oplus D)$

D_C :-

AB \ CD	00	01	11	10
00	0	1	1	0
01	1	1	0	0
11	1	0	0	1
10	0	0	0	0

D_D :-

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	0	1	0	1
10	1	1	1	0

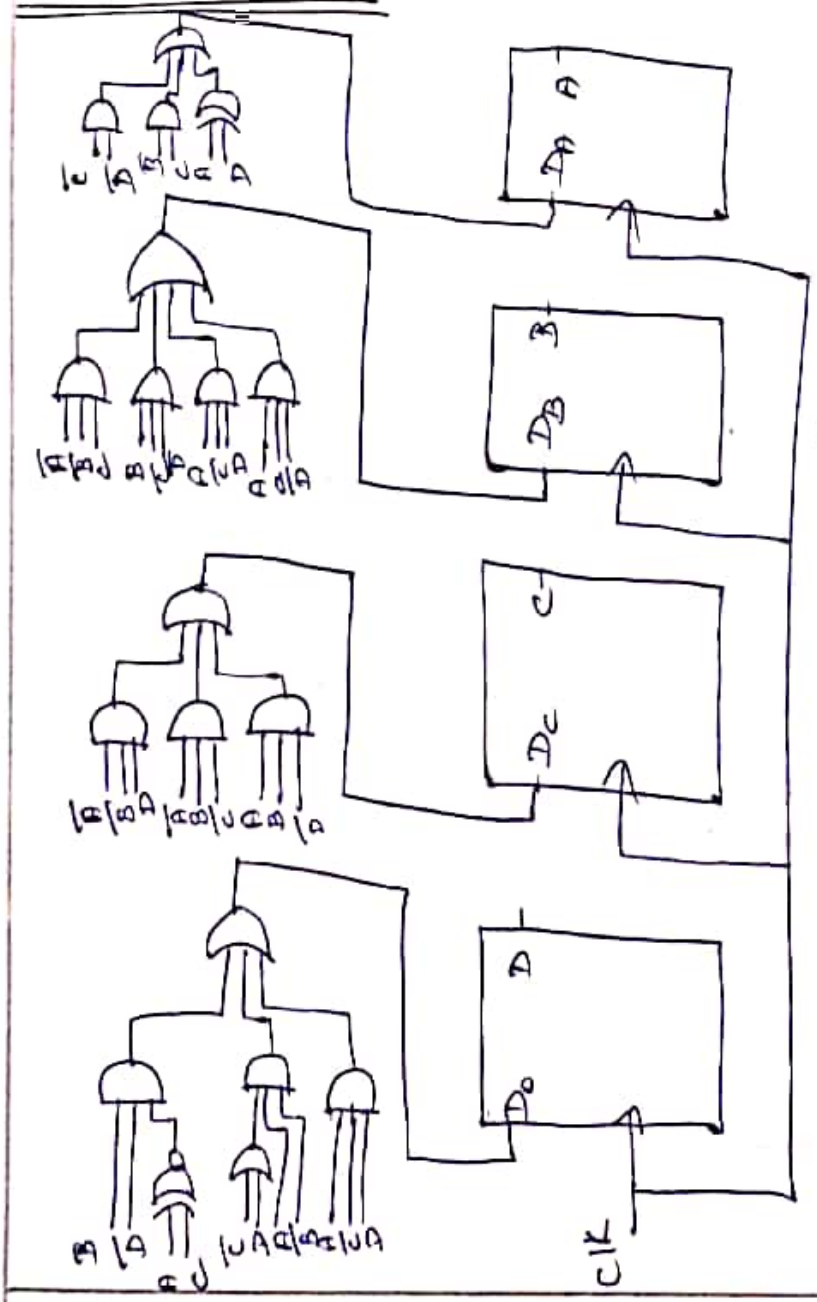
$$D_C = \bar{A}\bar{B}D + \bar{A}B\bar{C} + AB\bar{D}$$

$$D_D = \bar{A}B\bar{C}\bar{D} + ABC\bar{D} + A\bar{B}(\bar{C}+D) + A\bar{C}D$$

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

$$D_D = B\bar{D}(\overline{A \oplus C}) + A\bar{B}(\bar{C}+D) + A\bar{C}D$$

Implementation :-



REGISTERS :-

→ It is a group of F.F to store a binary number

→ A register capable of shifting binary information either to the right or to the left is called Shift Register.

→ This shift register permits the stored data to move from a particular location to some other location within a register.

Two methods of Shifting :-

1.) Serial Shifting :- (4 clk pulse required to shift a 4-bit data).

→ Shifting one bit at a time for each clock pulse, beginning with either MSB or LSB in a serial fashion.

2.) Parallel shifting :-

→ All the data get shifted simultaneously during a single clock pulse.

→ It is much faster than serial shifting.

Classification of Shift Registers :-

1. Serial-in Serial-out (SISO)
2. Serial-in Parallel-out (SIPO)
3. Parallel-in Serial-out (PISO)
4. Parallel-in Parallel-out (PIPO).

SERIAL-IN SERIAL-OUT :-

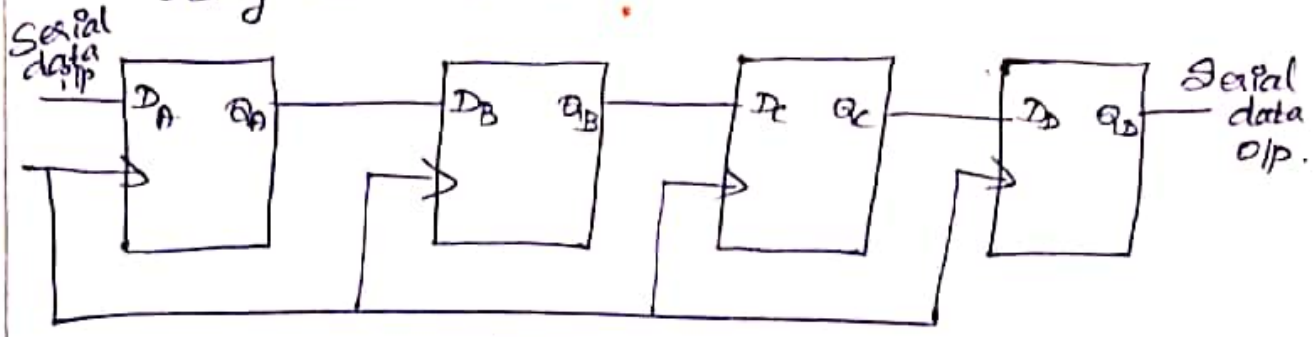
This type of Registers accept data serially (1) one bit at a time on a single I/p line.

Data can be shifted either right or left using

1. Shift Right Register
2. Shift Left Register.

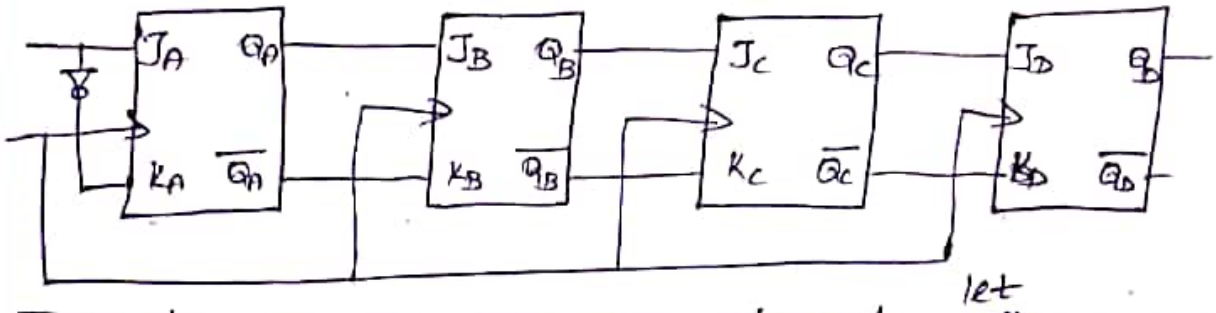
Shift Right Register

Using D F.F (4-bit Data).



(or)

Using JK F.F:



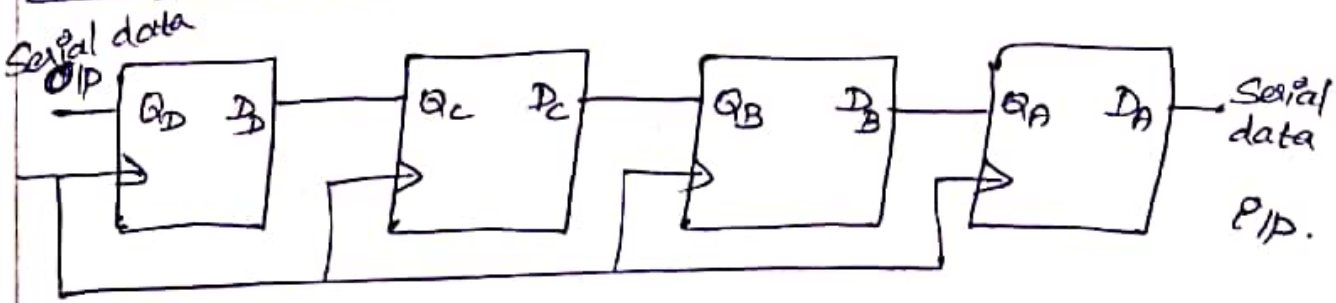
Initially all the F.Fs are cleared

let $D_{in} = 1101$

Data i/p	D_{in}	CLK	Q_A	Q_B	Q_C	Q_D
	1		0	0	0	0
	1	1	1	0	0	0
	0	2	1	1	0	0
	1	3	0	1	1	0
	0	4	1	0	1	1
		5	0	1	0	1
		6	0	0	1	0
		7	0	0	0	1

Data o/p.

Left Shift Registers



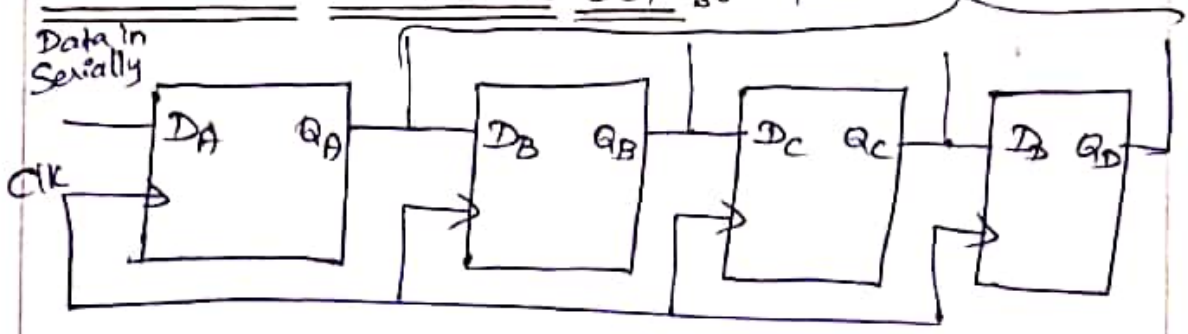
Initially all the F.Fs are cleared.

Q_D	Q_C	Q_B	Q_A	Clk	D_{in}
0	0	0	0		1
0	0	0	1	1	1
0	0	1	1	2	0
0	1	1	0	3	1
1	1	0	1	4	0
1	0	1	0	5	
0	1	0	0	6	
1	0	0	0	7	

Data O/P

We need 7 clock pulses to get serial o/p data with serial i/p.

SERIAL IN PARALLEL OUT :- parallel o/p data.

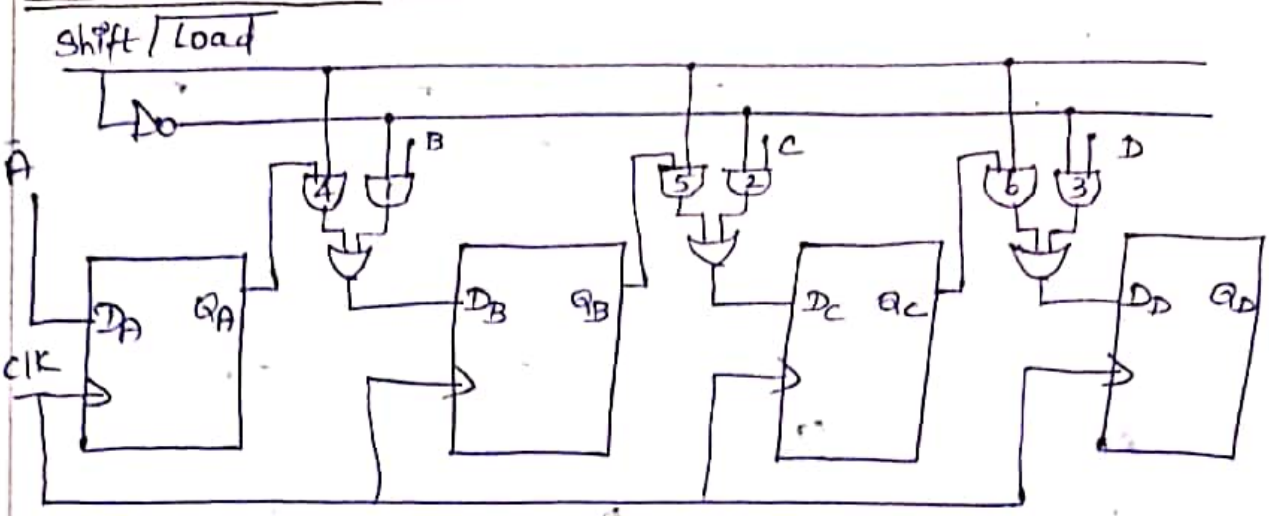


→ Data shifted in serial manner
→ O/p is taken in parallel manner / simultaneously.
→ 4 clk pulses needed to get the data.
→ Initially all the F.Fs are cleared.

D_{in}	Clk	Q_A	Q_B	Q_C	Q_D
1		0	0	0	0
1	1	1	0	0	0
0	2	1	1	0	0
1	3	0	1	1	0
	4	1	0	1	1

Parallel data O/P.

PARALLEL-IN SERIAL OUT :-



Shift / $\overline{Load}$ → 0 → It loads the data

→ The data will be available at i/p of all F.Fs

→ When clock pulse is given, data is simultaneously loaded in all F.Fs.

→ Gates 1, 2, 3 are enabled for loading.

Shift / $\overline{Load}$ → 1 → It performs the shifting operation.

→ Gates 4, 5, 6 are enabled.

→ For a 4-bit register, 4 clock pulses are required to get the data o/p in serial fashion.

Totally we required 4 clock pulses to get the o/p data for PISO.

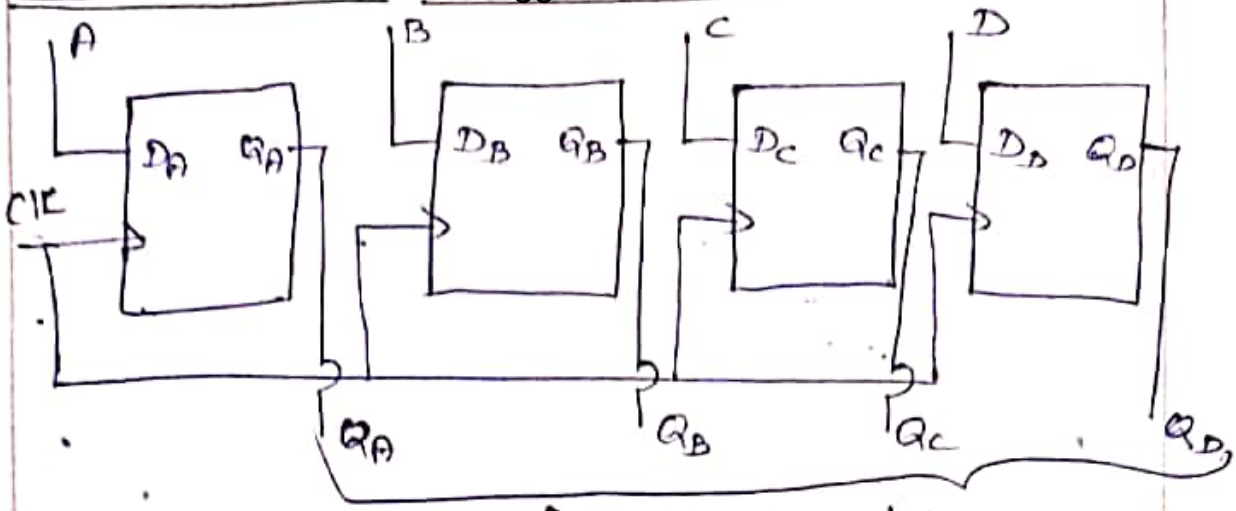
Shift / $\overline{Load}$	D_A	D_B	D_C	D_D
0	1	1	0	1

Shift / $\overline{Load}$ = 1

CLK	Q_A	Q_B	Q_C	Q_D
1	1	1	0	1
2	0	1	1	0
3	0	0	1	1
4	0	0	0	1

Data O/P.

PARALLEL-IN PARALLEL OUT :-



Parallel o/p data.

→ Single clock pulse is enough to get the o/p data simultaneously.

→ I/p data is also simultaneously applied.

UNIVERSAL SHIFT REGISTER :-

→ A register which is capable of shifting data both to the right and left is called bidirectional shift register.

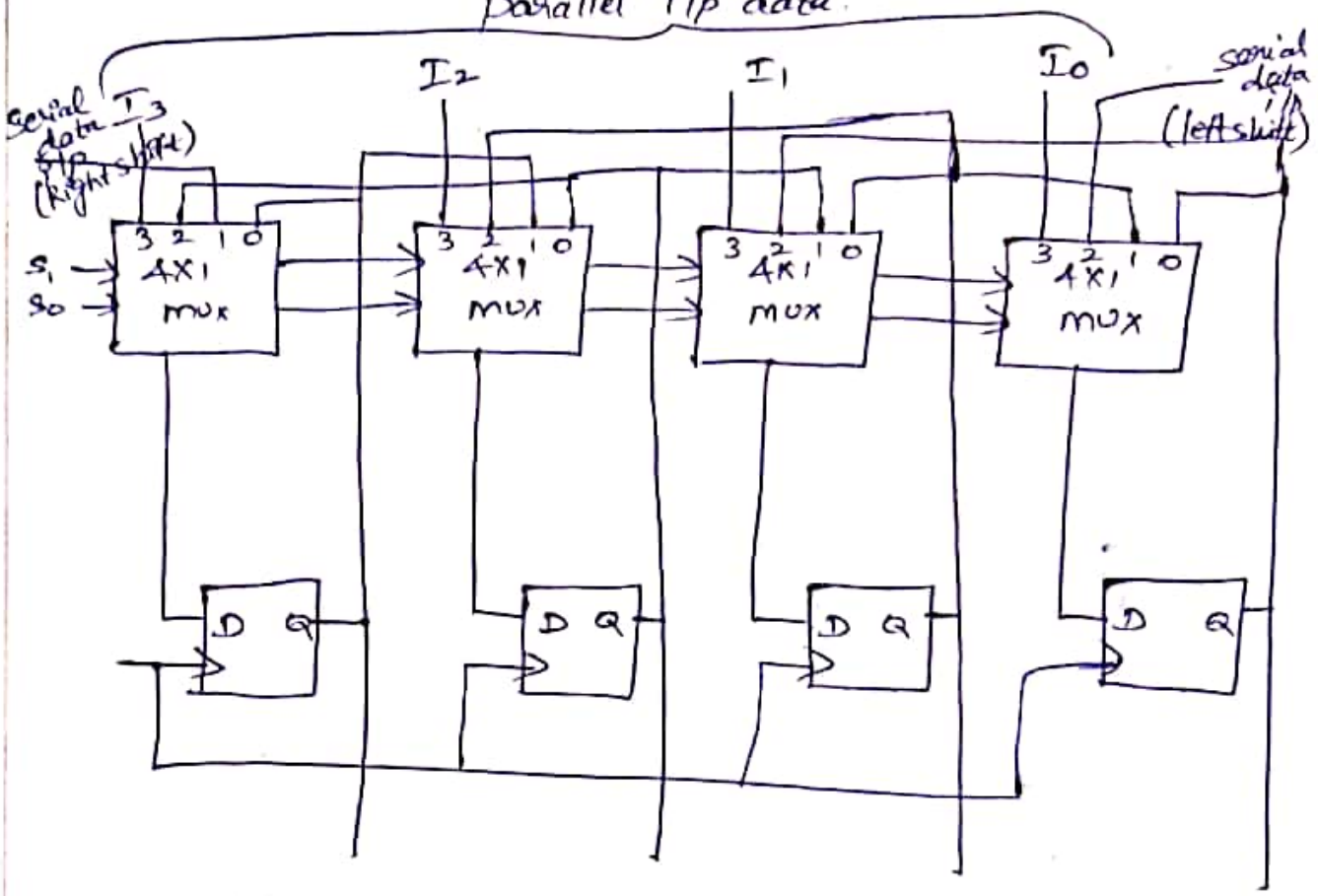
→ A register that can shift in only one direction is called unidirectional shift register.

→ If the register has shift and parallel load capabilities, it is called shift register with parallel load (or) Universal shift Register.

Mode Control S_1 S_0	Register Operation
0 0	No change.
0 1	Shift - Right
1 0	Shift - Left
1 1	Parallel load.

Universal Shift Register has 4 D.F.Fs which get I/p from a 4x1 mux. According to the selection line, it performs shifting and parallel loading.

Register operation for various possible Selection lines are listed above.



APPLICATION OF SHIFT REGISTERS :-

1.) Delay Time :-

A SISO shift registers can be used to introduce time delay, Δt in digital signals. The time delay can be given as,

$$\Delta t = N \times \frac{1}{f_c}$$

N → No. of stages (Flip-flops)

F_c → clock frequency.

The amount of delay is either controlled by No. of flipflops (or) clock frequency.

2.) Serial to Parallel conversion :-

Serial in parallel out shift registers are used.

3.) Parallel to Serial conversion :-

Parallel in serial out shift registers are used.

4.) Shift register counters

Used as counter. A shift register with the serial o/p connected back to the serial i/p is called shift register counter.

5.) Pseudo-Random Binary Sequence Generator (PRBS)

6.) Sequence Generator

7.) Sequence detector.

SHIFT REGISTER COUNTER :-

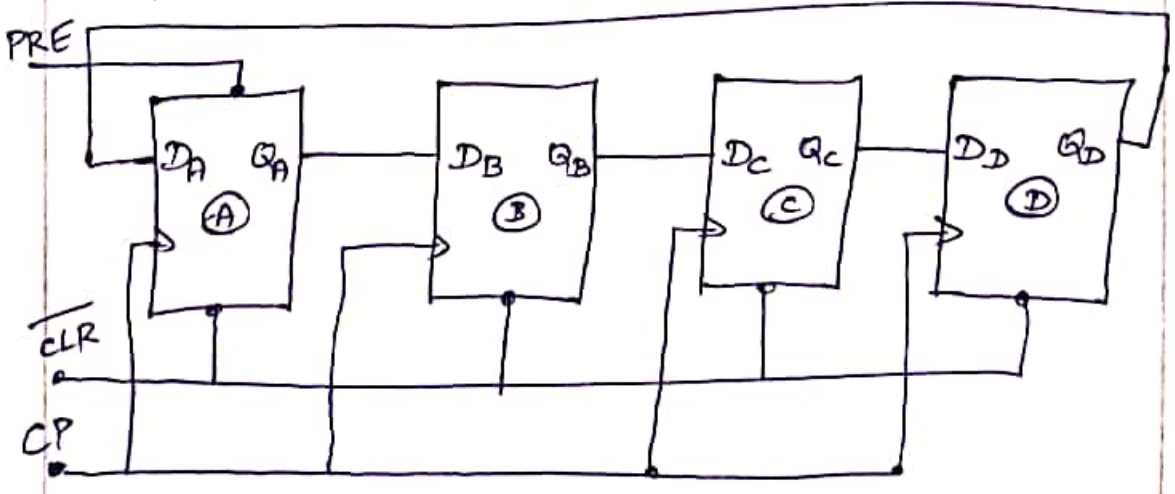
These registers are obtained from serial in serial out shift registers by providing feedback from last flipflop to the i/p of first F.F

Types: (1.) Simple Ring Counter (or) Ring Counter.

(2.) Twisted Ring counter / Johnson counter / switch-tail counter.

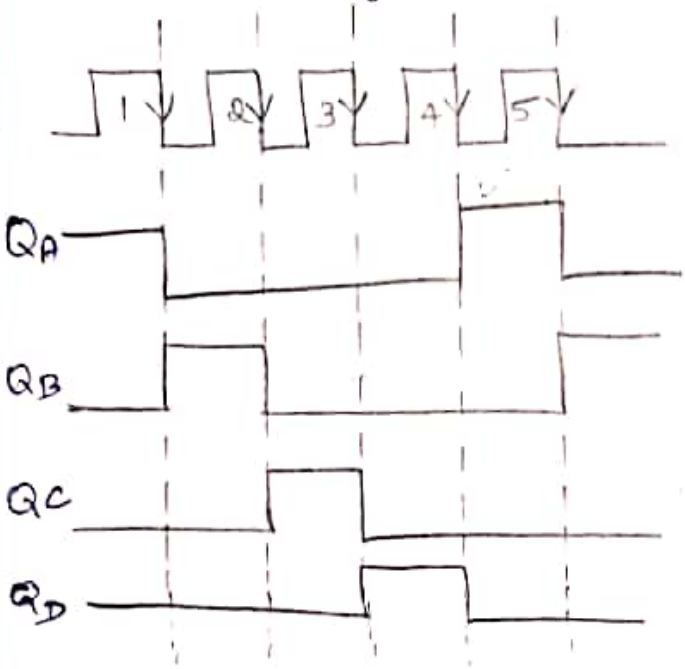
Ring Counters :-

Q o/p of each stage connected to D i/p of next stage and the o/p of last stage is fed back to the i/p of the first stage.



Clock Pulse	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

Initially all the F.Fs are cleared.



when $Q_A = 1$;

$D_B = 1$ & $Q_B Q_C Q_D = 0$.

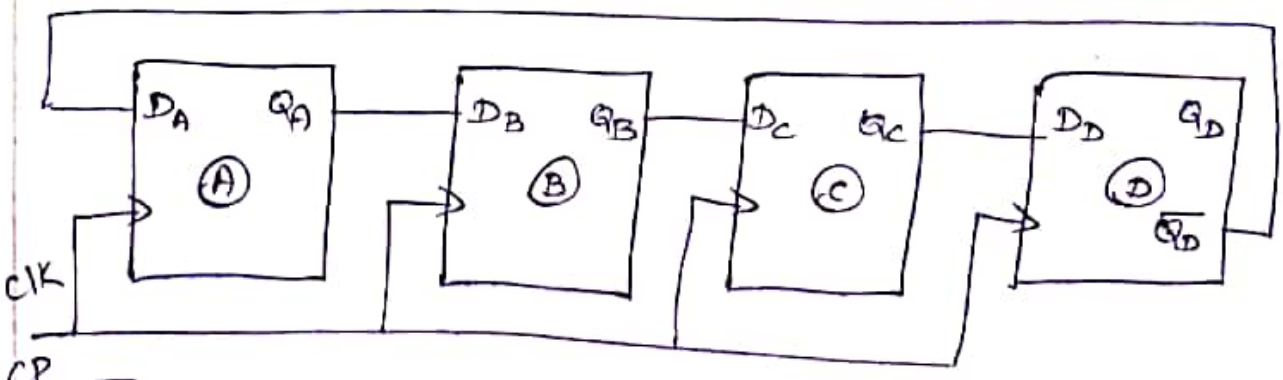
when $Q_B = 1$;

$Q_C = 0$; $Q_D = 0$; $Q_A = 0$.

It is always retained in the Counter and simply shifted "around the ring" advancing one stage for each clock pulse.

→ 4 stages of FFs are used.

Johnson (or) Twisted Ring (or) Switch Tail Counter :-



In Johnson counter, the Q o/p of each stage of FF is connected to the D i/p of next stage.

The single exception is that is complement o/p of last FF is connected back to the D i/p of the first FF.

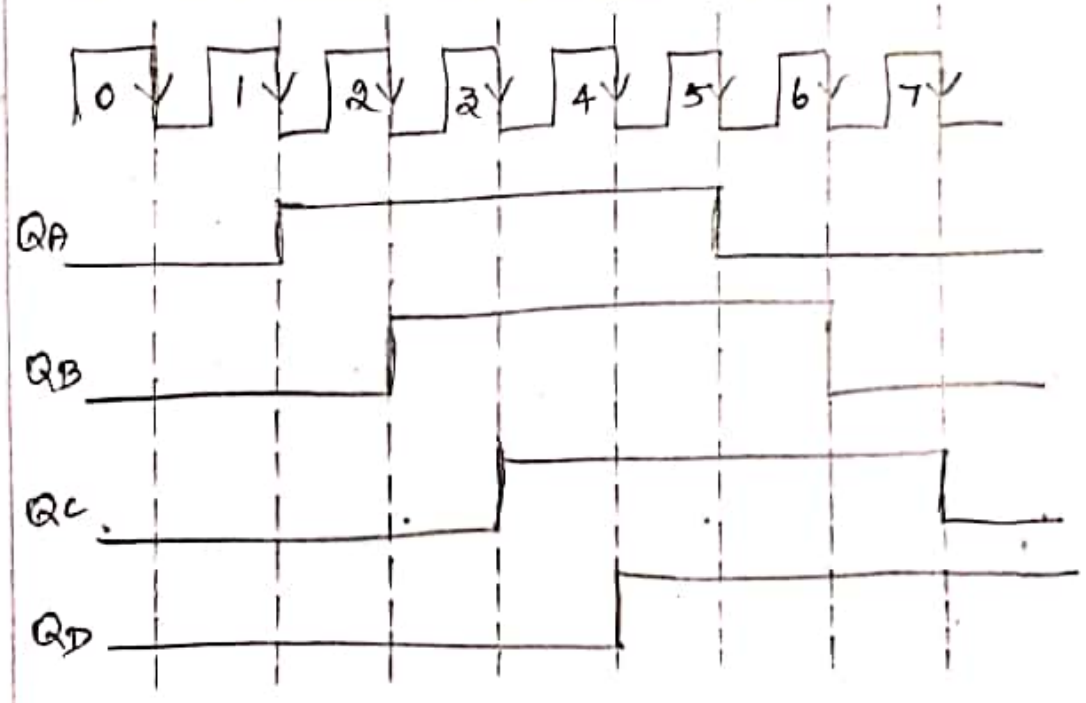
Johnson Counter implemented with SR or JK FF.

Clock Pulse	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

When $Q_D = 0$, $\overline{Q_D} = 1$, given to $D_A = 1$, $Q_A = 1$, $Q_B = Q_C = Q_D = 0$.

Similar for all,

After 8 states, the same sequence is repeated.



Advantages:-

1. It requires only half number of FF compared to standard ring counter.

Disadvantages:-

1. It require more FFs than binary counter.

SEQUENCE DETECTOR :-

A sequence detector accepts an input as a string of bits : either 0 or 1. Its output goes to 1 when a target sequence has been detected.

Types : 1.) Overlap
2.) Non - overlap.

Overlap :

The final bits of one sequence can be the start of another sequence.

(Eg): To detect the sequence 11011 for the I/P string

11011011011

11011 detector with overlap

X 1 1 0 1 1 0 1 1 0 1 1

Z 0 0 0 0 1 0 0 1 0 0 1

11011 detector without overlap

X 1 1 0 1 1 0 1 1 0 1 1

0 0 0 0 1 0 0 0 0 1

Without Overlap :-

The sequence detector with no overlap allowed resets itself to the start state when the sequence has been detected.

1.) Design a 11011 sequence detector using JK FF. Allow overlap.

Soln :-

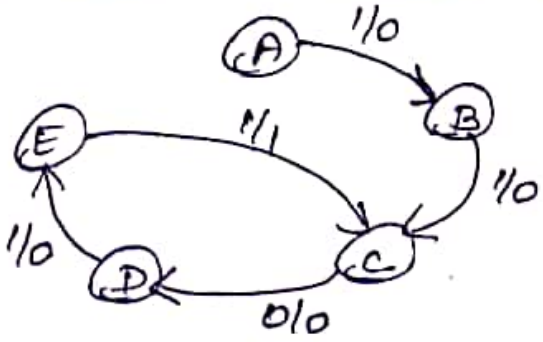
Step 1 :-

To detect a 5-bit sequence, we need 5 state variables. Let it be A, B, C, D, E.

State A → Initial State.

Step 2 :-

Do the transition for the expected sequence.



Sequence → 1 1 0 1 1

A B C D E

(1) If A detects 1 (part of the valid sequence) → moves to B. [A has 1]

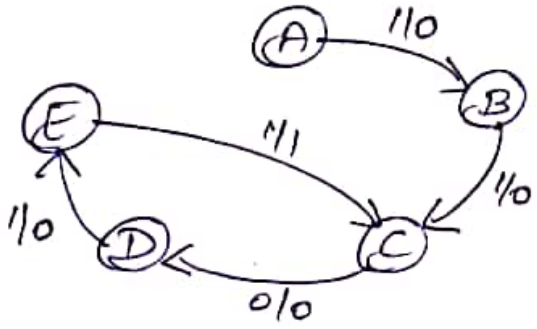
(2) If B detects 1 (second bit & Part of valid seq) → moves to C [B has 11]

(3) If C detects 0 (3rd bit & P.V.seq) → moves to D [C has 110]

(4) If D detects 1 (4th bit & PVS) → moves to E [D has 1101]

(5) If E detects 1 (5th bit & PVS), E has 11011, as last two bits are part of start of valid seq. As overlap is allowed, 2 bits (11) is available already. It has to detect 3rd bit. 3rd bit detector is C. So it moves to C.

Step 3: Insert the i/p that breaks the sequence:



11011

(1) If A receives 0, instead of 1, it is not valid sequence, it stays over there.

(2) If B detects 0, instead of 1, sequence → 10, it is not a valid seq, so it moves to A to detect Valid seq.

(3) Let B has 11 & C receives 1 instead 0, sequence is 111. Last two bits form the part of the valid seq. To detect 3rd bit, it has to move to 3rd bit detector. C is the 3rd bit detector. Hence it stays in C itself.

(4) Let C has 110, D receives 0 instead 1, seq → 1100, it is not a part of valid sequence. It moves to A again to detect the valid sequence.

(5) Let D has 1101, E receives 0, instead of 1, seq → 11010, it is not valid seq. Hence it moves to A.

State Table:

Present state	N.S		O/P	
	x=0	x=1	x=0	x=1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	E	0	0
E	A	C	0	1

We have five states, $N=5$; $N \leq 2^n \Rightarrow 5 \leq 2^3$

3 FFs are required.

	PS $y_1\ y_2\ y_3$	NS x=0 $y_1\ y_2\ y_3$	NS x=1 $y_1\ y_2\ y_3$	O/P z x=0	O/P z x=1
A →	0 0 0	0 0 0	0 0 1	0	0
B →	0 0 1	0 0 0	0 1 0	0	0
C →	0 1 0	0 1 1	0 1 0	0	0
D →	0 1 1	0 0 0	1 0 0	0	0
E →	1 0 0	0 0 0	0 1 0	0	1

PS x $y_1\ y_2\ y_3$	NS $y_1\ y_2\ y_3$	F.F Inputs $J_A\ K_A$	$J_B\ K_B$	$J_C\ K_C$	O/P z
0 0 0 0	0 0 0	0 X	0 X	0 X	0
0 0 0 1	0 0 0	0 X	0 X	X 1	0
0 0 1 0	0 1 1	0 X	X 0	1 X	0
0 0 1 1	0 0 0	0 X	X 1	X 1	0
0 1 0 0	0 0 0	X 1	0 X	0 X	0
0 1 0 1	X X X	X X	X X	X X	X
0 1 1 0	X X X	X X	X X	X X	X
0 1 1 1	X X X	X X	X X	X X	X
1 0 0 0	0 0 1	0 X	0 X	1 X	0
1 0 0 1	0 1 0	0 X	1 X	X 1	0
1 0 1 0	0 1 0	0 X	X 0	0 X	0
1 0 1 1	1 0 0	1 X	X 1	X 1	0
1 1 0 0	0 1 0	X 1	1 X	0 X	1
1 1 0 1	X X X	X X	X X	X X	X
1 1 1 0	X X X	X X	X X	X X	X
1 1 1 1	X X X	X X	X X	X X	X

K-map for J_A :

xy_1 \ y_2y_3	00	01	11	10
00	0	0	0	0
01	X	X	X	X
11	X	X	X	X
10	0	0	1	0

$$J_A = x y_2 y_3$$

K_A

xy_1 \ y_2y_3	00	01	11	10
00	X	X	X	X
01	1	X	X	X
11	1	X	X	X
10	X	X	X	X

$$K_A = 1$$

J_B

xy_1 \ y_2y_3	00	01	11	10
00	0	0	X	X
01	0	X	X	X
11	1	X	X	X
10	0	1	X	X

$$J_B = x y_1 + x y_3$$

K_B

xy_1 \ y_2y_3	00	01	11	10
00	X	X	1	0
01	X	X	X	X
11	X	X	X	X
10	X	X	1	0

$$K_B = y_3$$

J_C

xy_1 \ y_2y_3	00	01	11	10
00	0	X	X	1
01	0	X	X	X
11	0	X	X	X
10	1	X	X	0

$$J_C = \bar{x} y_2 + x \bar{y}_1 \bar{y}_2$$

K_C :

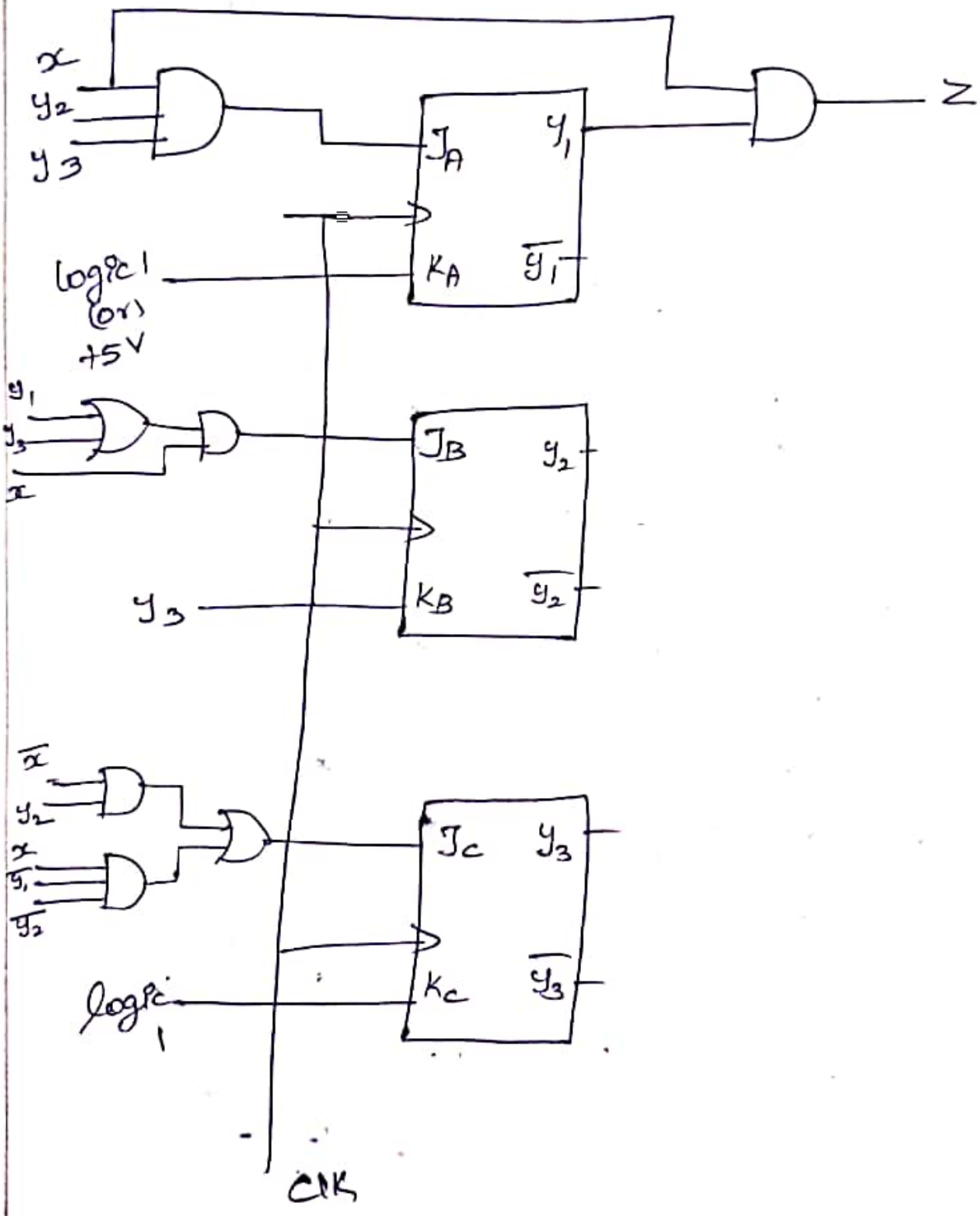
xy_1 \ y_2y_3	00	01	11	10
00	X	1	1	X
01	X	X	X	X
11	X	X	X	X
10	X	1	X	1

$$K_C = 1$$

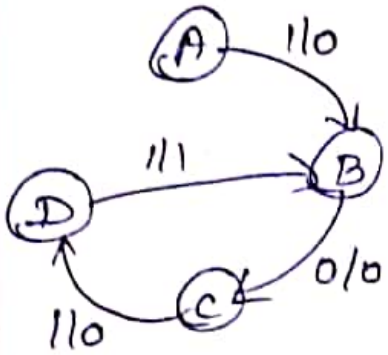
O/P, Z

xy_1 \ y_2y_3	00	01	11	10
00	0	0	0	0
01	0	X	X	X
11	1	X	X	X
10	0	0	0	0

$Z = xy_1$



(Eg): 1011 Sequence detector with Overlap:



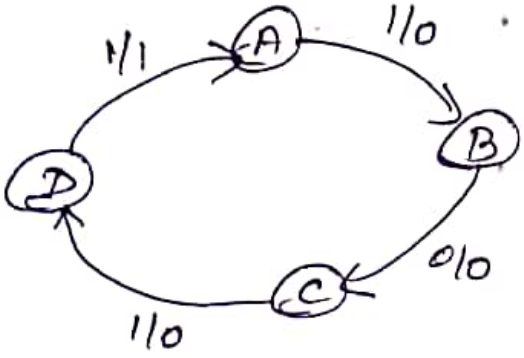
① If A → 0

② Let A=1, B→1; seq→11 (part of v-seq)

③ Let seq→10, C receives 0 instead 1.
seq→100, not a valid

④ Let seq→101, D receives 0 instead of 1, seq→1010 Part of valid seq. Hence moves to C.

1011 → Sequence Detector without Overlap:



① If A → detects 0, holds there

② seq 1, B receives 1, instead 0,
seq→11 part of valid seq.

③ seq→10, C receives 0 instead 1,
seq→100, invalid seq, C→A

④ Seq→101, D receives 0 instead 1,
seq→1010, invalid sequence, D→A

UNIT- III

COMPUTER FUNDAMENTALS

Introduction :-

Computer architecture acts as the interface between the hardware and the lowest level of software.

Computer Architecture refers to

* Attributes of a system visible to programmers like data types of variables.
* Attributes that have a direct impact on the execution of programs like clock cycle.

Computer Architecture is defined as study of the structure, behaviour and design of computers.

Computer Organisation :-

It refers to the operational units and their interconnections that realize the architecture specifications. It describes the function of and design of the various units of digital computers that store and process information.

The attributes in computer organisation refers to

→ Control signals, memory technology & peripheral interface
→ Data representation
→ Ilo Mechanisms
→ Addressing Techniques.

Computer Architecture :-

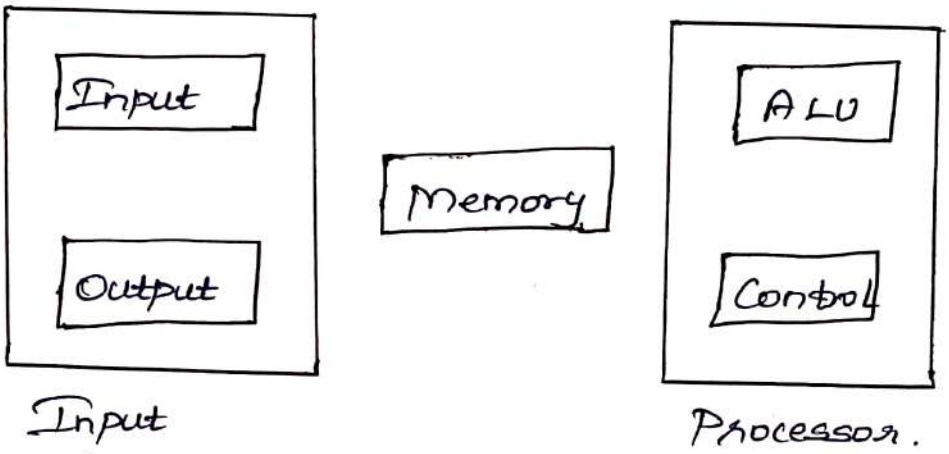
It is concerned with the structure and behaviour of the computer. It includes the information formats, the instruction set and techniques for addressing memory.

Architecture → Hidden to programmers

Organization → Implemented features in the system.

Basics of a computer system :-

↳ Von Neumann Architecture :-



Input

Processor.

Input Unit :-

- Computer accepts the coded information through input unit.
- Input devices accept data and instructions from the user or from another computer system (such as a computer on the Internet).
- The most common input device is the keyboard, which accepts letters, numbers and commands from the user.
- Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over cable to the memory of the computer.
- Another important type of input device is the mouse which lets you select options from on-screen menus.
- A mouse by moving it across a flat surface and pressing its button.

Memory Unit :-

→ Memory unit is used to store programs and data.

→ Memory is classified into primary and secondary storage.

Primary storage :-

→ It is also called main memory.

→ It operates at high speed and it is expensive.

→ It is made up of large number of semiconductor cells, each capable of storing one bit information.

→ These cells are grouped together in a fixed size called word. This facilitates reading and writing the content of one word.

→ Each word is associated with a distinct address that identifies word location. A given word is accessed by specifying its address.

Word length :-

→ The number of bits in each word is called word length of the computer.

→ The word lengths ranges from 16 to 64 bits. Programs must reside in the primary memory during execution.

RAM :- [Random Access Memory]

→ Memory in which any location can be reached in a short and fixed amount of time by specifying its address is called random-access memory.

Memory Access time :-

→ Time required to access one word is called memory access time.

→ This time is fixed and independent of the word being accessed.

→ It typically ranges from few nano seconds (ns) to about 100ns.

Caches :-

→ They are small and fast RAM units.
→ They are tightly coupled with the processor.
→ They are often contained on the same integrated circuits (IC) chip to achieve high performance

Secondary stage :-

→ Secondary memory is used 't' when large volumes of data programs have to be stored.
→ It is cheaper than primary memory and it's capacity is high.
→ Various secondary devices are magnetic tapes and disks, optical disks (CD-Roms) floppy etc....

Arithmetic and logic Unit :-

→ Most computer operations are executed in ALU.
→ The arithmetic logic section performs arithmetic operations such as addition, subtraction, multiplication and division.
→ A single processor to control a number of external devices such as keyboards, displays, magnetic and optical disks, sensors and mechanical controllers.

Control Unit :-

→ Control unit co-ordinates the operation of memory, arithmetic and logic unit, input unit and output unit in some proper way.
→ Control unit sends control signals to other units and senses their states.

→ The actual timing signals that govern the transfer are generated by the control circuits.

→ The data transfer between the processor and memory are also controlled by the control unit through timing signals.

UNIPROCESSORS TO MULTIPROCESSORS :-

The performance of the computer has drastically increased when the technology has drifted from uniprocessor systems to multiprocessor systems.

* As the core computing units were made more powerful, the performance of the processors also increased significantly.

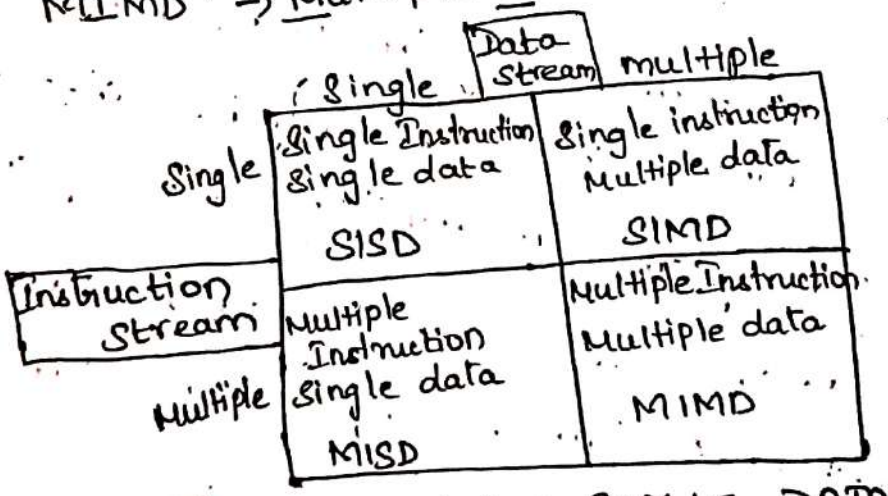
↳ uniprocessor system is a type of architecture that is based on a single computing unit. All the operations were done sequentially on the same unit. Multiprocessor systems are based on executing

instructions on multiple computing units.

* The multiprocessor architecture is based on Flynn taxonomy

"Flynn's taxonomy is a classification of parallel computer architecture that are based on the number of concurrent instruction and data streams available in the architecture".

SISD → Single Instruction, Single data
MISD → Multiple Instruction, Single data
SIMD → Single Instruction Multiple data
MIMD → Multiple Instruction Multiple data



SINGLE INSTRUCTION AND SINGLE DATA :-

* This is a uniprocessor machine which is capable of executing a single instruction operating on a single data stream.

* The machine instructions are processed in a sequential manner and computers adopting this model are popularly called Sequential computers.

* mast conventional computers have SISD architecture

* All the instructions and data to be processed have to be stored in primary memory.

* The speed of the processing element in the SISD model is limited by the rate at which the computer can transfer information internally.

MULTIPLE INSTRUCTION, SINGLE DATA (MISD)

* An MISD computing system is a multiprocessor machine capable of executing different instruction on different processing elements but all of them operating on the same dataset.

SINGLE INSTRUCTION, MULTIPLE DATA (SIMD)

* This machine capable of executing the same instructio on all the CPUs but operating on different data streams.

* machines based on an SIMD model are well suited to scientific computing since they involve lots of vector and Matrix operations. So that the information can be passed to all the processing Element (PEs) organized data elements of vectors can be divided into multiple sets and each PE can process one data set.

MULTIPLE INSTRUCTION, MULTIPLE DATA (MIMD)

*This is capable of executing Multiple Instructions on multiple data sets.

* Each PE in the MIMD model has separate instructions and data streams; therefore machine built using this model are capable to any kind of application

* unlike SIMD and MISD Machine, PEs in MIMD Machines work asynchronously.

INSTRUCTIONS :-

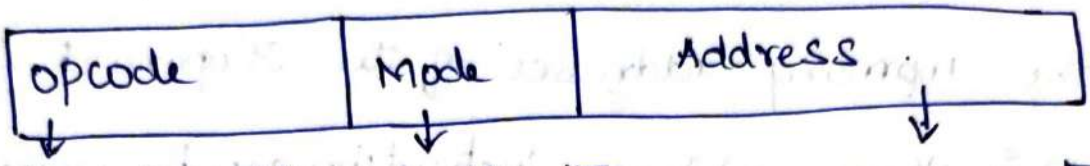
"An Instruction is a binary code, which Specifies a basic operation for the computer".

* A computer hardware must speak its language. The words of a computer language is called instructions. and its vocabulary is called an Instruction Set.

Operation code [OP-code] defines the operation type. operands define the operation of source and destinations.

Instruction Set Architecture [ISA] :- describes the processor in terms of what the assembly

language programmer sees ; ie; the instructions and registers.



Field that specifies the operation to be performed.

specifies the way the operant or the effective address is determined

designates a memory address or a processor register.

Types of operations :-

* Data transfer between the memory and the processor registers.
* Arithmetic and logic operations on data.
* program sequencing and control.
* I/o transfer.

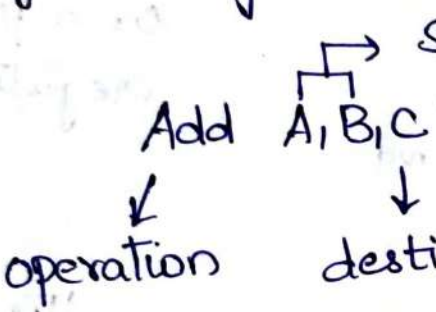
Performing a basic instruction is represented in many ways : They are

- 3-address instruction
- 2-address instruction
- 1- address instruction
- 0 - address instruction.

3- ADDRESS INSTRUCTION :-

Let us first assume that this action is to be

accomplished by a single machine instruction. Futhermore, assume that this instructions Contains the memory addresses of the 3 operands - A, B and C. This three address instruction can be represented Symbolically as.



The general Instruction format will be of this type

operation Destination Source 1, Source 2.

2-address Instruction :-

which performs the operation $B \leftarrow [A] + [B]$. when the sum is calculated the result is sent to the memory and stored in location B, replacing the original content of this location. This means that operand B is both a Source and a destination.

1 Address Instruction

Add A

* means → add the contents of memory location A to the contents of the accumulator register and place the sum back into the accumulator.

↳ Load A.
↳ and
↳ Store A.

Zero address Instruction :-

In these Instructions, the locations of all operands are defined implicitly. Such instruction are found in machines that store operands in a structure are called a pushdown stack.

Logical Instructions :-

AND, OR, XOR, Shift

ARITHMETIC INSTRUCTIONS :-

* Data types
* Integers : unsigned, Signed, Byte, short, long.
* Real numbers : Single precision (float), Double precision (double) operations
* Addition, subtraction, Multiplication, Devision.

DATA TRASFER INSTRUCTIONS :-

* Register transfer : Move
* memory transfer : Load, store
* I/o transfer : In, out
* control transfer instructions.
* unconditional branch.
* conditional brach
* procedure call
* Return.

OPERATIONS AND OPERANDS.

OPERATIONS OF THE COMPUTER HARDWARE :-

* Every computer must be able to perform arithmetic. The MIPS assessble to assembly language notation

```
add a, b, c
```

Instructs a computer to add the two variables b and c and to put their sum in 'a'.

Each MIPS arithmetic instruction performs only one operation and must always have exactly three variables.

* For example suppose want to place the sum of four variables b, c, d and e into variable 'a

The following sequence of instructions adds the four variables:

add a, b, c : The sum of 'b' and 'c' is placed in 'a'

add a, a, d : The sum of 'b', 'c' and 'd' is now in 'a'

add a, a, e : The sum of b, c, d and e is now in 'a'

Thus, it takes three instructions to sum the the four variables. The words to the right of the Sharp symbol (;) on each line above are comments for the human reader and the computer ignores them.

OPERANDS :-

↳ operand is a variable used to perform any kind of operations.

↳ operand must be from a limited number of Special locations, build directly in hardware called register.

↳ Registers are primitives used in a hardware design that also visible to the programmer when the computer is completed.

↳ MIPS architecture has 32 bits registers and group of 32 bits are called 'word'.

Word : The Natural unit to access in a computer, usually a group of 32 bits; corresponds to the size of a register in the MIPS architecture.

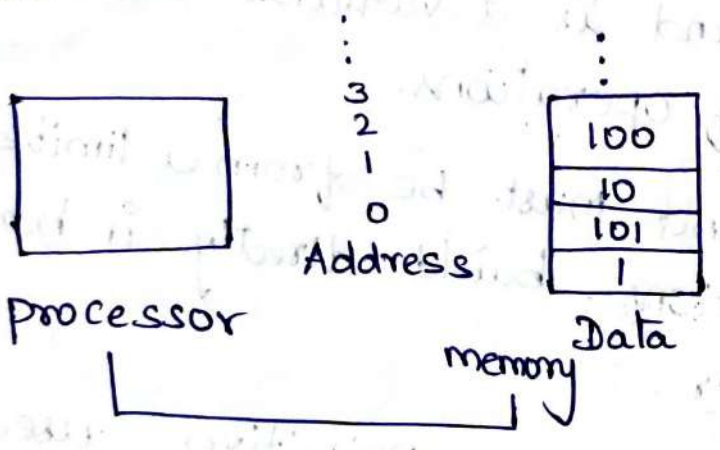
(i) MEMORY OPERANDS :-

programming languages have

* Simple variable that contain single data element.

* Complex data structures – array and structure

These complex data structures can contain many more data elements than there are registers in a computer.



Data transfer instructions :-

* Arithmetic operations occurs only on register in MIPS instructions.

* MIPS must include instructions that transfer data between memory and register. Such

instructions are called data transfer instructions.

* To access a word in memory, the instruction must supply the memory address.

memory :-

* memory is a large size and single dimensional array.

* The address acting as the index to that array starting at 0.

Load :-

* The data transfer instruction that copies data from memory to a register is traditionally called load.

* The format of the load instruction is the name of the operation followed by the register to be loaded, then a constant and register used to access memory.

MIPS Address (LOAD)

* The sum of the constant portion of the instruction and the content of the second register forms the memory address.

* The actual MIPS name for this instruction is lw, standing for load word.

Store :-

* The instruction complementary to load is called Store.

* It copies data from a register to memory.

* The format of a store is the name of the operation, followed by the register to be stored and then offset to select the array element and finally the base register.

* The MIPS address is specified in part by a constant and in part by the contents of a register. The actual MIPS name is SW standing for store word.

Name	Example	comments
32 Register	$s0, $s7 $t0, $t9 $zero, $a0 - $a3 $v0 - $v1	They can be accessed quickly, In MIPS architecture, the data must be loaded into the register to perform arithmetic operation
2^{30} memory words	Memory[0] Memory[i]	The contents can be accessed only after data transfer instruction. MIPS use byte addressing.

Category	Instruction	operation
Arithmetic	Add $S1, $S2, $S3	$S_1 = S_2 + S_3$. There are three operands in this instruction. The data resides in the register.
	Sub $S1, $S2, $S3	$S_1 = S_2 - S_3$ There are 3 operands in this register.
	Add $S1, $S2, 50	$S_1 = S_2 + 50$ This is add immediate instruction. It has two operands and one constant value.
Data transfer	Lw $S1, 50($S2)	S_1 = memory [S_2 + 50] Data is transferred from memory to regis
	Sw $S1, 50($S2)	memory [S_2 + 50] = S_1 Data is transferred from register to memory.

CONSTANT OR IMMEDIATE OPERANDS :-

↳ Constant Variables are used as one of the operand for many arithmetic operations in MIPS architecture.

↳ Many times a program will use a constant in an operation. For example incrementing an index to point to the next element of an array.

Eg : add I $S3, $S3, 10

This instruction is interpreted as addition of content of $S3 and the value 10. The sum is stored in $S3. Add I means add Immediate, since one of the operant is in Immediate addressing mode.

* As per the design principle "make common case faster" the constant operands must be loaded faster from the memory.

* Since constants occur more frequently in the instruction, they are mentioned in the instruction itself rather than to load from register.

REPRESENTATION OF INSTRUCTION

Signed and unsigned Numbers:-

* A single digit of a binary number is thus the 'atom' of computing, since all information is composed of binary digits or bits.

* This fundamental building block can be one or two values which can be thought of as several alternatives : high or low, on or OFF, true or false or '1' or '0'.

Generalizing the point, in any number base, the value of ith digit 'd' is

$$d \times Base^{i}$$

where i starts at 0 and increases from right to left.

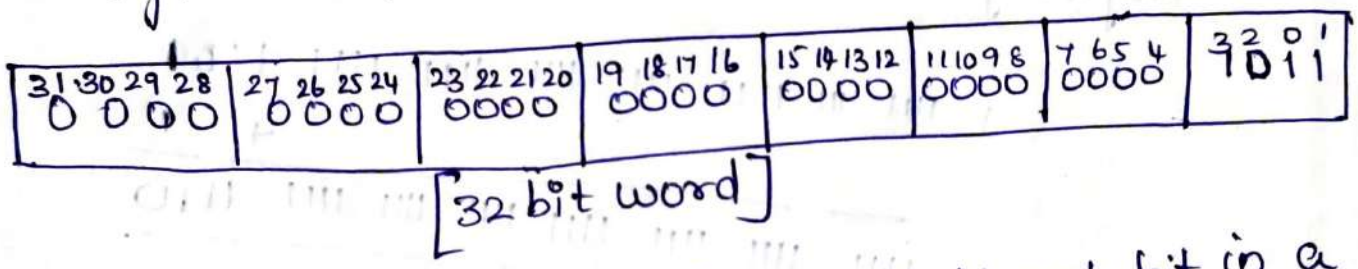
For example 1011_2 represents

$= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$

$= 8 + 0 + 2 + 2$

$= (11)_{10}$

We number the bits 0, 1, 2, 3.... from right to left in a word.



[32 bit word]

LSB [Least Significant bit] : The Right most bit in a MIPS word [Bit '0']

MSB [Most Significant bit] : The Left most bit in a MIPS word [Bit 31].

The MIPS word is 32 bits long. So we can represent 2^{32} different 32 bit pattern. It is natural to let these combinations represent the numbers from

0 to 2^{32}

* lb [load byte] sign-extends to fill the 24 left most bits of the register.

* lh (load half) sign extends to fill the 16 left-most bits of the register.
* lbu (load byte unsigned) and
* lhu (load half-word unsigned) for unsigned integers.

Problem:-

Negate 2_{ten} and then check the result by negating -2_{ten}.

$2_{ten} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{ten}$.

Negating this number by inverting the bits and adding one.

$$\begin{array}{r} 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101 \\ +\ \ 1 \\ \hline 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110 \end{array}$$

$\Rightarrow -2_{ten}$.

Going the other direction

1111 1111 1111 1111 1111 1111 1111 1110

First inversion

$$\begin{array}{r} 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001 \\ +\ \ 1 \\ \hline 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010 \end{array}$$

$\Rightarrow 2_{ten}$

In MIPS language,

↳ Register $S0 to $S7 map onto register 16 to 23.

↳ Register $t0 to $t7 map onto register 8 to 15.

Hence $S0 means register 16

↳ $S1 means register 17.

↳ $S2 means register 18

↳ $t0 means register 8.

↳ $t1 means register 9 and so on.

Examples: Translating a MIPS Assembly Instruction into a machine Instruction.

* The real MIPS language version of the instruction represented symbolically as

add $t0, $S1, $S2

First as a combination of decimal numbers and then a binary numbers.

The decimal representation is

from the about segment.

* Each of these segment of an instruction is called a Field.

* The First and last fields containing '0' and

* The second field gives the number of the register that is the first source operand of t

addition operation (17 = $s1)

* The third field gives the other source operand for the addition (18 = $s2)

* The fourth field contains the number of the register that is to receive the sum (8 = $t0)

* The fifth field is unused in this instruction, so it is set to '0'. Thus this instruction add register $s1 to register $s2 and places the sum in register $t0.

This instruction can also be represented as fields of binary numbers as opposed to decimal.

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits.

INSTRUCTION FORMAT :-

* A form of representation of an instruction composed of fields of binary number. This layout of the instruction is called the instruction format.

* From counting the number of bits, MIPS instruction takes exactly 32 bits - the same size as a data word.

* our design principle that simplicity favors regularity, all MIPS instructions are 32 bit long.

Machine language :-

Binary representation used for communication within a computer system.

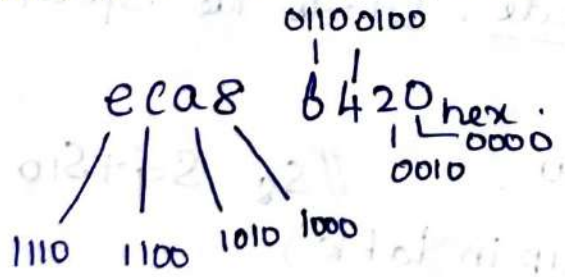
Hexadecimal :-

All computer data sizes are multiples of 4 hexadecimal (base 16) numbers are popular. Convert by replacing each group of four binary digits by a single hexadecimal digit and vice versa.

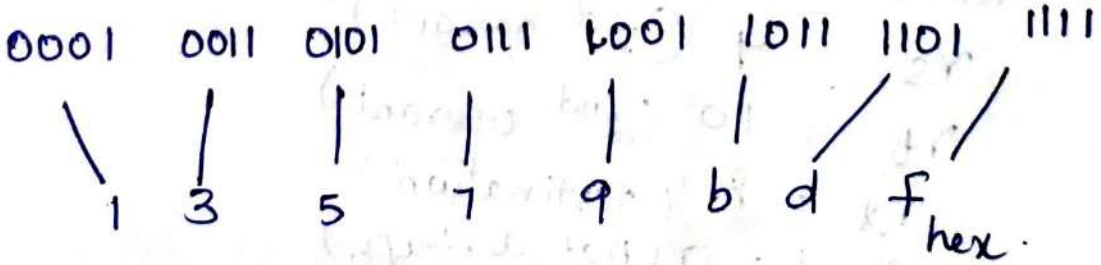
Hexadecimal	binary				
0hex	0000	7hex	0111	13	1101
1hex	0001	8hex	1000	14	1110
2hex	0010	9hex	1001	15	1111
3hex	0011	10hex	1010		
4hex	0100	11hex	1011		
5hex	0101	12hex	1100		
6hex	0110				

Example : Binary to hexadecimal and Back.

Convert the following hexadecimal and binary numbers into the other base.



and then the other direction.



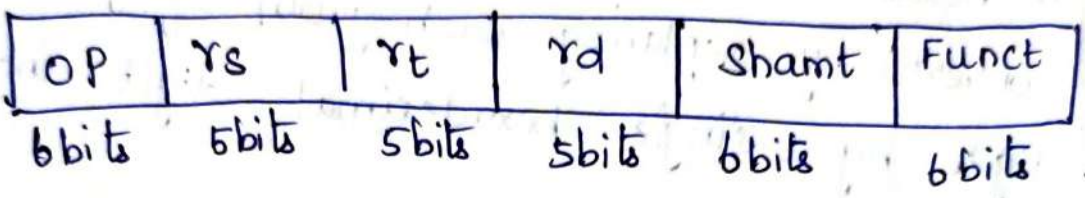
MIPS FORMAT :-

MIPS Fields has two kinds of format such as

(i) R-type (or) R Format (for register)

(ii) I-type (or) I-Format (for immediate).

(i) R-FORMAT



OP ⇒ Basic operation of the Instruction, traditionally called op code.

rs ⇒ The first register Source operand

rt ⇒ The Second register Source operand

rd ⇒ The register distination operand. It gets the result of the operation.

Shamt ⇒ Shift amount.

funct ⇒ function. This field, often called the function function code, selects the specific variant

Example

add $8, $9, $10 // $S_8 = S_9 + S_{10}$

opcode = 0 (look up in table)

funct = 32 (look up in table)

rs = 9 (1st operant)

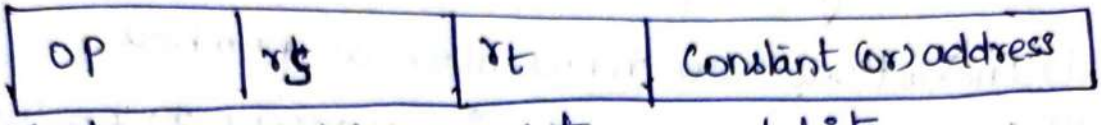
rt = 10 (2nd operant)

rd = 8 (destination)

Shamt = 0 (not a Shift)

I-Format :-

The fields of i-format are



* The 16 bit address means a load word instruction can load any word within region of $\pm 2^{15}$ or 32,768 bytes ($\pm 2^{13}$ or 8192 words) of the address in the base register rs.

* Similarly, add immediate is limited to constant no larger than $\pm 2^{15}$.

I-FORMAT Example :-

MIPS Instruction : lw $to, 1200($t1)

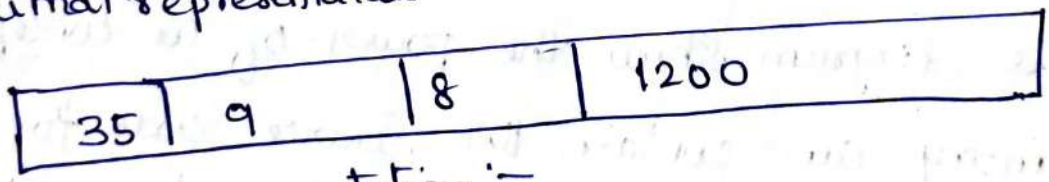
Opcode = 35 (look up in table)

rs = 9 (base register)

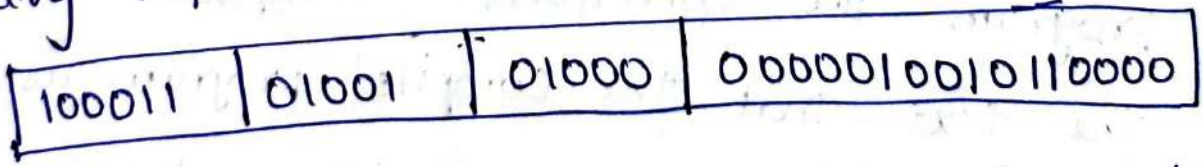
rt = 8 (destination register).

immediate = 1200 (offset).

decimal representation



binary representation :-



Note :- The meaning of the rt field has changed for this instruction : in a load word instruction.

The rt field specifies the destination register which receives the result of load.

Stored-program concept:

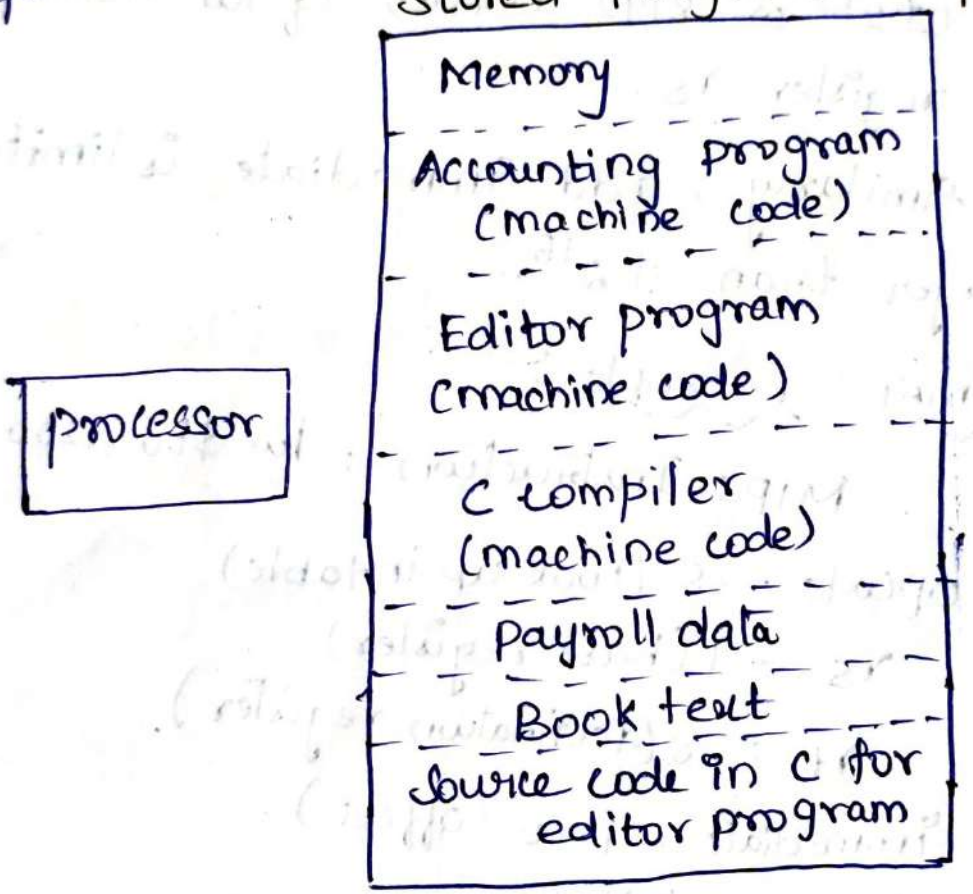
Today's computers are built on two key principles -

(i) Instructions are represented as numbers.

(ii) programs are stored in memory to be read or written, just like data.

↳ These principles lead to the stored-program concept.

Stored Program Concept.



* The diagram shows the power of the concept; Specifically memory can contain the source code for an editor program, the corresponding compiled machine code the text that the compiled program is using and even the compiler that generated the machine code.

* One consequence of instructions as number is that programs are often shipped as files of binary numbers.

* The commercial implications is that computers can inherit ready-made software, provided they are compatible with an existing instruction set.

* Such binary compatibility often leads industry to align around a small number of instructions set architecture.

LOGICAL OPERATION

* The first computer operated on full words. It was useful to operate on fields of bits within a word or even on individual bits.

* Examining characters within a word, each of which is stored as 8 bits.

* The programming languages and instruction set architecture to simplify, the packing and unpacking of bits into words. These instructions are called logical operations.

Logical operation	C operator	Java operator	MIPS Instruction
Shift left	<<	<<	sll.
Shift right	>>	>>>	srl
Bit by bit AND	&	&	and, andi

Logical operation	C-operator	Java operator	MIPS Instruction
Bit by bit OR	|	|	or, ori
Bit by bit NOT	~	~	nor

* The first class of such operations is called shifts
* They move all the bits in a word to the left or right, filling the emptied bits with '0's.

Shift

Left shift
Left shifting by i bit is equivalent to multiplying the number by 2^i

Right shift
Right shifting by i bit is equivalent to dividing the number by 2^i.

Category	Instruction	operation
AND	and $\$S_1, \$S_2, \$S_3$	$S_1 = S_2 \& S_3$
OR	or $\$S_1, \$S_2, \$S_3$	$S_1 = S_2 \mid S_3$
NOR	nor $\$S_1, \$S_2, \$S_3$	$S_1 = \sim(S_2 \mid S_3)$
NAND	nand $\$S_1, \$S_2, \$S_3$	$S_1 = \sim(S_2 \& S_3)$
ADD Immediate	andi $\$S_1, \$S_2, 100$	$S_1 = S_2 \& 100$
OR immediate	ori $\$S_1, \$S_2, 100$	$S_1 = S_2 \mid 100$
Shift left logical	Sll $\$S_1, \$S_2, 10$	$S_1 = S_2 << 10$
Shift right logical	Srl $\$S_1, \$S_2, 10$	$S_1 = S_2 >> 10$

CONTROL OPERATION :-

Decision making and branching makes the computer more powerful.

Decision making :- Decision making in MIPS assembly language includes two decision-making instructions.

- Conditional branches
  - Branch if Equal (BEQ)
    - Statement : Beq register 1, register 2, L1
    - In this instruction, go to the statement labeled L1 if the value in register 1 is equal to the value of register 2.
  - Branch if not Equal (BNE).
    - Eg : bne register 1, register 2, L1
    - In this instruction, go to the statement labeled L1 if the value of register 1 does not equal to value of register 2.

Conditional branch is an instruction that requires the comparison of two values and that allows for a subsequent transfer of control to a new address in the program based on the outcome of the comparison".

Loops :- Looping statement is used to execute the same task more than one time until certain condition gets failed.

Ex. Compiling a while loop in C

```
while (save[i] == k)
    i += [1;
```

BASIC BLOCK

* A basic block is a sequence of instruction without branches and without branch targes or branch labels.

* one of the first early phases of compilation is breaking the problem into basic blocks.

* It is useful to test for equality and inequality is probably the most popular test.

* It is useful to see if a variable is less than another variable.

For example, a for loop may want to test to see if the index variable is less than '0'.

ADDRESSING MODE :-

→ The different ways in which the operands of an instruction are specified are called as addressing modes.

→ multiple forms of addressing are generally called addressing modes.

The addressing modes are the following.

IMMEDIATE ADDRESSING :-

* where the operant is a constant within the instruction itself.

* has the advantage of not requiring an extra memory access to fetch the operant, hence will be executed faster, However the size of operand is limited to 16 bits.

* The Jump instruction format can also be considered as an example of immediate addressing

Eg : ADD 3 [Add 3 to contents of accumulator and 3 is operand]

Immediate mode

* DIRECT ADDRESSING :-

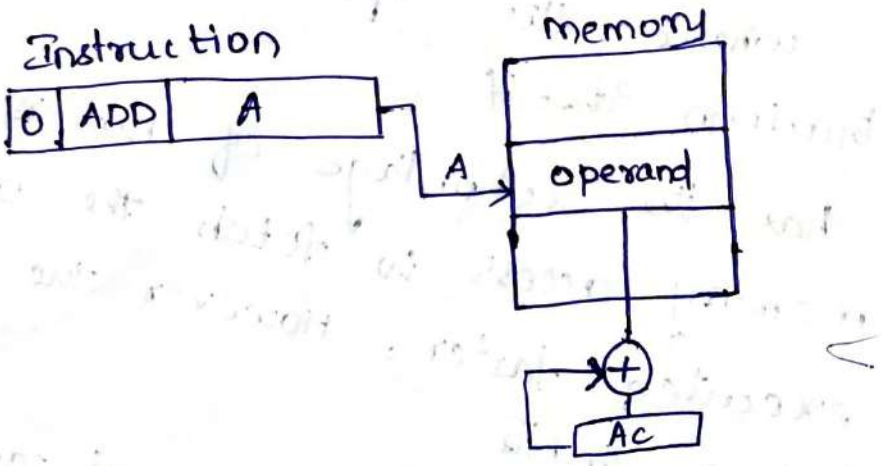
* In direct addressing mode, effective address

of the operant is given in the address field of the instruction.

* It requires one memory reference to read the operand from the given location and provides only a limited address space.

* Length of the address field is usually less than the word length.

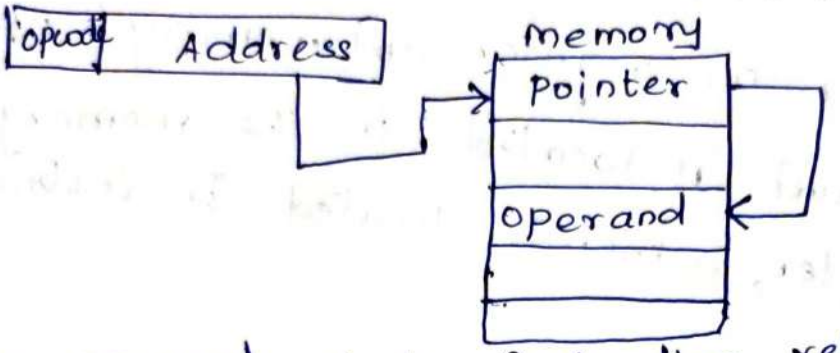
Example : move P, R0 → operants
Add Q, R0 → Register.



* INDIRECT OR PSEUDO DIRECT ADDRESSING :-

* In the mode, the offset value is specified indirectly into the memory by the pointer available in the instruction.

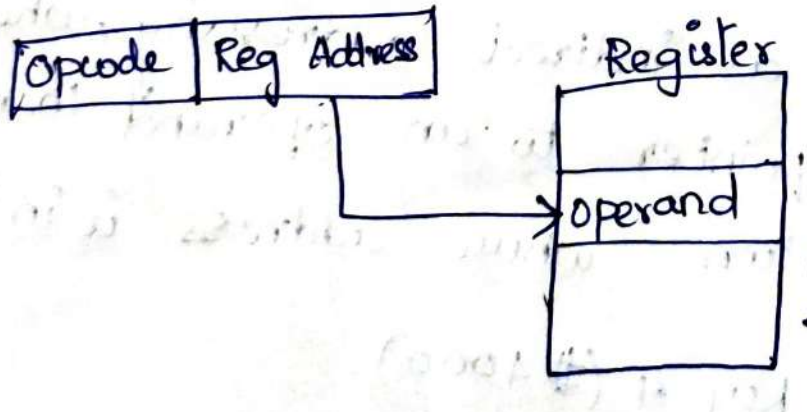
* The address field of the instruction contains the effective address.



* The operand of the instruction refers the control into the memory location indirectly by the pointed present inside the memory. Hence the pointer was direct to the operand.

REGISTER ADDRESSING :-

* It is the simplest addressing modes of all
* where the operand is a register.
* works must faster than other addressing modes because it does not involves with memory accesses.
* The register address is specified as a part of instruction.



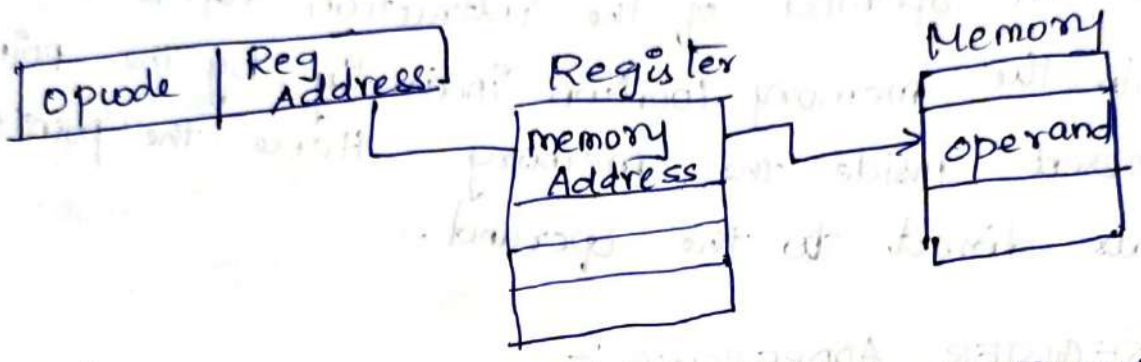
→ operand is directly obtained by the register which is present in the instruction.

E.g: ADD R_1, R_2, R_3.

(5) Register Indirect Addressing :-

↳ works on register and memory operands.

↳ operand is located in the memory pointed by the register, which is located in instruction



↳ The operand is obtained by the memory address which is located in register by searching that memory location into a memory.

(6) Base or Displacement Addressing :-

* The operand is at the memory location whose address is the sum of register and a constant in the instruction.

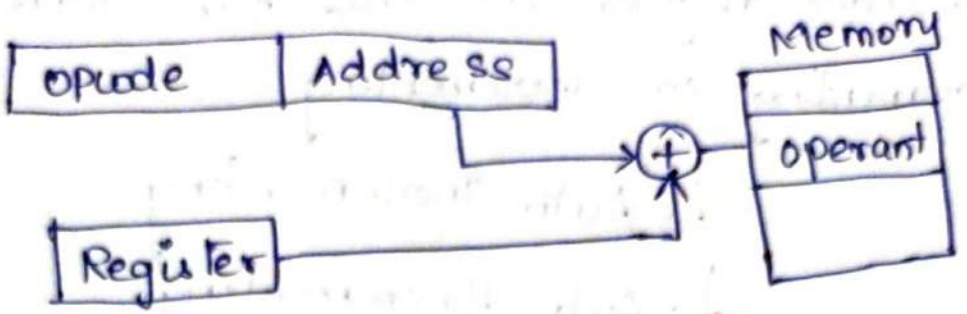
* Also known as indirect addressing, where register acts a pointer to an operand located at the memory location whose address is in the register

Ex:- LW R0, 4 (#4000)

R0 → Rs

4 → OFF set value

4000 → Memory address.

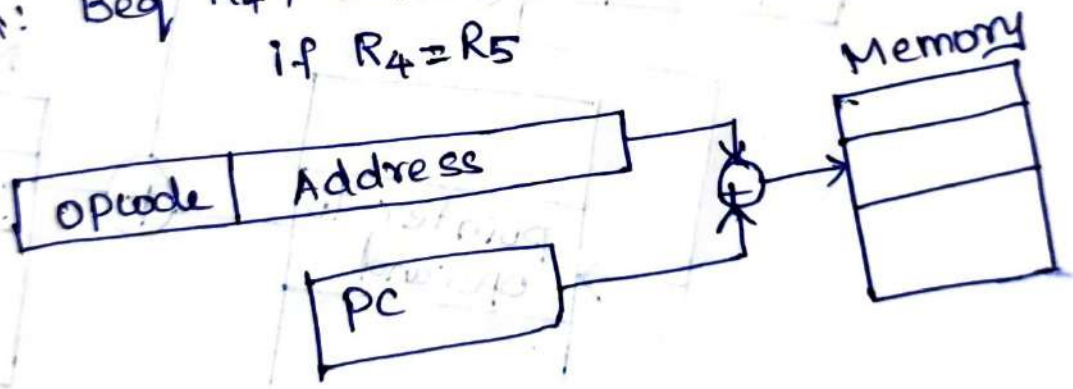


⑦ PC- Relative Addressing :-

* The branch address is the sum of the pc and a constant in the instruction.

* It is also known as program Counter addressing is a data or instruction memory location is specified as an offset value to the incremented.

Ex: Beq R_4, R_5, label.
if $R_4 = R_5$



⑧ INDEXing Addressing :-

* The address field refers to a main memory address and the ref reg contains a positive displacement from the address.

* The reg refers sometimes explit and implict.

* Index register are used for iterative task for incrementing or decrementing.

↳ Auto incrementing

↳ Auto Decrementing.

Ex:- Auto incrementing :-

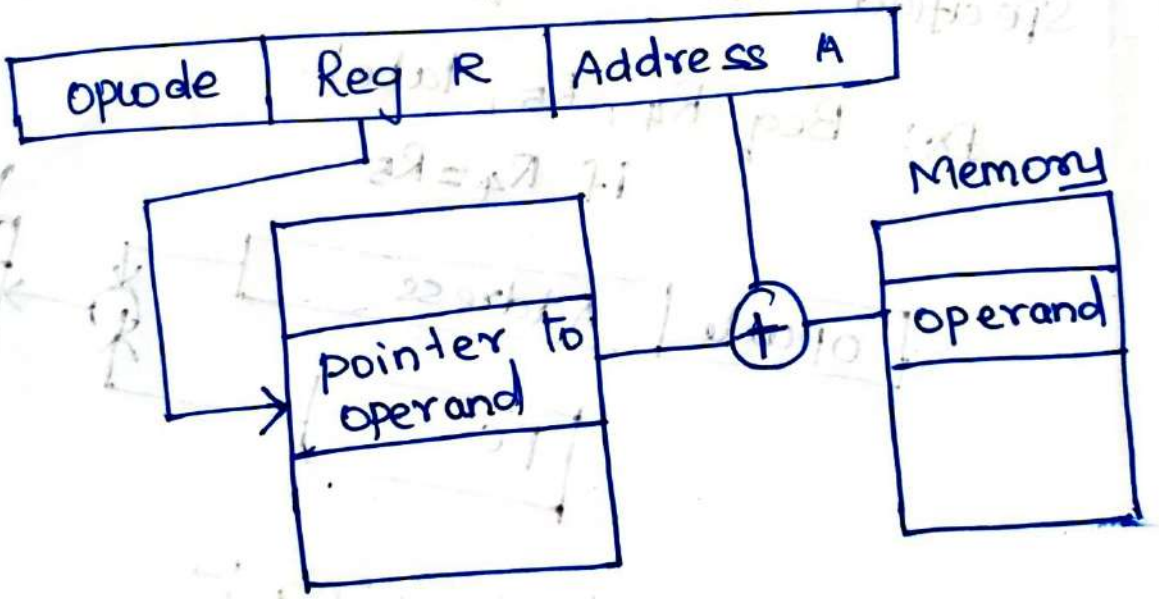
Effective Address = A+(R)

R = [R]+1

Auto decrementing :-

EA = A+[R]

R = [R]-1



Encoding of Machine Instructions :-

→ To be executed in a processor, an instruction must be encoded in a binary pattern. Such encoded instructions are referred to as Machine instructions.

→ The instructions that use symbolic-names and acronyms are called assembly language instructions.

→ The instructions that perform operations such as add, subtract, move, shift, rotate and branch may use operands of different sizes such as 32-bit and 8-bit numbers.

→ The instruction Add R1, R2 ; Has to specify the registers R1 and R2, in addition to the OP code. If the processor has 16 registers, then four bits are needed to identify each register. Additional bits are needed to indicate that the Register-addressing mode is used for each operand.

→ The instruction Move 24(R0), R5 ; Requires 16 bits to denote the OP code and the two registers, and some bits to express that the source operand uses the Index addressing mode and that the index value is 24.

In all these examples, the instructions can be encoded in a 32-bit word.

→ The OP code for given instruction refers to type of operation that is to be performed.

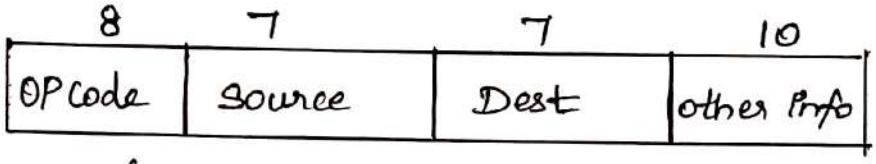
→ Source and destination field refers to source and destination operand respectively.

→ The "other info" field allows us to specify the additional information that may be needed such as an index value or an immediate operand.

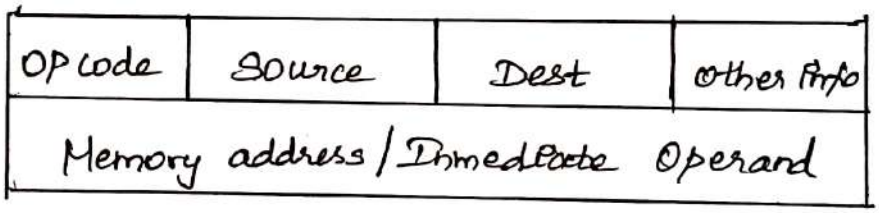
→ Using multiple registers, we can implement complex instructions, closely resembling operations in high level programming languages. The term complex instruction set computers (CISC) refers to processors that use.

→ CISC approach results in instructions of variable length, dependent on the number of operands and the type of addressing modes used.

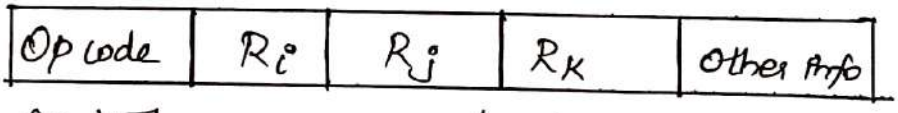
→ In RISC (Reduced instruction set computers), any instruction occupies only one word.



(a) one-word instruction



(b) Two-word Instruction



(c) Three-operand instruction.

(Fig) Encoding instructions into 32-bit words.

→ The RISC approach introduced other restrictions such as that all manipulation of data must be done on operands that are already in registers.

(Eg): Add R_1, R_2, R_3.

→ In RISC type machine, the memory references are limited to only Load/Store operations.

Assembly Level Language :-

It is a low-level language that allows users to write a program using alphanumeric mnemonic codes, instead of numeric code for a set of instructions. Examples of a large assembly language of the present time is IBM PC DOS.

High Level Language :-

It is a machine-independent type of language. It let users write various programs in such a language that resembles the English words (and alphabets) and all the familiar mathematical symbols. The very first high-level language was the COBOL language. C #, Python, etc -- are a few examples of high-level languages.

Difference between Assembly language and high-level language :-

Assembly Language	High Level Language.
1.) The assembley language requires an assembler for the process of conversion.	1.) High level language requires an interpreter/ compiler for the process of conversion.
2.) We perform the conversion of an assembly language into a machine language.	2.) We perform the conversion of a high-level language into an assembly language and then into a machine level language
3.) It is a machine-dependent type of language.	3.) It is a machine-independent type of language.
4.) It makes use of the mnemonic codes for operation.	4.) It makes use of the English statements for operation.
5.) It provides support for various low-level operations.	5.) It doesnot provide any support for low-level languages.
6.) The code is more compact in this case.	6) No code compactness is present in this case.

7.) Accessing the hardware component is very easy in this case.	7.) Accessing the hardware component is very difficult in this case.
8.) It is Processor-dependent.	8.) It is processor-independent
9.) It has better accuracy.	9.) Accuracy is much lesser in this case.
10.) An assembly language performs better than any high-level language, in general.	10.) The performance is comparatively not so good.
11.) It is shorter in assembly language.	11.) It is larger in a high-level language.
12.) Execution of code takes less time in this case because the code is not very large.	12.) It takes up more time for execution because it needs to execute a large code.
13.) It is way more efficient because of the shorter executable codes.	13.) It is comparatively less efficient because the executable codes are comparatively longer in length.
14.) We can do that directly at a physical address in the case of an assembly language. [Reading of pointers]	14.) It is not possible to do so in the case of a high-level language.
15.) It is very difficult to debug and understand the code of an assembly language.	15.) It is very easy to debug and understand the code of an assembly language.

UNIT –IV: PROCESSOR

4.1 BASIC MIPS IMPLEMENTATION

The basic MIPS implementation that has three kinds of core instruction set such as,

1. The memory-reference instructions ***load word*** **(lw) and** ***store word*** **(sw)**
2. The arithmetic-logical instructions **add, sub, AND, OR, and slt**
3. The instructions ***branch equal*** **(beq) and** ***jump*** **(j)**

To implement the above three types of instructions for same method, but independent of the exact class of instruction. For every instruction, the **first two steps are identical**:

1. Send the *program counter* (PC) to the memory that contains the code and fetch the instruction from that memory.
2. Read one or two registers, using fields of the instruction to select the registers to read. For the load word instruction, to read only one register, but most other instructions require reading two registers.

- After the above two steps, the actions required to complete the instruction depend on the instruction class.

    1. Memory Reference
    2. Arithmetic- Logical
    3. Branches

After using the ALU, the actions required to complete various instruction classes differ.

- A memory-reference instruction will need to access the memory either to read data for a load or write data for a store.
- An arithmetic-logical or load instruction must write the data from the ALU or memory back into a register.
- A branch instruction may need to change the next instruction address based on the comparison; otherwise, the PC should be incremented by 4 to get the address of the next instruction.

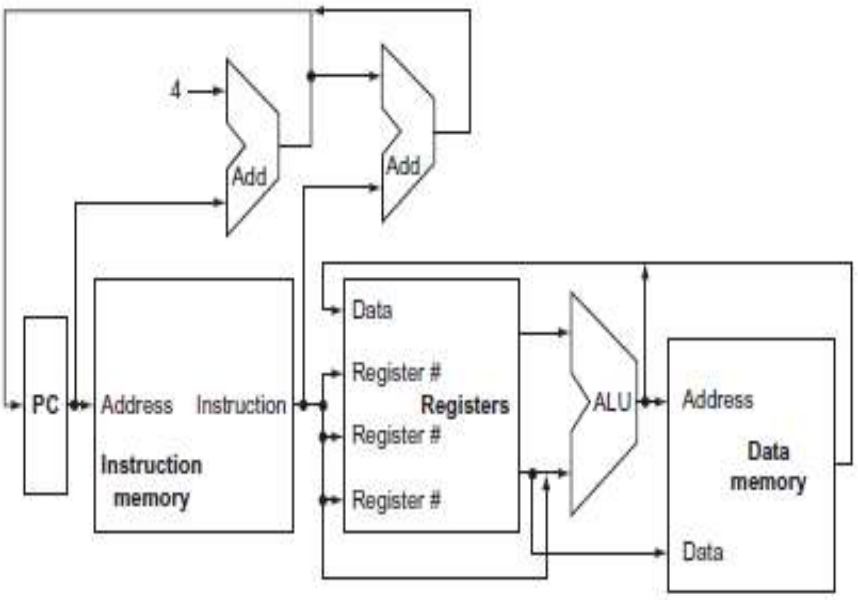


Fig 3.1 An abstract view of the implementation of the MIPS

Explanation of An abstract view of the implementation of the MIPS

- All instructions start by using the program counter to supply the instruction address to the instruction memory.
- After the instruction is fetched, the register operands used by an instruction are specified by fields of that instruction.
- Once the register operands have been fetched, they can be operated on
    1. To compute a memory address (for a load or store),
    2. To compute an arithmetic result (for an integer arithmetic-logical instruction),
    3. To compare (for a branch).
- If the instruction is an arithmetic-logical instruction, the result from the ALU must be written to a register.
- If the operation is a load or store, the ALU result is used as an address to either store a value from the registers or load a value from memory into the registers.
- The result from the ALU or memory is written back into the register file.
- Branches require the use of the ALU output to determine the next instruction address,

  which comes either from the ALU (where the PC and branch off set are summed) or from an adder that increments the current PC by 4.

The above figure 3.1 shows most of the flow of data through the processor, it omits **two important** aspects of instruction execution.

1. Data going to a particular unit as coming from two different sources.
2. Several of the units must be controlled depending on the type of instruction.

First aspect: Data going to a particular unit as coming from two different sources.

- The value written into the PC can come from one of two adders.
- The data written into the register file can come from either the ALU or the data memory
- The second input to the ALU can come from a register or the immediate field of the instruction.
- These data lines cannot simply be wired together so must add a logic element that chooses from among the multiple sources and steers one of those sources to its destination.
- This selection is commonly done with a device called a ***multiplexor*****;** it is also called as ***data selector*****.**
- The multiplexor, which selects from among several inputs based on the setting of its control lines. The control lines are set based primarily on information taken from the instruction being executed.

Second aspect: Several of the units must be controlled depending on the type of instruction.

Several of the units must be controlled depending on the type of instruction.
For example,

- The data memory must read on a load and written on a store.
- The register file must be written only on a load or an arithmetic-logical instruction.
- The ALU must perform one of several operations

The **following Figure 3.2 shows** the data path with the three required multiplexors added, as well as control lines for the major functional units.

Control unit:

- A control unit, which has the instruction as an input, is used to determine how to set the control lines for the functional units and two of the multiplexors.

Function of Third multiplexor

- The third multiplexor, which determines whether PC + 4 or the branch destination address is written into the PC.
- It is set based on the Zero output of the ALU, which is used to perform the comparison of a beq instruction.
- The regularity and simplicity of the MIPS instruction set means that a simple decoding process
- It is used to determine how to set the control lines.

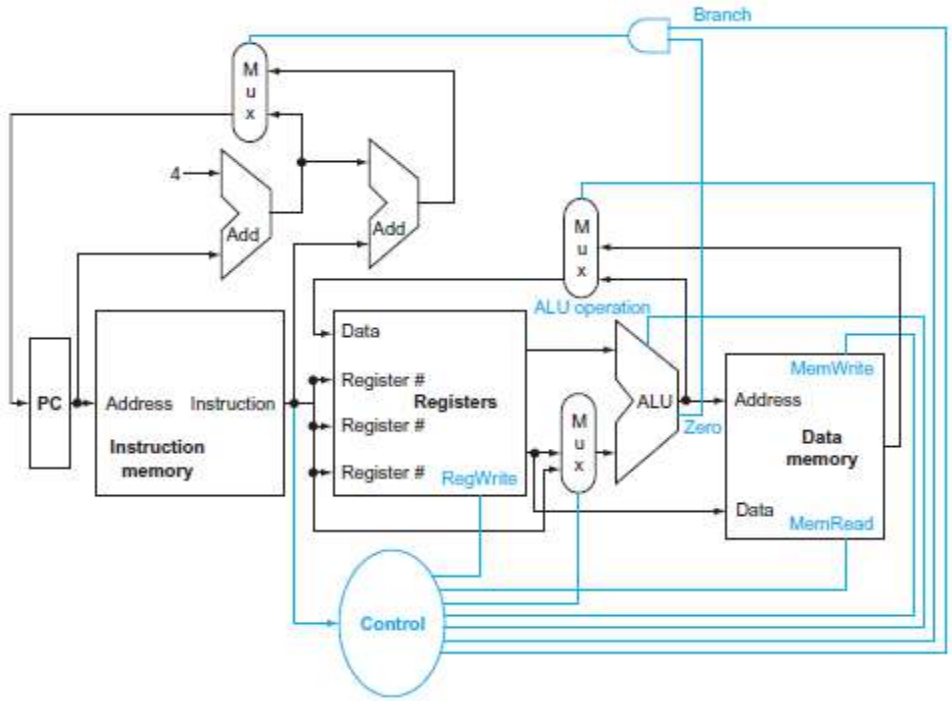


Fig 3.2 The basic implementation of the MIPS

Functions of Multiplexers:

The **top multiplexor** ("Mux") controls what value replaces the PC (PC + 4 or the branch destination address);
The multiplexor is controlled by the gate that "ANDs" together the Zero output of the ALU and a control signal that indicates that the instruction is a branch.

The **middle multiplexor**, whose output returns to the register file, is used to steer the output of the ALU or the output of the data memory (in the case of a load) for writing into the register file.

Finally, the **bottom most multiplexor** is used to determine whether the second ALU input is from the registers (for an arithmetic-logical instruction or a branch) or from the offset field of the instruction (for a load or store).

Function of Control line

Control lines are straight forward and determine the operation performed at the ALU.
ALU can perform following operations:
1. Data memory read
2. Data memory write
3. Write operation on register

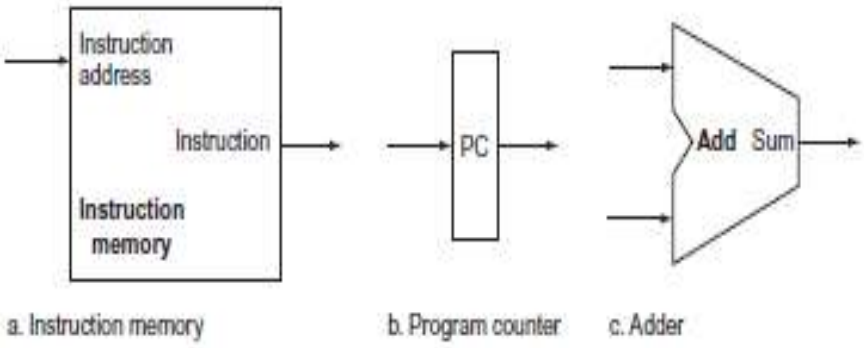
The control line determines the ALU perform which operations among three mentioned operations.

3.2 BUILDING DATA PATH

- Components required to form a data path is known as data path element.

Data path Element:

- It is a unit used to **operate on or hold data** within a processor.
- In the MIPS implementation, the data path elements includes
  1. The instruction and data memories
  2. The register file
  3. The ALU
  4. Adders.



Instruction memory:

- The instruction memory need only **provide read access** because the data path **does not write** instructions.
- The instruction memory is called as **combinational element** because it will **perform only read** access.
- The **output** at any time reflects the **contents of the location** specified by the address input,
- No **read control signal** is needed.

Program Counter:

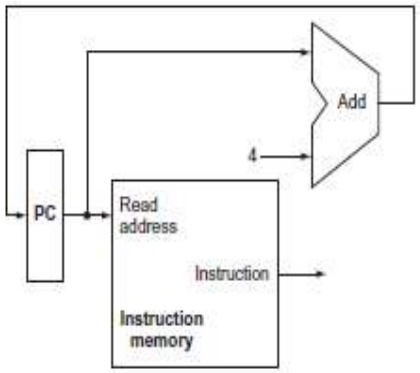
- The program counter is a **32-bit register** that is **written** at the **end of every clock cycle**
- **Does no**t need a **write control signal**.
- The register containing **the address of the instruction** in the program being executed.

Adder:

The adder is an ALU wired to always **add its two 32-bit inputs** and **place the sum on its output**.

Fetching Phase:

- To execute any instruction, must start by **fetching the instruction from memory**.
- To prepare for **executing the next instruction**, must also **increment the program counter**
- So that it points at the next instruction, 4 bytes (PC+4).



Portion of data path used for fetching instruction and incrementing the program counter

Instruction classes are,

1. Arithmetic and logical instructions
2. Memory reference instructions
3. Branch Instructions

3.2.1. ARITHMETIC LOGICAL INSTRUCTIONS:

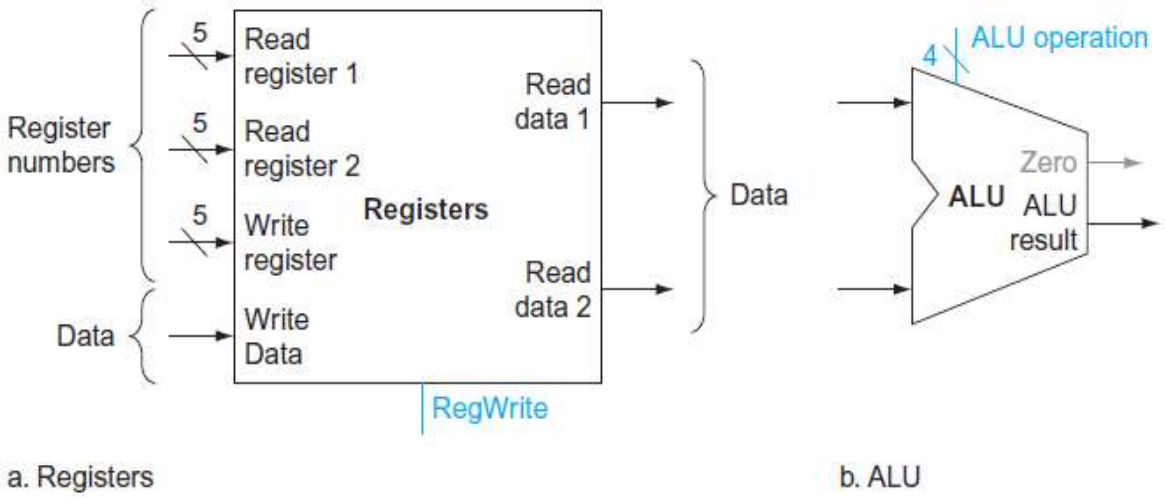
- It is also called as **R-format** instructions
- To perform an ALU operation these instructions **read two registers**, and **write** the result to a one **register**.
- This instruction class includes **add, sub, AND, OR, and slt**.
- The processors having **32 general-purpose registers and special purpose registers**.
- General-purpose registers and special purpose registers are stored in **separate space** of memory.
- These 32 general-purpose registers are stored in a **structure** called a **register file.**

Register File

- A register file is a state element that contains **set of registers** that can be **read or written** by **specifying the number** to be accessed.
- The register file contains the **register state** of the computer.

R-Type Instruction operand:

- R-format instructions have **three register operands** to perform ALU operation.
- **Two registers data** are read from the register file and **write one data word** into the register file for each instruction.
- To read data word from the registers, have to specify the **register number**.
- **Register number** specifies which data has to be read from which register present in the register file.
- To write a data word, we will need two inputs:
    1. One to specify the **register number to be written**
    2. One to supply the **data to be written into the register**.
- **Write operations** are **controlled** by the **write control signal**, which must be asserted for a write to occur at the clock edge.



- There are **two elements** need to implement the R-format ALU operations such as
  1. Register file
  2. ALU

Register file:

- The register file contains all the registers and has two read ports and one write port.
- The register file always outputs the contents of the registers corresponding to the Read register inputs on the outputs
- No other control inputs are needed.
- **Register write** must be explicitly indicated by asserting the **write control signal.**
- The register number to the **register file** is all **5 bits** wide to specify one of **32 bits wide**.

ALU:

- ALU takes **two 32 bit inputs** and **produces a 32 bit result** as well as 1 bit signal if the result is 0.
- The **operation** to be performed by the **ALU** is controlled with the **ALU operation signal**, which will be **4 bits wide**.
- The **Zero detection output** used to implement **branches**.

3.2.2. MIPS INSTRUCTIONS:(Memory reference instructions):

- Consider the MIPS load word and store word instructions, which have the general form.
  1. **lw $t1,offset_value($t2)**
  2. **sw $t1,offset_value ($t2).**

- These instructions compute a **memory address** by adding the **base register,** which is $t2, to the **16-bit signed off set field** contained in the instruction.

  Memory address= base register+16-bit signed off set field

- If the instruction is a **store**, the **value to be stored** must also be **read from the register file** where it resides in $t1.

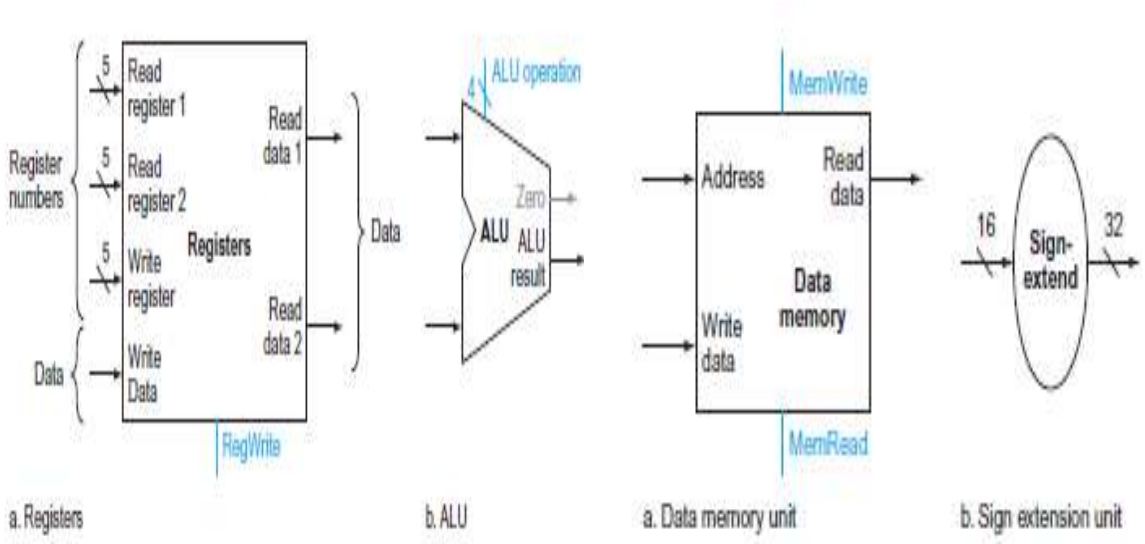
- If the instruction is a **load**, the **value read from memory** must be **written into the register file** in the specified register, which is $t1.

Units needed to implement loads and stores

Four Units needed to implement loads and stores instructions.

To implement MIPS load and store instructions we need following four units:

1. Register file
2. ALU
3. Data Memory
4. Sign extension unit



1. Register file:

- The register file contains all the registers and has **two read ports and one write port**.
- The register file always outputs the contents of the registers corresponding to the Read register inputs on the outputs
- No other control inputs are needed.
- **Register write** must be explicitly indicated by asserting the **write control signal**.
- The register number to the register file is all **5 bits** wide to specify one of **32 bits** wide.

2. ALU:

- ALU takes **two 32 bit inputs** and **produces a 32 bit result** as well as 1 bit signal if the result is 0.
- The **operation** to be **performed** by the **ALU** is **controlled** with the **ALU operation signal**, which will be **4 bits wide**.
- The **Zero detection output used** to implement **branches**.

3. Data Memory unit:

- The memory unit is a **state element** with inputs for the **address** and the **write data**.
- It produces a **single output** for the **read result**.
- Data Memory unit has **separate read and write control lines for read and write operation.**
- **Register** file **does not require read signal** but **memory** unit **needs** a **read signal**
- Because the register file, reading the value of an invalid address can cause problems.

4. Sign Extension unit:

The sign extension unit has a **16-bit data input and that sign-extended into a 32-bit result.**

3.2.3. BRANCH INSTRUCTIONS:

There are **two kinds** of **branch instructions** are available

1. **beq**-branch equal
2. **bnq**-branch unequal

The beq instruction has **three operands**, in that

1. **Two operands are registers** that are compared for **equality**
2. **One operand** is a **16-bit off set** used to compute the branch target address relative to the branch instruction address.

The beq instruction has the following form

beq $t1,$t2,offset

To implement this instruction, must compute the branch target address

Branch target address = sign-extended off set field of the instruction + PC

Branch target address:

- It is an **address specified** in a **branch**, which becomes the new program counter (PC) if the branch is taken.
- If the **operands** are **equal**, the **branch target address becomes the new PC**, and it is called as **branch is taken**.
- If the **operands are not equal**, the **incremented PC** should replace the current PC and it is called as **branch is not taken.**
- The branch data path will perform **two kinds** of **operations**:
    1. Compute the **branch target address**
    2. Compare the **register contents**.
- To compute the **branch target address**, the branch data path includes a **sign extension unit**
- To **perform** the **compare**, need to use **the register file**
- **Adder circuit** is used to **compute the branch target** and it is **sum** of the **incremented PC** and **sign extended lower 16 bits of the instruction shifted left 2 units.**
- **Control logic** is used to decide whether the in**cremented PC** or **branch target** should replace the **PC**, based on the **Zero output of the ALU**.

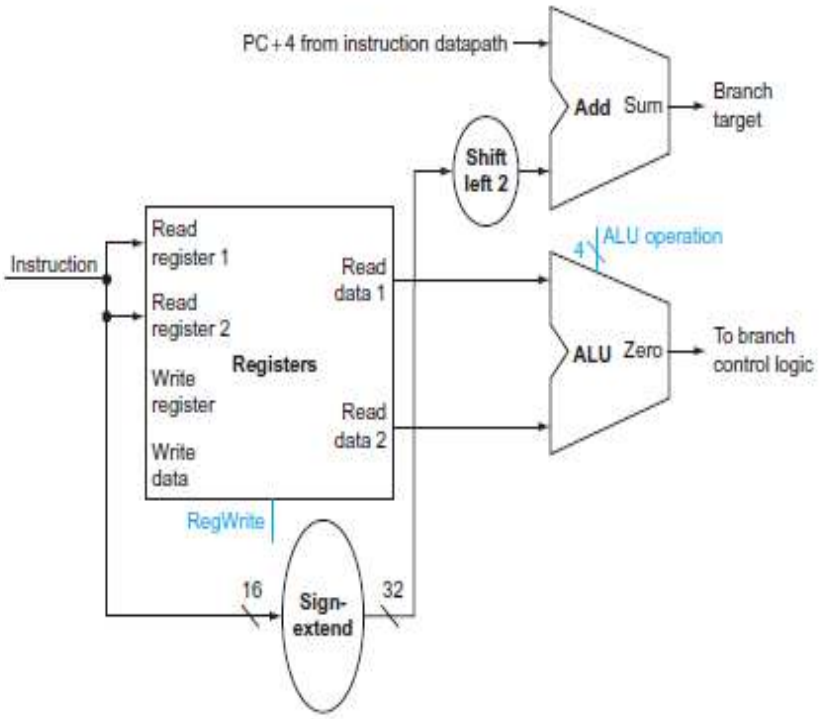


Fig 3.3 Structure of data path segment that handles branches

- **Shift left 2**is simply a direction of the **signals between input and output** that **adds 00two** to the low-order end of the sign-**extended off set field**
- **Control logic** is used to decide whether the **incremented PC** or **branch target** should **replace the PC**, **based on** the **Zero output** of the ALU as shown in fig 3.3

3.2.4 CREATING A SINGLE DATA PATH

- By combining individual instruction class data path components can form a single data path and add the control to complete the implementation.
- Single data path will execute **all instructions** in **one clock cycle**. This means that **no data path resource** can be used more than once per instruction
- In single data path if any **element needed more than once** must be duplicated.
- To **share** a data path element between **two different instruction classes,** need to allow **multiple connections** to the input of an element.
- To provide **multiple connections** we need to use **multiplexor** and **control signal** to select among the multiple inputs.
- For example consider two different instruction classes are
    1. Arithmetic and logical instructions (or) R-type
    2. Memory instructions

Difference between arithmetic and logical instructions and memory instructions

S.NO	R-type instruction	Memory instruction
1	It gets two operand s from register to perform LAU operation	It gets one operand from register and another operand from sign extended 16 bit offset field from the instruction to do address calculation
2	ALU result has stored in the destination register	ALU result has stored in the load

- For these two different kinds of instruction classes need to make single data path.
- It can be obtained by using single register file, single ALU to handle both types of instructions and multiplexers.

- To create a data path with only a **single register file** and a **single ALU**, need to provide **two different sources** for the **second input of the ALU.**
- Because both instructions has **first operand** as **register** and **second operand** is **different**.
- **Two instructions** have **two different formats** to store result so need to support **two different sources** for the data stored into the register file.
- For that need two multiplexers, **one multiplexor** is **placed** at the **ALU input** and another at the **data input** to the **register file**.

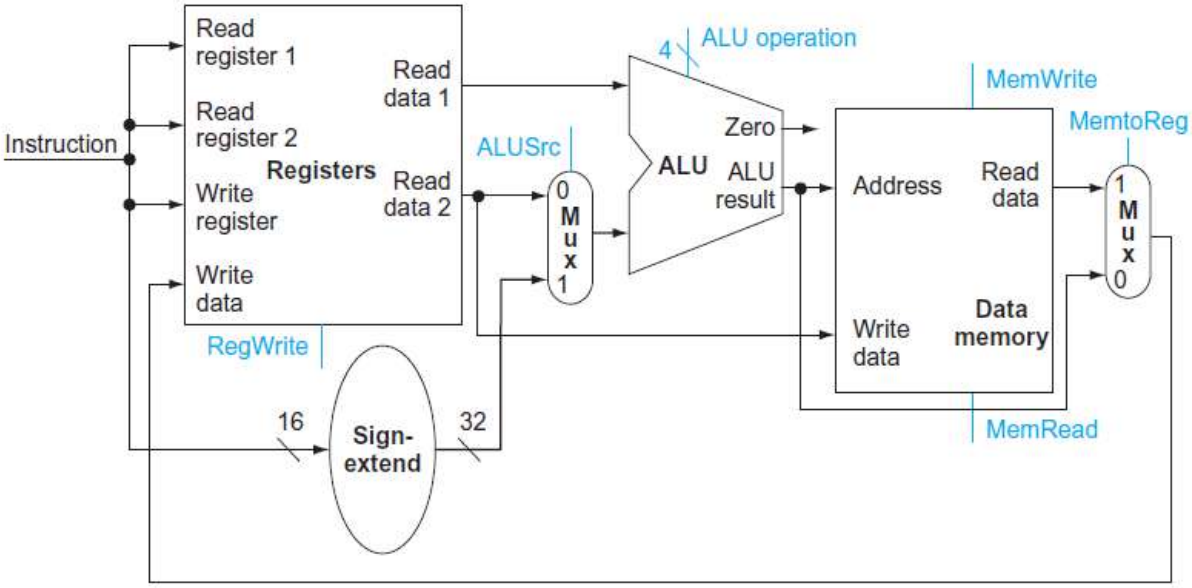


Fig 3.4 Data path for the memory and R-Type instructions

- Combine the simple data path for the core MIPS architecture.
- It can be obtained by adding **the data path for instruction fetch** and **the data path from R-type** and **memory instructions** and **the data path for branches** that is shown in the above figure 3.4
- The branch instruction uses the **main ALU** for **comparison of the register operands**. So need to use the adder circuit for the data path components of branch instruction
- An **additional multiplexor** is required to select either the sequentially following **instruction address (PC + 4) or the branch target address to be written into the PC.**
- To complete this simple data path, must add the control unit.
- The **control unit** must be able to take inputs and generate a write signal for each state element, the selector control for each multiplexor, and the ALU control.
- The ALU control is different in a number of ways, and it will be useful to design it first before design the rest of the control unit.
- The Simple data path for the core MIPS architecture by combining elements required by different instruction classes as shown in fig 3.5.

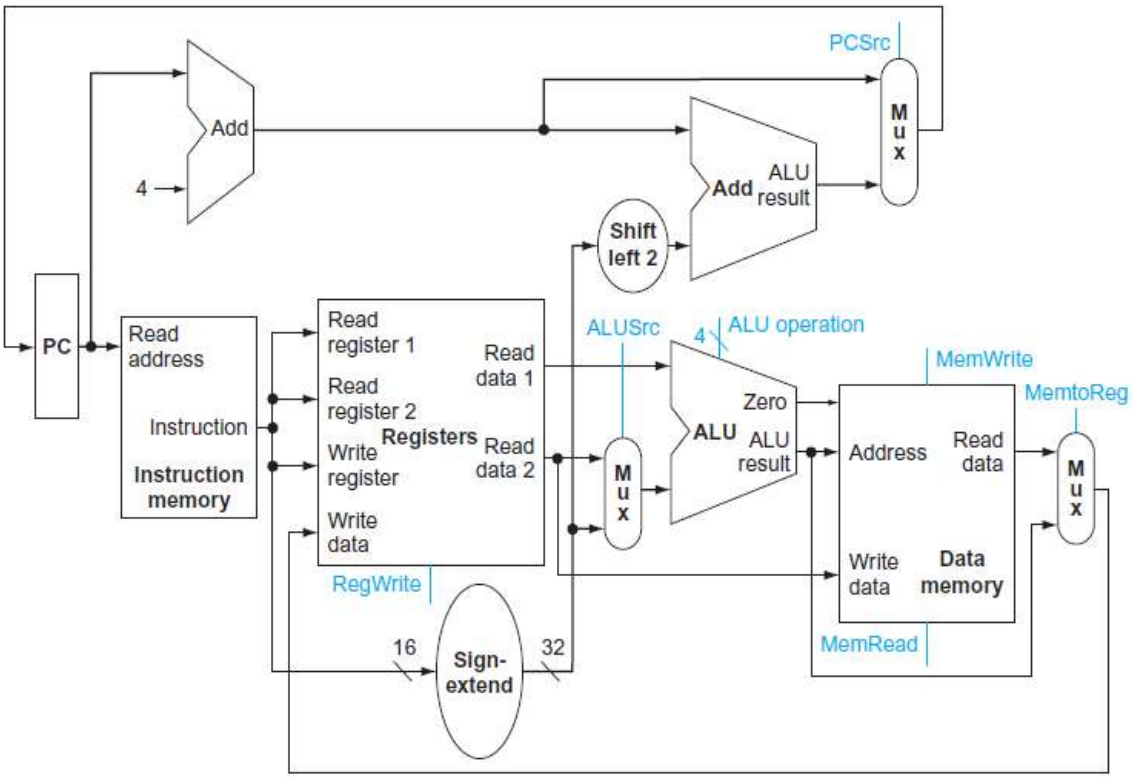


Fig 3.5 Simple data path for the core MIPS architecture by combining elements required by different instruction classes

3.3 CONTROL IMPLEMENTATION SCHEME

- Any instruction set can be implemented in many different ways like **single-cycle implementation and multicycle implementation.**
- In a basic single-cycle implementation all operations take the same amount of time-a single cycle.
- A multicycle implementation allows faster operations to take less time than slower ones, so overall performance can be increased**.**

3.3.1 The ALU Control

Depending on the instruction class, the ALU will need to perform one of these functions.

- **Load word and store word instructions**- ALU to compute the memory address by addition.
- **R-type instructions**- ALU needs to perform one of the five actions (AND,OR, subtract, add, or set on less than), depending on the value of the 6‑bit funct (or function) field in the low-order bits of the instruction.
- **Branch equal**-ALU must perform a subtraction.
- We can generate the 4-bit ALU control input using a small control unit.
- It has input function field of the instruction and a 2-bit control field, called ALUOp.
- ALUOp indicates three kinds of operations to be performed
  1. add (00) for loads and stores,
  2. subtract (01) for beq, or
  3. determined by the operation encoded in the funct field (10).

The output of the ALU control unit is a 4-bit signal that directly controls the ALU by generating one of the 4-bit combinations.

In following figure 3.6 shows how to set the ALU control inputs based on the 2-bit ALUOp control and the 6-bit function code.

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

Fig 3.6 The ALU control inputs based on the 2-bit ALUOp control and the 6-bit function code.

- When the ALuOP is 00 or 01, the ALU action does not depend on the function code field.
- We do not care about the value of the function code and the function field is shown as XXXXXX for 00 and 01 values.
- When the ALUOP value is 10, then the function code is used to set the ALU control input.

Multiple levels of decoding functions:

1. The main control unit generates the ALUOP bits.
2. ALUOP bit is used as a input to the ALU control.
3. That ALU control generates the actual signals to control the ALU unit.

Mapping 2-bit ALU Op field and 6-bit funct field

- There are several different ways to implement the mapping. From the 2-bit ALUOp field and the 6-bit funct field to the four ALU operation control bits.
- There are 64 possible values are available for function field in that small values are used more frequently.
- The function field is used only when the ALUOP bits equal to 10.
- A small piece of logic that recognizes the subset of possible values and causes the correct setting of the ALU control bits.
- To design logic first we have to create a truth table for the function code field and the ALUOP bits.

Truth table: It is a representation of a logical operation by listing all the values of the inputs and then in each case showing what the resulting outputs should be.

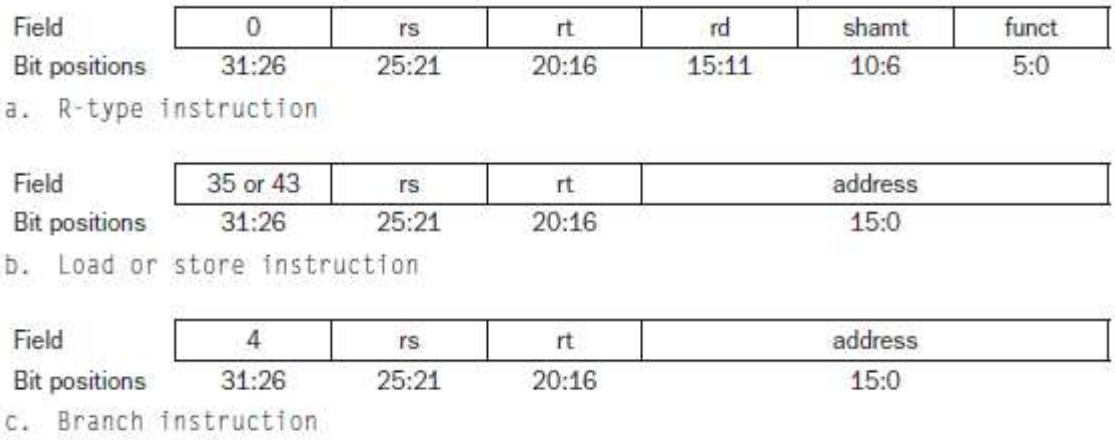
Don't-care term: An element of a logical function in which the output does not depend on the values of all the inputs. Don't-care terms may be specified in different ways.

ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

The truth table for the 4 ALU control bits (called Operation).

3.3.2 Designing the Main Control Unit:

- ALU control can be design that uses the function code and a 2-bit signal as its control inputs.
- Now we can design main control unit for that we have to identify the fields of an instruction and the control lines.
- Control lines are needed for the data path construction.
- Instruction format of R-type, load store and branch instructions are shown.



Opcode: The field that denotes the operation and format of an instruction.

(a) **Instruction format for R-format instructions**, which all have an opcode of 0. These instructions have three register operands: rs, rt, and rd. Fields rs and rt are sources, and rd is the destination. The ALU function is in the funct field and is decoded by the ALU control design in the previous section. The R-type instructions that we implement are add, sub, AND, OR, and slt. The shamt field is used only for shift

(b) **Instruction format for load** (opcode = 35ten) and store (opcode = 43ten) instructions. The register rs is the base register that is added to the 16-bit address field to form the memory address. For loads, rt is the destination register for the loaded value. For stores, rt is the source register whose value should be stored into memory.

(c) **Instruction format for branch equal** (opcode =4). The registers rs and rt are the source registers that are compared for equality. The 16-bit address field is sign-extended, shifted, and added to the PC + 4 to compute the branch target address.

There are several major observations about this instruction format:

- The op field, also called the opcode, is always contained in bits 31:26.
- The two registers to be read are always specified by the rs and rt fields, at positions 25:21 and 20:16.
- The base register for load and store instructions is always in bit positions 25:21 (rs).
- The 16-bit offset for branch equal, load, and store is always in positions 15:0.
- The destination register is in one of two places. For a load it is in bit positions 20:16 (rt), For R-type it is in bit positions 15:11(rd).
- So we need to add a multiplexor to select which field of the instruction is used to indicate the register number to be written.
- Using this information, we can add the instruction labels and extra multiplexor to the simple datapath as shown in fig 3.7

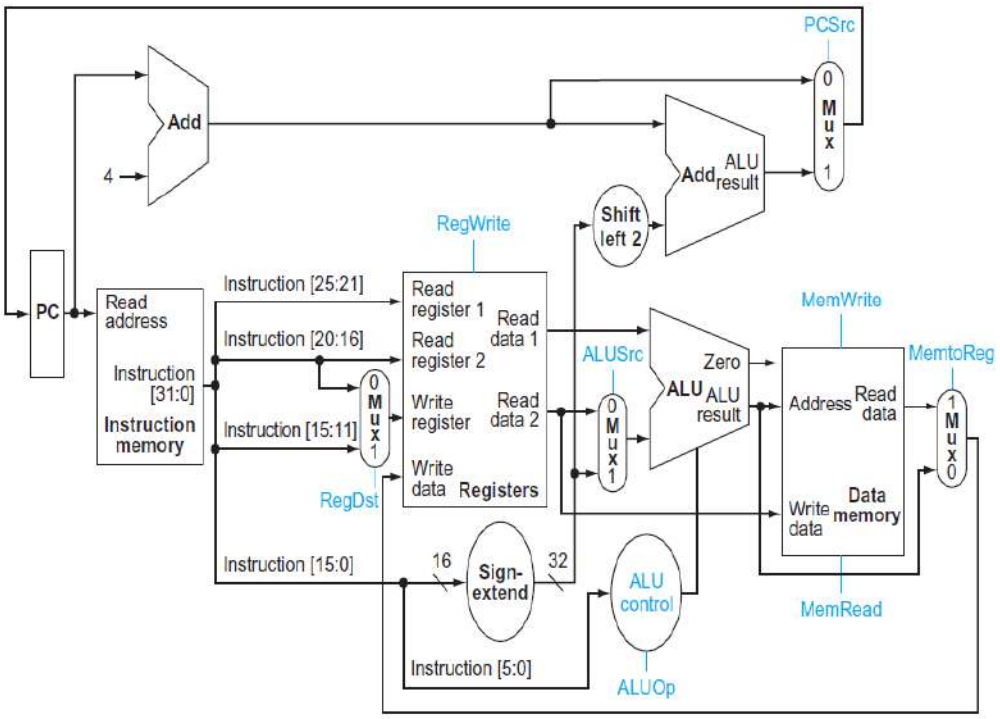


Fig 3.7 The datapath with all necessary multiplexors and all control lines identified.

- This shows these additions plus the ALU control block, the write signals for state elements, the read signal for the data memory, and the control signals for the multiplexors. Since all the multiplexors have two inputs, they each require a single control line.

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

The effect of each of the seven control signals.

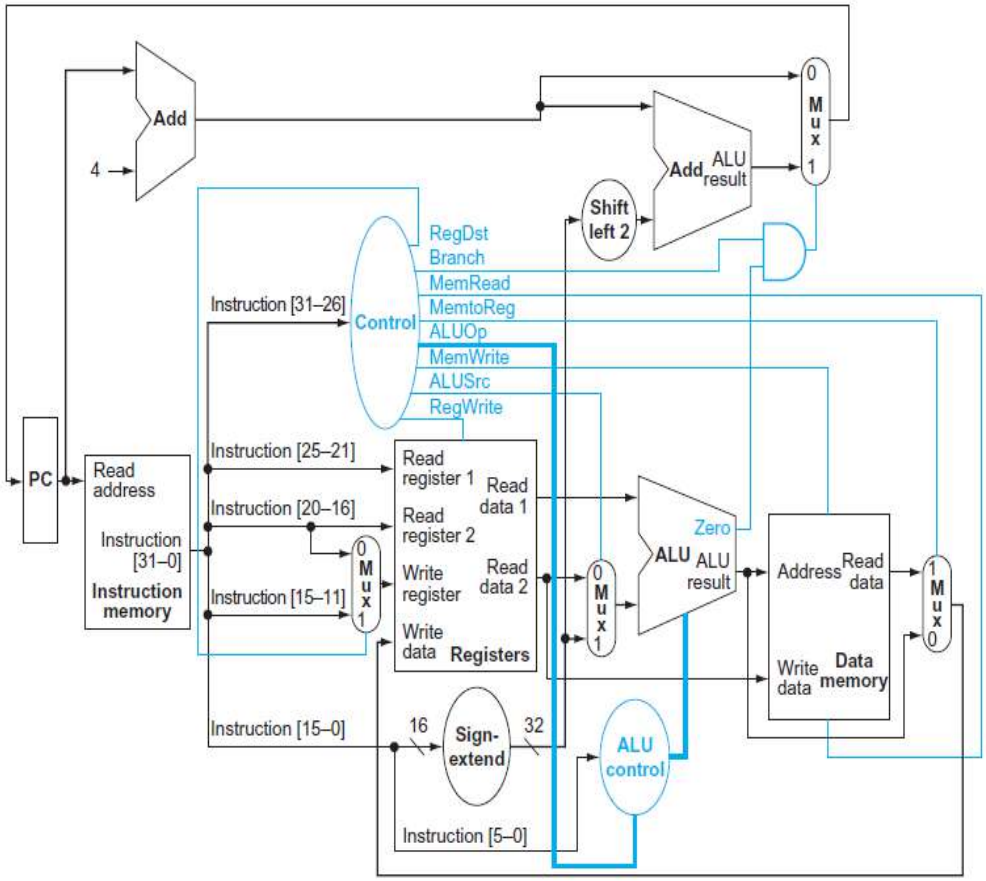


Fig 3.8 The simple datapath with the control unit

Truth table for the control unit:

Setting of the control lines depends only on the opcode and we have to define whether each control signal should be 0,1 or don't care (X) for each of the opcode values.

The below truth table shows how the control signals should be set for each opcode.

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

R-Format:

- ✓ The first row of the table corresponds to the R-format instructions (add, sub, AND, OR, and slt).
- ✓ For all these instructions, the source register fields are rs and rt, and the destination register field is rd; this defines how the signals ALUSrc and RegDst are set.
- ✓ R-type instruction writes a register (Reg-Write = 1), but neither reads nor writes data memory.
- ✓ The ALUOp field for R-type instructions is set to 10 to indicate that the ALU control should be generated from the funct field.

Load Word and store word:

- ✓ The second and third rows of this table give the control signal settings for lw and sw.
- ✓ These ALUSrc and ALUOp fi elds are set to perform the address calculation.
- ✓ The MemRead and MemWrite are set to perform the memory access.
- ✓ RegDst and RegWrite are set for a load to cause the result to be stored into the rt register.

Branch instruction:

- ✓ The branch instruction is similar to an R-format operation, since it sends the rs and rt registers to the ALU.
- ✓ The ALUOp field for branch is set for a subtract (ALU control = 01), which is used to test for equality. Notice that the MemtoReg field is irrelevant when the RegWrite signal is 0: since the register is not being written, the value of the data on the register data write port is not used.
- ✓ Thus, the entry MemtoReg in the last two rows of the table is replaced with X for don't care. Don't cares can also be added to RegDst when RegWrite is 0.

3.3.3 Operation of the DataPath:

We have to consider three kinds of instruction classes so far such as

1. R-type instructions
2. Load and store instructions
3. Branch instructions

3.3.3.1 R-type instruction operations in four steps:

- ✓ In R-type instruction consider add $t1,$t2,$t3 and remaining four operations (sub, AND, OR,slt) occurs in one clock cycle as shown in fig 3.9.

1. The instruction is fetched, and the PC is incremented.
2. Two registers, $t2 and $t3, are read from the register file; also, the main control unit

computes the setting of the control lines during this step.
3. The ALU operates on the data read from the register file, using the function code (bits 5:0, which is the funct field, of the instruction) to generate the ALU function.
4. The result from the ALU is written into the register file using bits 15:11 of the instruction to select the destination register ($t1).

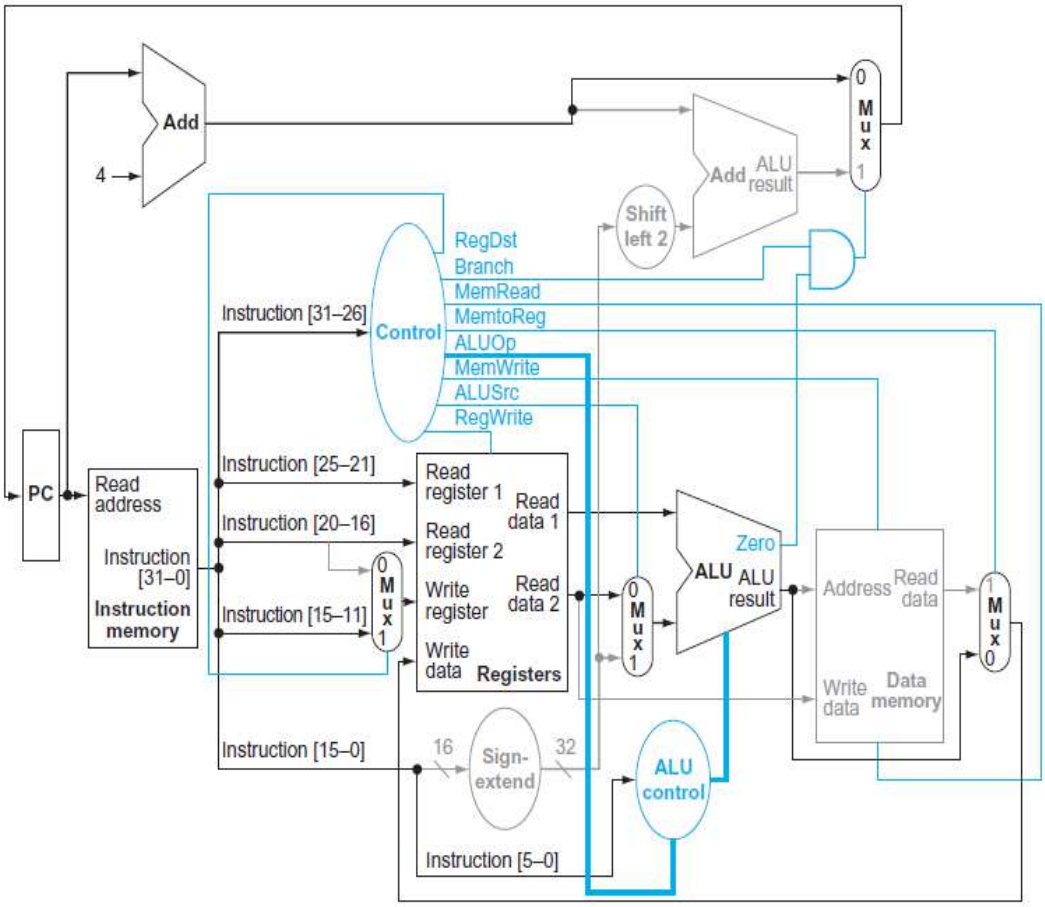


Fig 3.9 The datapath in operation for an R-type instruction

3.3.3.2 Load instruction operating in five steps:

1. An instruction is fetched from the instruction memory, and the PC is incremented.
2. A register ($t2) value is read from the register file.
3. The ALU computes the sum of the value read from the register file and the sign-extended, lower 16 bits of the instruction (offset).
4. The sum from the ALU is used as the address for the data memory.
5. The data from the memory unit is written into the register file; the register destination is given by bits 20:16 of the instruction ($t1).

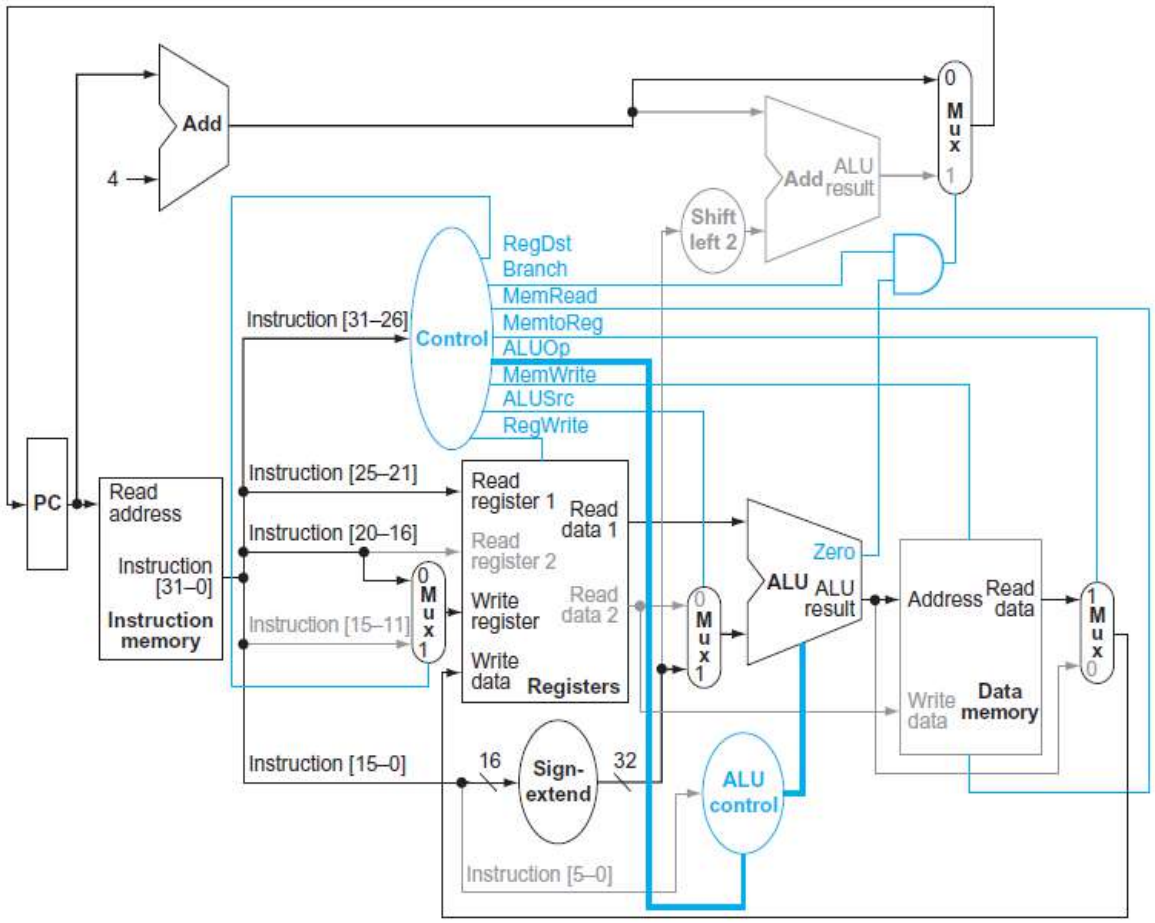


Fig 3.10 The datapath in operation for a load instruction.

3.3.3.3 Branch-on-equal instruction operation in five steps:

Branch on equal instruction has the following steps to execute a branch instruction such as shown in fig 3.11

beq, $t1, $t2, offset

1. An instruction is fetched from the instruction memory, and the PC is incremented.
2. Two registers, $t1 and $t2, are read from the register file. The data path in operation for a branch-on-equal instruction.
3. The ALU performs a subtract on the data values read from the register file. The value of PC + 4. It is added to the sign-extended, lower 16 bits of the instruction (offset) shifted left by two; the result is the branch target address.
5. The Zero result from the ALU is used to decide which adder result to store into the PC.

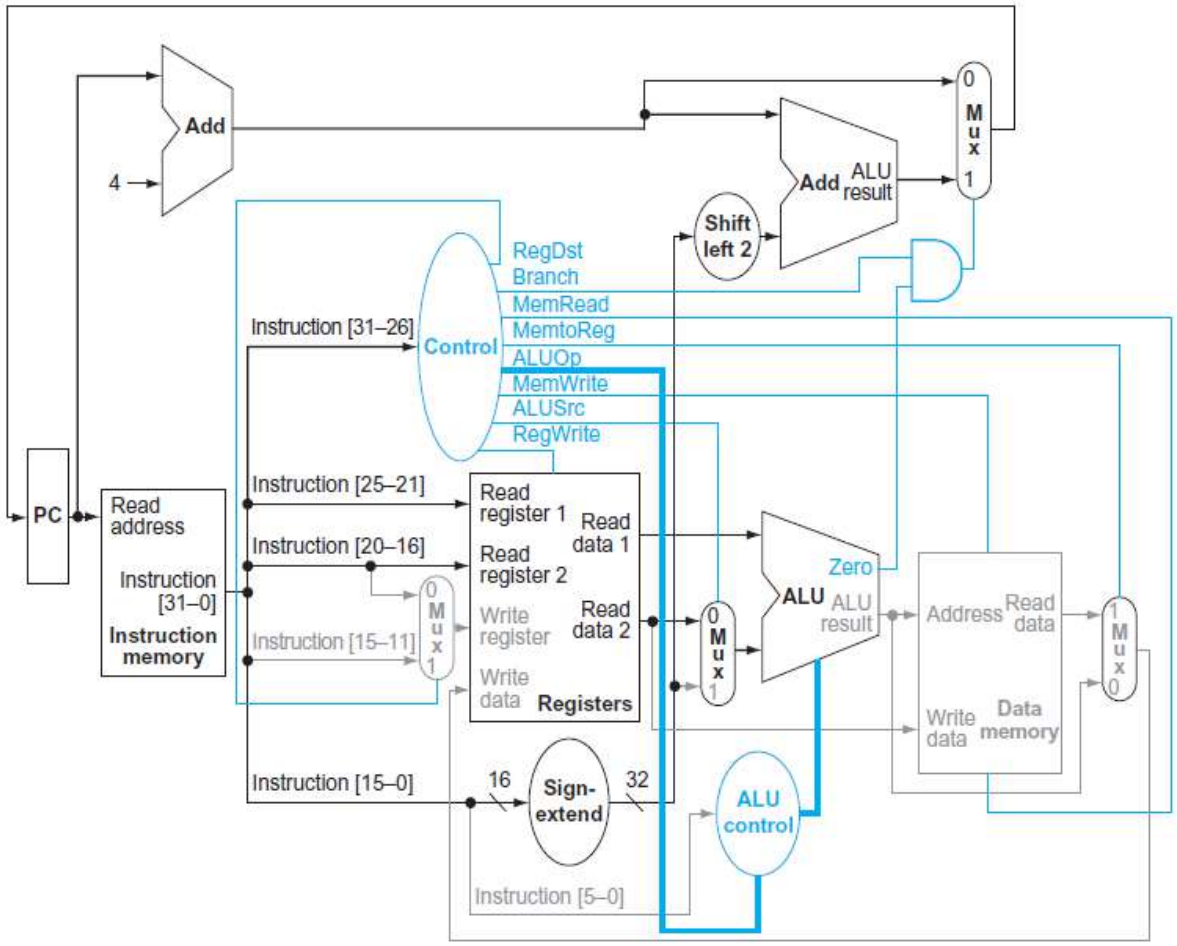


Fig 3.11 The datapath in operation for a branch-on-equal instruction.

3.3.4 Single cycle Implementation:

- ✓ Single cycle implementation also called single clock cycle implementation.
- ✓ Single cycle implementation is an instruction which is executed in one clock cycle.
- ✓ For example, consider jump instruction to show how the basic datapath and control can be extended to handle other instruction in the instruction set.

Field	000010	address
Bit positions	31:26	25:0

Instruction format for the jump instruction

- ✓ The jump instructionlooks somewhat like a branch instruction but computes the target PC differently and is not conditional.
- ✓ Like a branch, the low-order 2 bits of a jump address are always 00_{two}.
- ✓ The next lower 26 bits of this 32-bit address come from the 26-bit immediate field in the instruction.
- ✓ The upper 4 bits of the address that should replace the PC come from the PC of the jump instruction plus 4.
- ✓ Thus, we can implement a jump by storing into the PC the concatenation of

1. The upper 4 bits of the current PC + 4
2. The 26-bit immediate field of the jump instruction

3. The bits 00_{two}

✓ An additional multiplexor (at the upper right) is used to choose between the jump target and either the branch target or the sequential instruction following this one.

✓ This multiplexor is controlled by the jump control signal.

✓ The jump target address is obtained by shifting the lower 26 bits of the jump instruction left 2 bits, effectively adding 00 as the low-order bits, and then concatenating the upper 4 bits of PC + 4 as the high-order bits, thus yielding a 32-bit address.

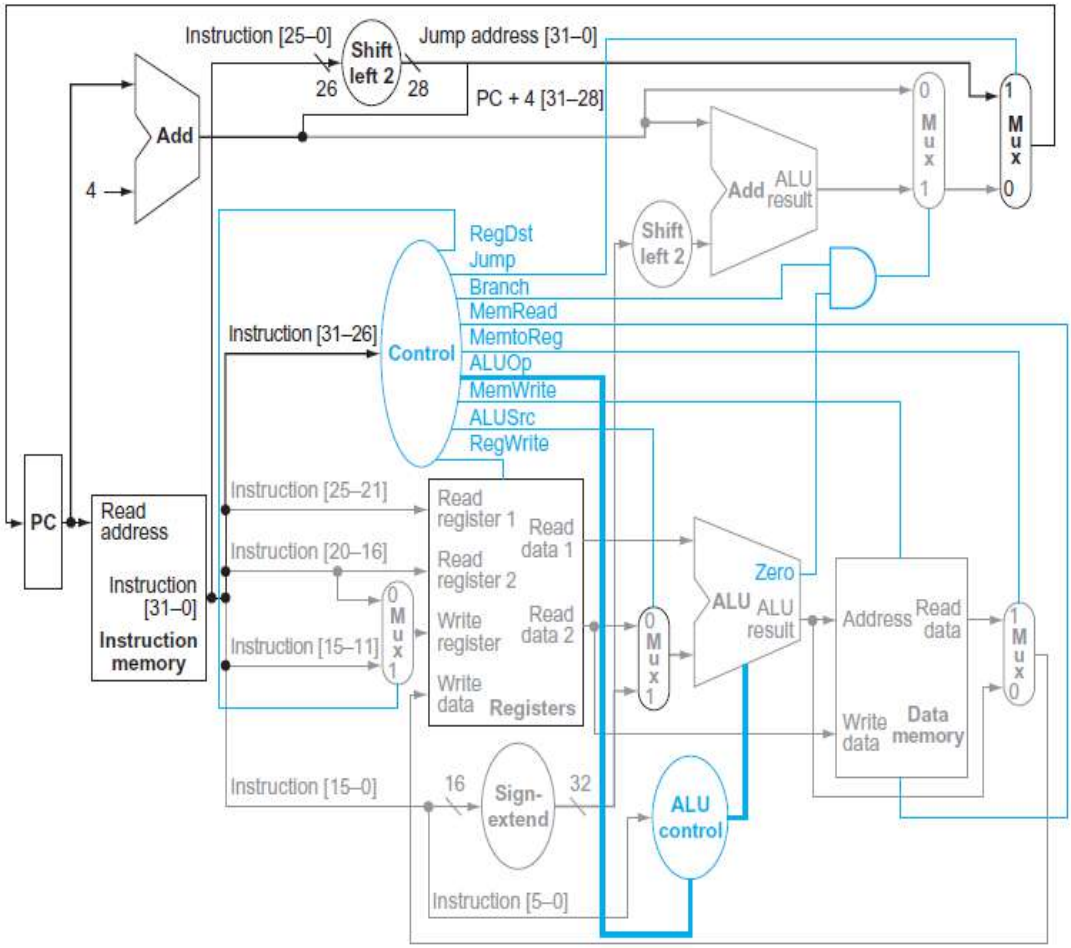


Fig 3.12 The simple control and datapath are extended to handle the jump instruction

Why single cycle implementation is not used today?

Single cycle implementation is not used mostly because of the following reasons:

1. It is inefficient.
2. Clock cycle have same length for every instruction.
3. Overall performance is very poor because it has too long clock cycle.

PIPELINING :-

Definition :-

Pipelining is an implementation technique in which multiple instructions are executed simultaneously by overlapping them in execution to save time and resource. The previous instruction will be in the execution phase when the current instruction is fetched from the memory.

NEED FOR PIPELINING :-

* <u>without a pipeline</u>, a computer processor fetches the first instruction from memory performs the operation mentioned in it, and then goes to fetch the next instruction from memory. while fetching the instruction, the arithmatic unit of the processor is idle. It must wait until it is loaded with next instruction.

* <u>with pipelining</u>, the computer architecture allows the next instructions to be fetched while the processor is performing arithmetic operations, holding them in a buffer close to the processor.

* The result is an increase in the number of instructions that can be performed during a given time period.

STAGES IN MIPS PIPELINING :-

The following are the various 5 stages in pipelining

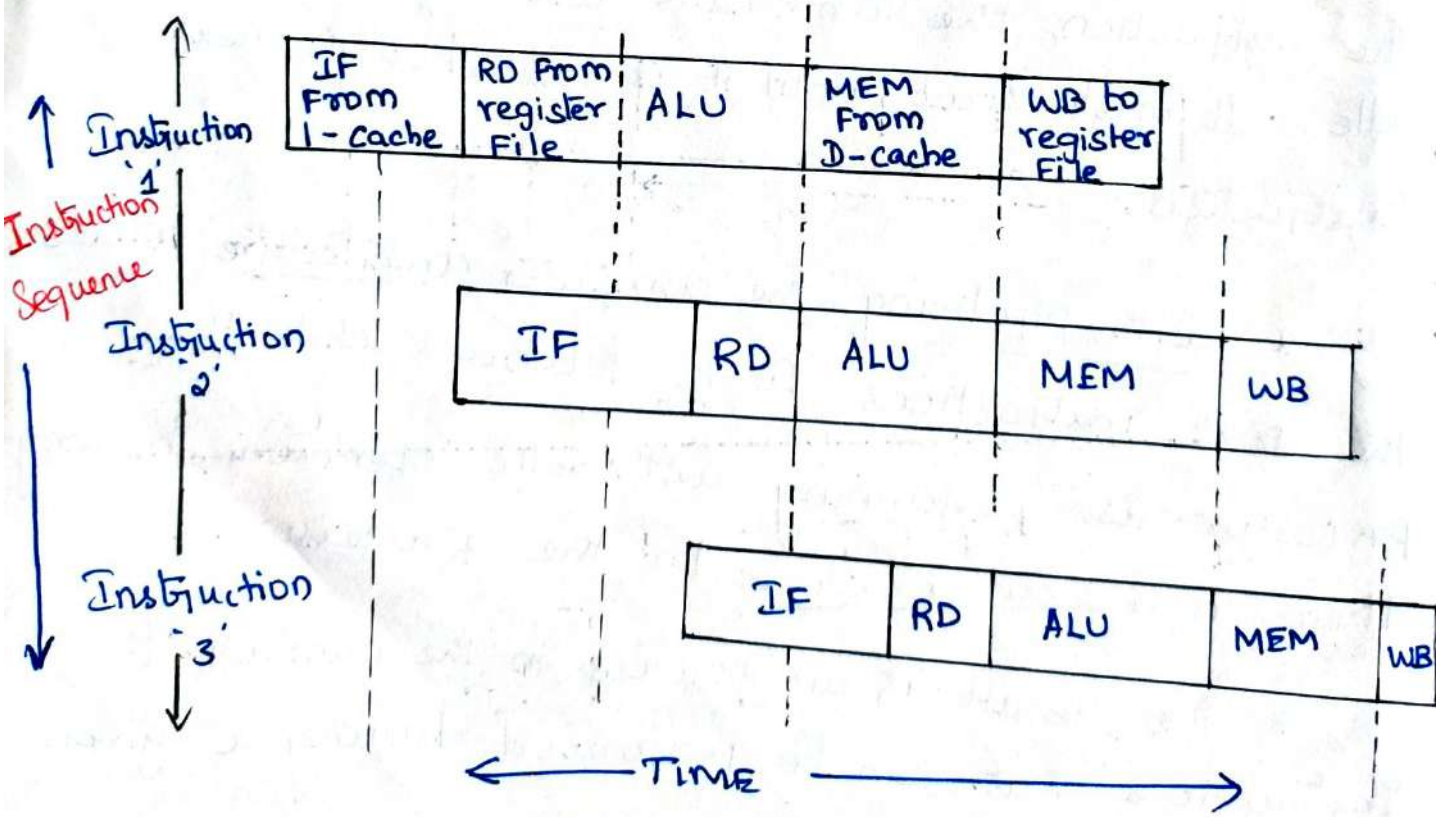
Instruction fetch (IF) :- Fetch instruction from memory

Instruction Decode (RD) :- Read register while decoding the instruction. The format of MIPS instruction allows reading and decoding to occur simultaneously.

Execute :- Execute the operation or calculate an address.

Memory access (MEM) :- Access an operant in data memory.

Write Back (WB) :- write the result into a register.

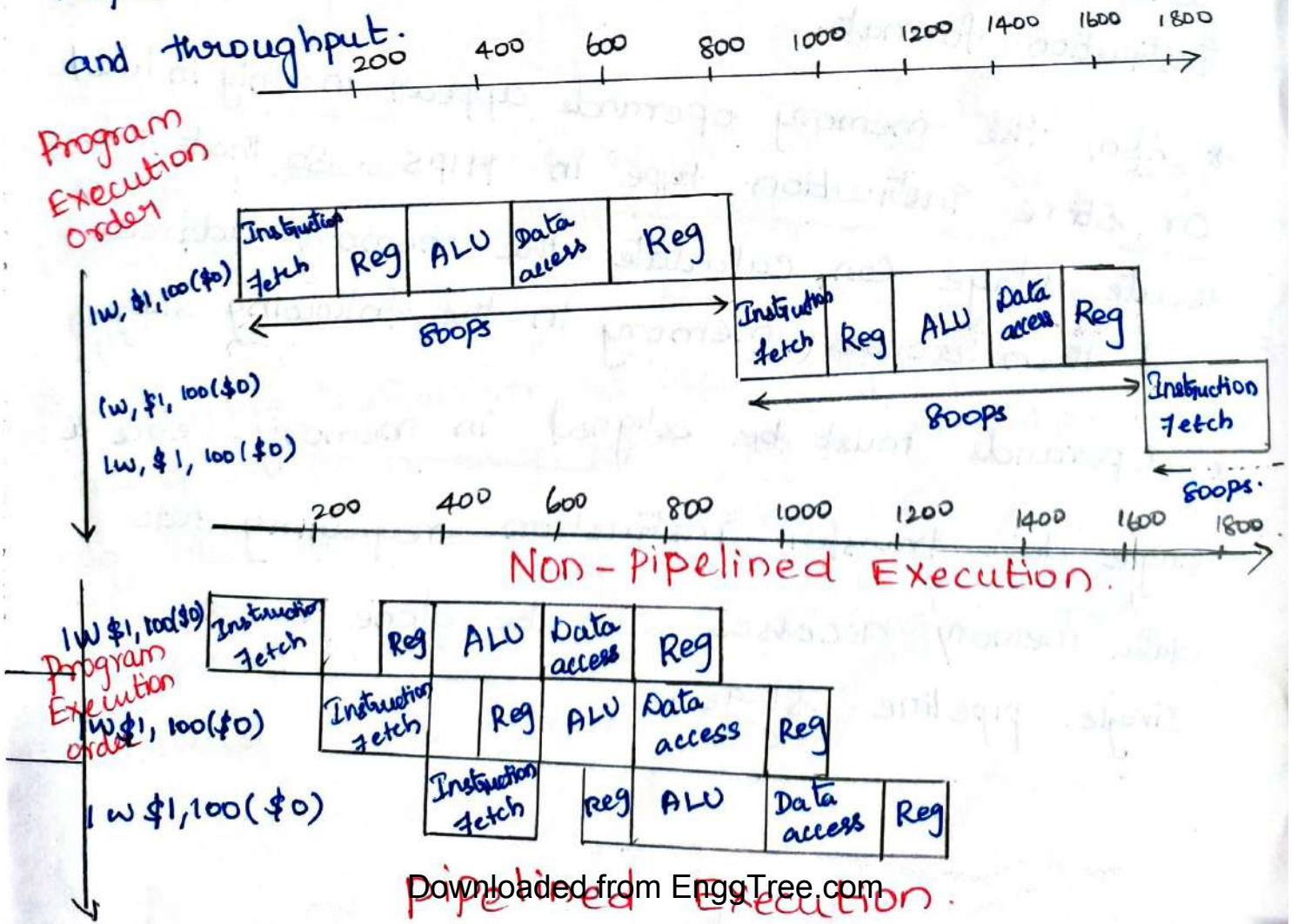


The pipelining speed can be manipulated using the expression.

$$\text{Time between Instructions}_{\text{pipelined}} = \frac{\text{Time between instruction}_{\text{non pipelined}}}{\text{Number of pipe stages.}}$$

* Pipeling improves performance by increasing instruction throughput. It is not decreasing the execution time of an individual instruction. but increases the number of instruction that complete its execution for a given time period.

* Thus the overall performance of the processor is improved both in terms of resource utilization and throughput.



DESIGNING INSTRUCTION SETS FOR PIPELINING :-

* The simplicity and generality of MIPS instructions are that they are of same length. This facilitates easy instruction fetching in the first stage of pipelining.

* MIPS has only a few instruction formats. In every instruction format, the source operand register is located at the same position in the instruction format.

* This symmetry eases the instruction decode stage by reading the register file simultaneously while the hardware is determining the type of instruction format.

* Also, the memory operands appear in only in load or store instruction type in MIPS. So that the execute stage can calculate the memory address and then access memory in the following stage.

* Operands must be aligned in memory. Hence a single data transfer instruction requiring two data memory accesses can be done in a single pipeline stage.

PIPELINE HAZARDS :

" Harads are situations that prevent the next instruction in the instruction cycle from being executing during its designated clock cycle. Hazards reduce the performance of the pipelining".

They are attempt to use same resource by two or more instructions at the same time.

Example :- In case of single memory is used for instruction and data access and when two instructions are accessing the same register one at instruction fetch stage and other at memory access stage. This leads to inconsistent data access.

Structural Hazard :-

* The hardware cannot support the combination of instructions that want to execute in the same clock cycle.

* The MIPS instruction set was designed to be pipelined making it fairly easy for designers to avoid structural hazards when designing a pipeline.

* Support, however that had a single memory instead of two memories.

* Data hazards occur in register files due to inconsistencies in file.

* This is an occurrences in which a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction is not yet available.

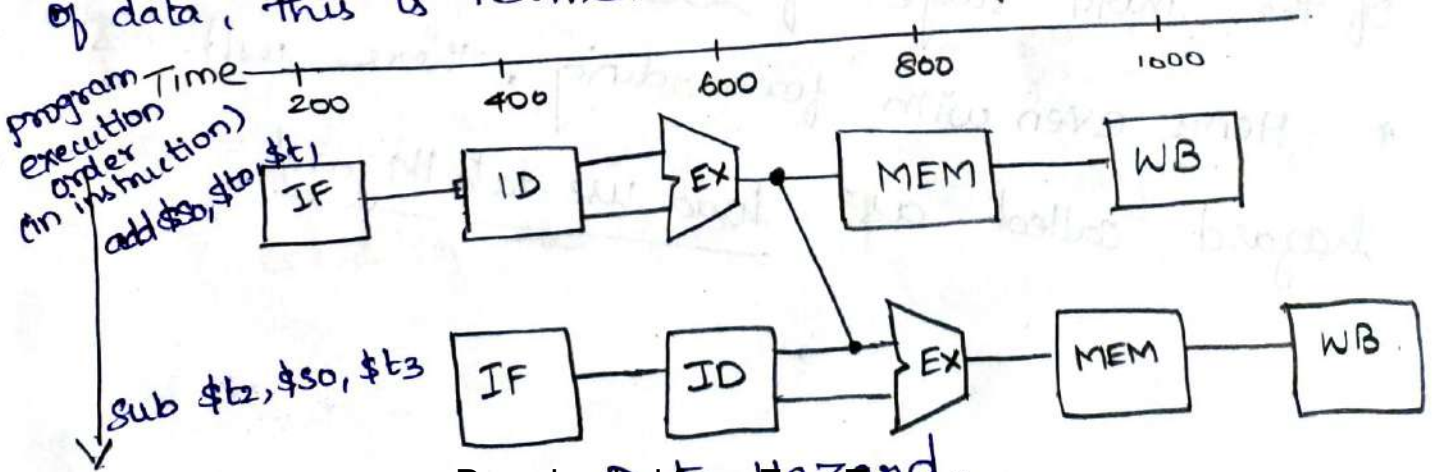
* In other words, data hazard occur when the pipeline must be stalled because one step must wait for another to complete. This is due to the data dependence.

Example:- Consider the following instructions

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```

Here the sub instruction uses the result to add instruction ($s0). The add instruction cannot not write its result until the fifth stage. This results in wasting three clock cycles in the pipeline.

* Since the stall occurs due to the non-availability of data, this is termed as data hazard.



Data Hazard

FORWARDING (OR) BYPASSING :- Solution to resolve data Hazard

* Forwarding also called Bypassing is a method of resolving data hazard by retrieving the missing data element from internal buffer rather than waiting for it to arrive from programmer-visible register or memory".

* This can be done by adding extra memory element or hardware that acts as an internal buffer.

* Forwarding cannot be a universal solution to solve data hazard. consider the following instructions

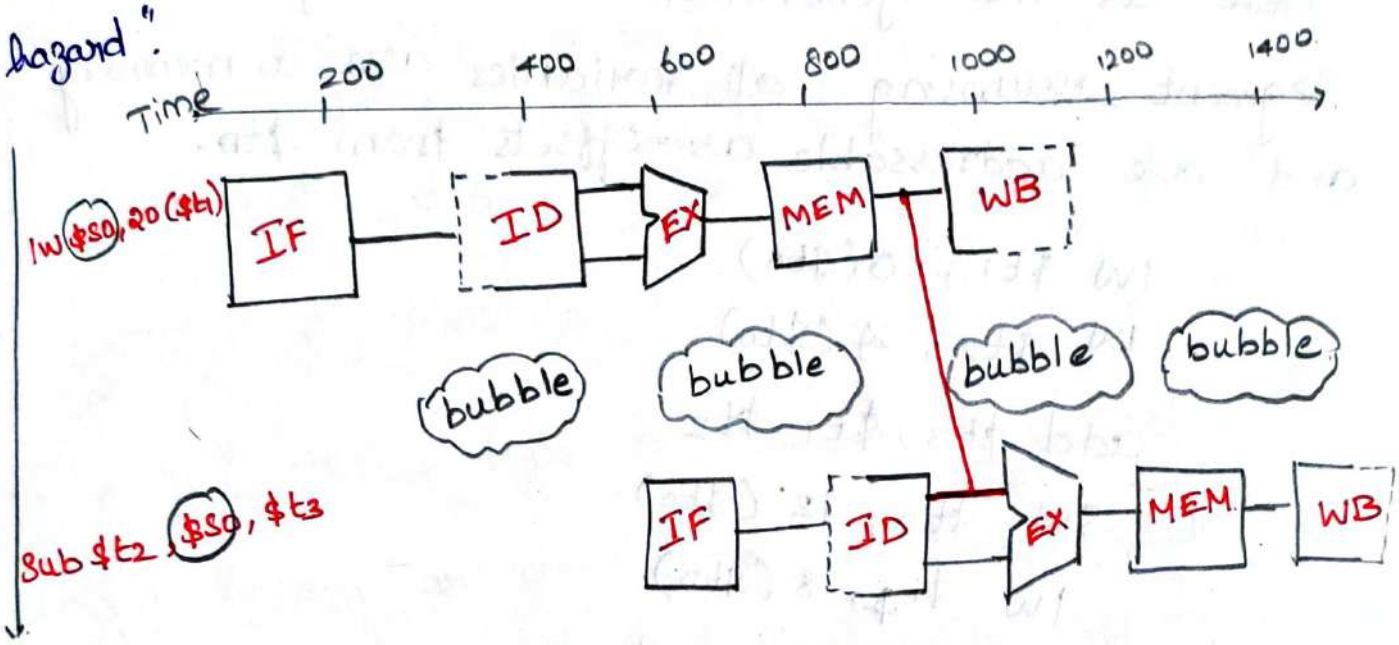
Eg:

```
lw $s0, 20($t1)
sub $t2, $s0, $t3
```

* The desired data would be available only after the fourth stage of the first instruction in the dependence, which is too late for the input of the third stage of "sub".

* Hence even with forwarding, there will be hazard called as load use data Hazard.

* A specific form of data hazard in which the data requested by an load instruction has not yet become available when it is requested. This is Load-use data hazard".



LOAD - USE DATA HAZARD.

* The stall mentioned is called bubble or pipeline stall. A pipeline stall is a delay in execution of an instruction in order to resolve a hazard.

* During the decoding stage the control unit will determine if the decoded instruction reads from a register that the instruction currently in the execution stages writes to.

PROBLEM :-
REORDERING CODE TO AVOID PIPELINE STALLS :-

Consider the following code segment in C:

```
a = b + e ;
c = b + f ;
```

Here is the generated MIPS code for this segment, assuming all variables are in memory and are addressable as offsets from $t0.

```
lw $t1, 0($t0)
lw $t2, 4($t0)
add $t3, $t1, $t2
sw $t3, 12($t0)
lw $t4, 8($t0)
add $t5, $t1, $t4
sw $t5, 16($t0)
```

Find the hazards in the preceding code segment and reorder the instructions to avoid any pipeline stall.

Soln Both add instructions have a hazard because of their respective dependence on the immediately preceding lw instruction.

Bypassing eliminates several other potential hazards, including the dependence of the first add on the first lw and any hazard for store instruction. Moving up the third lw instruction to become the third instruction eliminates both hazards.

```
lw   $t1, 0($to)
lw   $t2, 4($to)
lw   $t4, 8($to)
add  $t3, $t1, $t2
sw   $t3, 12($to)
add  $t5, $t1, $t4
sw   $t5, 16($to)
```

CONTROL HAZARDS :-

* The third type of hazard is called a control hazard, arising from the need to make a decision based on the results of one instruction while others are executing.

* Control hazard is called branch hazard consider the branch instruction.

* Begin fetching the instruction following the branch on the very next clock cycle.

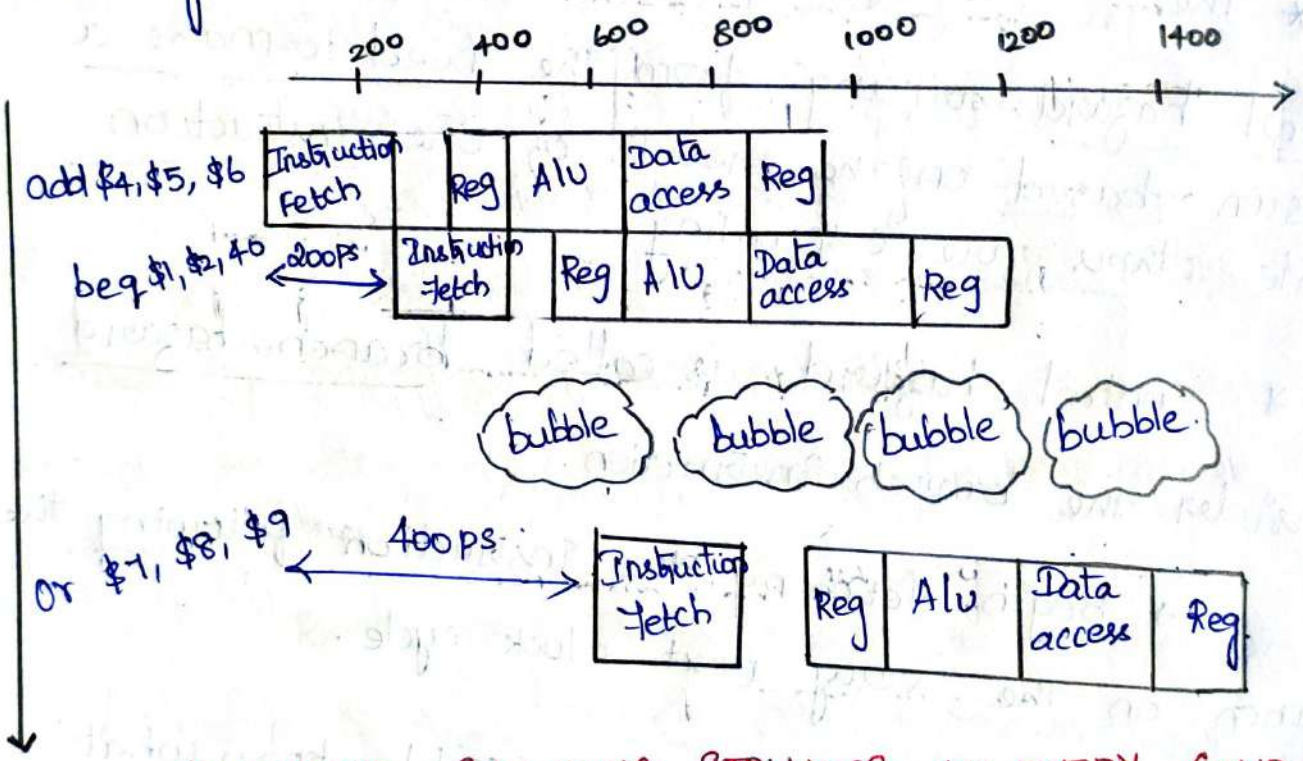
* The pipeline cannot possibly know what the next instruction should be since it only just received the branch instruction from memory.

* TO avoid stall fetch a branch, that waiting until the pipeline determines the outcome of the branch and knows what instruction addresses to fetch from.

* Extra hardware can test register, calculate the branch address and update the pc during the second stage of the pipeline.

* Even with this extra hardware, the pipeline involving conditional branches.

* The lw instructions, excuted if the branch fails, is stalled one extra 200ps clock cycles before starting.



PIPELINE SHOWING STALLING ON EVERY CONDITIONAL BRANCH AS SOLUTION TO CONTROL HAZARDS.

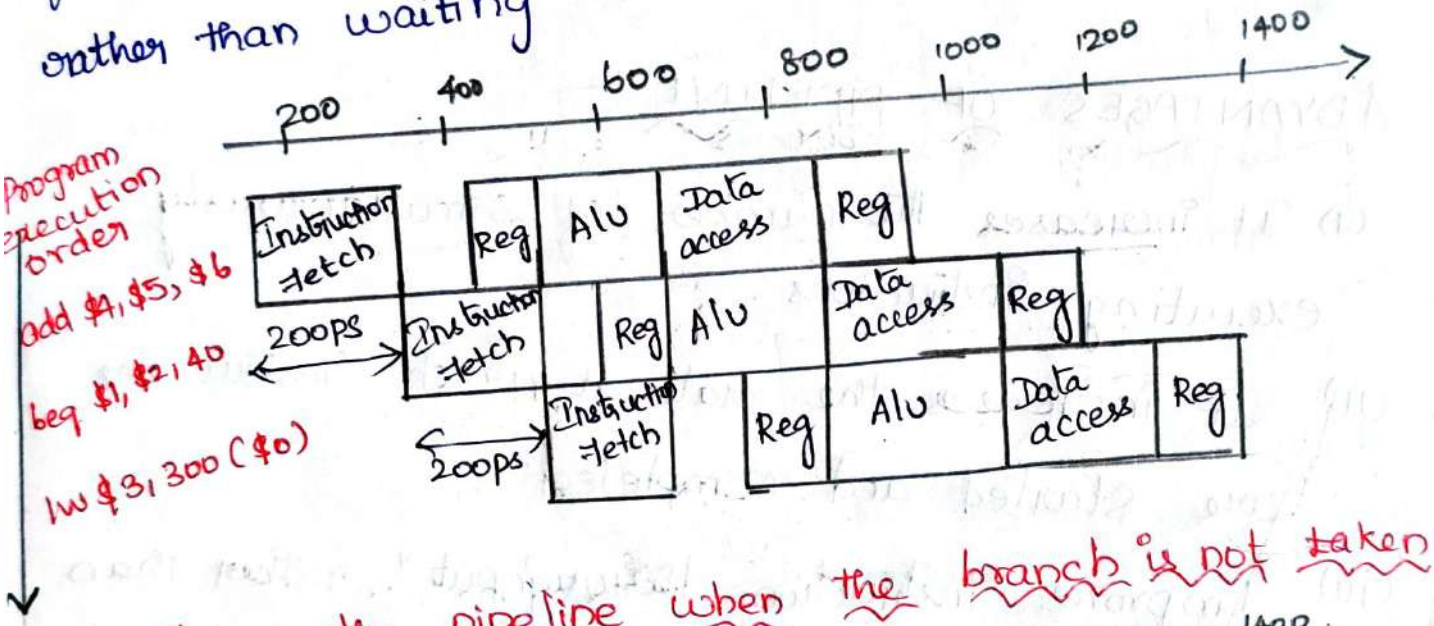
→ If cannot resolve the branch in the second stage, as is often the case for longer.

→ pipelines and it is too high cost for more computers.

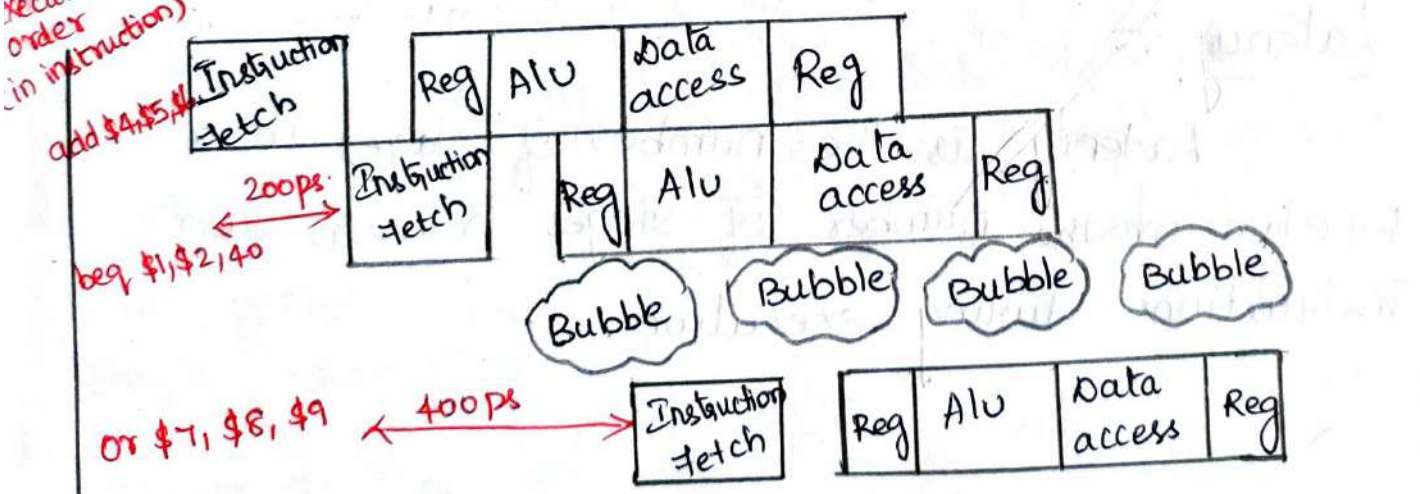
* So need to focus another method to resolve the control hazard that is called branch prediction.

BRANCH PREDICTION:

Branch prediction is a method of resolving a branch hazard that assume a given outcome for the branch and proceeds from that assumption rather than waiting to ascertain the actual outcome.



Shows the pipeline when the branch is not taken.



Shows the pipeline when the branch is taken.

Dynamic hardware Prediction :-

* Dynamic hardware predictors make their guesses on the behavior of each branch and may change predictions for a branch over life of a program.

* One popular approach to dynamic prediction of branches is keeping a history for each branch as taken or untaken.

ADVANTAGES OF PIPELINE :-

(i) It increases the number of simultaneously executing instructions.

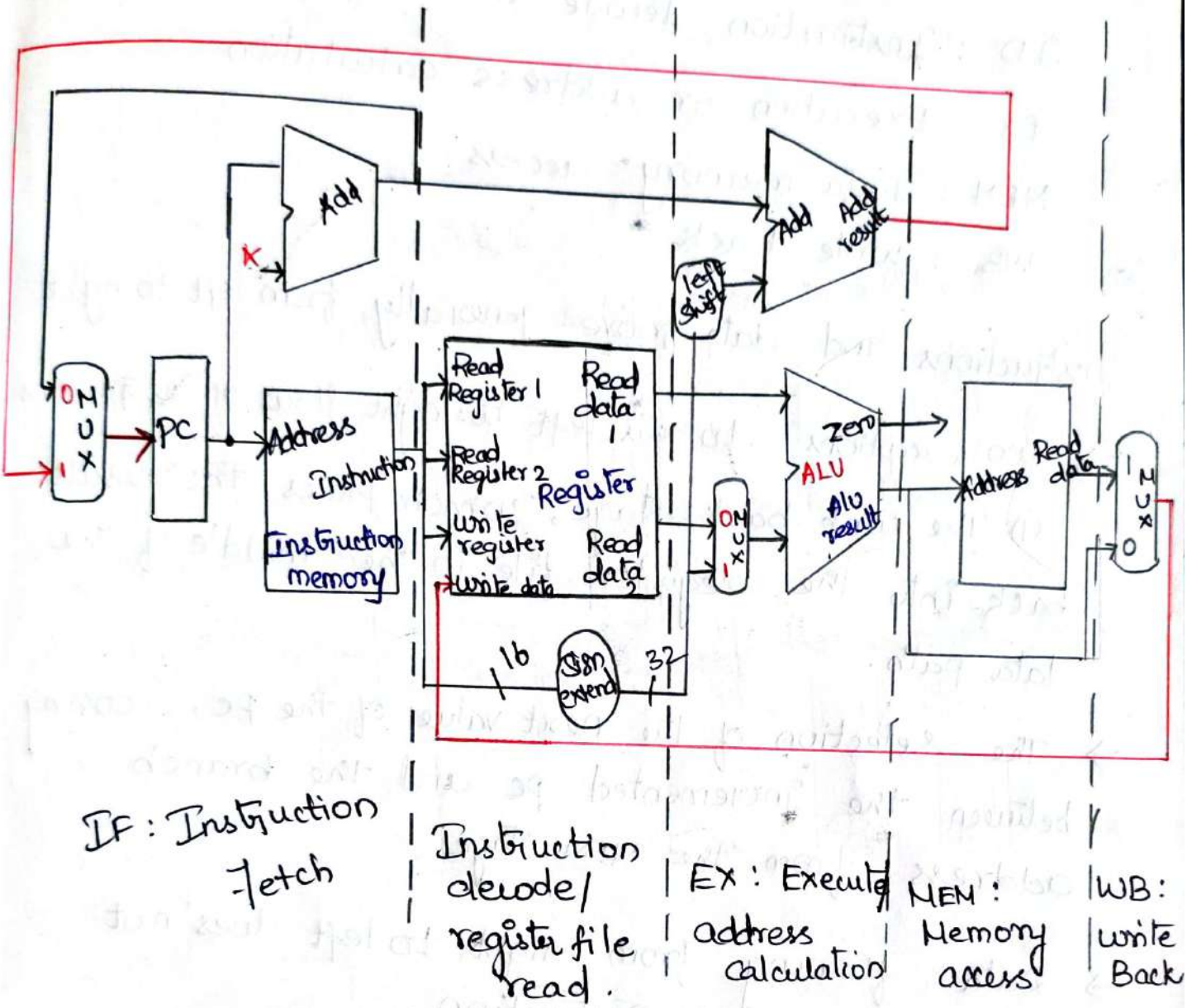
(ii) It increases the rate at which instructions are started and completed.

(iii) Improves instruction throughput rather than individual execution or latency.

Latency :-

Latency is the number of stages in a pipeline or the number of stages between two instructions during execution.

PIPELINED DATAPATH AND CONTROL :-



A Single-cycle Datapath.

· The division of an instruction into 5 stages means a five stage pipeline which in turn means that up to five instructions will be in execution during any single clock cycle.

Stages of instruction execution :-

IF : Instruction fetch

ID : Instruction decode and Register File read

EX : Execution or address calculation

MEM : Data memory access.

WB : write back.

Instructions and data move generally from left to right

→ Two exceptions to this left to right flow of instructions

(i) The write back stage, which places the result back into the register file in the middle of the data path.

→ The selection of the next value of the PC, choosing between the incremented pc and the branch address from the MEM stage.

→ Data flowing from right to left does not affect the current instruction.

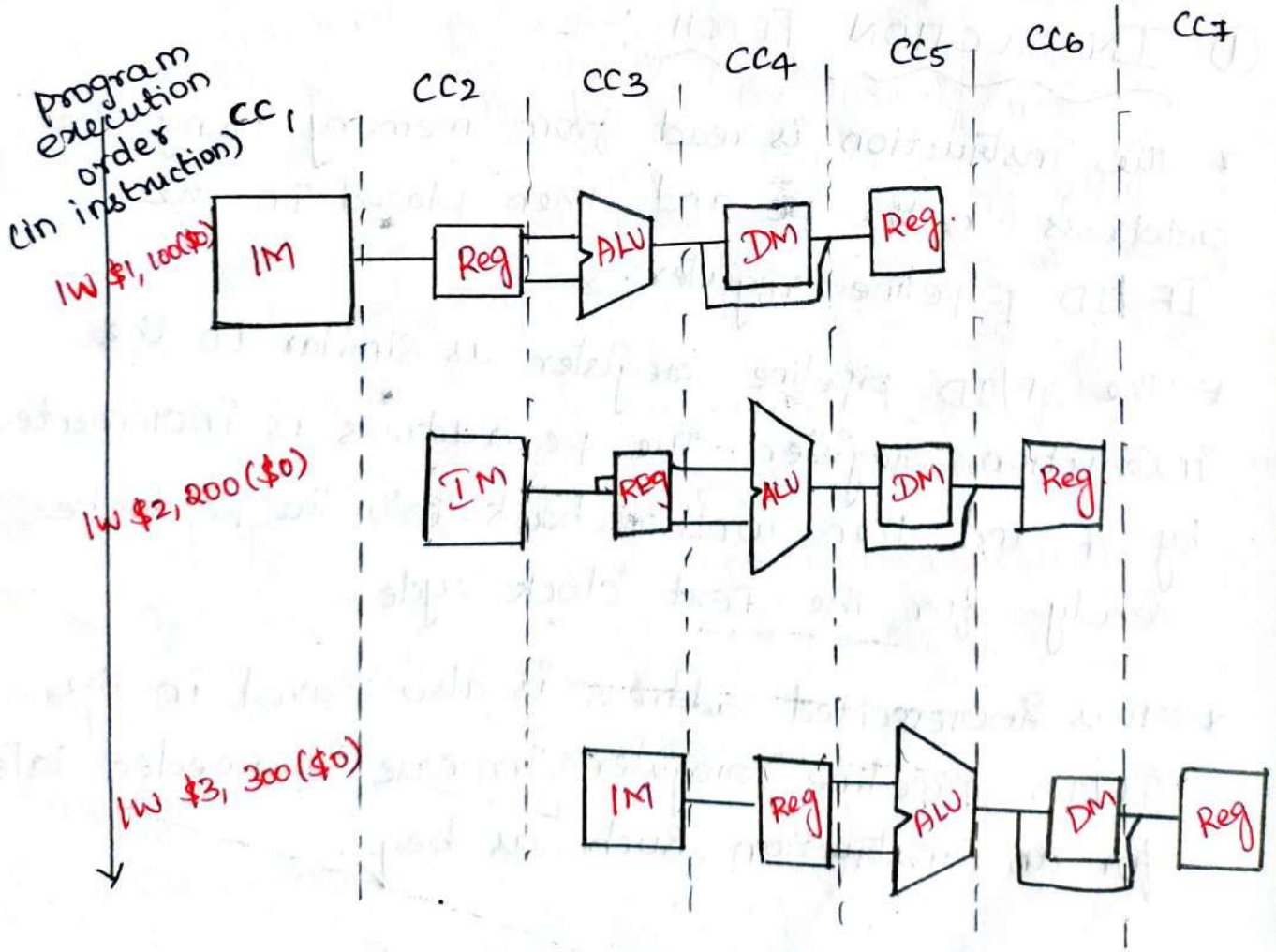
Note that (a) The first right to left flow of data can lead to data hazard.

(b) The second leads to control hazard.

Three load word instructions are

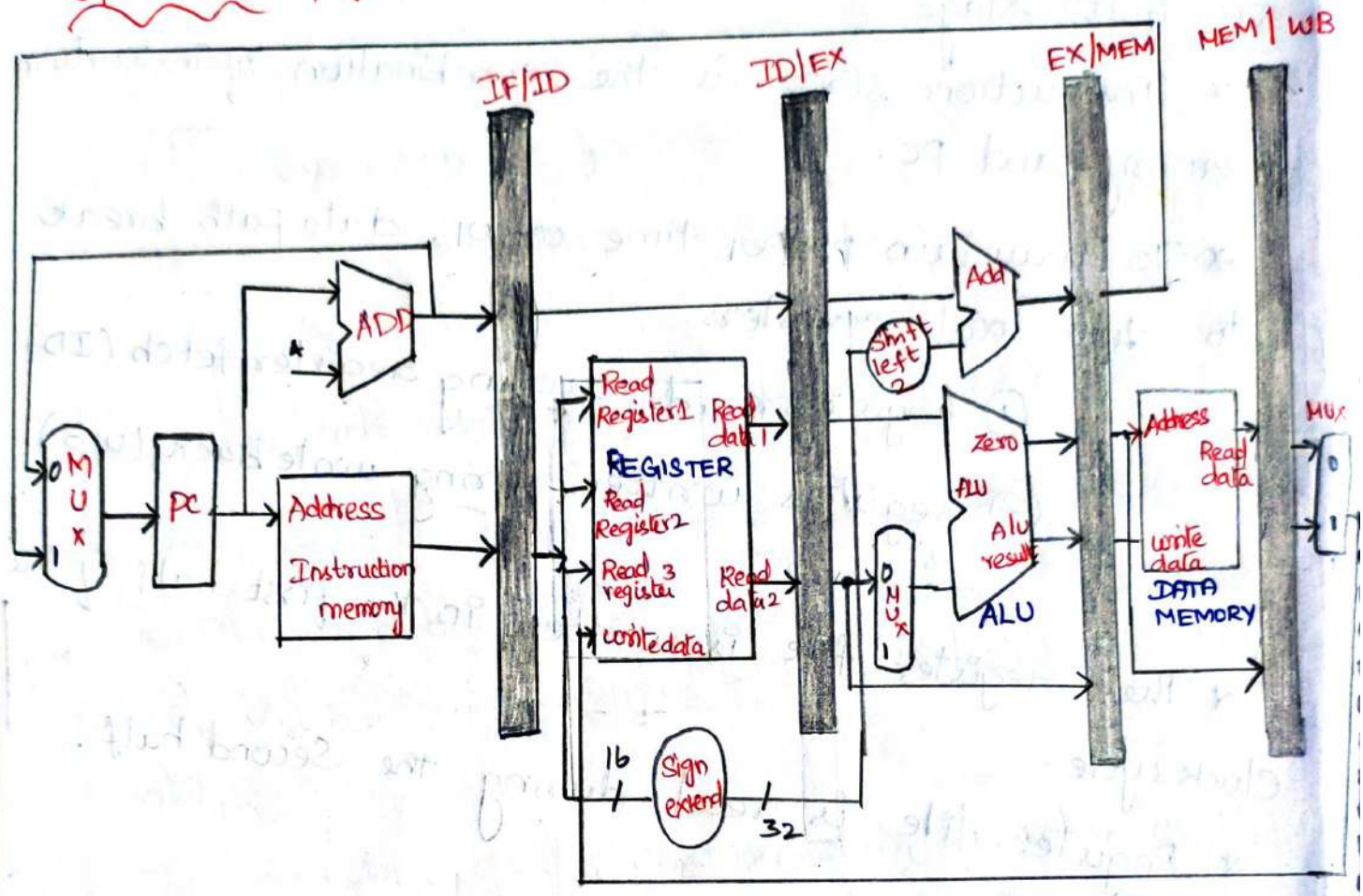
1. lw $1, 100 ($0)
2. lw $2, 200 ($0)
3. lw $3, 300 ($0)

* Each is labelled by the physical resource used in that stage.
* Instruction stage is the combination of instruction memory and PC.
* To maintain proper time order, data path break into two local registers
  1. Registers read during register fetch (ID)
  2. Registers written during write back (WB).
* The register file is written in the first half of the clock cycle.
* Register file is read during the second half.



INSTRUCTIONS EXECUTED USING SINGLE CLOCK CYCLE DATAPATH.

OPERATIONS IN EACH STAGE OF PIPELINE :-



① INSTRUCTION FETCH :-

* The instruction is read from memory using the address in the pc and then placed in the IF/ID pipeline register.

* The IF/ID pipeline register is similar to the instruction register. The pc address is incremented by 4 and then written back into the pc to be ready for the next clock cycle.

* This incremented address is also saved in the IF/ID pipeline register in case it needed later for an instruction such as beq.

* The computer cannot know which type of instruction is being fetched, so it must prepare for any instruction, passing potentially needed information down the pipeline.

② INSTRUCTION DECODE AND REGISTER FILE READ :-

* The instruction portion of the IF/ID pipeline register supplying the 16 bit immediate field, which is sign-extended to 32 bit and the register numbers to read the two registers.

* All three values are stored in the ID/EX pipeline register, along with the incremented PC address.

* Transfer everything that might be needed by any instruction during a later clock cycle.

* These first two stages are executed by all instructions, since it is too early to know that type of instruction.

③ EXECUTE OR ADDRESS CALCULATION :-

* The load instruction reads the content of register 1 and the sign extended immediate from the ID/EX pipeline register and adds them using the ALU.

* That sum is placed in the EX/MEM pipeline register.

(4) MEMORY ACCESS :-

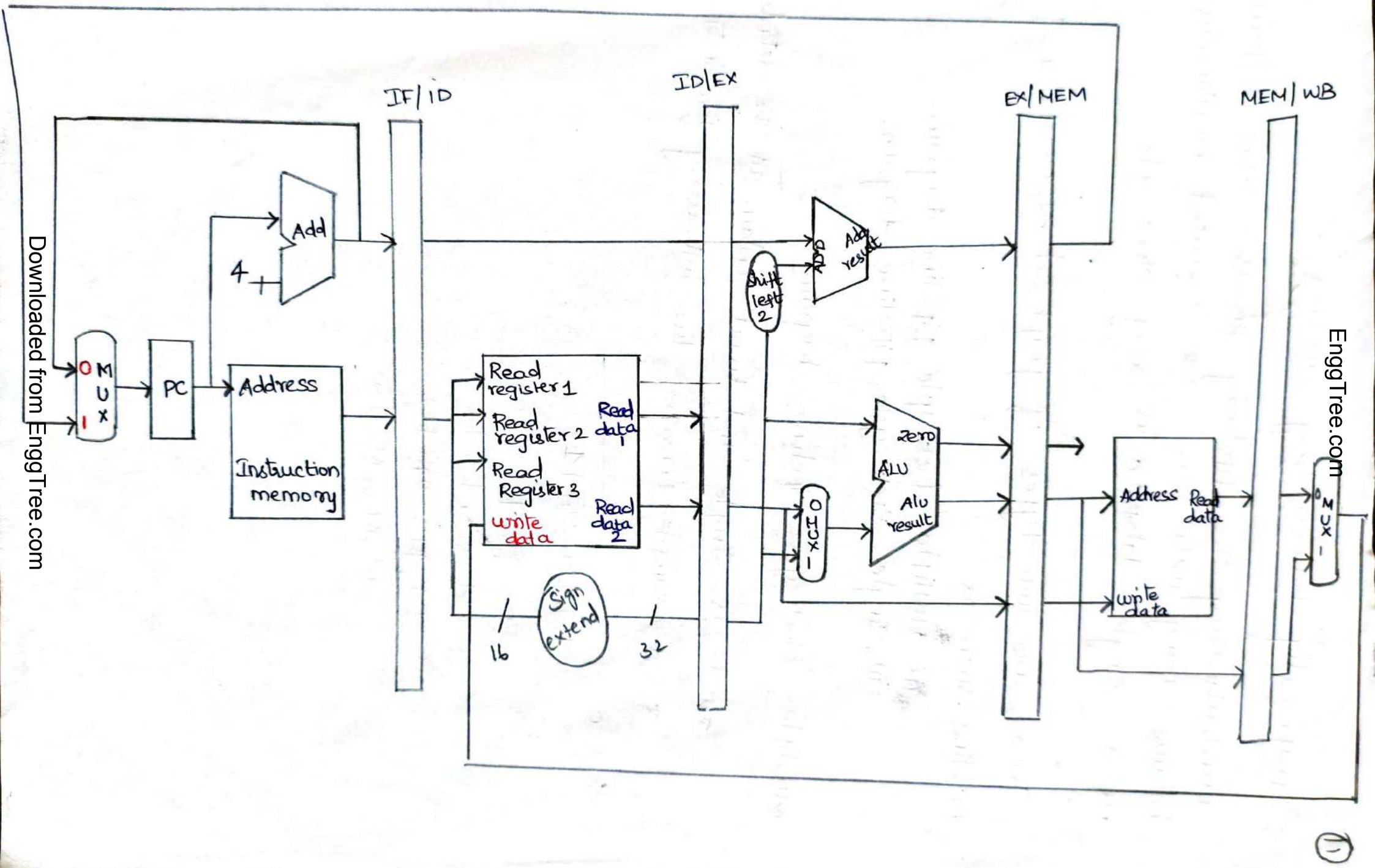
* The load instruction reading the data memory using the address from the EX/MEM pipeline register and loading the data into the MEM/WB pipeline register.

* The register containing the data to be stored was read in an earlier stage and stored in ID/EX.

* The only way to make the data available during the MEM stage is to place the data into the EX/MEM pipeline register in the EX stage, just as we store the effective address into EX/MEM.

(5) WRITE BACK :-

This involves reading the data from the mem/WB pipeline register and writing it into the register file.



Graphically Representing pipelines :-

* understanding the pipelining process is more difficult because many instructions are executed simultaneously in a single datapath in every clock cycle.

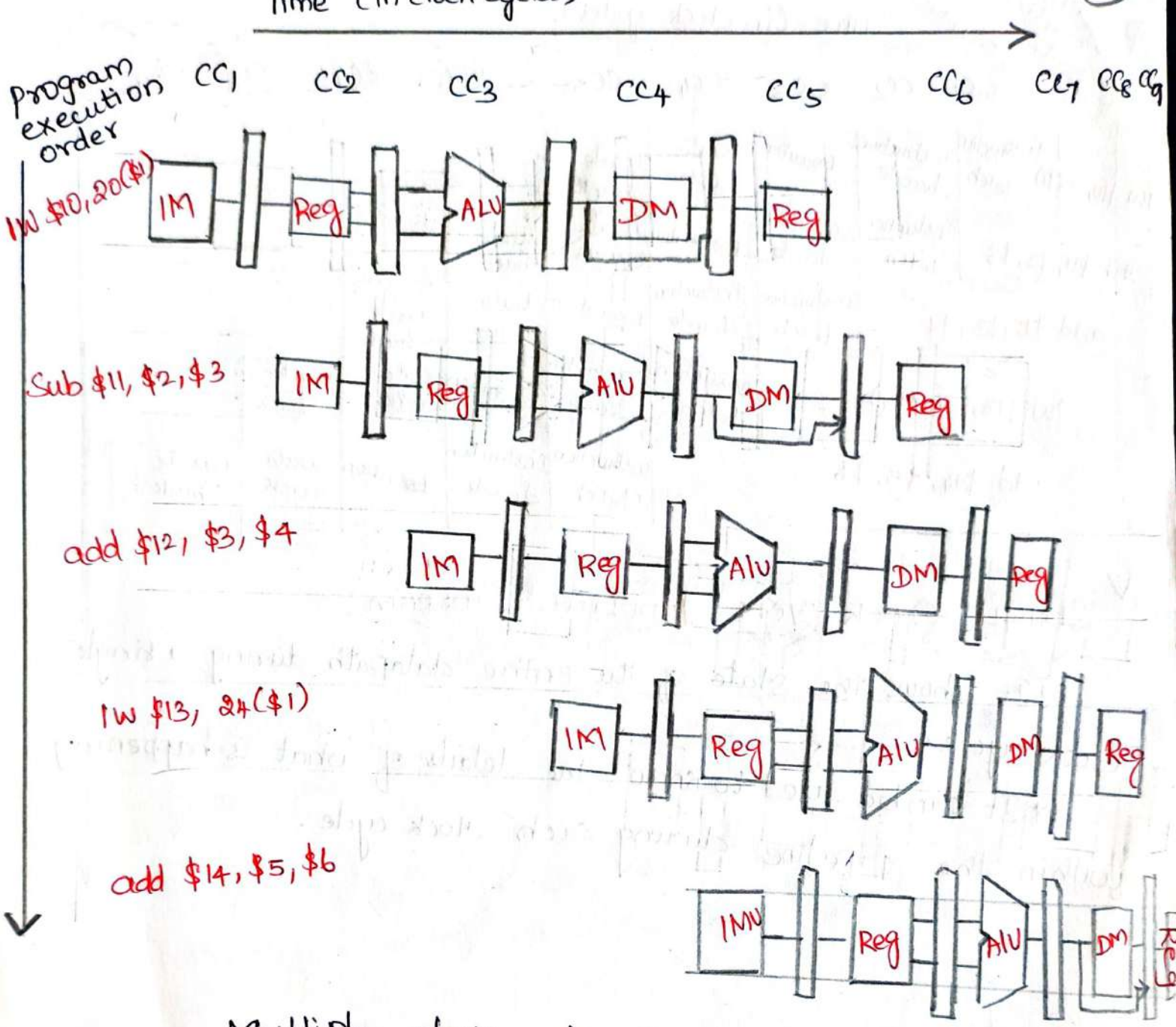
There are two types of graphical representations for pipeline such as

(i) Multiple clock cycle pipeline diagram.
(ii) Single clock cycle pipeline diagram.

Multiple clock cycle pipeline diagram :-

* It is simple and do not contain all the details
* For example consider the following five instruction Sequence.

Eg :
```
lw $10, 20($1)
Sub $11, $2, $3
add $12, $3, $4
lw $13, 24($1)
add $14, $5, $6
```



Multiple-clock cycle pipeline diagram of Five instruction.

* This style of pipeline representation shows the complete execution of instructions in a single picture.

* Instructions are listed in instruction execution order from top to bottom, and clock cycles move from left to right.

HANDLING OF DATA HAZARD & CONTROL HAZARD.

DATA HAZARDS :-

"Data Hazards occur when the pipeline must be stalled because one step must wait for another to complete".

FORWARDING OR BYPASSING :-

↳ Method to resolve a data hazard by retrieving the missing data element from internal buffer rather than waiting for it to arrive from programmer visible register to memory.

↳ This can be done by adding extra memory element or hardware that acts as an internal buffer.

Example Consider the following code.

```
Sub $2, $1, $3
and $12, $2, $5
or  $13, $6, $2
add $14, $2, $2
sw  $15, 100[$2]
```

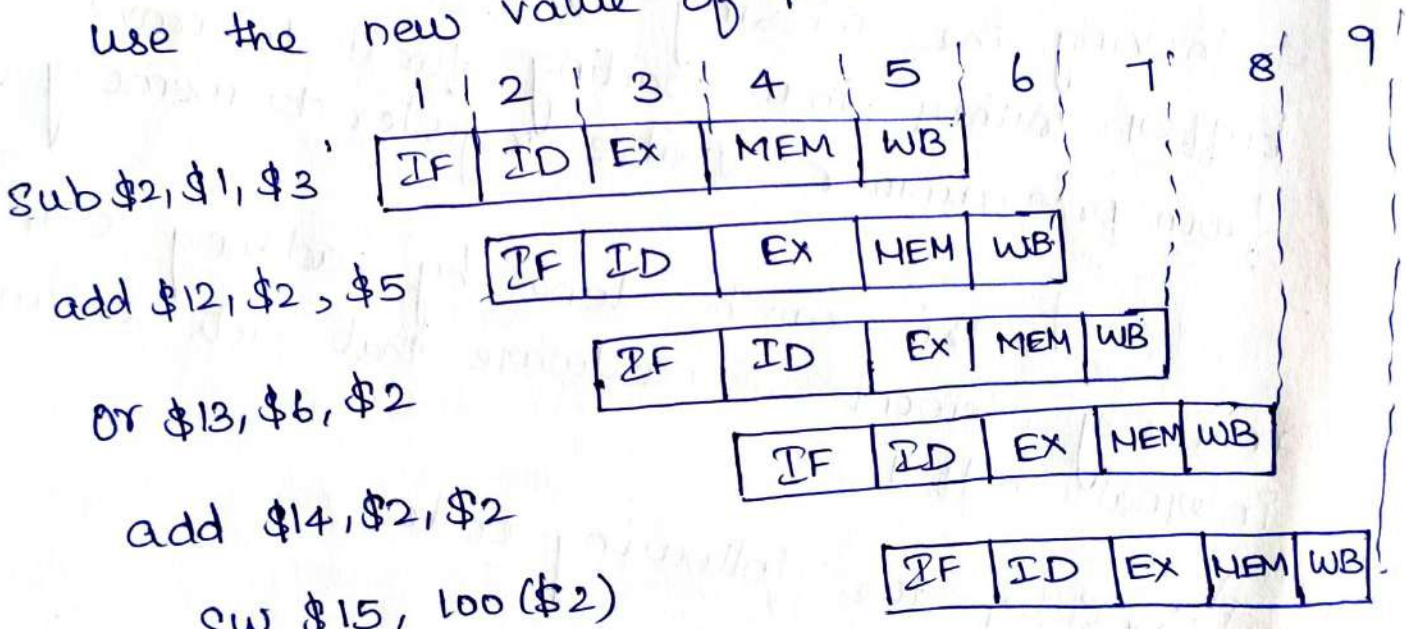
There are several dependences in the code fragment

* The First instruction, Sub, stores a value into $2.

* That register is used as a source in the rest of the instruction this is no problem for 1-cycle and multicycle data path.

* Each instruction executes completely before the next begins.

* This Ensures that instruction 2 through 5 above use the new value of $2.



* The SUB does not write to register $2 until clock cycle 5 causing a data hazard in our pipelined datapath.

* The AND reads register $2 in cycle 3. Since SUB hasn't modified the register yet, this is the old value of $2.

* The OR instruction uses register $2 in cycle 4, again before its actually updated by SUB.

To avoid data hazard, rewrite the instructions.

(Sll means stall)

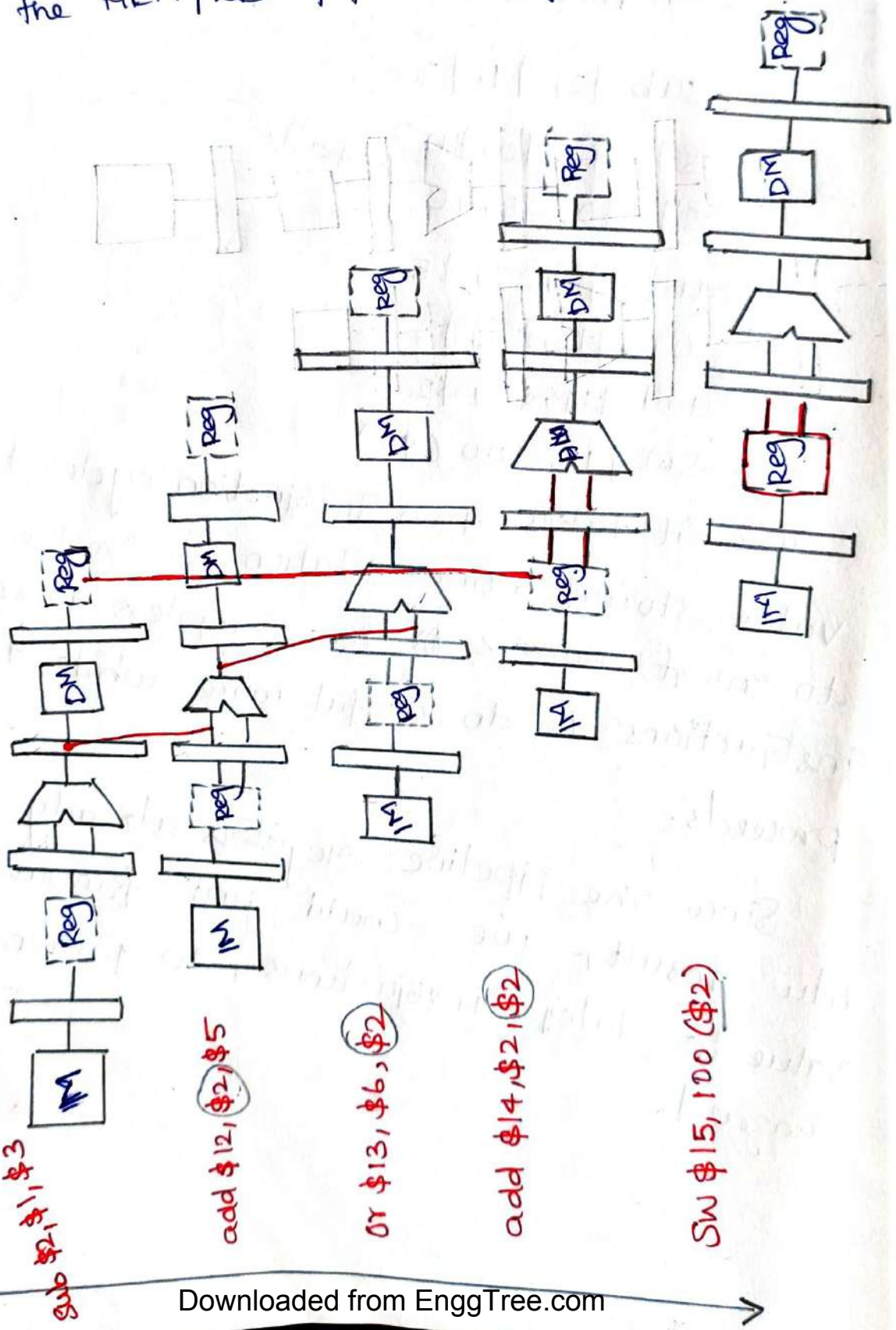
```
Sub $2, $1, $3
Sll $0, $0, $0   } stall.
Sll $0, $0, $0   }
and $12, $2, $5
 or $13, $6, $2
add $14, $2, $2
 SW $15, 100 ($2)
```

Since it takes two instruction cycles to get the value stored, one solution is for the assembler to insert no-ops or for compilers to reorder instructions to do useful work while the pipeline proceeds.

Since the pipeline registers already contain the ALU result, we could just forward the value to later instructions, to prevent data hazard.

* In clock cycle 4, AND instruction can get the value of $1 - $3, from the EX/MEM pipeline register used by SUB.

* Then in cycle 5, the OR can get that same result from the MEM/WB pipeline register being used by SUB.

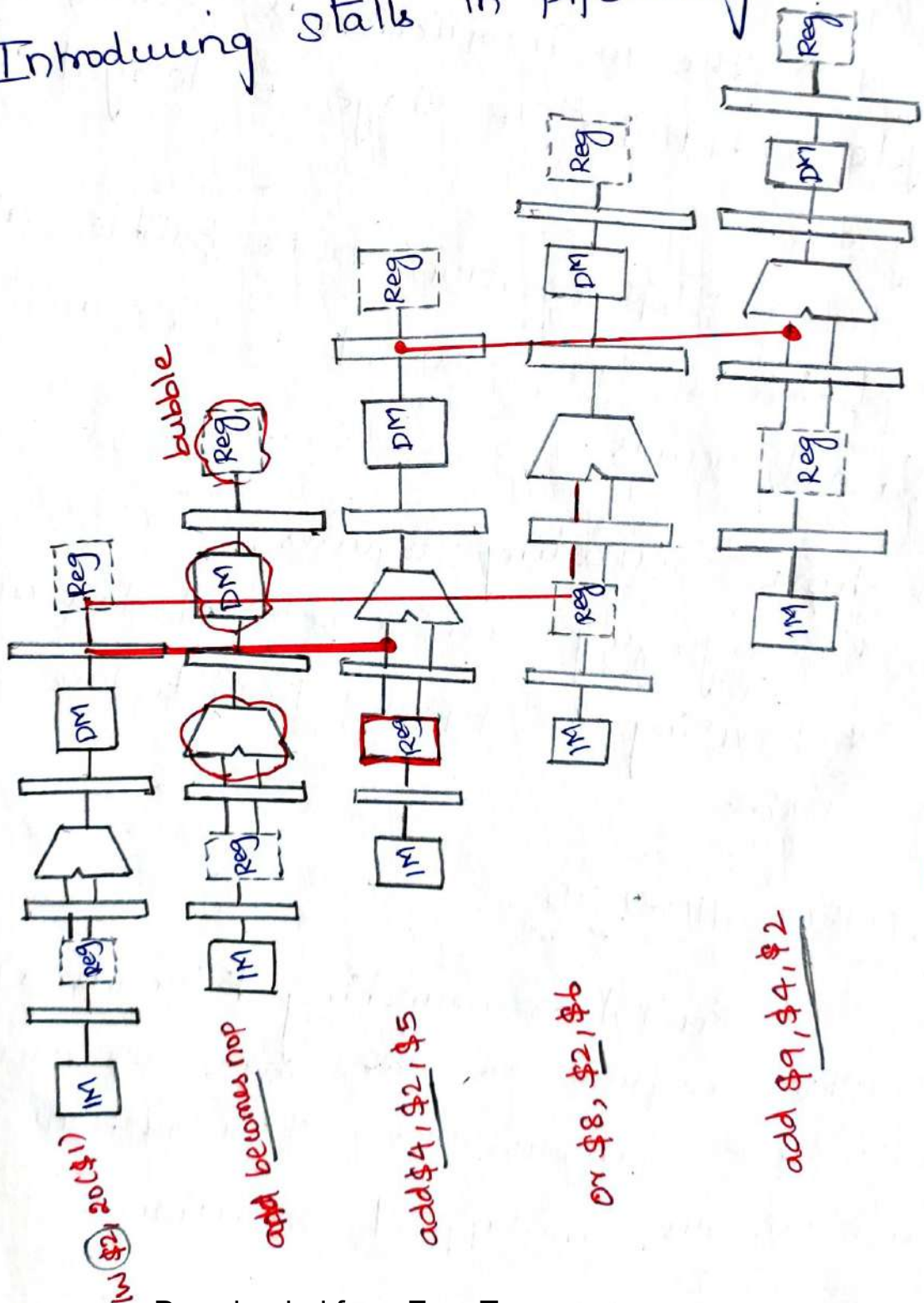


Pipelined dependencies.

"Forward the data as soon as it is available to any units that need it before it is available to read from the register file. This is forwarding in data hazards.

STALLING :-

Introducing stalls in pipelining :-



INTRODUCING STALLS IN PIPELINING.

A bubble is inserted beginning in clock cycle 4, by changing the AND instruction to a nop (no operation)

* Note that the and instruction is really fetched and decoded in clock cycle '2' and '3' but its EX stage is delayed until clock cycle 5.

* The or instruction is fetched in clock cycle 3, but its IF stage is delayed until clock cycle 5.

* After insertion of the bubble, all the dependences go forward in time and no futher hazard occurs.

In short forwarding requires;

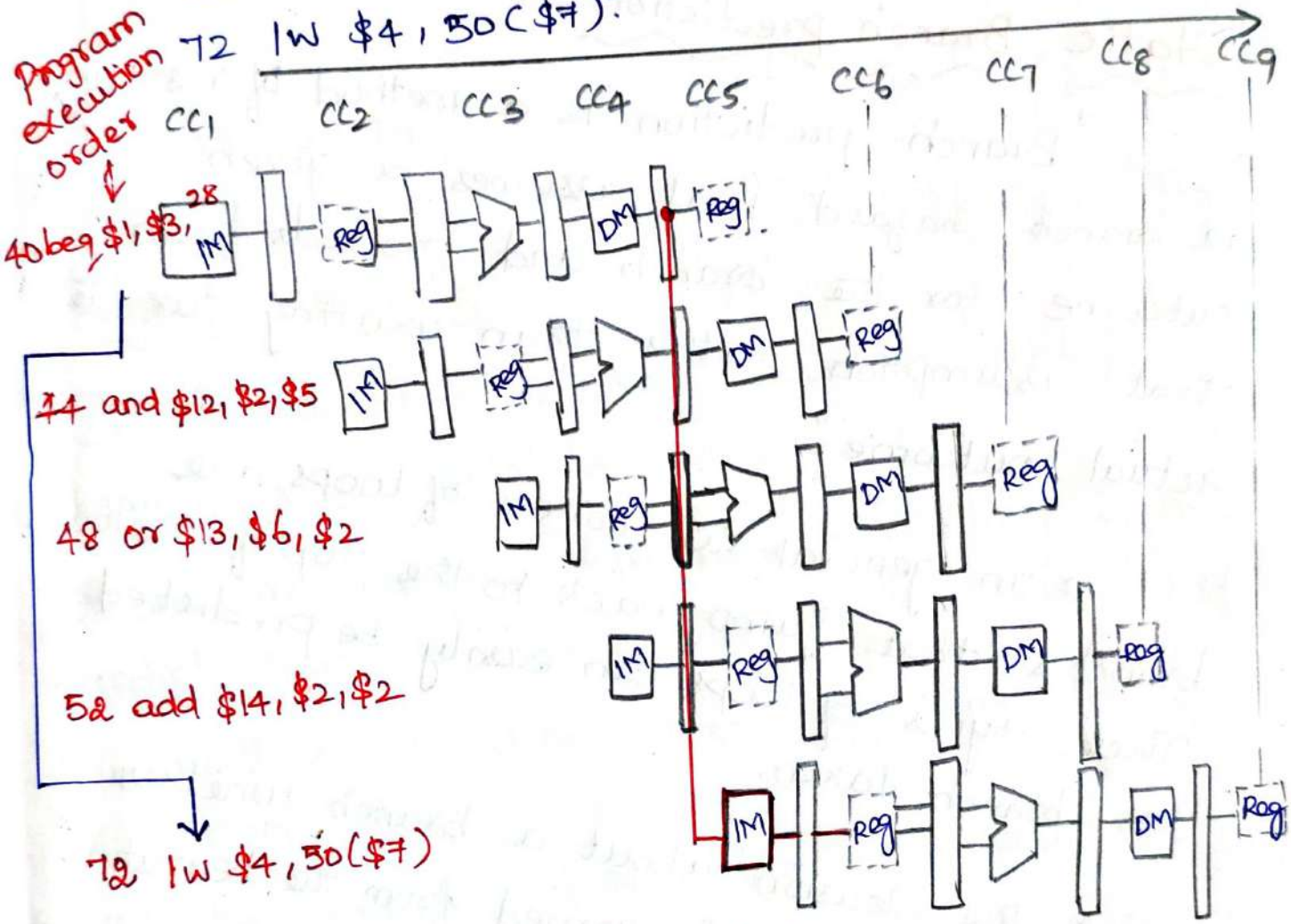
* Recognizing when a potential data hazard exists
* Revising the pipeline to introduce forwarding paths.

CONTROL HAZARD :-

" Control or branching hazards arise from resource conflicts when the hardware cannot support all possible combinations of instructions in simultaneous overlapped execution".

Eg 40 beq $1, $3, 28
44 and $12, $2, $5
48 or $13, $6, $2
52 add $14, $2, $2
72 lw $4, 50($7).

beq $1, $3
40 + 4 + 28
⇒ 72



Solution For Control hazard :-

1. Pipeline stall cycles.
2. Branch delay slots
3. Branch prediction
4. Indirect Branch prediction
5. Return address stack (RAS)

BRANCH PREDICTION :-

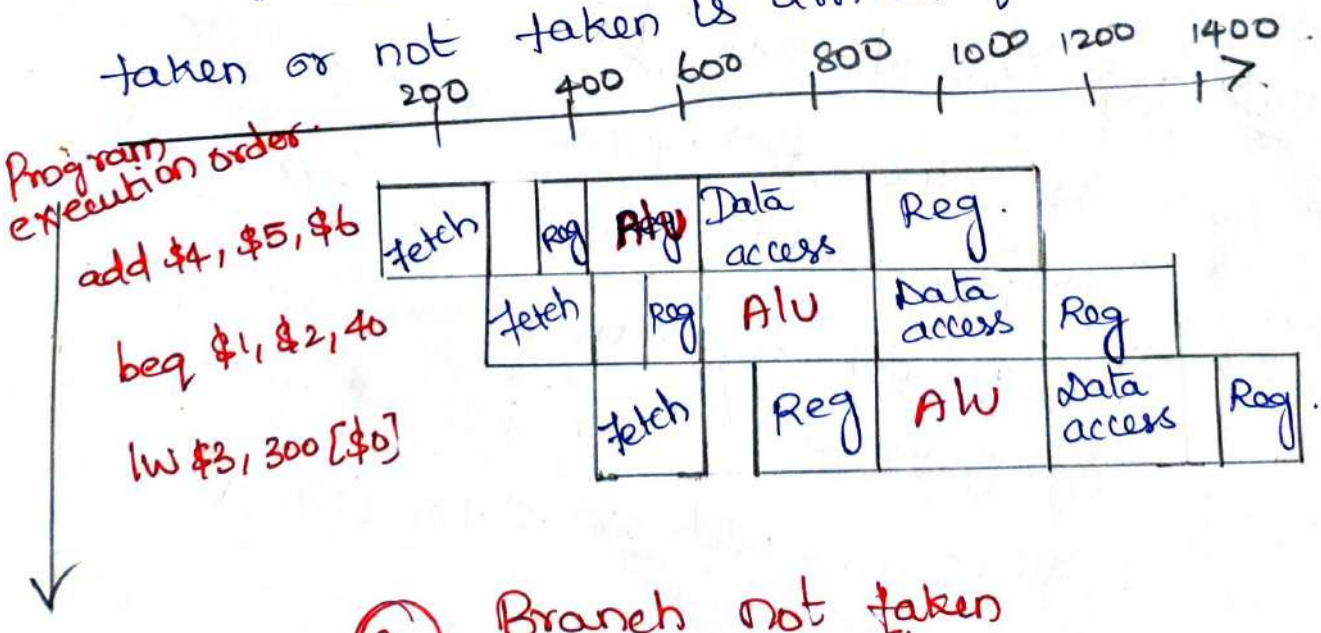
1. Static Branch prediction.
2. Dynamic Branch prediction.

Static Branch prediction :-

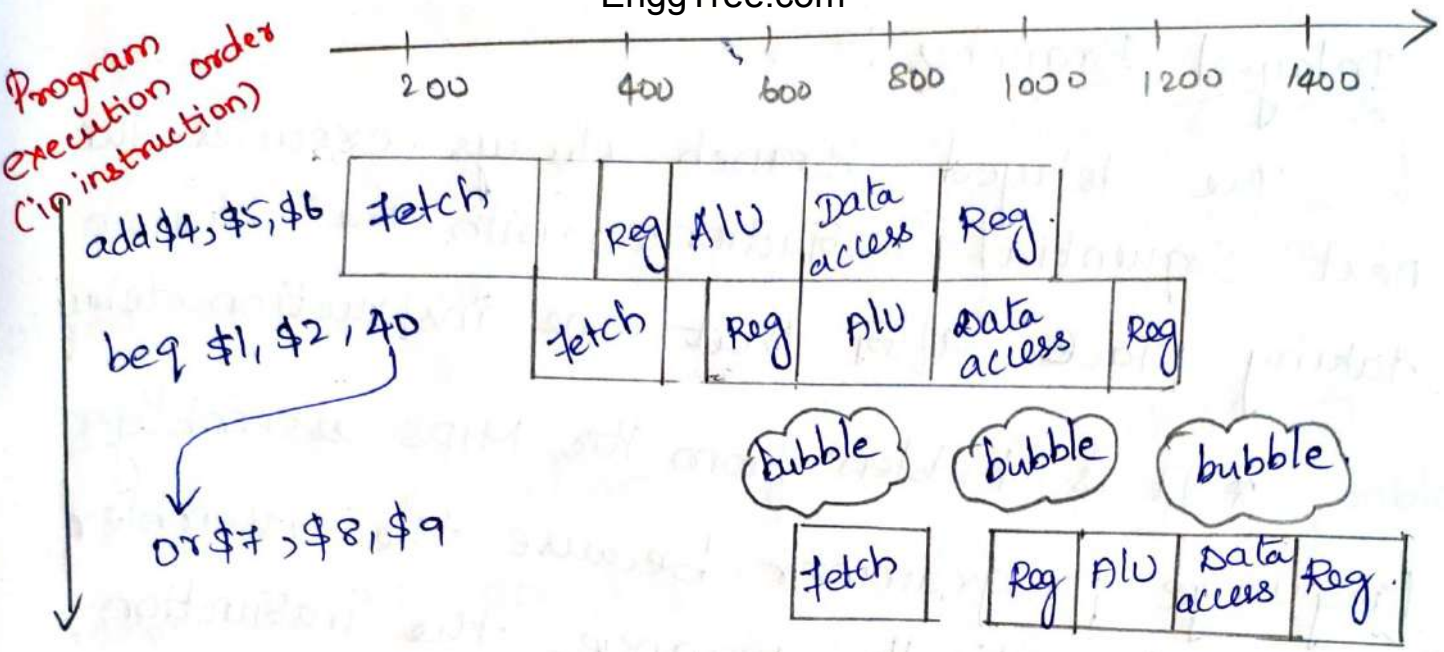
"Branch prediction is a method of resolving a branch hazard that assumes a given outcome for the branch and proceeds from that assumption rather than waiting for the actual outcome."

* In general the bottom of loops are branches that jump back to the top of the loop. These types of loops can easily be predicted as branch taken.

* The decision about a branch whether taken or not taken is arrived from the heuristics.



(a) Branch not taken



BRANCH STALLING :-

① This is stalling the instruction until the branch is complete is too slow.

② one improvement over branch stalling is to predict that the branch will not be taken and thus continue execution down the sequential instruction stream.

③ If the branch is taken, the instructions that are being fetched and decoded must be discarded. Execution continues at the branch target.

④ If branches are untaken, half the time, and if it costs little to discard the instructions, this optimization halves the cost of control hazard.

Delayed Branches :-

The delayed Branch always executes the next Sequential instructions, with the branch taking place after that one instruction delay.

* It is hidden from the MIPS assembly language programmer because the assembler can automatically arrange the instruction to get the branch behaviour desired by the programmer.

* one way to improve branch performance is to reduce the cost of the taken branch.

* The MIPS architecture was designed to support fast single cycle branches that could be pipelined with a small branch penalty.

* moving the branch decision up requires two actions to occur earlier.

* Computing the branch target address.

* Evaluating the branch decision.

DYNAMIC BRANCH PREDICTION :-

" prediction of branches at runtime using runtime information is called dynamic branch prediction".

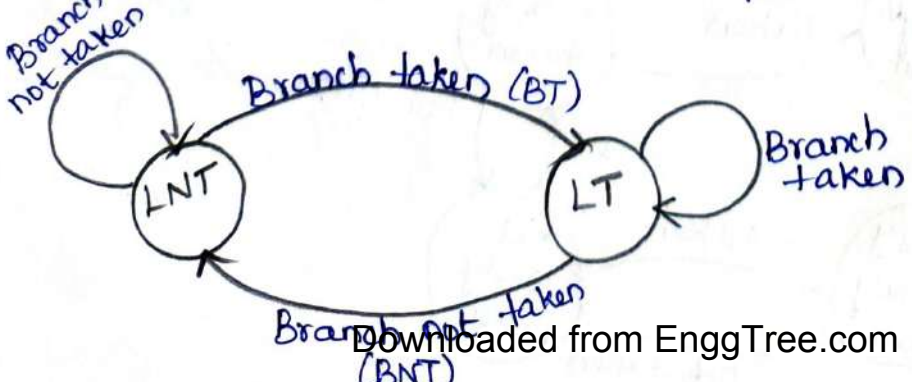
* One implementation of the approach is a branch prediction buffer or branch history table
* A branch prediction buffer is a small memory indexed by the lower portion of the address of the branch instruction.
* The memory contains a bit that says whether the branch was recently taken or not.

Types

1. one bit prediction scheme
2. Two bit prediction scheme.

one-bit prediction scheme :-

Two states (i) 2T :- Branch likely to be taken.
(ii) LNT :- Branch not likely to be taken.



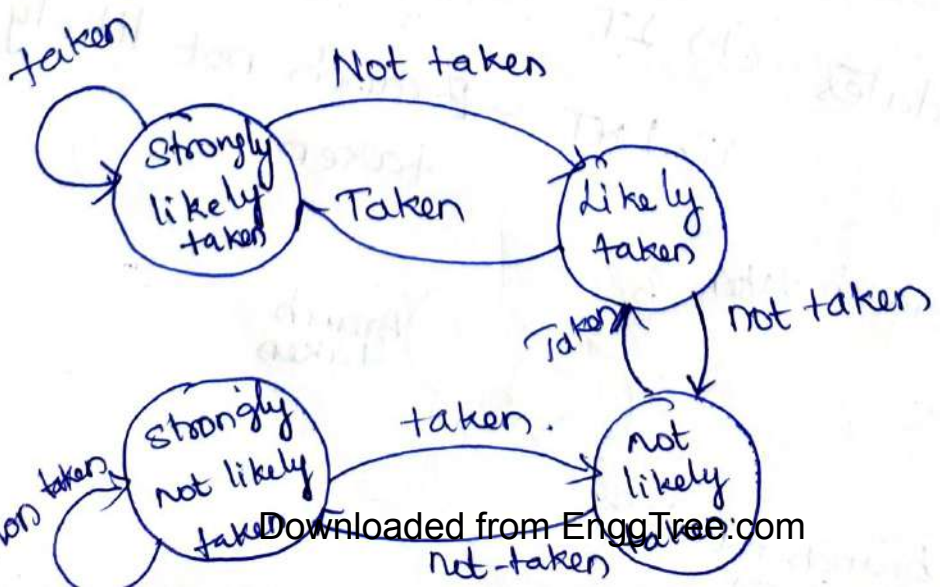
↳ Suppose that the algorithm is started in LNT, when the branch instruction is executed, and if the branch is taken the machine will move from LNT to LT. Otherwise it remains in LNT.

↳ During the next time, the same instruction is executed, the branch is predicted as taken if the machine is in LT. otherwise it is in LNT.

Eg :-

```
Second loop 2(100 times).
        R1 = 0  R2 = 9
Beq  R1, R2,  Second loop.
        :
ADD R1, R1, 1
Jump Loop.
```

2- bit prediction Scheme :-



* By using 2 bits rather than 1, a branch that strongly favours taken or not taken - as many branches do - will be mispredicted only once.

* The 2 bits are used to Encode the four states in the system.

* The two bit scheme is a general instance of a Counter based predictor, which is incremented when the prediction is accurate and decremented otherwise, and uses the midpoint of its range as the division between taken and not taken.

Branch delay slots :-

* The slot directly after a delayed branch instruction, which in the MIPS architecture is filled by an instruction that does not affect the branch.

* The limitations on delayed branch scheduling arise from the restriction on the instruction that are scheduled into the delay slots the ability to predict at compile time whether a branch is likely to be taken or not.

* Delayed branching is a simple and effective solution for a five-stage pipeline issuing one instruction each clock cycle.

* As processor go to both longer pipelines and issuing multiple instruction per clock cycle the branch delay becomes longer, and a single delay slot is insufficient.

* Hence, delay branching has lost popularity compared to more expensive but more flexible dynamic approaches.

EXCEPTIONS IN MIPS :-

The terminology used to refer the unexpected situation where the normal execution order of instruction is getting affected.

causes for exceptions :-

(i) I/O device request

(ii) Invoking an OS service from a user program.

(iii) Tracing Instruction execution.

(iv) Break point.

(v) Integer arithmetic overflow or underflow.

(vi) Page fault
(vii) Misaligned memory access.
(viii) Memory protection violation.
ix) Using an undefined instruction
(x) Hardware malfunction
xi) power failure.

What happens during an Exception occurs?

(i) The pipeline has to stop executing the offending instruction in the midstream.
(ii) Let all the preceeding instruction to complete.
(iii) Flush all succeeding instruction.
(iv) set a register to show the cause of the exception.
(v) save the address of the offending instruction and then
(vi) jump to the prearranged address (the address of the ~~exception~~ pipelined Instruction.

Exceptions in simple Five stage pipeline :-

① Instruction fetch & memory stages
↳ page fault on instruction / data fetch.
↳ misaligned memory access.
↳ memory protection violation.

2. Instruction Decode stage.
   - undefined / illegal opcode.
3. Execution stage.
   - Arithmetic exception.
4. write back stage.
   - No exception.

* The same clock cycle the first instruction is accessing data from memory while the fourth instruction is fetching an instruction from that same memory.

* without two memories, our pipeline could have a structural hazard.

Data Hazards :-

* They arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.

Control Hazard :-

* They arise from the pipelining of branches and other Instructions that change the pc. This is also known as branch hazard. The flow of instruction addresses is not what the pipeline had expected. This results in control hazard.

DATA HAZARDS :-

Definition

" Data hazard occur when the pipelined must be stalled because one step must wait for another to complete."

DESIGNING A CONTROL UNIT

CONTROL UNIT:

→ is the part of the computer's central processing unit which directs the operation of the processor.

→ It was included as part of the Von Neumann Architecture by John Von Neumann.

→ It is the responsibility of the control unit to tell the computer's memory, ALU, Input and output devices how to respond to the instructions that have been sent to the processor.

→ It fetches internal instructions of the programs from the main memory to the processor instruction register and based on this register contents, control unit generates a control signal that supervises the execution of these instructions.

→ The ~~functions~~ computer's processor then tells the attached hardware what operations to perform.

→ The functions that a control unit performs are dependent on the type of cpu because the architecture of cpu varies from manufacturer to manufacturer. Examples of devices that require a control unit are:

- control processing Units (cpu's)
- Graphics processing units (GPU's)

There are two methods in designing a control unit. They are,

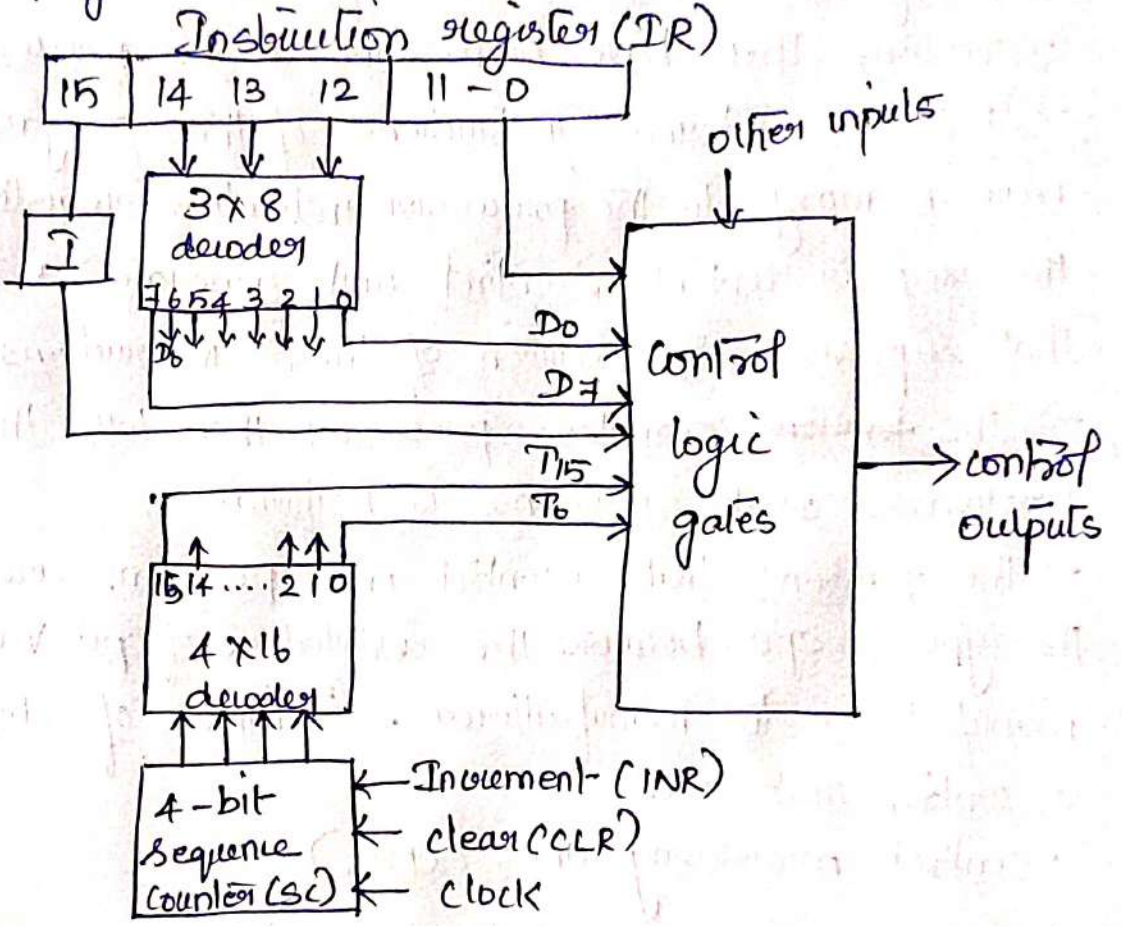
(i) Hardwired control unit

(ii) Microprogrammed control unit

HARDWIRED CONTROL UNIT:

DEFINITION: is a method of generating control signals with the help of finite state machines (FSM).

→ The control signals that are necessary for instruction execution in the hardwired control unit are generated by specially build hardware logical circuits and we can't change the signal production mechanism without physically changing the physical structure.



Control units of a Basic computer -

→ Two decoders, Sequence counter and logic gates make up a Hardwired control.

→ The instruction register stores an instruction retrieved from the memory unit (IR).

→ An Instruction register consists of operation code, the I bit and bits '0' through 11.

→ A 3X8 decoder is used to decode the operation code in bits 12 through 14.
→ The decoders outputs are denoted by the letters D_0 through D_7.
→ The bit 15 operation code is transferred to a flip flop with the symbol I.
→ The control logic gates are programmed with operation codes from bits 0 to 11.
→ The sequence counter can count from 0 to 15 in binary.

WORKING OF HARDWIRED CONTROL UNIT:

→ The basic data for control signal creation is contained in the operation code of an instruction.
→ The operation code is decoded in the instruction decoder.
→ As a result, only a few of the instruction decoder's output lines have active signal values.
→ These output lines are coupled to the matrix inputs, which provide control signals for the computer's execution units.
→ This matrix combines the decoded signals from the instruction opcode with the outputs from that matrix which generates signals indicating consecutive control unit states, as well as signals from the outside world, such as interrupt signals.
→ The sequence counter is a 4 bit counts in binary form 0 through 15. It can be incremented (or) cleared synchronously.
→ The timing signals from T_0 through T_{15} are the decoded outputs of the decoder.

DESIGNING OF HARDWIRED CONTROL UNIT:

The following are some of the ways for constructing hardwired control logic that have been proposed:

Sequence counter method: It is the most practical way to design a somewhat complex controller.

Delay Element Method:- This method relies on the usage of timed delay elements for creating the sequence of control signals.

State Table method: The standard algorithm method to designing the nodes controller utilising the classical state table is used in this method.

Pros of Hardwired control unit:

→ is quick due to the usage of combinational circuits to generate signals.

→ The amount of delay that can occur in the creation of control signals depends on the no. of gates.

→ Quicker than microprogrammed control unit.

Cons of Hardwired control unit:

→ The design becomes more complex.

→ Difficult & time consuming to add a new feature.

→ changes to control signals are challenging since they necessitate rearranging wires in the hardware circuit.

MICROPROGRAMMED CONTROL UNIT:

→ is a control unit that saves binary control values as words in memory.

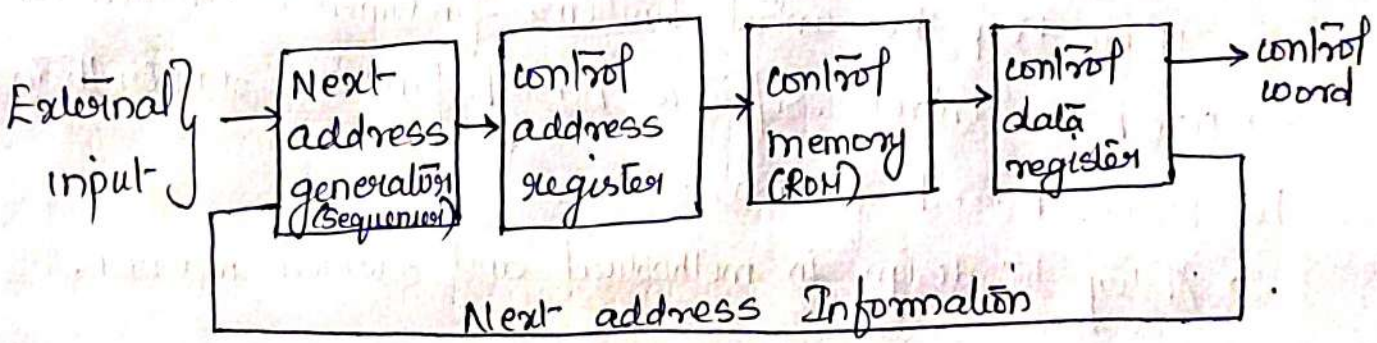
→ By collecting certain collection of signals at every system clock beat, a controller generates the instructions to be executed.

DEFINITION: The programming approach is used to implement a microprogrammed control unit. A program made up of micro instructions is used to carry out a series of microoperations.

→ The control memory in control unit stores a microprogram.

→ The creation of set of control signals depends on the execution of a micro instruction.

BLOCK DIAGRAM:



Microprogrammed control units of basic computer.

→ Micro instruction address is specified in control memory address register.

→ All the control information is saved in the control memory which is considered to be a ROM.

→ The micro instruction received from memory is stored in the control register.

→ A control word in the micro instruction specifies one (or) multiple micro operations for a data processor.

→ The next address is calculated in the circuit of the next address generator and then transferred to the control address register for reading the next micro instructions when the micro operations are being executed.

→ Because it determines the sequence of addresses received from control memory, the next address generator is also known as a microprogram sequencer.

Cons of microprogrammed control unit:

→ Adaptability comes at a higher price.

→ It is comparatively slower than a control unit that is hardwired.

pros of microprogrammed control unit

→ Its easier to trouble shoot and modify.

→ It can keep the control function's fundamental structure.

→ used to control software based functions rather than hardware based functions.

→ Ability to design in methodical and ordered manner.

UNIT – V

Memory and I/O Organisation

Memory Concepts and Hierarchy – Memory Management – Cache Memories: Mapping and Replacement Techniques – Virtual Memory – DMA – I/O – Accessing I/O: Parallel and Serial Interface – Interrupt I/O – Interconnection Standards: USB, SATA

5.1 INTRODUCTION

Memory unit enables us to store data inside the computer.The Computer memory always had here's to principle of locality .

Principle of locality or locality of reference is the tendency of a processor to access the same set of memory locations repetitively over a short period of time.

Two different types of locality are:

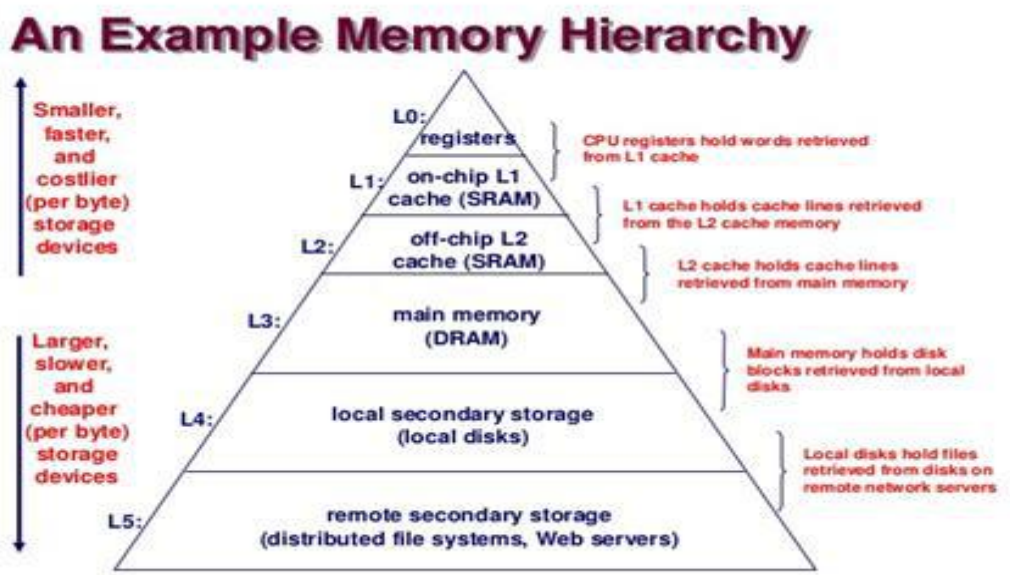
- Temporal locality:The principle stating that if a data location is referenced then it will tend to be referenced

address will tend to be referenced soon.

The locality of reference is useful in implementing the memory hierarchy.

Memory hierarchy is a structure that uses multiple levels of memories;as the distance from the cpu increases,the size of the memories and access time both increases.

A memory hierarchy consists of multiple levels of memory with different speed and sizes.The faster memories are more expensive per bit than the slower memories and thus smaller.



- Main memory is implemented from Dynamic Random Access Memory (DRAM).
- The levels closer to the processor (caches) use Static Random Access Memory (SRAM).
- DRAM is less costly per bit than SRAM, although it is substantially slower.
- For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- The computer programs tend to access the data at level k more often that at level k+1.
- The storage at level at k+1 can be slower

Cache memory (CPU memory) is high-speed SRAM that a computer Microprocessor can access more quickly than it can access regular RAM. This memory is typically integrated directly into the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU.

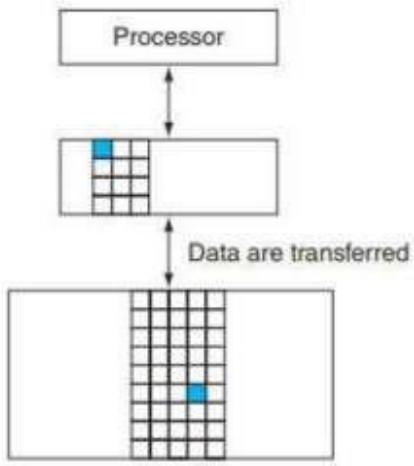


Fig 4.2: Data access by processor

The data transfer between various levels of memory is done through blocks. The minimum unit of information is called a **block.** If the data requested by the processor appears in some block

in the upper level, this is called a **hit.** If the data is not found in the upper level, the request is called a **miss.** The lower level in the hierarchy is then accessed to retrieve the block containing the requested data.

The fraction of memory accesses found in a cache is termed as hit rate or hit ratio.

Miss rate is the fraction of memory accesses not found in a level of the memory hierarchy. Hit time is the time required to access a level of the memory hierarchy, including the time needed to determine whether the access is a hit or a miss.

Miss penalty is the time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, and insert it in the level that experienced the miss.

Because the upper level is smaller and built using faster memory parts, the hit time will be much smaller than the time to access the next level in the hierarchy, which is the major component of the miss penalty.

MEMORY HIERARCHY

Principle of Locality

The locality of reference or the principle of locality is the term applied to situations where the same value or related storage locations are frequently accessed. There are three basic types of locality of reference:

- **Temporal locality:** Here a resource that is referenced at one point in time is referenced again soon afterwards.
- **Spatial locality:** Here the likelihood of referencing a storage location is greater if a storage location near it has been recently referenced.
- **Sequential locality:** Here storage is accessed sequentially, in descending or ascending order. The locality or reference leads to memory hierarchy.

Need for memory hierarchy

Memory hierarchy is an approach for organizing memory and storage systems. It consist of multiple levels of memory with different speeds and sizes. The following are the reasons for such organization:

- Fast storage technologies cost more per byte and have less capacity
- Gap between CPU and main memory speed is widening
- Well-written programs tend to exhibit good locality.

The memory hierarchy is shown in Fig 4.1. The entire memory elements of the computer fall under the following three categories:

- **Processor Memory:**

  This is present inside the CPU for high-speed data access. This consists of small set of registers that act as temporary storage. This is the costliest memory component.

- **Primary memory:**

  This memory is directly accessed by the CPU. All the data must be brought inside main memory before accessing them. Semiconductor chips acts as main memory.

- **Secondary memory:**

  This is cheapest, large and relatively slow memory component. The data from the secondary memory is accessed by the CPU only after it is loaded to main memory.

There is a trade-off among the three key characteristics of memory namely-

- Cost
- Capacity
- Access time

Terminologies in memory access

- **Block or line:** The minimum unit of information that could be either present or totally absent.
- **Hit**: If the requested data is found in the upper levels of memory hierarchy it is called hit.
- **Miss:** If the requested data is not found in the upper levels of memory hierarchy it is called miss.
- **Hit rate or Hit ratio:** It is the fraction of memory access found in the upper level .It is a performance metric.

Hit Ratio = Hit/ (Hit + Miss)

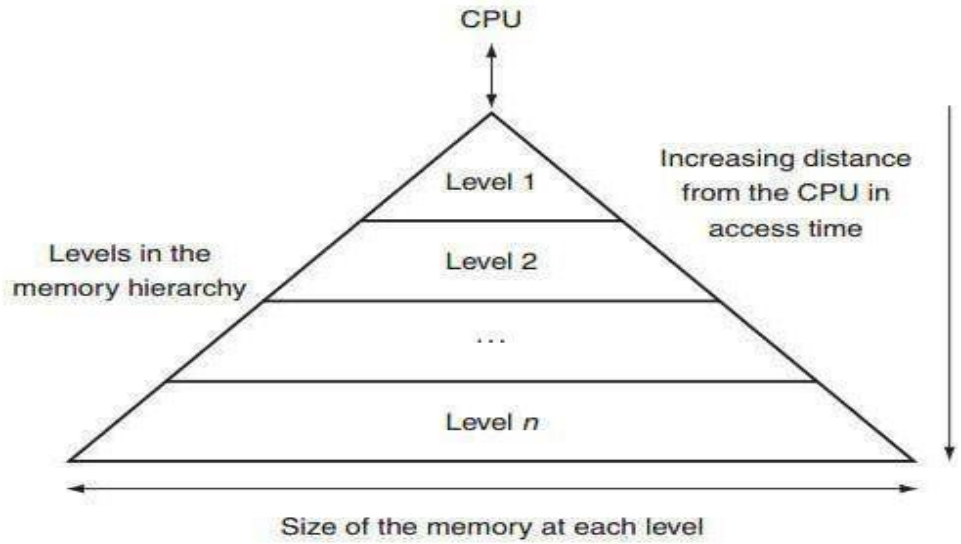


Fig 4.2: Memory level vs Access Time

The memory access time increases as the level increases. Since the CPU registers are located in very close proximity to the CPU they can be accessed very quickly and they are the more costly. As the level increases, the memory access time also increases thereby decreasing the costs.

Levels in Memory Hierarchy

The following are the levels in memory hierarchy:

- **CPU Registers:**

  They are at the top most level of this hierarchy, they hold the most frequently used data. They are very limited in number and are the fastest. They are often used by the CPU and the ALU for performing arithmetic and logical operations, for temporary storage of data.

- **Static Random Access Memory (SRAM):**

  Static Random Access Memory (Static RAM or SRAM) is a type of RAM that holds data in a static form, that is, as long as the memory has power. SRAM stores a bit of data on four transistors using two cross-coupled inverters. The two stable states characterize 0 and 1. During read and write operations another two access transistors are used to manage the availability to a memory cell.

- **Main memory or Dynamic Random Access Memory (DRAM):**

  Dynamic random access memory (DRAM) is a type of memory that is typically used for data or program code that a computer processor needs to function. In other words it is said to be the main memory of the computer. Random access allows processor to access any part of the memory directly rather than having to proceed sequentially from a starting place. The main advantages of DRAM are its simple design, speed and low cost in comparison to alternative types of memory. The main disadvantages of DRAM are volatility and high power consumption relative to other options.

Local Disks (Local Secondary Storage):

- **A local drive is a computer disk drive that is installed directly within the host or the local computer.)t is a computer's native hard disk drive)(DD), which is directly**

- **accessed by the computer for storing and retrieving data. It is a cheaper memory with more memory access time.**

- **Remote Secondary Storage:**

  This includes Distributed file system (DFS) and online storage like cloud. The storage area is vast with low cost but larger access time.

Distinction between Static RAM and Dynamic RAM

Static RAM	Dynamic RAM
➢ SRAM uses transistor to store a single bit of data	➢ DRAM uses a separate capacitor to store each bit of data
➢ SRAM does not need periodic refreshment to maintain data	➢ DRAM needs periodic refreshment to maintain the charge in the capacitors for data
➢ SRAM's structure is complex than DRAM	➢ DRAM's structure is simplex than SRAM
➢ SRAM are expensive as compared to DRAM	➢ DRAM's are less expensive as compared to SRAM
➢ SRAM are faster than DRAM	➢ DRAM's are slower than SRAM
➢ SRAM are used in Cache memory	➢ DRAM are used in Main memory

CLASSIFICATION OF MEMORY

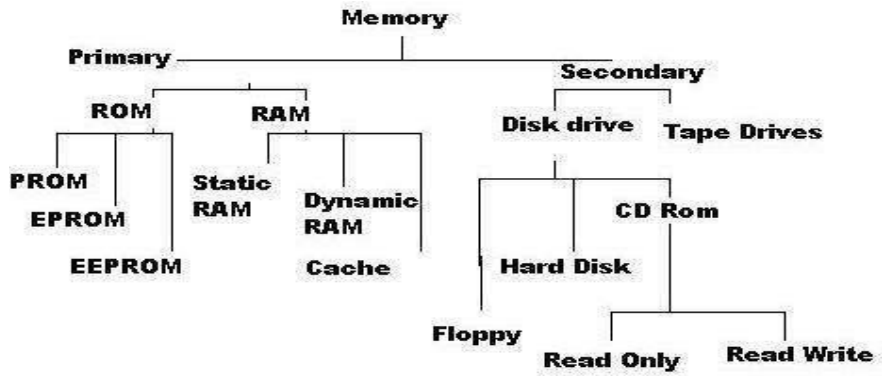


Fig 4.3: Classification of Memory

The instructions and data are stored in memory unit of the computer system are divided into following main groups:

- Main or Primary memory
- Secondary memory.

Primary Memory:

Primary memory is the main area in a computer in which data is stored for quick access by the computer¦s processor.)t is divided into two parts:

i) Random Access Memory (RAM):

RAM is a type of computer primary memory. It accessed any piece of data at any time. RAM stores data for as long as the computer is switched on or is in use. This type of memory is volatile. The two types of RAM are:

- **Static RAM:** This type of RAM is static in nature, as it does not have to be refreshed at regular intervals. Static RAM is made of large number of flip-flops on IC. It is being costlier and having packing density.
- **Dynamic RAM:** This type of RAM holds each bit of data in an individual capacitor in an integrated circuit. It is dynamic in the sense that the capacitor charge is repeatedly refreshed to ensure the data remains intact.

ii) Read Only Memory (ROM):

The ROM is nonvolatile memory. It retains stored data and information if the power is turned off.)n ROM, data are stored permanently and can¦t alter by the programmer. There are four types of ROM:

- **MROM (mask ROM):** MROM (mask ROM) is manufacturer-Programmed ROM in which data is burnt in by the manufacturer of the electronic equipment in which it is used and it is not possible for a user to modify programs or data stored inside the ROM chip.

- **PROM (programmable ROM):** PROM is one in which the user can load and store ¦read-only¦ programs and data.)n PROM the programs or data are stored only fast time and the stored data cannot modify the user.
- **EPROM (erasable programmable ROM):** EPROM is one in which is possible to erase information stored in an EPROM chip and the chip can be reprogrammed to store new information. When an EPROM is in use, information stored in it can only be ¦read¦ and the information remains in the chip until it is erased.
- **EEPROM (electronically erasable and programmable ROM):** EEPROM is one type of EPROM in which the stored information is erased by using high voltage electric pulse. It is easier to alter information stored in an EEPROM chip.

Secondary Memory:

Secondary memory is where programs and data are kept on a long time basis. It is cheaper from of memory and slower than main or primary memory. It is non-volatile and cannot access data directly by the computer processor. It is the external memory of the computer system.

Example: hard disk drive, floppy disk, optical disk/ CD-ROM.

MEMORY CHIP ORGANISATION

A memory consists of cells in the form of an array. The basic element of the semiconductor memory is the **cell**. Each cell is capable of storing one bit of information. Each row of the cells constitutes a memory words and all cells of a row are connected to a common line referred to as a **word line.** AW×b memory has w words, each word having |b| number of bits.

The basic memory element called cell can be in two states (0 or 1). The data can be written into the cell and can be read from it.

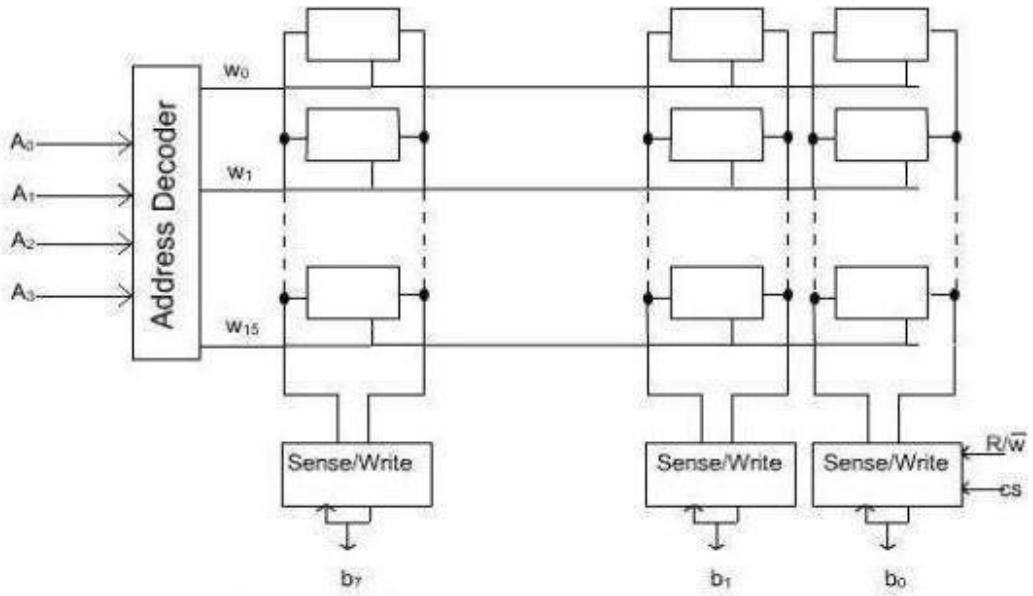


Fig 4.4: Organization of 16 x 8 memory

- In the above diagram there are 16 memory locations named as w_0, w_1, w_3...w_{15}. Each location can store at most 8 bits of data (b_0, b_1, b_3... b_7). Each location (w_n) is the word line. The word line of Fig 4.4 is 8.
- Each row of the cell is a memory word. The memory words are connected to a common line termed as word line. The word line is activated based on the address it receives from the address bus.
- An address decoder is used to activate a word line.
- The cells in the memory are connected by two **bit lines** (column wise). These are connected to data input and data output lines through sense/ write circuitry.
- **Read Operation:** During read operation the sense/ write circuit reads the information by selecting the cell through word line and bit lines. The data from this cell is transferred through the output data line.
- **Write Operation:** During write operation, the sense/ write circuitry gets the data and writes into the selected cell.
- The data input and output line of sense / write circuit is connected to a bidirectional data line.
- It is essential to have n bus lines to read 2^nwords.

Organization of 1M x 1 memory chip:

The organization of 1024 x 1 memory chips, has 1024 memory words of size 1 bit only. The size of data bus is 1 bit and the size of address bus is 10 bits. A particular memory location is identified by the contents of memory address bus. A decoder is used to decode the memory address.

Organization of memory word as a row:

- ❖ The whole memory address bus is used together to decode the address of the specified location.

.

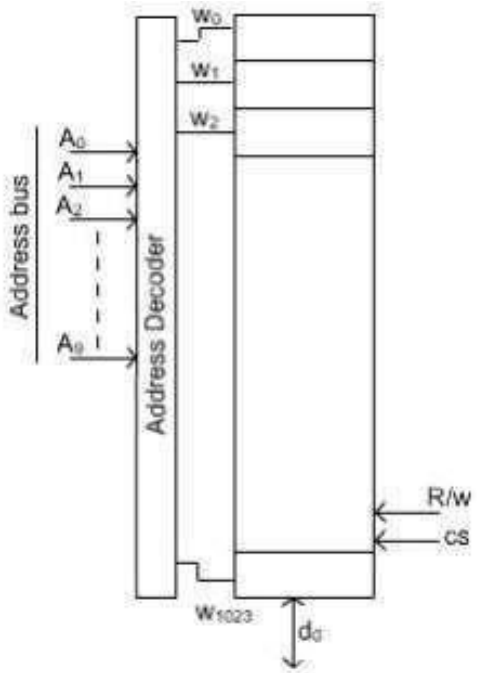


Fig 4.5: Organization of memory word as row

Organization of several memory words in row:

- One group is used to form the row address and the second group is used to form the column address.
- The 10-bit address is divided into two groups of 5 bits each to form the row and column address of the cell array.
- A row address selects a row of 32 cells, all of which could be accessed in parallel.
- Regarding the column address, only one of these cells is connected to the external data line via the input output multiplexers

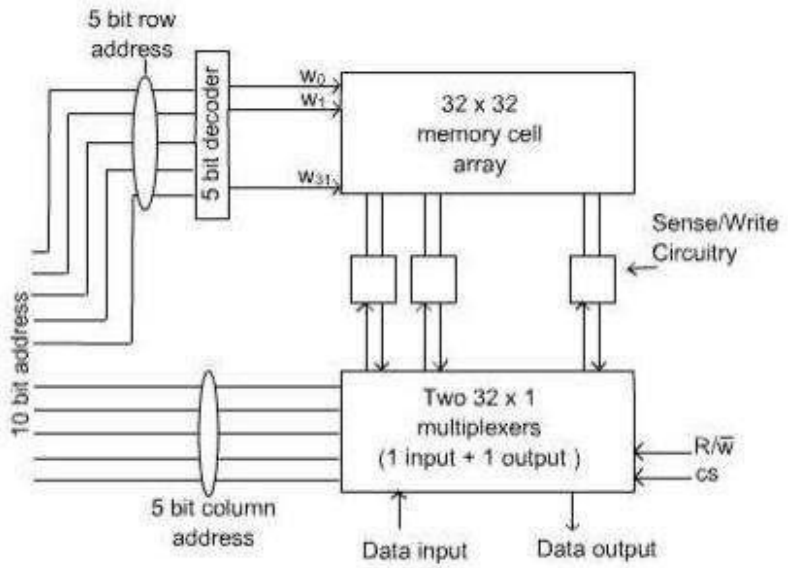


Fig 4.6: Organization of several memory words in row

Signals used in memory chip:

- A memory unit of 1MB size is organized as 1M x 8 memory cells.
- It has got220 memory location and each memory location contains 8 bits of information.
- The size of address bus is20 and the size of data bus is 8.
- The number of pins of a memory chip depends on the data bus and address bus of the memory module.
- To reduce the number of pins required for the chip, the cells are organized in the form of a square array.
- The address bus is divided into two groups, one for column address and other one is for row address.
- In this case, high- and low-order 10 bits of 20-bitaddress constitute of row and column address of a given cell, respectively.
- In order to reduce the number of pin needed for external connections, the row and column addresses are multiplexed on tenpins.

- During a Read or a Write operation, the row address is applied first. In response to a signal pulse on the Row Address Strobe (RAS) input of the chip, this part of the address is loaded into the row address latch.
- All cell of this particular row is selected. Shortly after the row address is latched, the column address is applied to the address pins.
- It is loaded into the column address latch with the help of Column Address Strobe (CAS) signal, similar to RAS.
- The information in this latch is decoded and the appropriate Sense/Write circuit is selected.

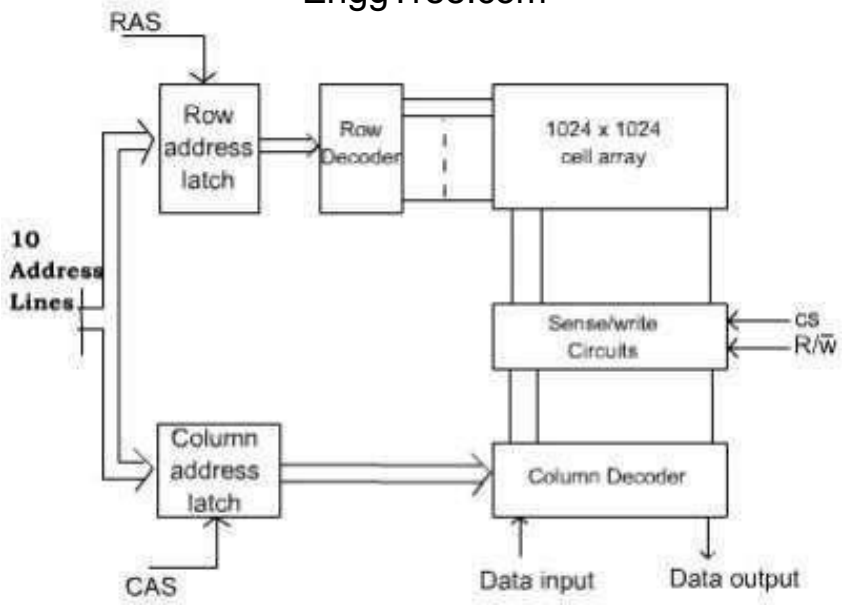


Fig 4.7: Signals in accessing the memory

- Each chip has a control input line called Chip Select (CS). A chip can be enabled to accept data input or to place the data on the output bus by setting its Chip Select input to 1.
- The address bus for the 64K memory is 16 bits wide.
- The high order two bits of the address are decoded to obtain the four chip select control signals.
- The remaining 14 address bits are connected to the address lines of all the chips.
- They are used to access a specific location inside each chip of the selected row.
- The R/ W inputs of all chips are tied together to provide a common read / write control.

CACHE MEMORY

The cache memory exploits the locality of reference to enhance the speed of the processor.

Cache memory or CPU memory, is high-speed SRAM that a processor can access more quickly than a regular RAM. This memory is integrated directly into the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU.

The cache memory stores instructions and data that are more frequently used or data that is likely to be used next. The processor looks first in the cache memory for the data. If it finds the instructions or data then it does perform a more time-consuming reading of data from larger main memory or other data storage devices.

The processor do not need to know the exact location of the cache. It can simply issue read and write instructions. The cache control circuitry determines whether the requested data resides in the cache.

- **Cache and temporal reference:** When data is requested by the processor, the data should be loaded in the cache and should be retained till it is needed again.
- **Cache and spatial reference:** Instead of fetching single data, a contiguous block of data is

- loaded into the cache.

Terminologies in Cache

- **Split cache:** It has separate data cache and a separate instruction cache. The two caches work in parallel, one transferring data and the other transferring instructions.
- **A dual or unified cache:** The data and the instructions are stored in the same cache. A combined cache with a total size equal to the sum of the two split caches will usually have a better hit rate.
- **Mapping Function:** The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

- **Cache Replacement:** When the cache is full and a memory word that is not in the cache is referenced, the cache control hardware must decide which block should be removed to create space for the new block that contains the referenced word. The collection of rules for making this decision is the replacement algorithm.

Cache performance:

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache. If the processor finds that the memory location is in the cache, a **cache hit** has said to be occurred. If the processor does not find the memory location in the cache, a **cache miss** has occurred. When a cache miss occurs, the cache replacement is made by allocating a new entry and copies in data from main memory. The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

Miss penalty or cache penalty is the sum of time to place a bock in the cache and time to deliver

Hit ratio = hit / (hit + miss) = Number of hits/ Total accesses to the cache

the block to CPU.

Miss Penalty= time for block replacement + time to deliver the block to CPU

Cache performance can be enhanced by using higher cache block size, higher associativity, reducing miss rate, reducing miss penalty, and reducing the time to hit in the cache. CPU execution Time of a given task is defined as the time spent by the system executing that task, including the time spent executing run-time or system services.

CPU execution time=(CPU clock cycles + memory stall cycles (if any)) x Clock cycle time

The **memory stall cycles** are a measure of count of the memory cycles during which the CPU is waiting for memory accesses. This is dependent on caches misses and cost per miss (cache penalty).

Memory stall cycles = number of cache misses x miss penalty

- Instruction Count x (misses/ instruction) x miss penalty
- Instruction Count (IC) x (memory access/ instruction) x miss penalty
- IC x Reads per instruction x Read miss rate X Read miss penalty + IC x Write per instruction x Write miss rate X Write miss penalty

---

Misses / instruction = (miss rate x memory access)/ instruction

Issues in Cache memory:

- **Cache placement:** where to place a block in the cache?
- **Cache identification:** how to identify that the requested information is available in the cache or not?
- **Cache replacement:** which block will be replaced in the cache, making way for an incoming block?

Cache Mapping Policies:

These policies determine the way of loading the main memory to the cache block. Main memory is divided into equal size partitions called as **blocks or frames.** The **c**ache memory is divided into fixed size partitions called as **lines.** During cache mapping, block of main memory is copied to the cache and further access is made from the cache not from the main memory.

Cache mapping is a technique by which the contents of main memory are brought into the cache memory.

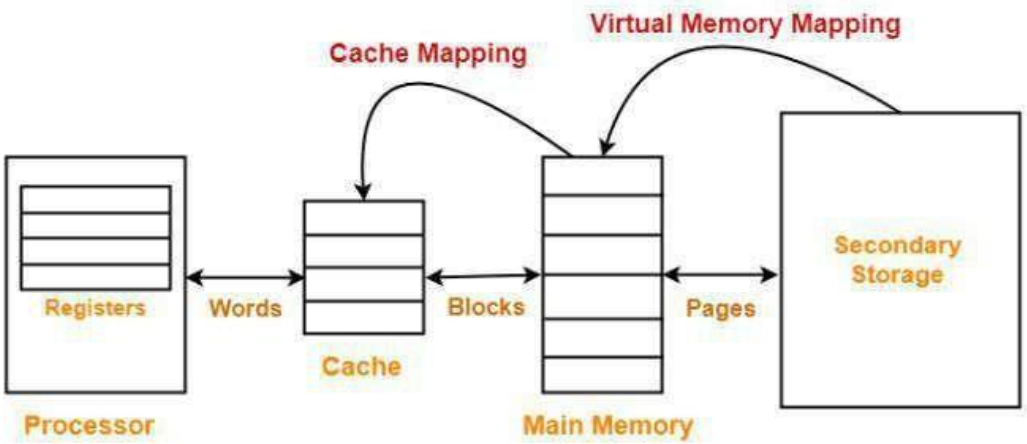


Fig 4.8: Cache mapping

There are three different cache mapping policies or mapping functions:

- Direct mapping
- Fully Associative mapping
- Set Associative mapping

Direct Mapping

- The simplest technique is direct mapping that maps each block of main memory into only one possible cache line.
- Here, each memory block is assigned to a specific line in the cache.

- If a line is previously taken up by a memory block and when a new block needs to be loaded, then the old block is replaced.
- Direct mapping's performance is directly proportional to the (it ratio.

The direct mapping concept is if the i^{th} block of main memory has to be placed at the j^{th} block of cache memory j = i % (number of blocks in cache memory)

- Consider a 128 block cache memory. Whenever the main memory blocks 0, 128, 256 are loaded in the cache, they will be allotted cache block 0, since j= (0 or 128 or 256) % 128 is zero).
- Contention or collision is resolved by replacing the older contents with latest contents.
- The placement of the block from main memory to the cache is determined from the 16 bit memory address.
- The lower order four bits are used to select one of the 16 words in the block.
- The 7 bit block field indicates the cache position where the block has to be stored.
- The 5 bit tag field represents which block of main memory resides inside the cache.
- This method is easy to implement but is not flexible.
- **Drawback:** The problem was that every block of main memory was directly mapped to the cache memory. This resulted in high rate of conflict miss. Cache memory has to be very frequently replaced even when other blocks in the cache memory were present as empty.

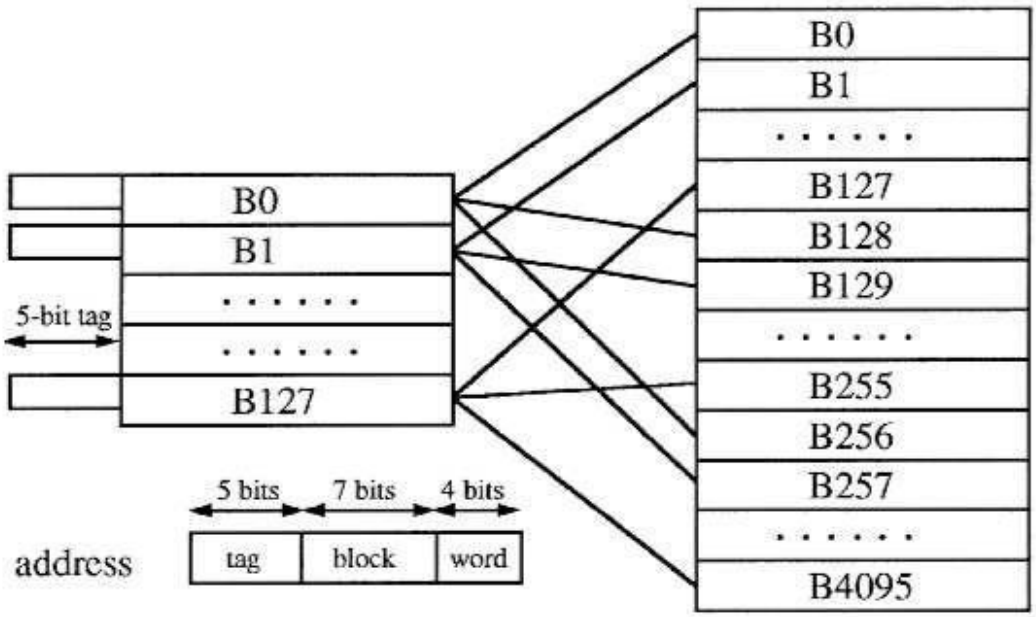


Fig 4.9: Direct memory mapping

Associative Mapping:

- The associative memory is used to store content and addresses of the memory word.
- Any block can go into any line of the cache. The 4 word id bits are used to identify which word in the block is needed and the remaining 12 bits represents the tag bit that identifies the main memory block inside the cache.
- This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.
- The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to check, if the desired block is present. Hence it is known as Associative Mapping technique.
- Cost of an associated mapped cache is higher than the cost of direct-mapped because of the need to search all 128 tag patterns to determine whether a block is in cache.

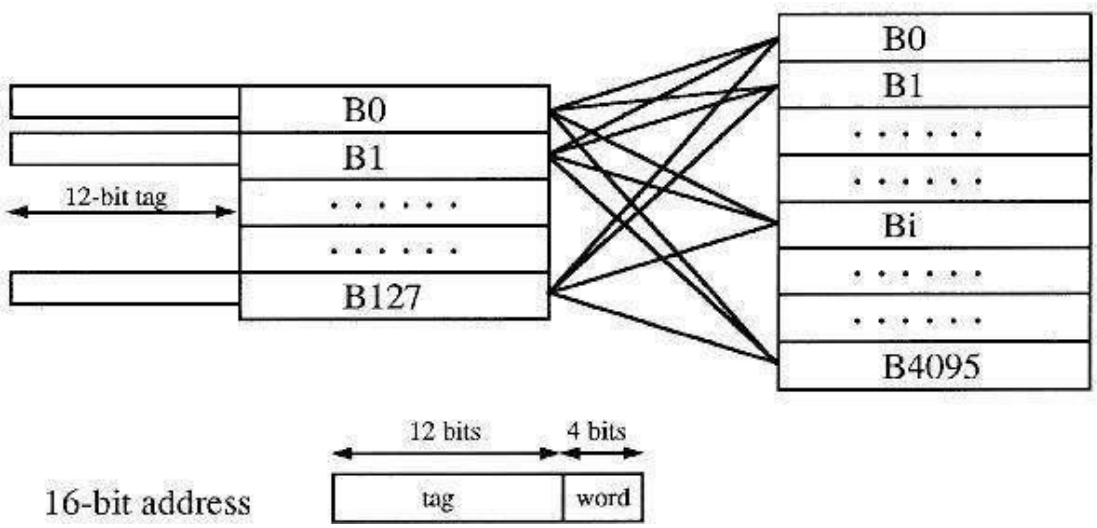


Fig 4.10: Associative Mapping

Set associative mapping:

- It is the combination of direct and associative mapping technique.
- Cache blocks are grouped into sets and mapping allow block of main memory to reside into any block of a specific set.
- This reduces contention problem (issue in direct mapping) with low hardware cost (issue in associative mapping).
- Consider a cache with two blocks per set.)n this case, memory block ど, はね, なにぱ,.....,ねどぬに map into cache set 0 and they can occupy any two block within this set.
- It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a set. Then a block in memory can map to any one of the lines of a specific set.
- The 6 bit set field of the address determines which set of the cache might contain the desired block. The tag bits of address must be associatively compared to the tags of the two blocks of the set to check if desired block is present.

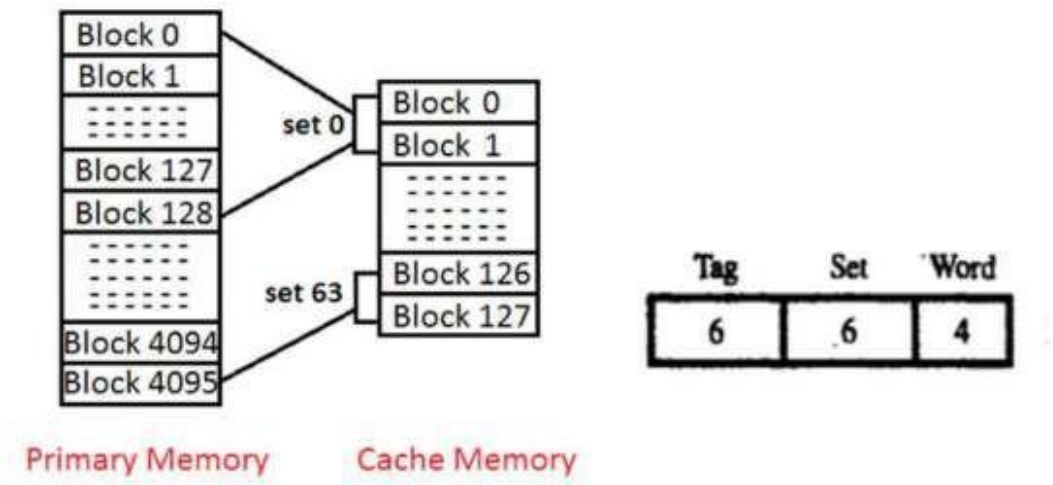


Fig 4.11: Set associative mapping

Handling Cache misses:

When a program accesses a memory location that is not in the cache, it is called a cache miss. The performance impact of a cache miss depends on the latency of fetching the data from the next cache level or main memory. The cache miss handling is done with the processor control unit and with a separate controller that initiates the memory access and refills the cache. The following are the steps taken when a cache miss occurs:

- Send the original PC value (PC - 4) to the memory.
- Instruct main memory to perform a read and wait for the memory to complete its access.
- Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.
- Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.

Writing to a cache:

- Suppose on a store instruction, the data is written into only the data cache (without changing main memory); then, after the write into the cache, memory would have a different value from that in the cache. This leads to inconsistency.

- The simplest way to keep the main memory and the cache consistent is to always write the data into both the memory and the cache. This scheme is called write-through.

Write through is a scheme in which writes always update both the cache and the memory, ensuring that data is always consistent between the two.

- With a write-through scheme, every write causes the data to be written to main memory. These writes will take a long time.
- A potential solution to this problem is deploying write buffer.
- A write buffer stores the data while it is waiting to be written to memory.
- After writing the data into the cache and into the write buffer, the processor can continue

execution.

- When a write to main memory completes, the entry in the write buffer is freed.
- If the write buffer is full when the processor reaches a write, the processor must stall until there is an empty position in the write buffer.
- If the rate at which the memory can complete writes is less than the rate at which the processor is generating writes, no amount of buffering can help because writes are being generated faster than the memory system can accept them.

Write buffer is a queue that holds data while the data are waiting to be written to memory.

iii) The rate at which writes are generated may also be less than the rate at which the memory can accept them, and yet stalls may still occur. To reduce the occurrence of such stalls, processors usually increase the depth of the write buffer beyond a single entry.

iv) Another alternative to a write-through scheme is a scheme called write-back. When a write occurs, the new value is written only to the block in the cache.

v) The modified block is written to the lower level of the hierarchy when it is replaced.

vi) Write-back schemes can improve performance, especially when processors can generate writes as fast or faster than the writes can be handled by main memory; a write-back scheme is, however, more complex to implement than write-through.

Write-back is a scheme that handles writes by updating values only to the block in the cache, then writing the modified block to the lower level of the hierarchy when the block is replaced.

Cache Replacement Algorithms

When a main memory block needs to be brought into the cache while all the blocks are occupied, then one of them has to be replaced. This selection of the block to be replaced is using cache replacement algorithms. Replacement algorithms are only needed for associative and set associative techniques. The following are the common replacement techniques:

- **Least Recently Used (LRU):** This replaces the cache line that has been in the cache the longest with no references to it.
- **First-in First-out (FIFO):** This replaces the cache line that has been in the cache the longest.
- **Least Frequently Used (LFU):** This replaces the cache line that has experienced the fewest references.
- **Random:** This picks a line at random from the candidate lines.

Example 4.1: Program P runs on computer A in 10 seconds. Designer says clock rate can be increased significantly, but total cycle count will also increase by 20%. What clock rate do we need on computer B for P to run in 6 seconds? (Clock rate on A is 100 MHz). The new machine is B. We want CPU Time_B = 6 seconds.

We know that Cycles count_B = 1.2 Cycles count_A. Calculate Cycles count_A. CPU Time_A = 10 sec. = ; Cycles count_A = 1000 x 106 cycles Calculate Clock rate_B:

CPU Time_B = 6 sec. = ; Clock rate_B = = 200 MHz

Machine B must run at twice the clock rate of A to achieve the target execution time.

Example 4.2: We have two machines with different implementations of the same ISA. Machine A has a clock cycle time of 10 ns and a CPI of 2.0 for program P; machine B has a clock cycle time of 20 ns and a CPI of 1.2 for the same program. Which machine is faster? Let IC be the number of instructions to be executed. Then Cycles count_A = 2.0 IC

Cycles count_B = 1.2 IC

calculate CPU Time for each machine:
CPU Time_A = 2.0 IC x 10 ns = 20.0 IC ns
CPU Time_B = 1.2 IC x 20 ns = 24.0 IC ns
» Machine A is 20%faster.

Example 4.3: Consider an implementation of MIPS ISA with 500 MHz clock and

– each ALU instruction takes 3 clock cycles,

– each branch/jump instruction takes 2 clock cycles,

– each sw instruction takes 4 clock cycles,

– eachlw instruction takes 5 clock cycles.

Also, consider a program that during its execution executes:

– x=200 million ALU instructions

– y=55 million branch/jump instructions

– z=25 million sw instructions

– w=20 million lw instructions

Find CPU time. Assume sequentially executing CPU.
Clock cycles for a program = (3x + 2y + 4z + 5w)
= 910 x 10^6 clock cycles CPU_time = Clock cycles for a program / Clock rate
= 910 x 10^6 / 500 x 10^6 = 1.82 sec

Example 4.4: Consider another implementation of MIPS ISA with 1 GHz clock and

– each ALU instruction takes 4 clock cycles,

– each branch/jump instruction takes 3 clock cycles,

– each sw instruction takes 5 clock cycles,

– eachlw instruction takes 6 clock cycles.

Also, consider the same program as in Example 1.

Find CPI and CPU time. Assume sequentially executing CPU.
CPI = (4x + 3y + 5z + 6w) / (x + y + z + w)
= 4.03 clock cycles/ instruction

CPU time = Instruction count x CPI / Clock rate
= (x+y+z+w) x 4.03 / 1000 x 10^6
= 300 x10^6 x 4.03 /1000 x 10^6
= 1.21 sec

VIRTUAL MEMORY

Virtual memory is a memory management capability of an operating system that uses hardware and software to allow a computer to compensate for physical memory shortages by temporarily transferring data from RAM to disk storage.

The concept of virtual memory in computer organization is allocating memory from the hard disk and making that part of the hard disk as a temporary RAM. In other words, it is a technique that uses main memory as a cache for secondary storage. The motivations for virtual memory are:

- To allow efficient and safe sharing of memory among multiple programs
- To remove the programming burdens of a small, limited amount of main memory.

Virtual memory provides an illusion to the users that the PC has enough primary memory left to run the programs. Sometimes the size of programs to be executed may sometimes very bigger than the size of primary memory left, the user never feels that the system needs a bigger primary storage to run that program. When the RAM is full, the operating system occupies a portion of the hard disk and uses it as a RAM. In that part of the secondary storage, the part of the program which not currently being executed is stored and all the parts of the program that are executed are first brought into the main memory. This is the theory behind **virtual memory.**

Terminologies:

- **Physical address** is an address in main memory.
- **Protection** is a set of mechanisms for ensuring that multiple processes sharing the processor, memory, or I/O devices cannot interfere, with one another by reading or writing each other's data.
- Virtual memory breaks programs into fixed-size blocks called **pages.**
- **Page fault** is an event that occurs when an accessed page is not present in main memory.
- **Virtual address** is an address that corresponds to a location in virtual space and is translated by address mapping to a physical address when memory is accessed.
- **Address translation or address mapping** is the process by which a virtual address is mapped to an address used to access memory.

Working mechanism

- In virtual memory, blocks of memory are mapped from one set of addresses (virtual addresses) to another set (physical addresses).

- The processor generates virtual addresses while the memory is accessed using physical addresses.
- Both the virtual memory and the physical memory are broken into pages, so that a virtual page is really mapped to a physical page.
- It is also possible for a virtual page to be absent from main memory and not be mapped to a physical address, residing instead on disk.
- Physical pages can be shared by having two virtual addresses point to the same physical address. This capability is used to allow two different programs to share data or code.
- Virtual memory also simplifies loading the program for execution by providing relocation. **Relocation** maps the virtual addresses used by a program to different physical addresses before the addresses are used to access memory. This relocation allows us to load the program anywhere in main memory.

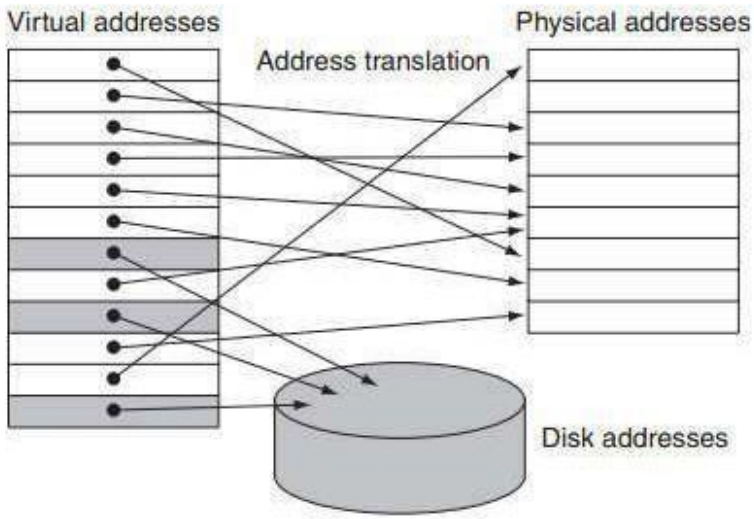


Fig 4.12: Mapping of virtual and physical memory

Addressing in virtual memory

- A virtual address is considered as a pair (p,d) where lower order bits give an offset d within the page and high-order bits specify the page p.
- The job of the Memory Management Unit (MMU) is to translate the page number p to a frame number f.

- The physical address is then (f,d), and this is what goes on the memory bus.
- For every process, there is a page and page-number p is used as an index into this array for the translation.
- The following are the entries in page tables:

1. Validity bit: Set to 0 if the corresponding page is not in memory
2. Frame number: Number of bits required depends on size of physical memory
3. Protection bits: Read, write, execute accesses
4. Referenced bit is set to 1 by hardware when the page is accessed: used by page replacement policy
5. Modified bit (dirty bit) set to 1 by hardware on write-access: used to avoid writing when swapped out.

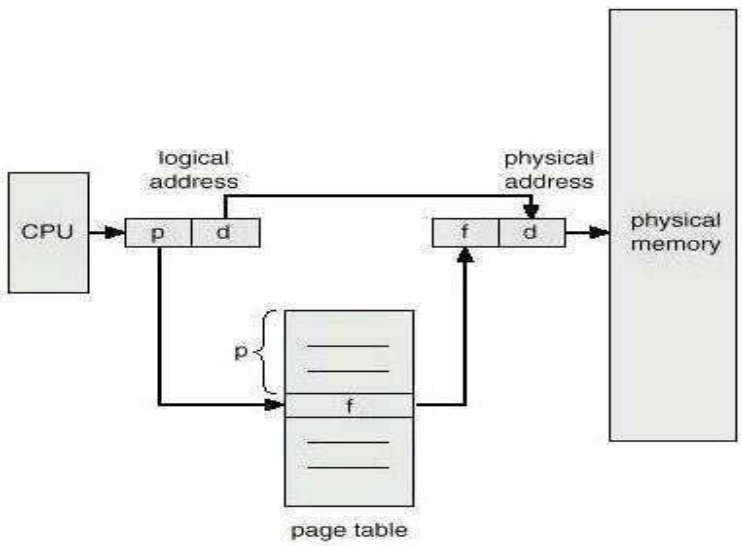


Fig 4.13: Conversion of logical address to physical address

Role of control bit in page table

The control bit (v) indicates whether the page is loaded in the main memory. It also indicates whether the page has been modified during its residency in the main memory. This information is crucial to determine whether to write back the page to the disk before it is removed from the main memory during next page replacement.

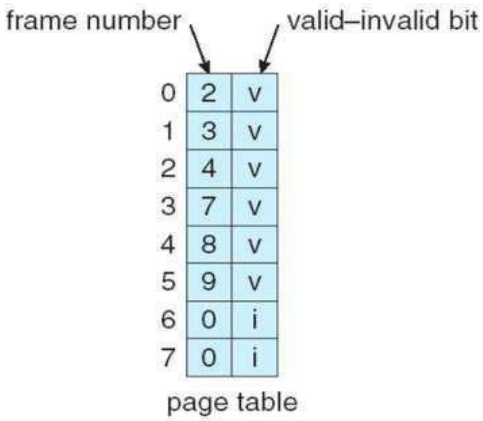


Fig 4.14: Page table

Page faults and page replacement algorithms

A page fault occurs when a page referenced by the CPU is not found in the main memory. The required page has to be brought from the secondary memory into the main memory. A page that is currently residing in the main memory, has to be replaced if all the frames of main memory are already occupied.

Page replacement is a process of swapping out an existing page from the frame of a main memory and replacing it with the required page.

Page replacement is done when all the frames of main memory are already occupied and a page has to be replaced to create a space for the newly referenced page. A good replacement algorithm will have least number of page faults.

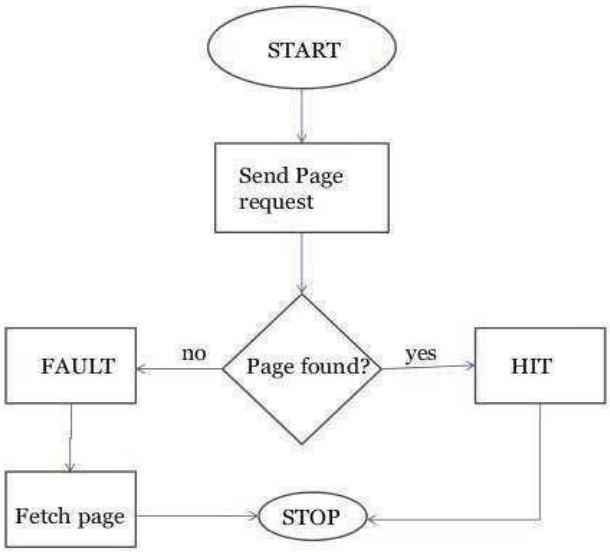


Fig 4.14: Occurrence of page fault

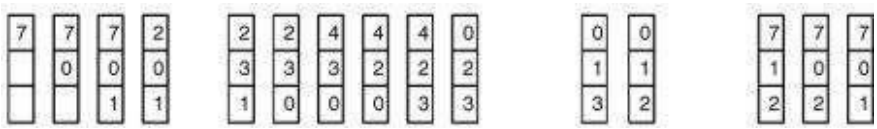
The following are the page replacement algorithms:

1. FIFO Page Replacement Algorithm
2. LIFO Page Replacement Algorithm
3. LRU Page Replacement Algorithm
4. Optimal Page Replacement Algorithm
5. Random Page Replacement Algorithm

1. First In First Out (FIFO) page replacement algorithm

It replaces the oldest page that has been present in the main memory for the longest time. It is implemented by keeping track of all the pages in a queue.

Example 4.5. Find the page faults when the following pages are requested to be loaded in a page frame of size 3: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



Page faults= 15

2. Last In First Out (LIFO) page replacement algorithm

It replaces the newest page that arrived at last in the main memory. It is implemented by keeping track of all the pages in a stack.

3. Least Recently Used (LRU) page replacement algorithm The new page will be replaced with least recently used page.

Example 4.6: Consider the following reference string. Calculate the number of page faults when the page frame size is 3 using LRU policy.7, 0, 1, 2, 0, 3, 4, 2, 3, 0, 3, 2,1,2,0,1,7,0,1

7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
F	F	F	F		F		F	F	F	F			F		F		F		

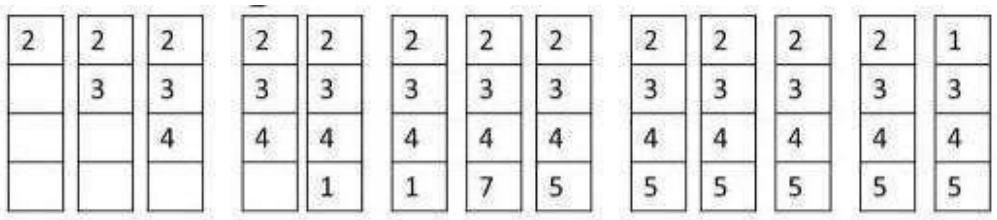
Page faults= 12 (F bit indicates the occurrence of page faults)

---

4. Optimal page replacement algorithm

In this method, pages are replaced which would not be used for the longest duration of time in the future.

Example 4.7: Find the number of misses and hits while using optimal page replacement algorithm on the following reference string with page frame size as 4: 2, 3, 4, 2, 1, 3, 7, 5, 4, 3, 2, 3, 1.



Page fault=13Number of page hit= 6Number of page misses=7

5. Random page replacement algorithms

Random replacement algorithm replaces a random page in memory. This eliminates the overhead cost of tracking page references.

Translation Look aside Buffer (TLB)

A translation look aside buffer (TLB) is a memory cache that stores recent translations of virtual memory to physical addresses for faster retrieval.

The page tables are stored in main memory and every memory access by a program to the page table takes longer time. This is because it does one memory access to obtain the physical address and a second access to get the data. The virtual to physical memory address translation occurs twice. But a TLB will exploit the locality of reference and can reduce the memory access time.

TLB hit is a condition where the desired entry is found in translation look aside buffer.

If this happens then the CPU simply access the actual location in the main memory.

If the entry is not found in TLB (TLB miss) then CPU has to access page table in the main memory and then access the actual frame in the main memory. Therefore, in the case of TLB hit, the effective access time will be lesser as compare to the case of TLB miss.

If the probability of TLB hit is P% (TLB hit rate) then the probability of TLB miss (TLB miss rate) will be (1-P) %. The effective access time can be defined as

Effective access time = P (t + m) + (1 - p) (t + k.m + m)

Where, p is the TLB hit rate, t is the time taken to access TLB, m is the time taken to access main memory. K indicates the single level paging has been implemented.

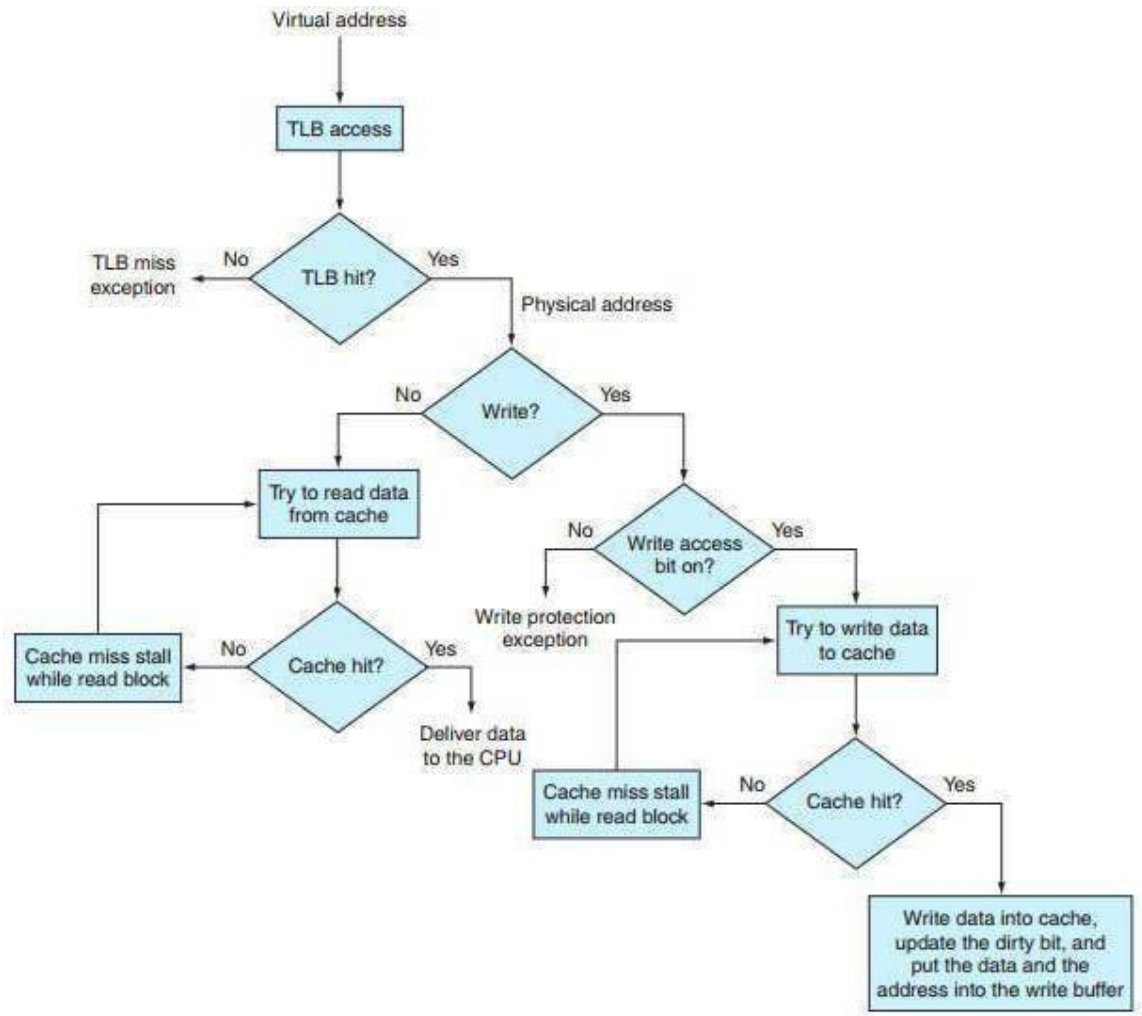


Fig 4.15: Cache access levels

4.3.5 Protection in Virtual memory

- Virtual memory allows sharing of main memory by multiple processes. So protection mechanisms, while providing memory protection.
- The protection mechanism must ensure one process cannot write into the address space of another user process or into the operating system.
- Memory protection can be done at two levels: hardware and software levels.

Hardware Level:

Memory protection at hardware level is done in three methods:

- The machine should support two modes: supervisor mode and user mode. This indicates whether the current running process is a user or supervisory process. The processes running in supervisor or kernel mode is an operating system process.

- Include user / supervisor bit in TLB to indicate whether the process is in user or supervisor mode. This is an access control mechanism imposed on the user process only to read from the TLB and not write to it.
- The processors can switch between user and supervisor mode. The switching from user to

system mode is done through system calls that transfers control to a dedicated location in supervisor code space.

System call is a special instruction that transfers control from user mode to a dedicated location in supervisor code space, invoking the exception mechanism in the process.

PARALLEL BUS ARCHITECTURES

Single bus architectures connect multiple processors with their own cache memory using shared bus. This is a simple architecture but it suffers from latency and bandwidth issues. This naturally led to deploying parallel or multiple bus architectures. Multiple bus multiprocessor systems use several parallel buses to interconnect multiple processors with multiple memory modules. The following are the connection schemes in multi bus architectures:

1. Multiple-bus with full bus–memory connection (MBFBMC)

This has all memory modules connected to all buses. The multiple-bus with single bus

memory connection has each memory module connected to a specific bus. For N processors with M memory modules and B buses, the number of connections requires are: B(N+M) and the load on each bus will ne N+M.

2. Multiple bus with partial bus–memory connection (MBPBMC)

The multiple-bus with partial bus–memory connection, has each memory module connected to a subset of buses.

3. Multiple bus with class-based memory connection (MBCBMC)

The multiple-bus with class-based memory connection (MBCBMC), has memory modules grouped into classes whereby each class is connected to a specific subset of buses. A class is just an arbitrary collection of memory modules.

4. Multiple bus with single bus memory connection (MBSBMC)

Here, only single bus will be connected to single memory, but the processor can access all the buses. The numbers of connections:

$$BN + \sum_{j=1}^{k} M_j(j + B - k)$$

And load on each bus is given by

$$N + \sum_{j=\max(i+k-B,1)}^{k} M_j,\ 1 \le i \le B$$

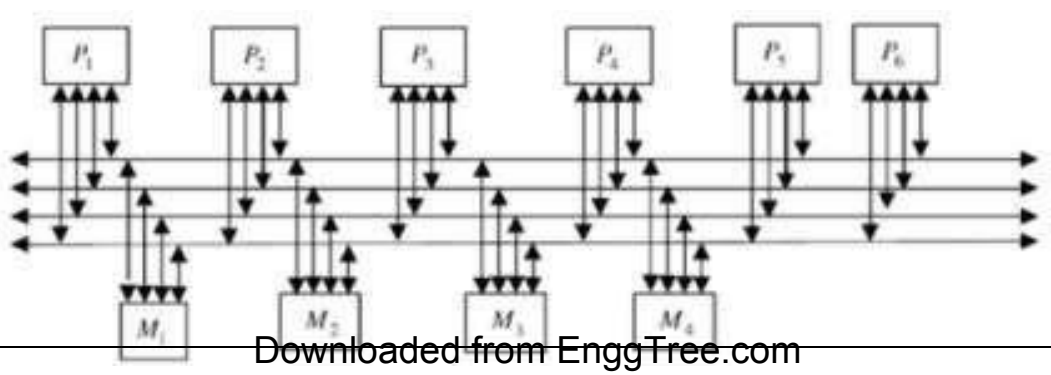


Fig 4.16 a) Multiple-bus with full bus–memory connection (MBFBMC)

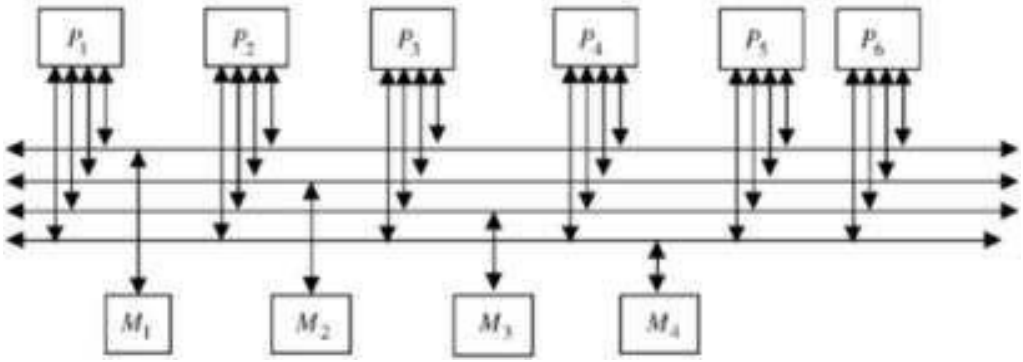


Fig 4.16 b) Multiple bus with single bus memory connection (MBSBMC)

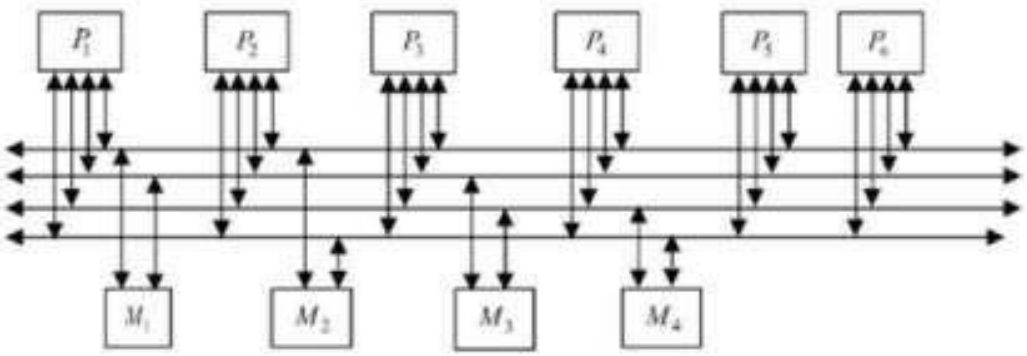


Fig 4.16 c) Multiple bus with partial bus–memory connection (MBPBMC)

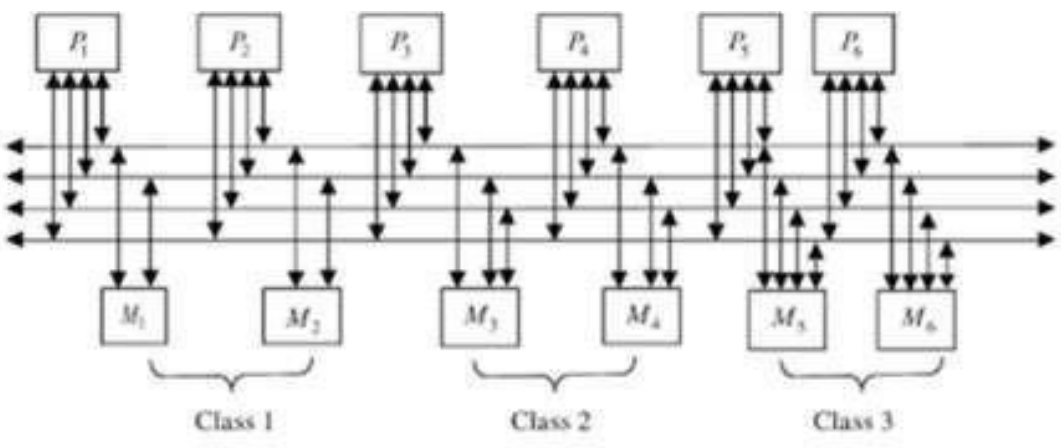


Fig 4.16 d) Multiple bus with class-based memory connection (MBCBMC)

Bus Synchronization:

- In a single bus multiprocessor system, bus arbitration is required in order to resolve the bus contention that takes place when more than one processor competes to access the bus.

- A bus can be classified as synchronous or asynchronous. The time for any transaction over a synchronous bus is known in advance. Asynchronous bus depends on the availability of data and the readiness of devices to initiate bus transactions.
- The processors that want to use the bus submit their requests to bus arbitration logic. The latter decides, using a certain priority scheme, which processor will be granted access to the bus during a certain time interval (bus master).
- The process of passing bus mastership from one processor to another is called **handshaking,** which requires the use of two control signals: bus request and bus grant.
- Bus request indicates that a given processor is requesting mastership of the bus.
- Bus grant: indicates that bus mastership is granted.
- Bus busy: is usually used to indicate whether or not the bus is currently being used.
- In deciding which processor gains control of the bus, the bus arbitration logic uses a predefined priority scheme.
- Among the priority schemes used are random priority, simple rotating priority, equal priority, and least recently used (LRU) priority.
- After each arbitration cycle, in simple rotating priority, all priority levels are reduced one place, with the lowest priority processor taking the highest priority. In equal priority, when two or more requests are made, there is equal chance of any one request being processed.
- In the LRU algorithm, the highest priority is given to the processor that has not used the bus for the longest time.

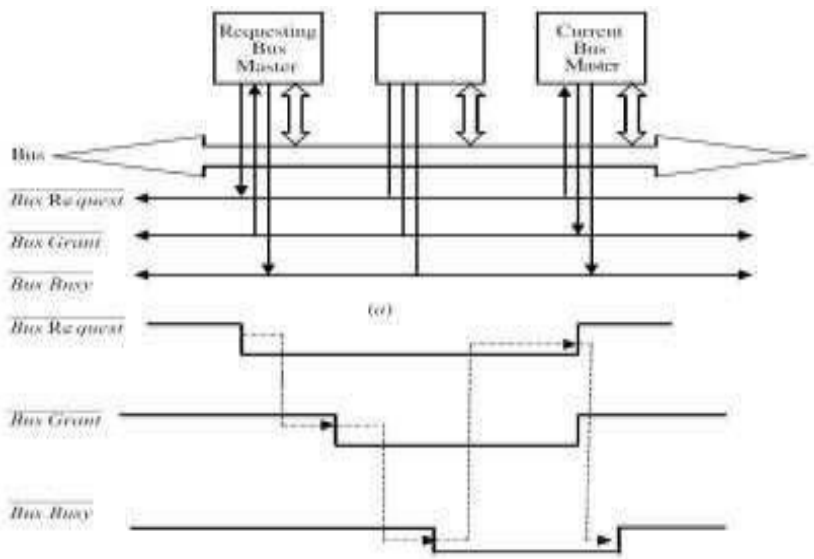


Fig 4.17: Bus synchronization

INTERNAL COMMUNICATION METHODOLOGIES

CPU of the computer system communicates with the memory and the I/O devices in order to transfer data between them. The method of communication of the CPU with memory and I/O devices is different. The CPU may communicate with the memory either

directly or through the Cache memory. However, the communication between the CPU and I/O devices is usually implemented with the help of interface. There are three types of internal communications:

- Programmed I/O
- Interrupt driven I/O
- Direct Memory Access (DMA)

Programmed I/O

- Programmed I/O is implicated to data transfers that are initiated by a CPU, under driver software control to access Registers or Memory on a device.
- With programmed I/O, data are exchanged between the processor and the I/O module.
- The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data.
- When the processor issues a command to the I/O module, it must wait until the I/O operation is complete.
- If the processor is faster than the I/O module, this is wasteful of processor time. With interrupt-driven I/O, the processor issues I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work.
- With both programmed and interrupt I/O, the processor is responsible for extracting data from main memory for output and storing data in main memory for input.
- The alternative is known as direct memory access. In this mode, the I/O module and main memory exchange data directly, without processor involvement.
- With programmed I/O, the I/O module will perform the requested action and then set the appropriate bits in the I/O status register.
- The I/O module takes no further action to alert the processor.
- When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module. In particular, it does not interrupt the processor.
- It is the responsibility of the processor periodically to check the status of the I/O module. Then if the device is ready for the transfer (read/write).
- The processor transfers the data to or from the I/O device as required. As the CPU is faster than the I/O module, the problem with programmed I/O is that the CPU has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data.
- The CPU, while waiting, must repeatedly check the status of the I/O module, and this process is known as **Polling.**
- The level of the performance of the entire system is severely degraded.

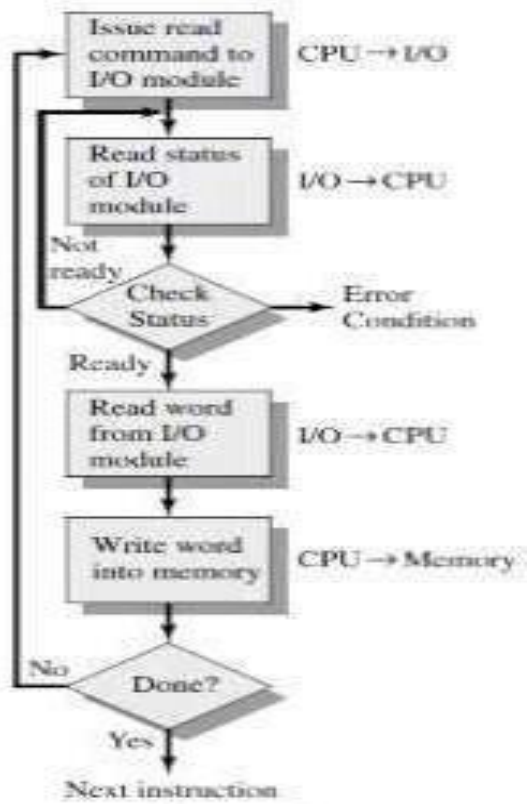


Fig 4.18: Workflow in programmed I/O

Interrupt Driven I/O

- The CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.
- For input, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports, memory mapping.
- For output, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.
- Although Interrupt relieves the CPU of having to wait for the devices, but it is still inefficient in data transfer of large amount because the CPU has to transfer the data word by word between I/O module and memory.
- Below are the basic operations of Interrupt:

1. CPU issues read command
2. I/O module gets data from peripheral whilst CPU does other work
3. I/O module interrupts CPU
4. CPU requests data
5. I/O module transfers data

Direct Memory Access (DMA)

- Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement.
- DMA module controls exchange of data between main memory and the I/O device.
- Because of DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus relieve congestion on the bus.
- CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

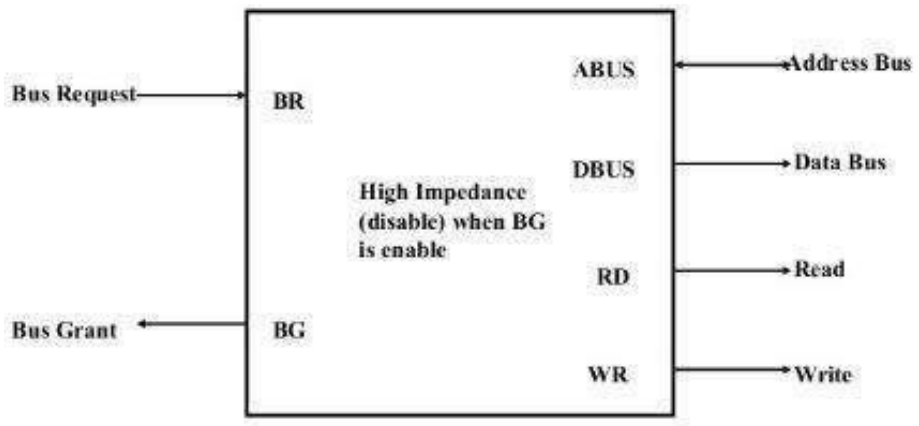


Fig 4.19: CPU bus signals for DMA transfer

- The CPU programs the DMA controller by setting its registers so it knows what to transfer where.
- It also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum.

- When valid data are in the disk controller's buffer, DMA can begin. The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller.
- This read request looks like any other read request, and the disk controller does not know whether it came from the CPU or from a DMA controller.
- The memory address to write to is on the bus address lines, so when the disk controller fetches the next word from its internal buffer, it knows where to write it.
- The write to memory is another standard bus cycle.
- When the write is complete, the disk controller sends an acknowledgement signal to the DMA controller, also over the bus.
- The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0, steps 2 through 4 are repeated until the count reaches 0.
- At that time, the DMA controller interrupts the CPU to let it know that the transfer is now complete.
- When the operating system starts up, it does not have to copy the disk block to memory; it

is already there.

- The DMA controller requests the disk controller to transfer data from the disk controller's buffer to the main memory. In the first step, the CPU issues a command to the disk controller telling it to read data from the disk into its internal buffer.

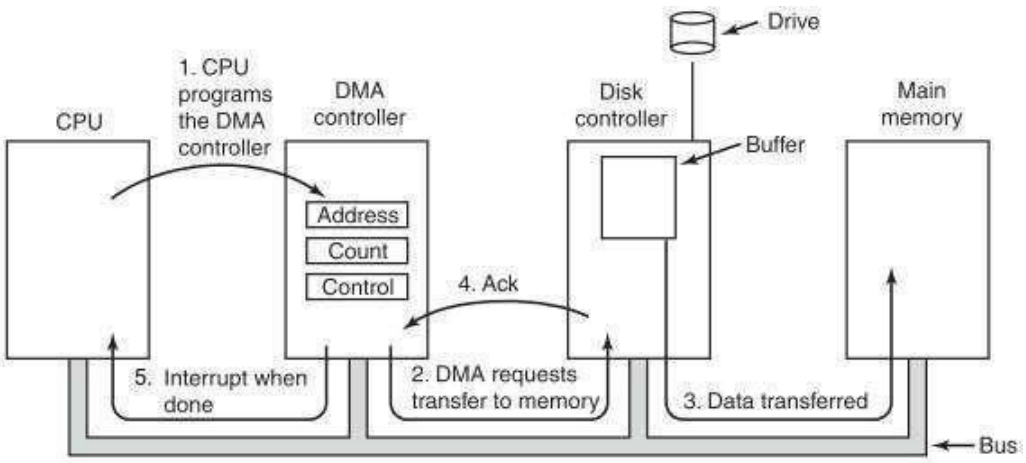


Fig 4.20: Operations in DMA

SERIAL BUS ARCHITECTURES

The peripheral devices and external buffer that operate at relatively low frequencies communicate with the processor using serial bus. There are two popular serial buses: Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I^2C).

Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is an interface bus designed by Motorola to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device.

- A standard SPI connection involves a master connected to slaves using the serial clock (SCK), Master Out Slave In (MOSI), Master In Slave Out (MISO), and Slave Select (SS) lines.
- The SCK, MOSI, and MISO signals can be shared by slaves while each slave has a unique SS line.
- The SPI interface defines no protocol for data exchange, limiting overhead and allowing for high speed data streaming.
- Clock polarity (CPOL) and clock phase (CP(A) can be specified as 0 or 1 to form four unique modes to provide flexibility in communication between master and slave.
-)f CPOL and CP(A are both 0 (defined as Mode 0) data is sampled at the leading rising edge of the clock. Mode 0 is by far the most common mode for SPI bus slave communication.
-)f CPOL is 1 and CP(A is 0 (Mode 2), data is sampled at the leading falling edge of the clock.Likewise, CPOL = 0 and CP(A = 1 (Mode 3) results in data sampled at on the

trailingfalling edge and CPOL = ┼ﾅ┼ with CP(A = ┼ﾅ┼ ⌀Mode ⁂ results in data sampled on the trailingrising edge.

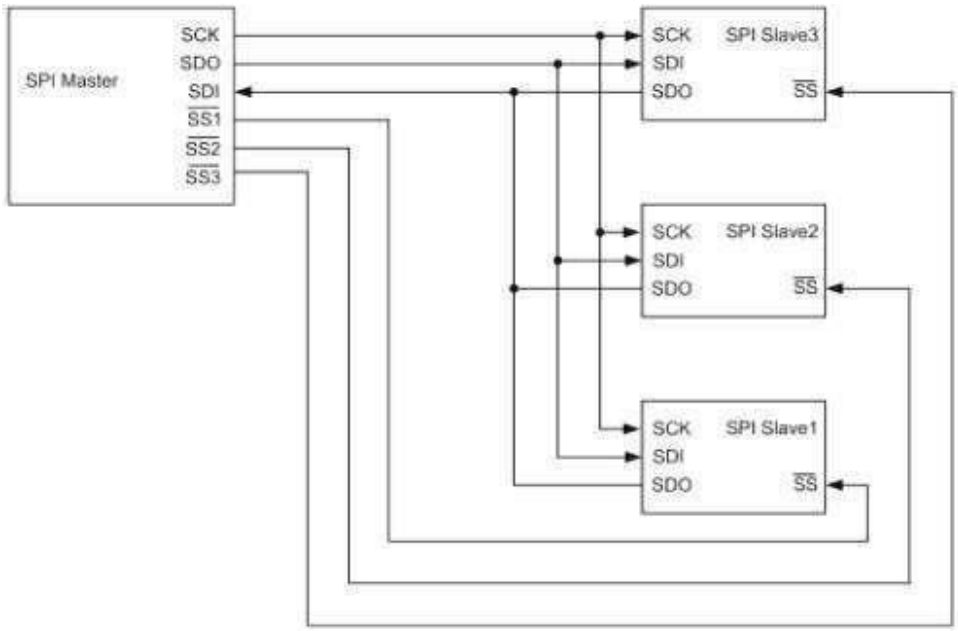


Fig 4.21: SPI master with three slaves

Mode	CPOL	CPHA
0	0 0	
1	0 1	
2	1 0	
3	1 1	

Fig 4.22: Modes in SPI

- In addition to the standard 4-wire configuration, the SPI interface has been extended to include a variety of IO standards including 3-wire for reduced pin count and dual or quad I/O for higher throughput.
- In 3-wire mode, MOSI and MISO lines are combined to a single bidirectional data line.
- Transactions are half-duplex to allow for bidirectional communication. Reducing the number of data lines and operating in half-duplex mode also decreases maximum possible throughput; many 3-wire devices have low performance requirements and are instead designed with low pin count in mind.

- Multi I/O variants such as dual I/O and quad I/O add additional data lines to the standard for increased throughput.
- Components that utilize multi I/O modes can rival the read speed of parallel devices while still offering reduced pin counts. This performance increase enables random access and direct program execution from flash memory (execute-in-place).

Inter-Integrated Circuit (I^2C)

An inter-integrated circuit (Inter-IC or I2C) is a multi-master serial bus that connects low-speed peripherals to a motherboard, mobile phone, embedded system or other electronic devices.

- Philips Semiconductor created I²C with an intention of communication between chips reside on the same Printed Circuit Board (PCB).
- It is a multi-master, multi-slave protocol.
- It is designed to lessen costs by streamlining massive wiring systems with an easier interface for connecting a central processing unit (CPU) to peripheral chips in a television.
- It had a battery-controlled interface but later utilized an internal bus system.
- It is built on two lines
- SDA (Serial Data) – The line for the master and slave to send and receive data
- SCL (Serial Clock) – The line that carries the clock signal.
- Devices on an I2C bus are always a master or a slave. Master is the device which always initiates a communication and drives the clock line (SCL). Usually a microcontroller or microprocessor acts a master which needs to read data from or write data to slave peripherals.
- Slave devices are always responds to master and won't initiate any communication by itself. Devices like EEPROM, LCDs, RTCs acts as a slave device. Each slave device will have a unique address such that master can request data from or write data to it.
- The master device uses either a 7-bit or 10-bit address to specify the slave device as its partner of data communication and it supports bi-directional data transfer.

Working of I²C

- The I2C, data is transferred in messages, which are broken up into frames of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted.
- The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame.
- The following are the bits in data frames:

1. **Start Condition:** The SDA line switches from a high voltage level to a low voltage level before the SCL line switches from high to low.
2. **Stop Condition:** The SDA line switches from a low voltage level to a high voltage level after the SCL line switches from low to high.
3. **Address Frame:** A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.
4. **Read/Write Bit**: A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).
5. **ACK/NACK Bit:** Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.

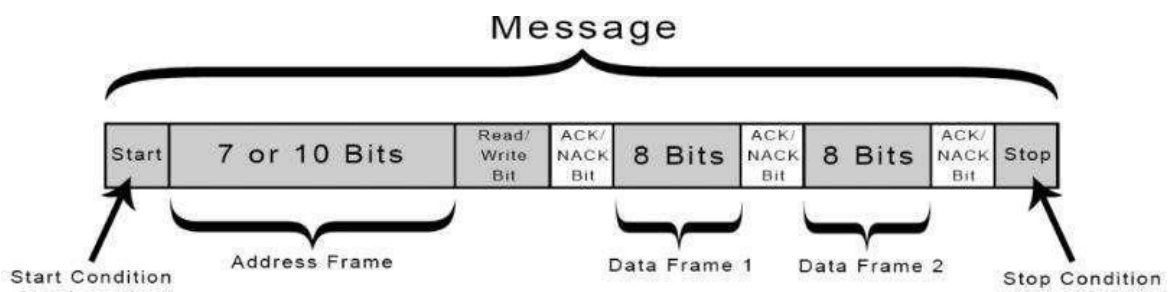


Fig 4.23: I²C Message Format

Addressing:

- I²C doesn't have slave select lines like SP), so it needs another way to let the slave know that data is being sent to it, and not another slave. It does this by addressing. The address frame is always the first frame after the start bit in a new message.

- The master sends the address of the slave it wants to communicate with to every slave connected to it. Each slave then compares the address sent from the master to its own address.
- If the address matches, it sends a low voltage ACK bit back to the master. If the address doesn't match, the slave does nothing and the SDA line remains high.

Read/Write Bit

- The address frame includes a single bit at the end that informs the slave whether the master wants to write data to it or receive data from it. If the master wants to send data to the slave, the read/write bit is a low voltage level. If the master is requesting data from the slave, the bit is a high voltage level.

Data Frame

- After the master detects the ACK bit from the slave, the first data frame is ready to be sent.
- The data frame is always 8 bits long, and sent with the most significant bit first.
- Each data frame is immediately followed by an ACK/NACK bit to verify that the frame has been received successfully.
- The ACK bit must be received by either the master or the slave (depending on who is sending the data) before the next data frame can be sent.
- After all of the data frames have been sent, the master can send a stop condition to the slave to halt the transmission.
- The stop condition is a voltage transition from low to high on the SDA line after a low to high transition on the SCL line, with the SCL line remaining high.

Steps in Data transmission

1. The master sends the start condition to every connected slave by switching the SDA line from a high voltage level to a low voltage level before switching the SCL line from high to low.
2. The master sends each slave the 7 or 10 bit address of the slave it wants to communicate with, along with the read/write bit.
3. Each slave compares the address sent from the master to its own address. If the address matches, the slave returns an ACK bit by pulling the SDA line low for one bit. If the address from the master does not match the slave's own address, the slave leaves the SDA line high.
4. The master sends or receives the data frame.
5. After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful receipt of the frame.
6. To stop the data transmission, the master sends a stop condition to the slave by switching

SCL high before switching SDA high.

Advantages

- It uses two wires.
- This supports multiple masters and multiple slaves.
- ACK/NACK bit gives confirmation that each frame is transferred successfully.
- Well known and widely used protocol

Disadvantages

- Slower data transfer rate than SPI.
- The size of the data frame is limited to 8 bits
- More complicated hardware needed to implement than SPI.

MASS STORAGE

Mass storage refers to various techniques and devices for storing large amounts of data. Mass storage is distinct from memory, which refers to temporary storage areas within the computer. Unlike main memory, mass storage devices retain data even when the computer is turned off.

The mass storage medium includes:

- solid-state drives (SSD)
- hard drives
- external hard drives
- optical drives
- tape drives
- RAID storage
- USB storage
- flash memory cards

Solid State Devices

- Solid-state devices are electronic devices in which electricity flows through solid semiconductor crystals like silicon, gallium arsenide, and germanium rather than through vacuum tubes.
- It do not involve any moving parts or magnetic materials.
- RAM is a solid state device that consists of microchips that store data on non-moving components, providing for fast retrieval of that data.
- Transistors are the most important solid state devices. The transistors contain two p– n junctions, have three contacts or terminals.
- They require the action of perpendicular electrical fields, their behavior is more difficult to understand than that of diodes.
- The different types of transistors are: bipolar junction transistor (BJT) where the current is

amplified, while in the field effect transistor (FET) a voltage controls a current.

- In a solid-state component, the current is confined to solid elements and compounds engineered specifically to switch and amplify it.
- Current flows in two forms: as negatively charged electrons, and as positively charged electron deficiencies called holes.
- In some semiconductors, the current consists mostly of electrons; in other semiconductors, it consists mostly of holes. Both the electron and the hole are called charge carriers.

Hard Drives

- A hard disk drive is a non-volatile memory hardware device that permanently stores and retrieves data on a computer.
- A hard drive is a secondary storage device that consists of one or more platters to which data is written using a magnetic head, all inside of an air-sealed casing.
- Internal hard disks reside in a drive bay, connect to the motherboard using an ATA, SCSI, or SATA cable, and are powered by a connection to the power supply unit.

External Hard Drives

- An external hard drive is a portable storage device that can be attached to a computer through a USB or FireWire connection, or wirelessly.
- External hard drives typically have high storage capacities and are often used to back up computers or serve as a network drive.

Optical Drives

- An Optical Drive refers to a computer system that allows users to use DVDs, CDs and Blu-ray optical drives.
- The drive contains some lenses that project electromagnetic waves that are responsible for reading and writing data on optical discs.
- An optical disk drive uses a laser to read and write data. A laser in this context means an electromagnetic wave with a very specific wavelength within or near the visible light spectrum.
- An optical drive that works with all types of discs will have two separate lenses: one for CD/DVD and one for Blu-ray.
- An optical drive has a rotational mechanism to spin the disc. Optical drives were originally designed to work at a constant linear velocity (CLV) (i.e.) the disc spins at varying speeds depending on where the laser beam is reading, so the spiral groove of the disc passes by the laser at a constant speed.
- An optical drive also needs a loading mechanism: A **tray-loading mechanism,** where the disc is placed onto a motorized tray, which moves in and out of the computer case and **slot-loading mechanism,** where the disc is slid into a slot and motorized rollers are used to move the disc in and out.

Tape disks

- A tape drive is a device that stores computer data on magnetic tape, especially for backup and archiving purposes.
- Tape drives work either by using a traditional helical scan where the recording and playback heads touch the tape, or linear tape technology, where the heads never actually

touch the tape.

- Drives can be rewinding, where the device issues a rewind command at the end of a session, or non-rewinding.
- Rewinding devices are most commonly used when a tape is to be unmounted at the end of a session after batch processing of large amounts of data.
- Non-rewinding devices are useful for incremental backups and other applications where new files are added to the end of the previous session's files.
- The different types of tapes are audio, video and data storage tape.

Redundant Array of Inexpensive Disks (RAID) Storage

- RAID is a way of storing the same data in different places on multiple hard disks to protect data in the case of a drive failure.
- RAID works by placing data on multiple disks and allowing input/output (I/O) operations to overlap in a balanced way, improving performance. Because the use of multiple disks increases the mean time between failures (MTBF), storing data redundantly also increases fault tolerance.
- A RAID controller can be used as a level of abstraction between the OS and the physical disks, presenting groups of disks as logical units. Using a RAID controller can improve performance and help protect data in case of a crash.
- Levels in RAID:

1. **RAID 0 (Disk striping):**

   RAID 0 splits data across any number of disks allowing higher data throughput. An individual file is read from multiple disks giving it access to the speed and capacity of all of them. This RAID level is often referred to as striping and has the benefit of increased performance.

2. **RAID 1 (Disk Mirroring):**

   RAID 1 writes and reads identical data to pairs of drives. This process is often called data mirroring and it's a primary function is to provide redundancy. If any of the disks in the array fails, the system can still access data from the remaining disk(s).

3. **RAID 5 (Striping with parity):**

   RAID 5 stripes data blocks across multiple disks like RAID 0, however, it also stores parity information (Small amount of data that can accurately describe larger amounts of data) which is used to recover the data in case of disk failure. This level offers both speed (data is accessed from multiple disks) and redundancy as parity data is stored across all of the disks.

4. **RAID 6 (Striping with double parity):**

   Raid 6 is similar to RAID 5, however, it provides increased reliability as it stores an extra parity block. That effectively means that it is possible for two drives to fail at once without breaking the array.

5. **RAID 10 (Striping + Mirroring):**

   RAID 10 combines the mirroring of RAID 1 with the striping of RAID 0. Or in other words, it combines the redundancy of RAID 1 with the increased performance of RAID 0. It is best suitable for environments where both high performance and security is required.

Universal Serial Bus (USB) Devices

- USB is a system for connecting a wide range of peripherals to a computer, including pointing devices, displays, and data storage and communications products.
- The Universal Serial Bus is a network of attachments connected to the host computer.
- These attachments come in two types known as **Functions and Hubs.**
- Functions are the peripherals such as mice, printers, etc.
- Hubs basically act like a double adapter does on a power-point, converting one socket, called a port, into multiple ports.
- Hubs and functions are collectively called devices.
- When a device is attached to the USB system, it gets assigned a number called its address. The address is uniquely used by that device while it is connected.
- Each device also contains a number of endpoints, which are a collection of sources and
- destinations for communications between the host and the device.
- The combination of the address, endpoint number and direction are what is used by the host and software to determine along which pipe data is travelling.

Flash Drives

- A flash drive stores data using flash memory. Flash memory uses an electrically erasable programmable read-only (EEPROM) format to store and retrieve data.
- Flash drives are non-volatile, which means they do not need a battery backup.
- Most computers come equipped with USB ports, which detect inserted flash drives and install the necessary drivers to make the data retrievable.
- Computer users can store and retrieve data once the operating system has detected a connection to the USB port.
- Flash drives have a USB mass storage device classification, which means they do not require additional drivers.
- The computer's operating system recognizes a block-structured logical unit, which means it can use any file system or block addressing system to read the information on the flash drive.
- A flash drive enters emulation mode, or acts a hard drive, once it has connected to the USB port. This makes it easier to transfer data between the flash drive and the computer.
- Flash memory is known as a solid state storage device, meaning there are no moving parts — everything is electronic instead of mechanical.

INPUT AND OUTPUT DEVICES

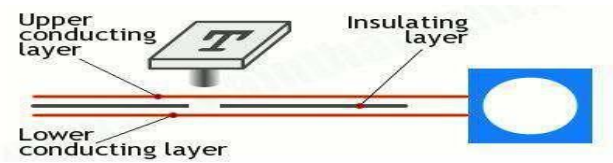
The common input and output devices are discussed here:

Input Devices

Keyboard

- A keyboard has its own processor and circuitry that carries information to and from that processor.

- A large part of this circuitry makes up the **key matrix which is arranged in rows and columns.**
- The key matrix is a grid of circuits underneath the keys.
- In all keyboards each circuit is broken at a point below each key. When a key is presses, it presses a switch, completing the circuit and allowing a tiny amount of current to flow through.
- The mechanical action of the switch causes some vibration, called **bounce,** which the processor filters out.
- If the key is pressed and held continuously, the processor recognizes it as the equivalent of pressing a key repeatedly.
- Another type of keyboard has three layers: top plasticized layer with key positions marked on the top surface and conducting traces on another side; middle layer made of rubber with hole for key positions; bottom metallic layer with raised bumps for key positions.
- When a key is pressed the trace underneath the top layer comes in contact with the bump in



- the last layer, thus completing an electrical circuit. The current flow is sensed by the microcontroller.

Fig 4.24: Layers in keyboard

Mouse

- A computer mouse is a hand-held pointing device that detects two-dimensional motion relative to a surface.
- This motion is typically translated into the motion of a pointer on a display, which allows a smooth control of the graphical user interface.
- There are two main kinds of mice: rolling rubber ball mouse or optical mouse.
- As the mouse is moved, the ball rolls under its own weight and pushes against two plastic rollers linked to thin wheels.
- One of the wheels detects movements in an up-and-down direction (y-axis) and the other detects side-to-side movements (x-axis).
- If the mouse is moved straight up, only the y-axis wheel turns. If the mouse is moved to the right, only the x-axis wheel turns.

- The optical mouse shines a bright light down onto the desk from an LED mounted on the bottom of the mouse.
- The light bounces straight back up off the desk into a photocell also mounted under the mouse, a short distance from the LED.
- The photocell has a lens in front of it that magnifies the reflected light, so the mouse can respond more precisely to your hand movements.
- As the mouse is pushed, the pattern of reflected light changes, and the chip inside the

mouse uses this to figure out the motion.

Trackball, Joystick and Touch pad

- A trackball can also be used as an alternative to a mouse. This device also has buttons similar to those on a mouse.
- It holds a large moving ball on the top. The body of the trackball is not moved. The ball is rolled with fingers. The position of the cursor on the screen is controlled by rotating the ball.
- The main benefit of the trackball over a mouse is that it takes less space to move. The trackball is often included in laptop computers. The standard desktop computer also uses a trackball operated as a separate input device.
- A touchpad is a small, plane surface over which the user moves his finger. The user controls the movement of the cursor on the screen by moving his fingers on the touchpad. It is also known as a track pad.
- A touchpad also has one or more buttons near it. These button work like mouse buttons. Touchpads are commonly used with notebook computers.
- A joystick consists of a base and a stick. The stick can be moved in several directions to shift an object anywhere on the computer screen.
- A joystick can perform a similar function to a mouse or trackball. It is often considered less comfortable and efficient. The most common use of a joystick is for playing computer games.

Scanners

- Scanners operate by shining light at the object or document being digitized and directing the reflected light onto a photosensitive element.
- In most scanners, the sensing medium is an electronic, light-sensing integrated circuit known as a charged coupled device (CCD).
- Light-sensitive photo sites arrayed along the CCD convert levels of brightness into electronic signals that are then processed into a digital image.
- A scanner consists of a flat transparent glass bed under which the CCD sensors, lamp, lenses, filters and also mirrors are fixed.
- The document has to be placed on the glass bed. There will also be a cover to close the scanner.
- The lamp brightens up the text to be scanned. Most scanners use a cold cathode fluorescent lamp (CCFL).
- A stepper motor under the scanner moves the scanner head from one end to the other. The movement will be slow and is controlled by a belt.
- The scanner head consists of the mirrors, lens, CCD sensors and also the filter. The scan head moves parallel to the glass bed and that too in a constant path.
- As the scan head moves under the glass bed, the light from the lamp hits the document and is reflected back with the help of mirrors angled to one another.
- According to the design of the device there may be either 2-way mirrors or 3-way mirrors.
- The mirrors will be angled in such a way that the reflected image will be hitting a smaller surface.
- In the end, the image will reach a lens which passes it through a filter and causes the image

to be focused on CCD sensors.

- The CCD sensors convert the light to electrical signals according to its intensity.
- The electrical signals will be converted into image format inside a computer.

Output Devices

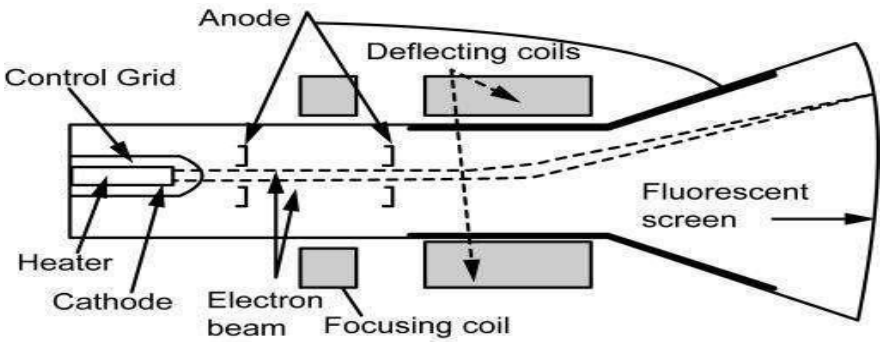
Video Displays

- The CRT monitors were fundamental output display device.
- The CRT or cathode ray tube, is the picture tube of a monitor.
- The back of the tube has a negatively charged cathode.
- The electron gun shoots electrons down the tube and onto a charged screen.
- The screen is coated with a pattern of dots using phosphor that glow when struck by the electron stream.
- The image on the monitor screen is usually made up from at least tens of thousands of such tiny dots glowing on command from the computer.
- The closer together the pixels are, the sharper the image on screen.
- The distance between pixels on a computer monitor screen is called its dot pitch and is measured in millimeters. Most monitors have a dot pitch of 0.28 mm or less.
- There are two electromagnets around the collar of the tube which deflect the electron beam.
- The beam scans across the top of the monitor from left to right, is then blanked and moved back to the left-hand side slightly below the previous trace (on the next scan line), scans across the second line and so on until the bottom right of the screen is reached.
- The beam is again blanked, and moved back to the top left to start again.
- This process draws a complete picture, typically 50 to 100 times a second.
- The number of times in one second that the electron gun redraws the entire image is called the refresh rate and is measured in hertz (cycles per second).
- It is common, particularly in lower priced equipment, for all the odd-numbered lines of an image to be traced, and then all the even-numbered lines; the circuitry of such an interlaced display need to be have only half the speed of a non-interlaced display.
- An interlaced display, particularly at a relatively low refresh rate, can appear to some observers to flicker, and may cause eye strain and nausea.

- The intensity or strength of the electron beam is controlled by setting the voltage levels.
- The number of electrons that hits the screen determines the light emitted by the screen. When the voltage is varied in the electron gun, the brightness of the display also varies.
- The focusing hardware focuses the beam at all positions on the screen.
- The deflection of electron beam is controlled by electric or magnetic fields.
- Two pairs of coils mounted on the CRT to produce the necessary defection.

- The coils are placed in such a way that, the magnetic field produced by them results in traverse deflection force that is perpendicular to the magnetic field and electron beam.

Fig 4.25: CRT Monitor



- An LED screen is an LCD screen, but instead of having a normal CCFL backlight, it uses light-emitting diodes (LEDs) as a source of light behind the screen.
- An LED is more energy efficient and a lot smaller than a CCFL, enabling a thinner television screen.

Printers

- A printer is an electromechanical device which converts the text and graphical documents from electronic form to the physical form.
- They are the external peripheral devices which are connected with the computers or laptops through a cable or wirelessly to receive input data and print them on the papers.
- Quality of printers is identified by its features like color quality, speed of printing, resolution etc. Modern printers come with multipurpose functions i.e. they are combination of printer, scanner, photocopier, fax, etc.
- Broadly printers are categorized as impact and non impact printers.

Daisy Wheel Printers

- Daisy wheel printers print only characters and symbols and cannot print graphics. They are generally slow with a printing speed of about 10 to 75 characters per second.
- A circular printing element is the heart of these printers that contains all text, numeric characters and symbols mould on each petal on the circumference of the circle.
- The printing element rotates rapidly with the help of a servo motor and pauses to allow the printing hammer to strike the character against the paper.

Dot Matrix Printers

- It is a popular computer printer that prints text and graphics on the paper by using tiny dots to form the desired shapes.
- It uses an array of metal pins known as print head to strike an inked printer ribbon and produce dots on the paper.
- These combinations of dots form the desired shape on the paper.
- The key component in the dot matrix printer is the print head which is about one inch long and contains a number of tiny pins aligned in a column varying from 9 to 24.
- The print head is driven by several hammers which force each pin to make contact with the paper at the certain time. These hammers are pulled by small electromagnet which is energized at a specific time depending on the character to be printed.

- The timings of the signals sent to the solenoids are programmed in the printer for each character.

Inkjet printers

- Inkjet printers are most popular printers for home and small scale offices as they have a reasonable cost and a good quality of printing as well.
- An inkjet printer is made of the following parts:

i) Print head – It is the heart of the printer which holds a series a nozzles which sprays the ink drops over the paper.
ii) Ink cartridge – It is the part that contains the ink for printing. Generally monochrome (black & white) printers contain a black colored ink cartridges and a color printer

contains two cartridges – one with black ink and other with primary colors (cyan, magenta and yellow).

iii) Stepper motor – It is housed in the printer to move the printer head and ink cartridges back and forth across the paper.

iv) Stabilizer bar – A stabilizer bar is used in printer to ensure the movement of print head is précised and controlled over the paper.

v) Belt – A belt is used to attach the print head with the stepper motor.

vi) Paper Tray – It is the place where papers are placed to be printed.

vii) Rollers – Printers have a set of rollers that helps to pull paper from the tray for printing purpose.

viii) Paper tray stepper motor- another stepper motor is used to rotate the rollers in order to pull the paper in the printer.

ix) Control Circuitry – The control circuit takes the input from the computer and by decoding the input controls all mechanical operation of the printer.

Laser Printers

- Laser printers are the most popular printers that are mainly used for large scale qualitative printing.
- They are among the most popularly used fastest printers available in the market.
- A laser printer uses a slight different approach for printing. It does not use ink like inkjet printers, instead it uses a very fine powder known as Toner.
- The control circuitry is the part of the printer that talks with the computer and receives the printing data.
- A Raster Image Processor (RIP) converts the text and images in to a virtual matrix of dots.
- The photo conducting drum which is the key component of the laser printer has a special coating which receives the positive and negative charge from a charging roller.
- A rapidly switching laser beam scans the charged drum line by line. When the beam flashes on, it reverses the charge of tiny spots on the drum, respecting to the dots that are to be printed black.

- As soon the laser scans a line, a stepper motor moves the drum in order to scan the next line by the laser.
- A developer roller plays the vital role to paste the tonner on the paper. It is coated with charged tonner particles.
- As the drum touches the developer roller, the charged tonner particles cling to the discharged areas of the drum, reproducing your images and text reversely.
- Meanwhile a paper is drawn from the paper tray with help of a belt. As the paper passes through a charging wire it applies a charge on it opposite to the toner's charge.
- When the paper meets the drum, due to the opposite charge between the paper and toner particles, the toner particles are transferred to the paper.
- A cleaning blade then cleans the drum and the whole process runs smoothly continuously.
- Finally paper passes through the fuser which is a heat and presser roller, melts the toner and fixes on the paper perfectly.