DIGITAL ELECTRONICS

LECTURE NOTE

Based on New syllabus (2020-21) circulated by SCTE&VT, Odisha for 5th Semester Diploma in Electrical Engineering courses approved by AICTE, New Delhi

PREPARED BY

SIBA PRASAD SAMANTARAYA

Lecturer (Electronics), Department of Electrical Engineering

Govt. Polytechnic Kalahandi

Advantages of Digital signal on Analog signal

1. storage of digital signal is possible
2. Processing of digital signal is easy.
3. With less noise interference digital signal can be transmited to long distance

Logic level

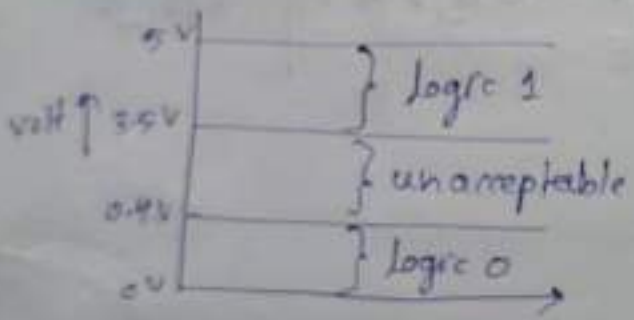
There are 2 types of logic level

1. Positive logic

   +ve logic = logic 1 = 1

2. Negative logic

   -ve logic = logic 0 = 0



(Practical case) Range

0V → 0.4V (due to leakage voltage)

3.5 → 5V

Number system

There are 4 types of number system

1. Binary
2. Octal
3. Decimal
4. Hexadecimal

Type of no system	Base/ Radix	Symbol	example
Binary	2	0,1	$(101\cdot11)_2$
Octal	8	0,1,2,3,4,5,6,7	$(756\cdot23)_8$
Decimal	10	0,1,2,3,4,5,6,7,8,9	$(910\cdot23)_{10}$
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F	$(A2B\cdot7A)_{16}$

Conversion

1. Number in base 'r' to decimal

$$a_n r^n + a_{n-1} r^{n-1} + \cdots + a_1 r^1 + a_0 r^0 \cdot + a_{-1} r^{-1} + \cdots + a_{-m} r^{-m}$$

where $a_j \rightarrow$ coefficient

$r \rightarrow$ radix

Binary to decimal

Eg → $(101\cdot11)_2$

$= 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2}$

$= 4 + 0 + 1 + \frac{1}{2} + \frac{1}{4}$

$= 5 + \frac{2+1}{4}$

$= 5 + \frac{3}{4}$

$= (5.75)_{10}$

Octal to decimal

Eg → $(756\cdot23)_8$

$= 7\times8^2 + 5\times8^1 + 6\times8^0 + 2\times8^{-1} + 3\times8^{-2}$

$= 448 + 40 + 6 + \frac{2}{8} + \frac{3}{64}$

$= 494 + \frac{16+3}{64}$

$= 494.2968$

2. Number in decimal to base (r)

decimal to binary

Eg → $(910.23)_{10}$

2	910	
2	455	0
2	227	1
2	113	1
2	56	1
2	28	0
2	14	0
2	7	0
2	3	1
2	1	1
	0	1

For integer

$(910)_{10} = 1110001110$

	Integer	Fraction
$0.23 \times 2 =$	0	.46
$0.46 \times 2 =$	0	.92
$.92 \times 2 =$	1	.84
$.84 \times 2 =$	1	.68
$.68 \times 2 =$	1	.36

Here fraction can never be expressed exactly in binary. This process may terminated after a few steps.

$(0.23)_{10} = (.00111)_2$

For fraction

$\therefore (910.23)_{10} = (1110001110.00111)_2$

decimal to octal

Eg → $(910.23)_{10}$

8	910	
8	113	6
8	14	1
8	1	6
	0	1

$0.23 \times 8 = 1 \quad .84$

$0.84 \times 8 = 6 \quad .72$

$0.72 \times 8 = 5 \quad .76$

$0.76 \times 8 = 6 \quad .08$

$(910.23)_{10} = (1616.1656)_8$

decimal to Hexadecimal

Eg → $(910.23)_{10}$

16 | 910		
16 | 56	14	↑
16 | 3	8	
0	3	

$0.23 \times 16 =$	3	.68
$0.68 \times 16 =$	10	.88
$.88 \times 16 =$	14	.08
$.08 \times 16 =$	1	.28

10 → A
11 → B
12 → C
13 → D
14 → E
15 → F

$(910.23)_{10} = (38E.3AE1)_{16}$

3. Number in base 'r' to base 'x'

octal to Binary

Eg → $(756.23)_8$

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

$(756.23)_8 = (111\,101\,110 \cdot 010\,011)_2$

Binary to octal

Eg → $(1011.1010)_2$

1011 . 1010

Put 0 at required position to make group of 3 bits

001 011 · 101 000

1 3 · 5 0

$\therefore (1011 \cdot 1010)_2 = (13 \cdot 50)_8$

Hexadecimal to binary

eg → $(A2B \cdot 7A)_{16}$

0 → 0000
1 → 0001
2 → 0010
3 → 0011
4 → 0100
5 → 0101
6 → 0110
7 → 0111
8 → 1000
9 → 1001
A → 1010
B → 1011
C → 1100
D → 1101
E → 1110
F → 1111

A 2 B · 7 A

1010 0010 1011 0111 1010

$\therefore (A2B \cdot 7A)_{16} = (1010\ 0010\ 1011 \cdot 01111010)_2$

Binary to Hexadecimal

eg → 101011 · 10101

10 1011 · 1010 1

put 0 at needed blank position to make group of 4 bits

0010 1011 · 1010 1000

2 B A 8

$\therefore (101011 \cdot 10101)_2 = (2B \cdot A8)_{16}$

Binary Arithmetics

Binary Addition

Rules :- 0+0 = 0
0+1 = 1
1+0 = 1
1+1 = 0, with carry 1

Eg :- $(1101.101)_2 + (111.011)_2$

$$\begin{array}{rl} 1101.101 & \longrightarrow 13.625 \\ +\ 111.011 & \longrightarrow 7.375 \\ \hline 10101.000 & \qquad 21.000 \end{array}$$

Binary subtraction

Rules :- 0-0 = 0
1-1 = 0
0-1 = 1, with borrow of 1
1-0 = 1

Eg → $(1010.01)_2 - (111.111)_2$

$$\begin{array}{rl} 1010.010 & \longrightarrow 10.250 \\ -\ 0111.111 & \longrightarrow 7.875 \\ \hline 0010.011 & = 2.375 \end{array}$$

Binary multiplication

Rules :- 0×0 = 0
0×1 = 0
1×0 = 0
1×1 = 1

Eg → $(1101)_2 \times (110)_2 = ?$

$$\begin{array}{r} 1101 \\ \times\ 110 \\ \hline 0000 \\ 1101 \\ 1101 \\ \hline 1001110 \end{array} \longrightarrow \begin{array}{r} 13 \\ \times\ 6 \\ \hline 78 \end{array}$$

Binary Division

eg → $(101101)_2 \div (110)_2 = ?$

$$\begin{array}{r|l} 110 & 101101 \\ & 110 \\ \hline & 1010 \\ & 110 \\ \hline & 1001 \\ & 110 \\ \hline & 0110 \\ & 110 \\ \hline & 0000 \end{array} \quad 111 \cdot 1$$

Assignment

1) $(1101110 \cdot 011)_2 = (\quad)_{10}$

2) $(197 \cdot 56)_{10} = (\quad)_2$

3) $(30)_8 = (\quad)_{10}$

4) $(562 \cdot 3)_{10} = (\quad)_8$

5) $(1110111 \cdot 10)_2 = (\quad)_8$

6) $(13265)_8 = (\quad)_2$

7) $(94Fc)_{16} = (\quad)_2$

8) $(11011.01)_2 = (\quad)_{16}$

9) $(3c94)_{16} = (\quad)_{10}$

10) $(15514)_{10} = (\quad)_{16}$

11) $(732)_8 = (\quad)_{16}$

12) $(A13)_{16} = (\quad)_8$

(13) $100110.10 + 10111.01 = (\quad)$

(14) $100110.10 - 11011.10 = (\quad)$

(15) $1011.101 \times 101.11 = (\quad)$

(16) $11010.11 \div 101 = (\quad)$

(17) $10110 \times 10.1 = (\quad)$

Representation of Signed Numbers

1) sign magnitude form
2) Complement form → 1's, 2's

Sign Magnitude Form

sign bit	Magnitude bits
(system)	

- An additional bit called the sign bit is placed in front of number

$+ \rightarrow 0$

$- \rightarrow 1$

Eg → +7 = 0111

−7 = 1,111

1's complement form

- +ve number representation is same as sign magnitude form

sign bit	magnitude bits

Eg → +9 = 01001

- −ve number representation syntax is as follows

sign bit	1's complement of actual binary

- The 1's complement of a binary number is obtained by changing each 0 to 1 & each 1 to 0.

Eg → −7 = 1000

2's complement form

- +ve number representation is same as sign magnitude form

Eg → +12 = 01100

- −ve number representation syntax is as follows

sign bit	2's complement of actual binary

- 2's complement of Binary number = 1's complent of Binary number + 1.

Eg → −7 = 1001

2's complement subtraction

(1) Subtraction smaller number from larger

steps

1. Determine the 2's complement of subtrahend.
2. Add 2's complent of subtrahend with minuend.
3. If carry generates discard the carry. Result is +ve & is in normal binary form.

Eg → Subtract 14 from 46.

Sol^n/ $46 - 14 = ?$

	32	16	8	4	2	1
Binary of 46 =	1	0	1	1	1	0
Binary of 14 =	0	0	1	1	1	0

2's complement of 14 = $\begin{array}{r} 1\,1\,0\,0\,0\,1 \\ +\quad\quad 1 \\ \hline 1\,1\,0\,0\,1\,0 \end{array}$

$$\begin{array}{r} 1\,0\,1\,1\,1\,0 \\ +\;1\,1\,0\,0\,1\,0 \\ \hline \boxed{1}\,1\,0\,0\,0\,0\,0 \end{array}$$

↑ discard it

The result is +ve & is in normal binary form.

(2) Subtraction larger number from smaller

steps

1. Determine 2's complement of subtrahend
2. Add it to minuend.
3. If carry is not generated the result is -ve & is in 2's complent form.

Eg → subtract 75 from 26.

Soln 26 − 75 = ?

64 32 16 8 4 2 1

Binary of 26 = 0 0 1 1 0 1 0

Binary of 75 = 1 0 0 1 0 1 1

2's complement of 75 = 0 1 1 0 1 0 1

$$\begin{array}{r} 0\,0\,1\,1\,0\,1\,0 \\ +\;0\,1\,1\,0\,1\,0\,1 \\ \hline 1\,0\,0\,1\,1\,1\,1 \end{array}$$ (No carry)

$$\begin{array}{r} 26 \\ -75 \\ \hline =49 \end{array}$$

The result is -ve & is in 2's complement form.

Codes

1. Weighted code
2. Non-weighted code

Weighted code

- In these code each digit position has a particular weight
- Ex:- Binary, octal, Decimal, Hexadecimal, BCD.
- The sum of all digit multiplied by a weight gives the total amount being represented.

Non-weighted code

- In these code no positional weight is assigned to any binary bit.
- Ex:- Excess-3 code, Gray code.

BCD code

- The name "BCD code" is the short form of Binary coded Decimal code.
- In this code each decimal digit is represented by a 4-bit binary number.
- We can convert only (0 to 9) number in to BCD.

eg → $(92.3)_{10} = (1001\ 0010 \cdot 0011)_{BCD}$

Excess-3 code / XS-3 code

- XS-3 code is used to represent decimal digit

steps

1. Convert each decimal digit into BCD
2. Add 0011 to each four bit group in BCD

eg → $(92.3)_{10} = (\quad)_{XS-3}$

$92 \cdot 3 \longrightarrow 1001\ 0010 \cdot 0011$

$+\ 0011\ 0011\ 0011$

$1100\ 0101 \cdot 0110$

- XS-3 code is self complimenting code.

Decimal	BCD	XS-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

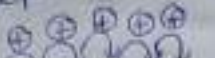
- This code is used for error checking & correction in communi

Gray code

- Binary code is converted into Gray code.

steps

1. MSB of Gray is same as MSB of Binary.
2. To get next Gray code bit x-OR the MSB of Binary with next bit of Binary.
3. To get further next bit of Gray code do x-OR the consequtive next bit & previous bit. Repeat it until get LSB of gray code.



Eg → $(1011101)_2 = (1110111)_{Gray}$

Decimal	Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

- Gray code is also known as Reflected binary code.
- It is a unit distance code & Minimum error code.
- Two successive values differ in only one bit.
- Binary number is converted to Gray code to reduce switching operation.
- Gray code is also known as cyclic code.
- This code is used for error checking & correction in digital communication.

Gray to Binary conversion

Eg → (1 0 1 1 0 1 1)$_{Gray}$

(1 1 0 1 1 0 1)$_{Binary}$

Assignment

1. Subtract 130 from 128 using 2's complement.
2. Subtract 18 from 20 using 2's complement.
3. Subtract $(11100)_2$ from $(10011)_2$ using 2's complement.
4)
5)
6)

Parity bit

- Parity bit is an extra bit that to be added with data bits to detect the errors appears in the digital trasmission.
- A bit either 1 or 0 is added as a parity bits with data bit.
- Parity bit is two types
  1. Even parity
  2. Odd parity

Eaven Parity

- In even parity the parity bit is selected as '0' if number of 1's present in the data is even, while the parity bit is selected '1' if number of 1's present in the data is odd.

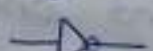
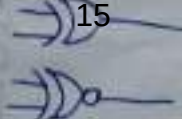
Odd parity

- In odd parity, the parity bit is selected as '0' if number of 1's present in the data is odd, while the parity bit is selected as '1' if number of '1' present in the data is even.

Data bit	Even parity	odd parity
0000	0	1
0001	1	0
0010	1	0
0011	0	1
0100	1	0
0101	0	1
0110	0	1
0111	1	0
1000	1	0
1001	0	1
1010	0	1
1011	1	0
1100	0	1
1101	1	0
1110	1	0
1111	0	1

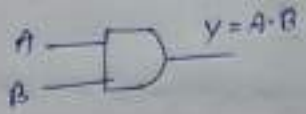
Logic Gate

- It is an electronic circuit having one one more than one input and only one output.
- The relationship between the input and output is based on a certain logic.
- Logic gates are the basic building blocks of any digital system.

- Basic
  - AND 
  - OR 
  - NOT 
- Universal
  - NAND 
  - NOR 
- other.
  - X-OR 
  - X-NOR 

AND Gate

- This gate perform an AND operation
- AND operation is represented as '.'
- It has n inputs ($n \geq 2$) and one output



Input		output
A	B	Y=A·B
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

- This gate perform an OR operation
- OR operation is represented as '+'
- It has n inputs ($n \geq 2$) and one output

Input		output
A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

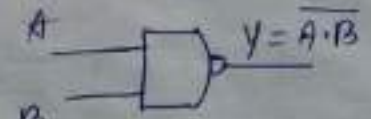
NOT Gate

- NOT gate is also known as inverter gate.
- It has one input and one output.

input	output
A	Y=Ā
0	1
1	0

NAND Gate

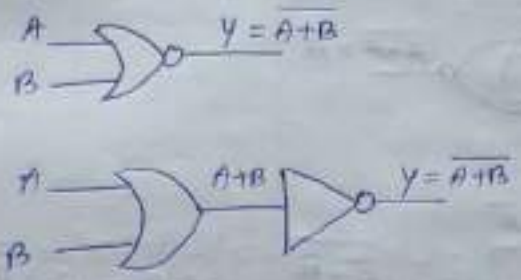
- A NOT-AND operation is known as NAND operation.
- It has n input ($n \geq 2$) and one output.



Input			output
A	B	A·B	A·B
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

NOR Gate

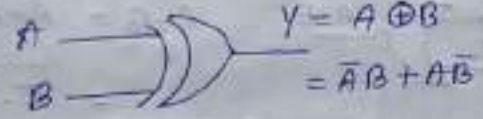
- A NOT-OR operation is known as NOR operation.
- It has n input ($n>=2$) and one output.



Inputs		output
A	B	A+B
0	0	1
0	1	0
1	0	0
1	1	0

X-OR Gate

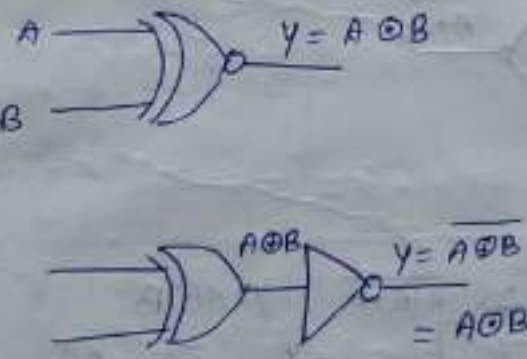
- X-OR or Ex-OR is an odd one detector.
- XOR operation is represented as '⊕'.
- It has n inputs ($n>=2$) and one output.



Inputs						output
A	B	Ā	ĀB	B̄	AB̄	ĀB + AB̄
0	0	1	0	1	0	0
0	1	1	1	0	0	1
1	0	0	0	1	1	1
1	1	0	0	0	0	0

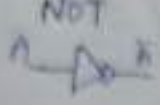
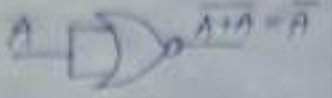
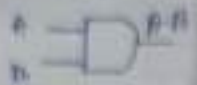
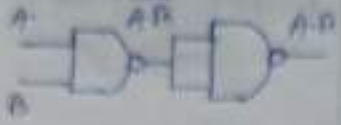
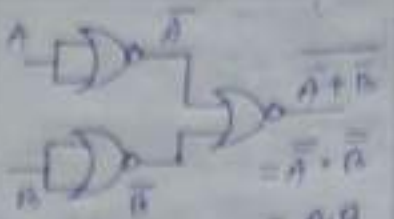
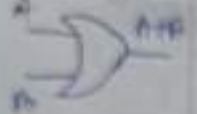
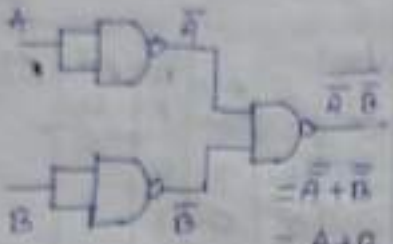
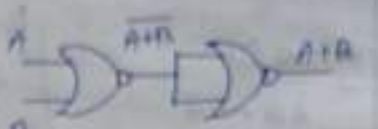
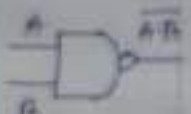
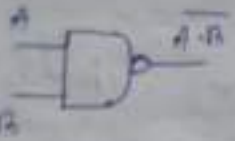
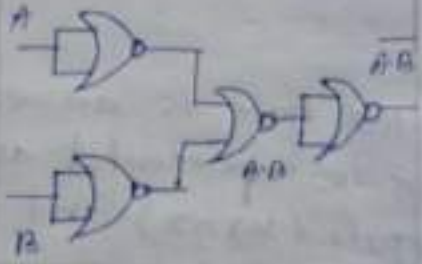
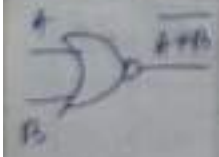
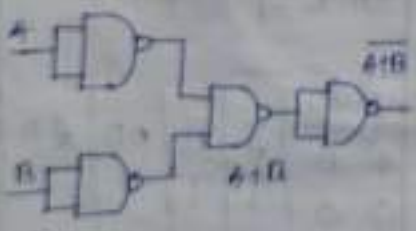
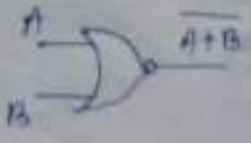
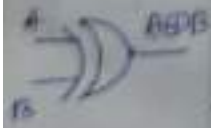
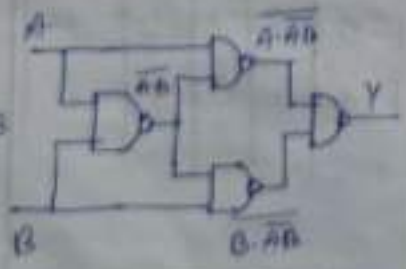
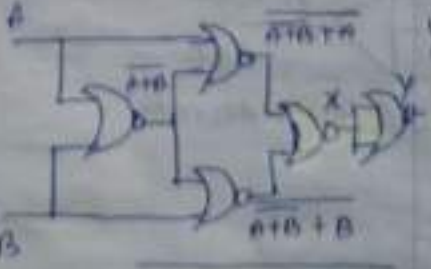
XNOR Gate

- XNOR or EX-NOR gate is an even one detector.
- XNOR operation is represented as '⊙'
- It has n inputs ($n>=2$) and one output.
- NOT-XOR operation is known as XNOR operation.



Inputs						output
A	B	Ā	B̄	AB	ĀB̄	AB + ĀB̄
0	0	1	1	0	1	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	1

Realisation of Logic gates using universal gates

Logic gates	using NAND gate	using NOR gate	
NOT			
AND				$A \cdot B = \overline{\overline{A} \cdot \overline{B}}$ $= \overline{\overline{A} + \overline{B}}$ According to De-morgan's theorem
OR				$A+B = \overline{\overline{A}+\overline{B}}$ $= \overline{\overline{A} \cdot \overline{B}}$ According to Demorgan's theorem
NAND				
NOR				
XOR		 $Y = \overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}$ $= A \oplus B$	 $Y = \overline{\overline{A+B+A} + \overline{A+B+B}}$ $Y = \overline{\overline{A+B+A}} + \overline{\overline{A+B+B}}$ $= A \oplus B$	$Y = A \oplus B$ $= \overline{A}B + A\overline{B}$ $= \overline{\overline{A}B} + \overline{A\overline{B}}$ $= \overline{\overline{A}B} \cdot \overline{A\overline{B}}$ $= \overline{\overline{A}B + B\overline{B}} \cdot \overline{A\overline{B} + A\overline{A}}$ $= \overline{B(\overline{A}+\overline{B})} \cdot \overline{A(\overline{A}+\overline{B})}$

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

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

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

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

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

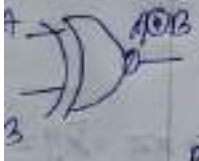
$$= (\overline{\overline{A}}+\overline{\overline{B}})\cdot\overline{A} + (\overline{\overline{A}}+\overline{\overline{B}})\cdot\overline{B}$$

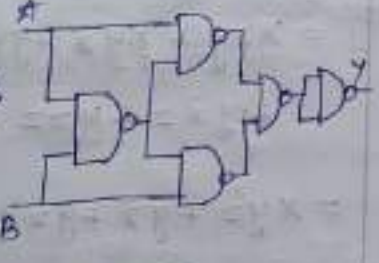
$$= (A+B)\cdot\overline{A} + (A+B)\cdot\overline{B}$$

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

$$= \overline{A}B + A\overline{B} = A \oplus B$$

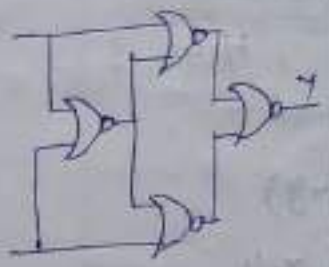
XNOR





$$Y = \overline{A \odot B}$$

$$= A \odot B$$



$$Y = A \odot B$$

Laws OF Boolean Algebra

1. $\overline{0} = 1$
2. $\overline{1} = 0$
3. If $x=0$, $\overline{x}=1$
4. If $x=1$, $\overline{x}=0$
5. $\overline{\overline{x}} = x$

complementation laws

6. $x \cdot 0 = 0$
7. $x \cdot 1 = x$
8. $x \cdot x = x$
9. $x \cdot \overline{x} = 0$

AND laws

10. $x + 0 = x$
11. $x + 1 = 1$
12. $x + x = x$
13. $x + \overline{x} = 1$

OR laws

14. $x + y = y + x$
15. $x \cdot y = y \cdot x$

commutative law

16. $x + (y+z) = (x+y) + z$
17. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$

Associative law

18. $x \cdot (y+z) = (x \cdot y) + (x \cdot z)$
19. $x + (y \cdot z) = (x+y) \cdot (x+z)$
20. $x + (\overline{x} \cdot y) = x + y$

Distributive law

21. $x + x \cdot y = x$
22. $x \cdot (x+y) = x$

Absorption law

23. $x \cdot y + \overline{x} \cdot z + yz = x \cdot y + \overline{x} z$
24. $(x+y) \cdot (\overline{x}+z) \cdot (y+z) = (x+y) \cdot (\overline{x}+z)$

consensus law

25. $\bar{x}\cdot y + \bar{x}z = (x+z)(\bar{x}+y)$
26. $(x+y)\cdot(\bar{x}+z) = (x\cdot z)+(\bar{x}\cdot y)$

] Transposition law

27. $\overline{x+y} = \bar{x}\cdot\bar{y}$
28. $\overline{x\cdot y} = \bar{x}+\bar{y}$

] Demorgan's law

Q) $x + x\cdot y = x$, How ?

Proof

$L.H.S = x + x\cdot y$
$= x(1+y)$
$= x\cdot 1 = x$ R.H.S

Q) $x\cdot(x+y) = x$, How ?

Proof

$L.H.S = x\cdot(x+y)$
$= x\cdot x + x\cdot y$
$= x + x\cdot y$
$= x(1+y)$
$= x\cdot 1$
$= x$ R.H.S

Q) $x\cdot y + \bar{x}z + y\cdot z = x\cdot y + \bar{x}z$
How ?

Proof

$L.H.S = x\cdot y + \bar{x}z + y\cdot z$
$= x\cdot y + \bar{x}z + y\cdot z(x+\bar{x})$
$= x\cdot y + \bar{x}z + yzx + yz\bar{x}$
$= xy(1+z) + \bar{x}z(1+y)$
$= xy\cdot 1 + \bar{x}z\cdot 1$
$= xy + \bar{x}z$ R.H.S

Q) $(x+y)\cdot(\bar{x}+z)\cdot(y+z)$
$= (x+y)\cdot(\bar{x}+z)$
How ?

$L.H.S = (x+y)\cdot(\bar{x}+z)\cdot(y+z)$
$= (x\bar{x} + xz + y\bar{x} + yz)\cdot(y+z)$
$= (xz + y\bar{x} + yz)\cdot(y+z)$
$= xzy + y\cdot y\bar{x} + yyz + xzz$
$+ y\bar{x}z + yzz$
$= xyz + y\bar{x} + yz + xz + \bar{x}yz$
$+ yz$
$= yz(x+1) + \bar{x}y(1+z)$
$+ xz$
$= yz + \bar{x}y + xz$
$= yz + \bar{x}y + xz + x\bar{x}$
$= z(x+y) + \bar{x}(x+y)$
$= (x+y)\cdot(\bar{x}+z)$ R.H.S

Q) $\bar{x}y + \bar{x}z = (x+z)(\bar{x}+y)$
How ?

Proof

R.H.S $= (x+z)\cdot(\bar{x}+y)$
$= x\bar{x} + xy + \bar{x}z + zy$
$= xy + \bar{x}z + zy$
$= xy + \bar{x}z + zy(x+\bar{x})$
$= xy + \bar{x}z + xyz + \bar{x}yz$
$= xy(1+z) + \bar{x}z(1+y)$

$= xy + \bar{x}z$ L.H.S

Q) $(x+y)\cdot(\bar{x}+z) = (x\cdot z) + (\bar{x}\cdot y)$

How ?

Proof

L.H.S $= (x+y)(\bar{x}+z)$

$= x\bar{x} + xz + y\bar{x} + yz$

$= xz + y\bar{x} + yz$

$= xz + y\bar{x} + yz(x+\bar{x})$

$= xz + y\bar{x} + xyz + \bar{x}yz$

$= xz(1+y) + \bar{x}y(1+z)$

$= xz + \bar{x}y$ R.H.S

Q) $\overline{x+y} = \bar{x}\,\bar{y}$

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

How ?

Proof

x	y	$\bar{x}$	$\bar{y}$	$x+y$	$\overline{x+y}$	$\bar{x}\,\bar{y}$	$x\cdot y$	$\overline{x\cdot y}$	$\bar{x}+\bar{y}$
0	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	0	0	1	1
1	0	0	1	1	0	0	0	1	1
1	1	0	0	1	0	0	1	0	0

Representation of Boolean Function

Boolean function can be represented in four form

1) SOP → sum of product form
2) POS → Product of sum form
3) SSOP → standard sum of product form or disjunctive canonical form / canonical SOP / expanded SOP
4) SPOS → standard Product of sum form or conjunctive canonical form / canonical POS / expanded POS

SOP

→ In this form products of variables are added.

Ex :- $F(A,B,C) = A + BC + A\bar{C}$

POS

→ In this form sums of variables are multiplied.

Ex :- $F(A,B,C) = (A+B)(B+\bar{C})$

SSOP

→ This is a SOP form in which each product term contains all the variables of the function either in complemented or uncomplemented form.

→ Here each product term is called minterm.

eg → $F(A,B,C) = AB\bar{C} + \bar{A}BC + ABC$

$= m_6 + m_3 + m_7$

$= \sum m(3,6,7)$

SPOS

→ This is a POS form in which each sum term contains all the variables of the function either in complemented or uncomplemented form.

→ Here each sum term is called Maxterm.

eg → $F(A,B,C) = (A+\bar{B}+C)\cdot(\bar{A}+B+C)\cdot(\bar{A}+B+\bar{C})$

$= M_2 \cdot M_4 \cdot M_5$

$= \Pi M(2,4,5)$

Conversion SOP to SSOP

Steps

1. Find out the number of variables
2. Identify the variables missing in each minterm.
3. Multiply (variable + its complement)
4. Neglect the repeated terms

eg → $F = ABC + \bar{A}B + CD$

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

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

$+ (CDA + CD\bar{A})(B+\bar{B})$

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

$+ ABCD + \bar{A}BCD + A\bar{B}CD + \bar{A}\bar{B}CD$

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

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

or

$= m_{15} + m_{14} + m_{13} + m_{12} + m_7 + m_{11} + m_3$

or

$= \sum M(3, 7, 11, 12, 13, 14, 15)$

Conversion POS to SPOS

steps

1. Find out the number of variables.
2. Identify the variables missing in each maxterm.
3. Add (variable × its complement)
4. Neglect the repeated terms.

eg → $F = (A+B) \cdot C$

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

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

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

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

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

or

$= M_0 \cdot M_4 \cdot M_1 \cdot M_2 \cdot M_6$

or

$= \Pi M(0, 1, 2, 4, 6)$

Conversion between canonical forms

SSOP to SPOS

eg → $F(A,B,C) = \sum m(0,2,4,6,7)$

Convert into maxterm.

sol^n/ $F(A,B,C) = \sum m(0,2,4,6,7)$

$\overline{F(A,B,C)} = \sum m(1,3,5)$

$= m_1 + m_3 + m_5$

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

$\overline{\overline{F(A,B,C}} = \overline{\bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C}$

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

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

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

OR/

$= M_1 \cdot M_3 \cdot M_5$

$= \Pi M(1,3,5)$

Note

- Complement of minterm = Maxterm, i.e, $\overline{m_j} = M_j$
- Complement of maxterm = minterm

SPOS to SSOP

eg → $F(A,B,C) = (A+B+\bar{C}) \cdot (\bar{A}+\bar{B}+C) \cdot (A+\bar{B}+\bar{C})$

convert it into SSOP form.

sol^n/ $F(A,B,C) = (A+B+\bar{C}) \cdot (\bar{A}+\bar{B}+C) \cdot (A+\bar{B}+\bar{C})$

$= M_1 \cdot M_6 \cdot M_3$

$= \Pi M(1,3,6)$

$$\overline{F(A,B,C)} = \Pi M(0,2,4,5,7)$$

$$= M_0 \cdot M_2 \cdot M_4 \cdot M_5 \cdot M_7$$

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

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

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

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

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

OR/

$$= m_0 + m_2 + m_4 + m_5 + m_7$$

OR/

$$= \Sigma m(0,2,4,5,7)$$

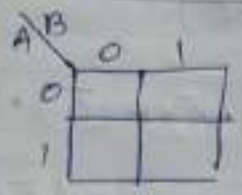
Karnaugh's Map / K-Map

- K-map is used to simplify Boolean function without using Boolean Algebra laws.
- This can be used to simplify Boolean expression up to six variables.
- In this method it is required to transfer Boolean expression into SSOP form or SPOS form.
- When Boolean expression in SSOP form 1 is placed in the square for minterm which are present in Boolean expression.
- When Boolean expression is in SPOS form 0 is placed in square for maxterm which are present in Boolean expression.

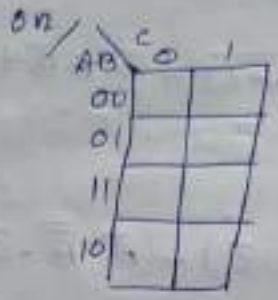
- No. of square (cell) required $= 2^n$

  where n = number of variables.

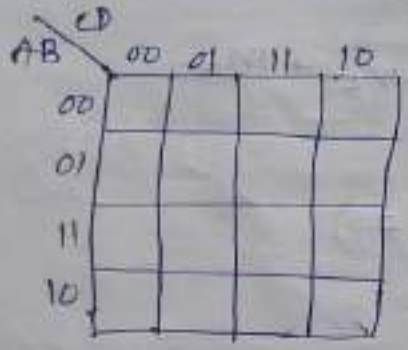
2-Variable K-map



3-Variable K-map



4-Variable K-map



Rules For K-map Simplification

1. Cells containing 1 must be grouped
2. We can group 1, 2, 4, 8, 16 cells
3. zero should not be included.
4. Each group should be as large as possible.
5. Groups may over lap.
6. Opposite grouping and corner groupings are allowed
7. Combination of variable of each group is called Prime implicante.
8. Write all the prime implicantes and add them.

Q) Draw a K-map for the following

1) $F(A,B) = 0$
2) $F(A,B) = 1$
3) $F(A,B) = A$
4) $F(A,B) = A + B$
5) $F(A,B) = (A+B)(A+\bar{B})$
6) $F = AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C}$
7) $F = \pi M(0,1,2,6)$

Soln/

1)

A \ B	0	1
0	0	0
1	0	0

2)

A \ B	0	1
0	1	1
1	1	1

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

A \ B	0	1
0	0	0
1	1	1

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

A \ B	0	1
0	0	1
1	1	1

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

A \ B	0	1
0	0	0
1	1	1

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

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

7) $F = \Pi M(0,1,2,6)$

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

8) Simplify using K-map

1) $F = \sum m(0,3,5,7,6)$

2) $F = A\bar{B} + \bar{A}\bar{B}C + AB\bar{C}D + \bar{A}\bar{B}\bar{C} + A\bar{C}\bar{D}$

Sol^n/

1) $F = \sum m(0,3,5,7,6)$

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

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

2) $F = A\bar{B} + \bar{A}\bar{B}C + AB\bar{C}D + \bar{A}\bar{B}\bar{C} + A\bar{C}\bar{D}$

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

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

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

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

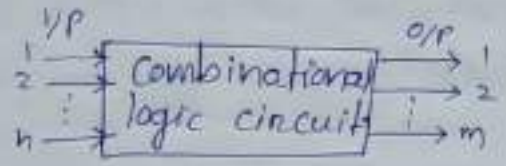
$= A\bar{B}CD + A\bar{B}C\bar{D} + A\bar{B}\bar{C}D + A\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}\bar{B}C\bar{D} + AB\bar{C}D + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + AB\bar{C}\bar{D} + A\bar{B}C\bar{D}$

AB \ CD	00	01	11	10
00	1	1	1	1
01				
11	1	1		
10	1	1	1	1

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

Combinational Logic Circuit

- Combinational logic circuit is a circuit consist of logic gates.
- It's output depend on combination of input variables
- It does not have memory.
- It can have n numbers of input & m no.s of output



Ex:- Adder, Subtractor, multiplexers, demultiplexer, encoder, Decoder, comparator.

Combinational circuit Designing

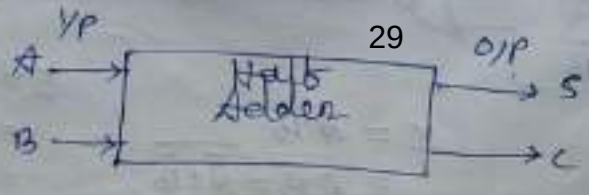
- To design a combinational circuit we have to follow certain steps.

steps

1. Understand the problem
2. Determine the no.s of inputs & no.s of outputs
3. Give name to inputs & outputs
4. Make truth table depending on relation between inputs and outputs.
5. Draw K-map for individual output from truth table.
6. Write simplified boolean expression of individual output using K-map
7. Draw logic diagram.

Designing of Half Adder

- Half Adder is a combinational logic circuit used to add two numbers each have single bit.
- It has two outputs i.e 'sum(s)' and 'carry(c)'



I/Ps		O/Ps	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

K-map for S

A \ B	0	1
0		1
1	1	

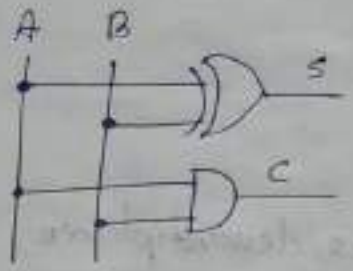
$S = A\bar{B} + \bar{A}B$

$= A \oplus B$

K-map for C

A \ B	0	1
0		
1		1

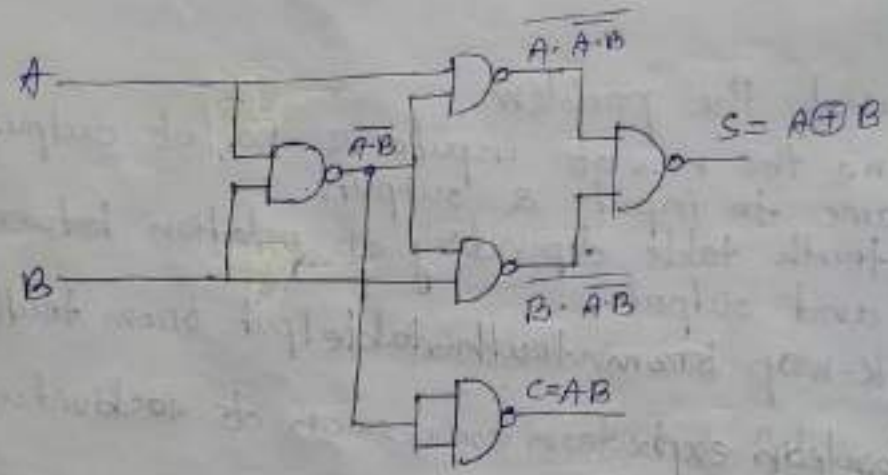
$C = AB$



Realization of Half Adder using NAND gates only

Sum, $S = A \oplus B$

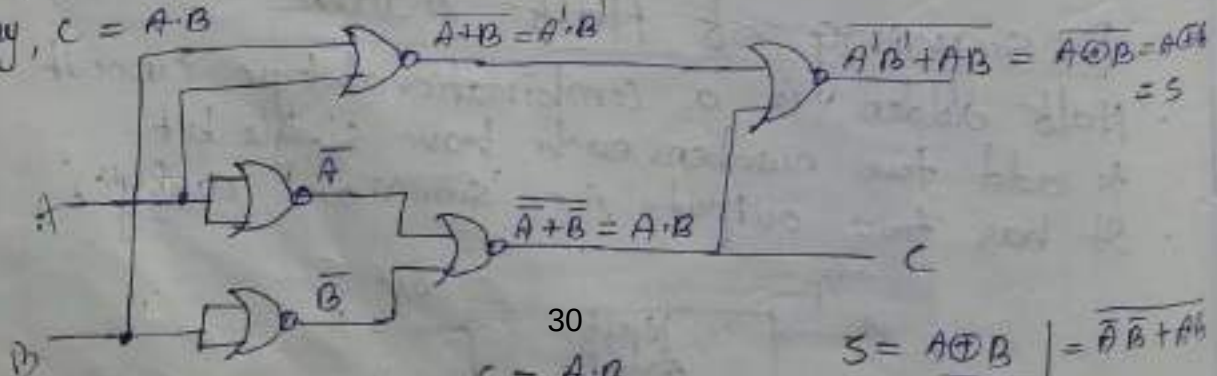
Carry, $C = A \cdot B$



Realization of Half Adder using NOR gates only

Sum, $S = A \oplus B$

Carry, $C = A \cdot B$



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

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

$S = A \oplus B$

$= \overline{A \odot B}$

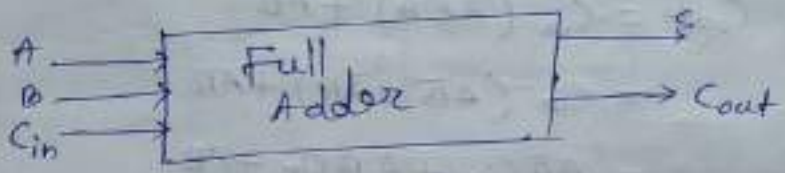
$= A \odot B$

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

Designing of Full Adder

- Full adder is a combinational logic circuit used to add two single bit numbers with carry.
- It has three inputs & two output "Sum (S)" and "output carry "Cout".

* The input carry is the carry from the previous addition in case of multiple bit



I/Ps O/Ps

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

K-map For S

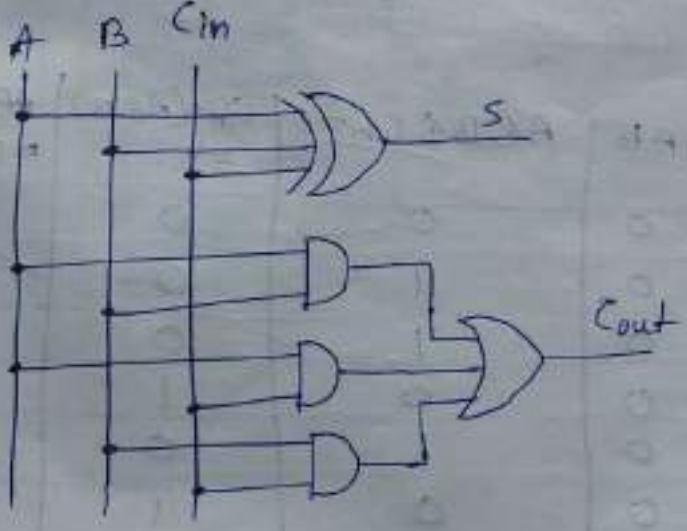
A \ BCin	00	01	11	10
0		1		1
1	1		1	

$S = A \oplus B \oplus C_{in}$

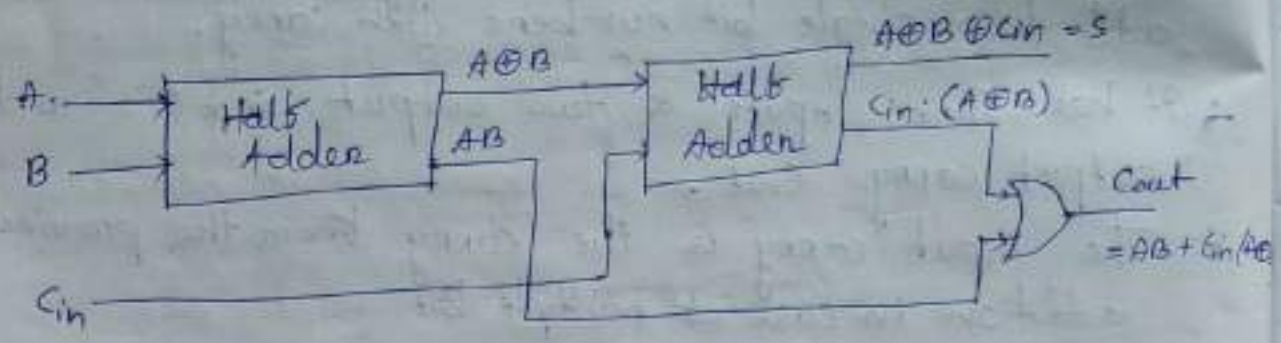
K-map for Cout

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

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



Full Adder using two Half Adders



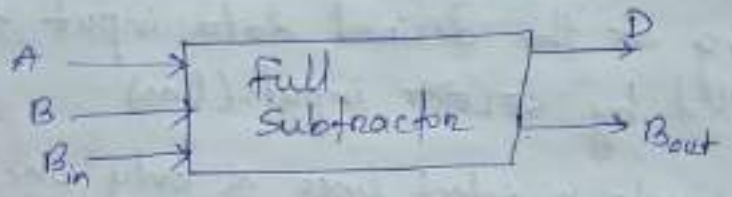
$$
\begin{aligned}
C_{out} &= C_{in}(A \oplus B) + AB \\
&= C_{in}(A\bar{B} + \bar{A}B) + AB \\
&= A\bar{B}C_{in} + \bar{A}BC_{in} + AB \\
&= A(\bar{B}C_{in} + B) + \bar{A}BC_{in} \\
&= A((B+\bar{B})\cdot(B+C_{in})) + \bar{A}BC_{in} \\
&= A(B + C_{in}) + \bar{A}BC_{in} \\
&= AB + AC_{in} + \bar{A}BC_{in} \\
&= AB + C_{in}(A + \bar{A}B) \\
&= AB + C_{in}((A+\bar{A})\cdot(A+B)) \\
&= AB + C_{in}(A+B) \\
&= AB + AC_{in} + BC_{in}
\end{aligned}
$$

Truth table

A	B	Cin	A⊕B	AB	A⊕B⊕Cin = S	Cin·(A⊕B) = y	AB + y = Cout
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	1	0	0	1	1
1	0	0	1	0	1	0	0
1	0	1	1	0	0	1	1
1	1	0	0	1	0	0	1
1	1	1	0	1	1	0	1

Full subtractor designing

- Full subtractor performs the subtraction involving three bits i.e minuend bit, subtrahend bit & the borrow to the previous state.
- The outputs are the difference D & the borrow signal.

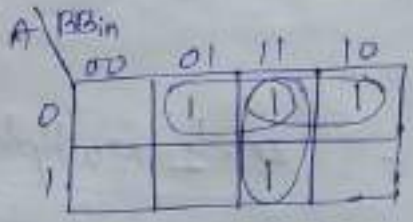


A	B	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

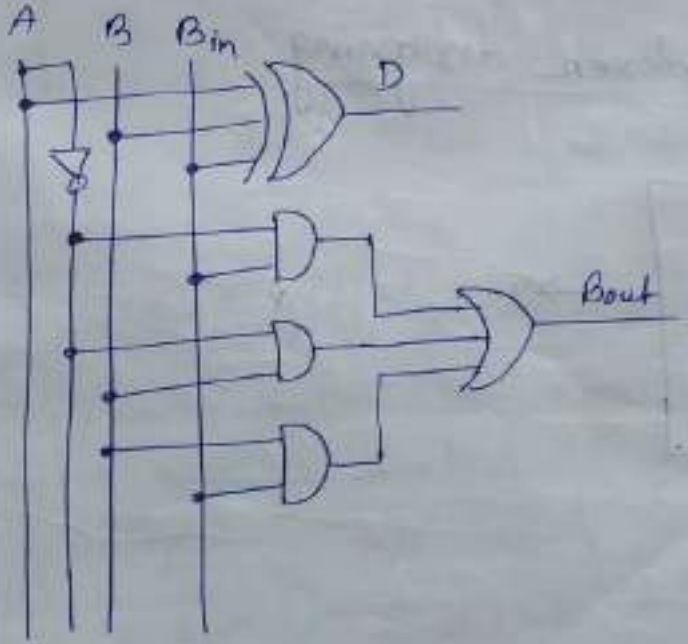
K-map for D

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

K-map for B_{out}

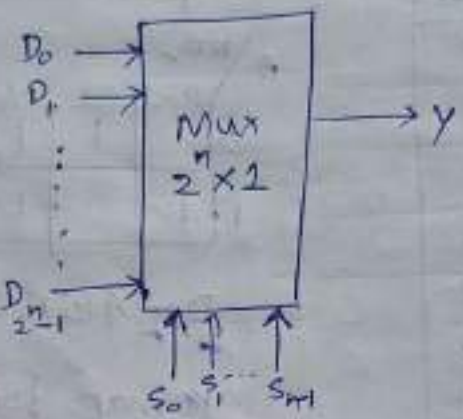


$B_{out} = \bar{A}B_{in} + \bar{A}B + BB_{in}$

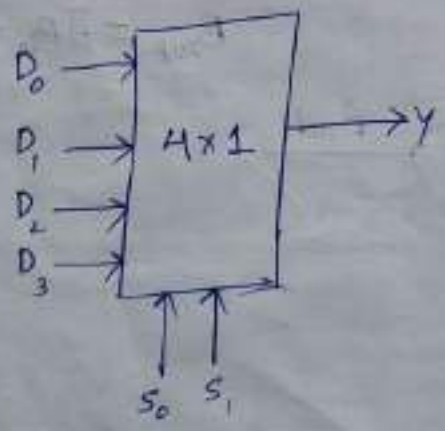


4×1 Multiplexer

- A multiplexer or data selector is a combinational logic circuit that accepts several data inputs & allows only one of them at a time to go through to the output.
- The routing of the desired data input to the output is controlled by SELECT inputs (lines)
- It has 2^n inputs, n select lines & only one outputs.



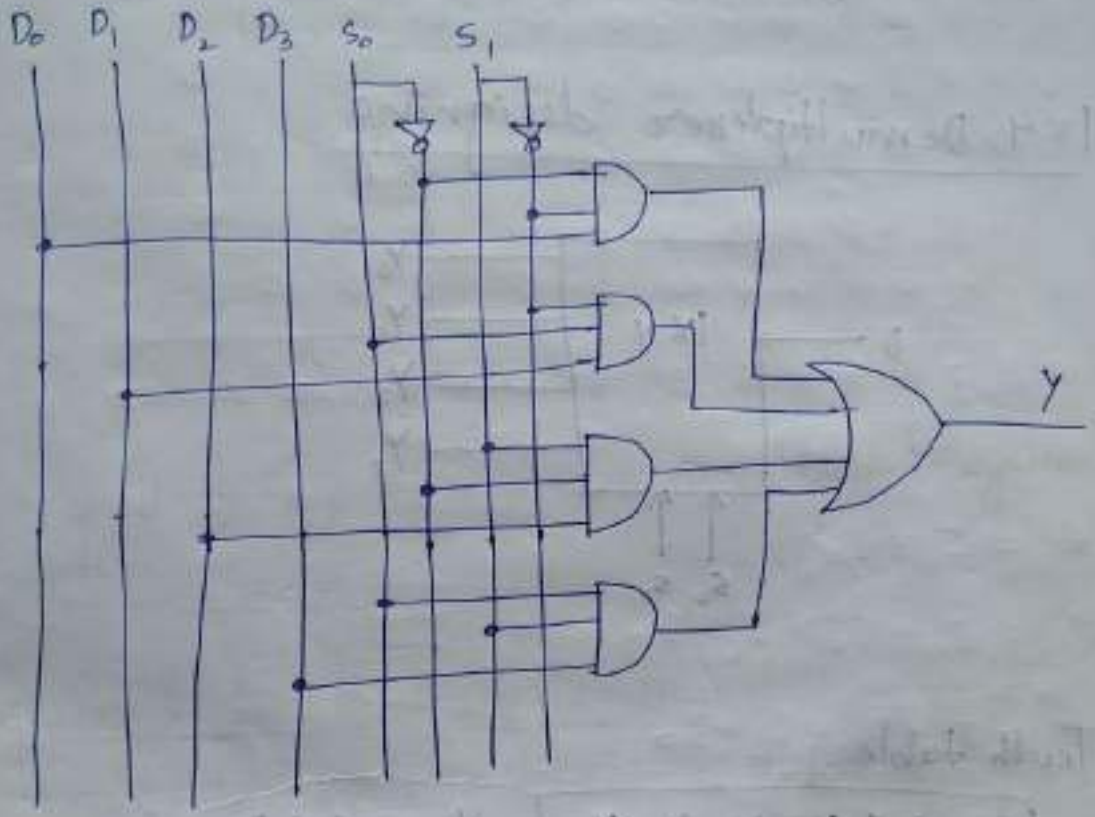
Ex:- 4×1 Multiplexer designing



Truth table

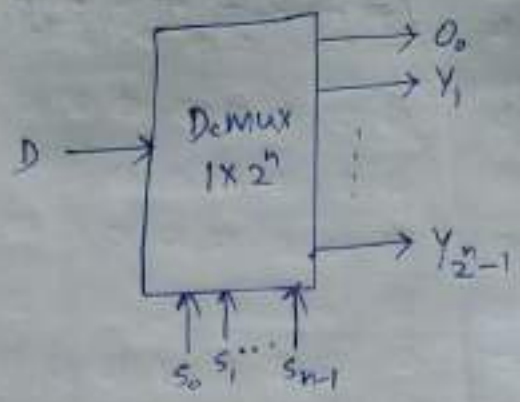
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

$$Y = \bar{S_1}\bar{S_0}D_0 + \bar{S_1}S_0D_1 + S_1\bar{S_0}D_2 + S_1S_0D_3$$

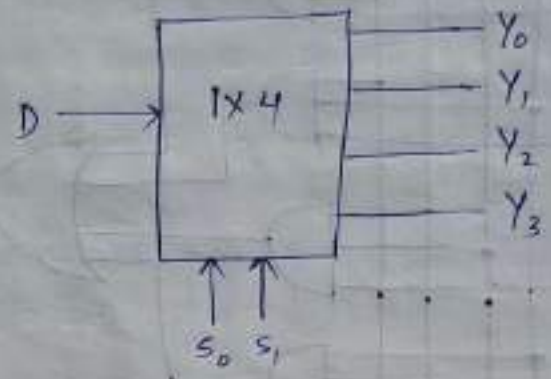


1 × 4 Demultiplexer

- Demultiplexers perform the reverse operation of multiplexer.
- It takes a single input & distributes it over several outputs. So demultiplexer is also called data distributor.
- It has one input, 2^n outputs & n select lines.
- The select line determines the output line to which the input data will be transmitted.



Ex:- 1×4 Demultiplexer designing



Truth table

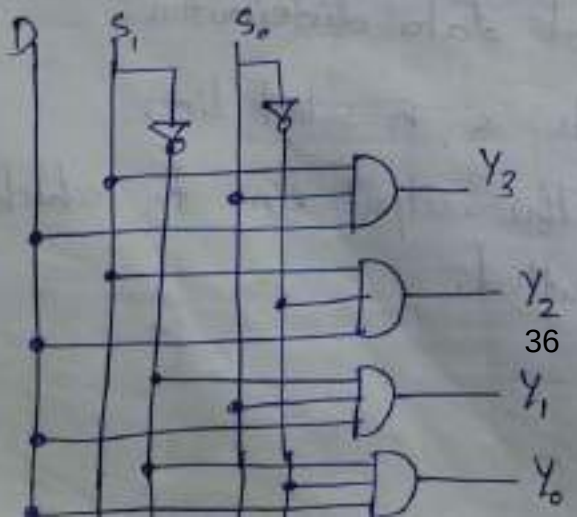
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

$Y_3 = S_1 S_0 D$

$Y_2 = S_1 \overline{S_0} D$

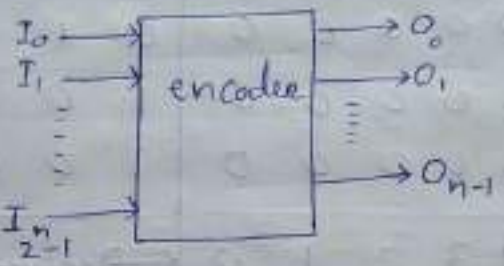
$Y_1 = \overline{S_1} S_0 D$

$Y_0 = \overline{S_1}\,\overline{S_0} D$



Encoders

- An encoder is a device whose inputs are decimal digits or alphabetic characters & whose outputs are the coded representation of those inputs.
- It converts familiar numbers or symbols into coded format.
- An encoder has 2^n input lines & n output lines.
- Among the inputs only one is active at a time.



- It is used in input circuit of any digital system. like computer.

Decimal to BCD Encoder

- This encoder has inputs '0' to '9' so it has 10 input lines & produce o/p of 4 bits for each input.

(b)

Truth table

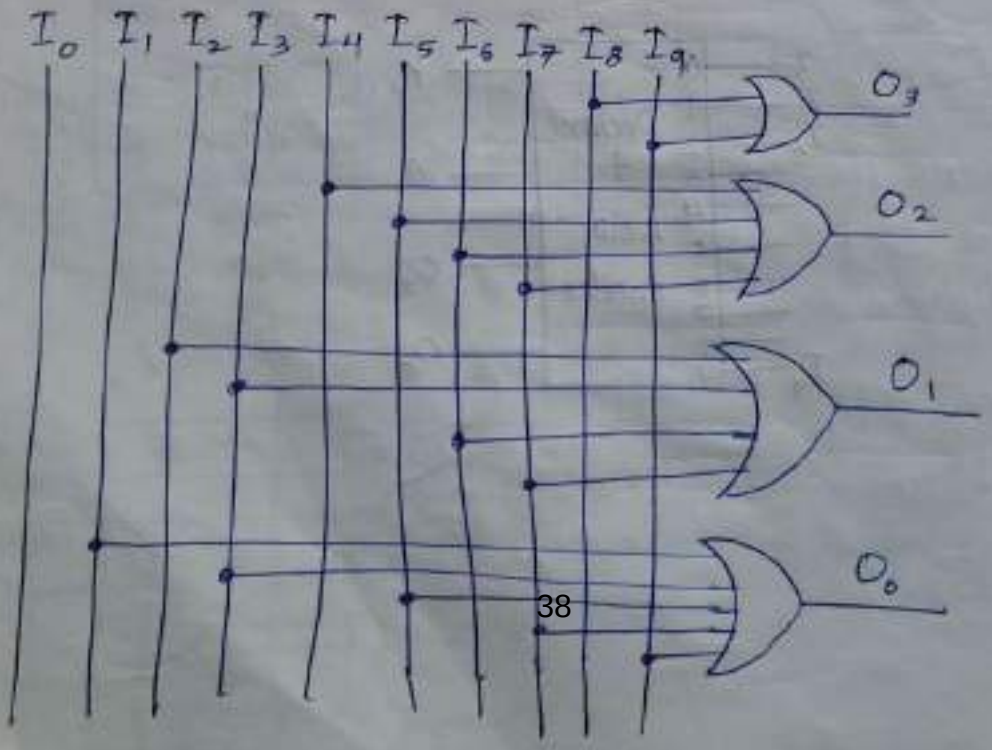
I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1
9	8	7	6	5	4	3	2	1	0	← Decimal number			

$$O_3 = I_8 + I_9$$

$$O_2 = I_4 + I_5 + I_6 + I_7$$

$$O_1 = I_2 + I_3 + I_6 + I_7$$

$$O_0 = I_1 + I_3 + I_5 + I_7 + I_9$$



Decoders

- Decoder do the reverse operation of encoder.
- This is a logic circuit that converts binary codes to familiar symbols or numbers.
- It has n input lines & 2^n output lines.
- For each combination of inputs, only one of the output will be active high.

- It is used in output circuit of any digital system like computer.

3 line to 8 line Decoder / Binary to octal

- In this decoder input lines are 3 nos & output lines are 8 nos.
- It convert binary input to octal output.

Truth table

I_2	I_1	I_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	10	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0
octal number →			7	6	5	4	3	2	1	0

$O_0 = \overline{I_0}\,\overline{I_1}\,\overline{I_2}$

$O_1 = I_0\,\overline{I_1}\,\overline{I_2}$

$O_2 = \overline{I_0}\,I_1\,\overline{I_2}$

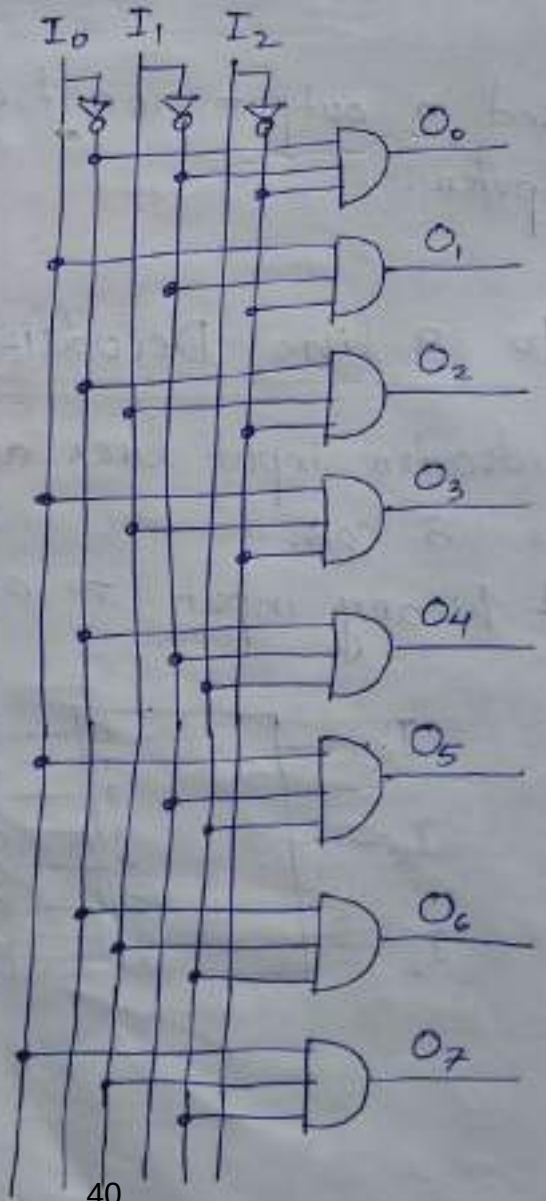
$O_3 = I_0\,I_1\,\overline{I_2}$

$O_4 = \overline{I_0}\,\overline{I_1}\,I_2$

$O_5 = I_0\,\overline{I_1}\,I_2$

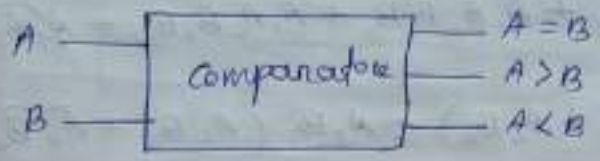
$O_6 = \overline{I_0}\,I_1\,I_2$

$O_7 = I_0\,I_1\,I_2$



2 bit Magnitude comparator

- A comparator is a combinational logic circuit used to compare the magnitude of two binary number.
- It's output become high if two numbers are equal or greater than or smaller than other number.
- 2 bit magnitude comparator compare two 2 bits numbers.



- Here A is a binary number having two bits $A_1 A_0$
  B is a binary number having two bits $B_1 B_0$

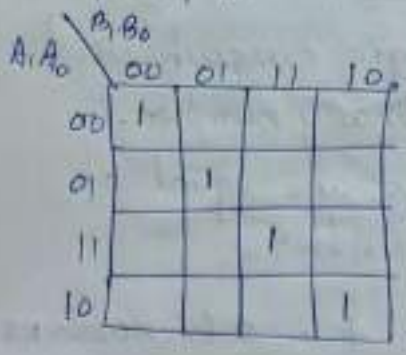
$A = A_1 A_0$

$B = B_1 B_0$

Truth table

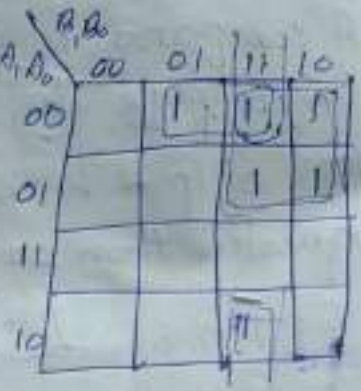
A_1	A_0	B_1	B_0	A = B	A > B	A < B
0	0	0	0	1	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	0	0	1
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	1	0
1	1	1	1	1	0	0

K map for $A=B$



K-map for $A>B$

K-map for $A<B$



$$(A=B) = \bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0 + A_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0$$

$$= \bar{A}_1\bar{B}_1(\bar{A}_0\bar{B}_0 + A_0B_0) + A_1B_1(A_0B_0 + \bar{A}_0\bar{B}_0)$$

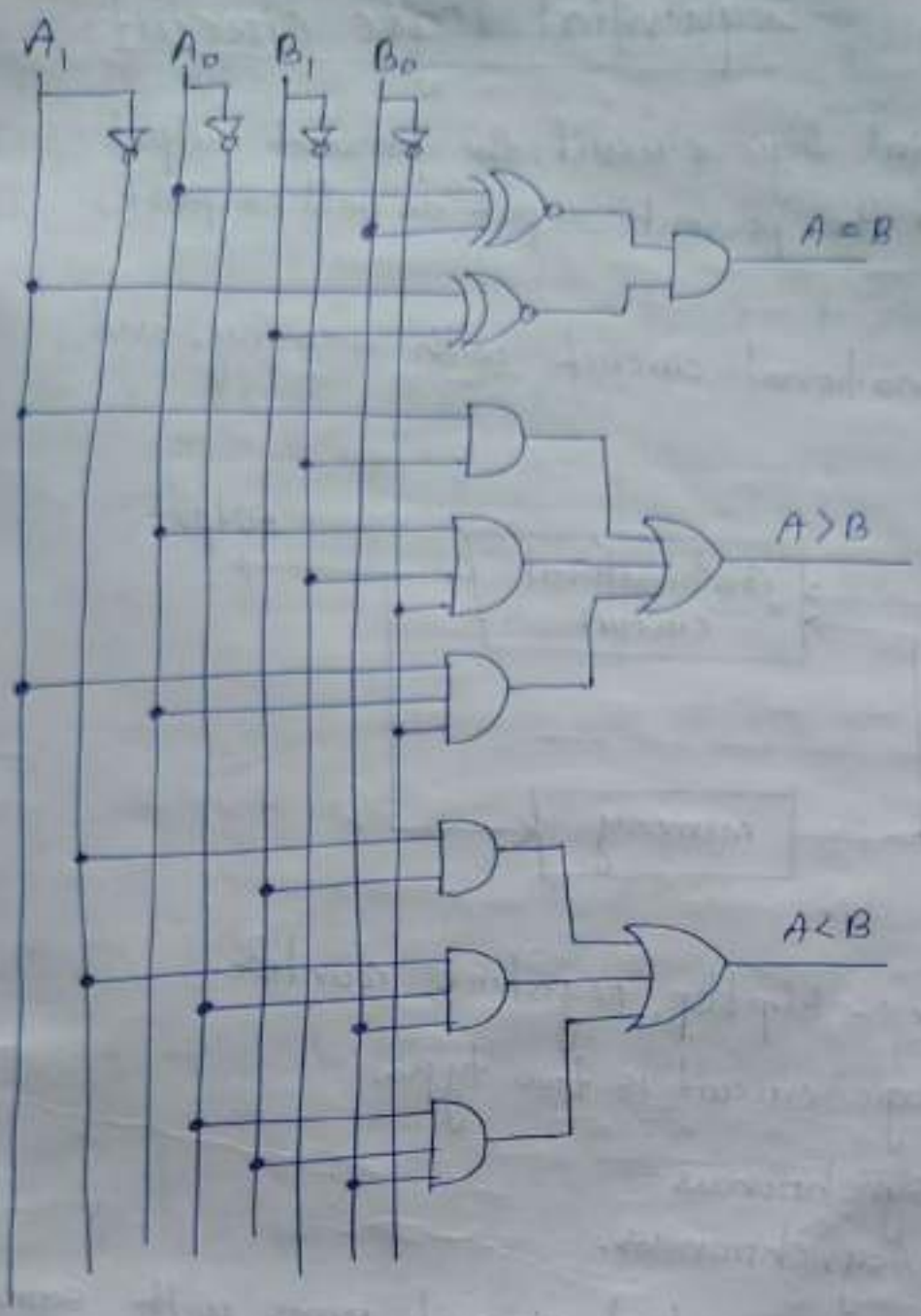
$$= \bar{A}_1\bar{B}_1(A_0 \odot B_0) + A_1B_1(A_0 \odot B_0)$$

$$= (A_0 \odot B_0)(\bar{A}_1\bar{B}_1 + A_1B_1)$$

$$= (A_0 \odot B_0)(A_1 \odot B_1)$$

$$A>B = A_1\bar{B}_1 + A_0\bar{B}_1\bar{B}_0 + A_1A_0\bar{B}_0$$

$$A<B = \bar{A}_1B_1 + \bar{A}_1\bar{A}_0B_0 + \bar{A}_0B_1B_0$$


A_1
A_0
B_1
B_0
A=B
A>B
A<B

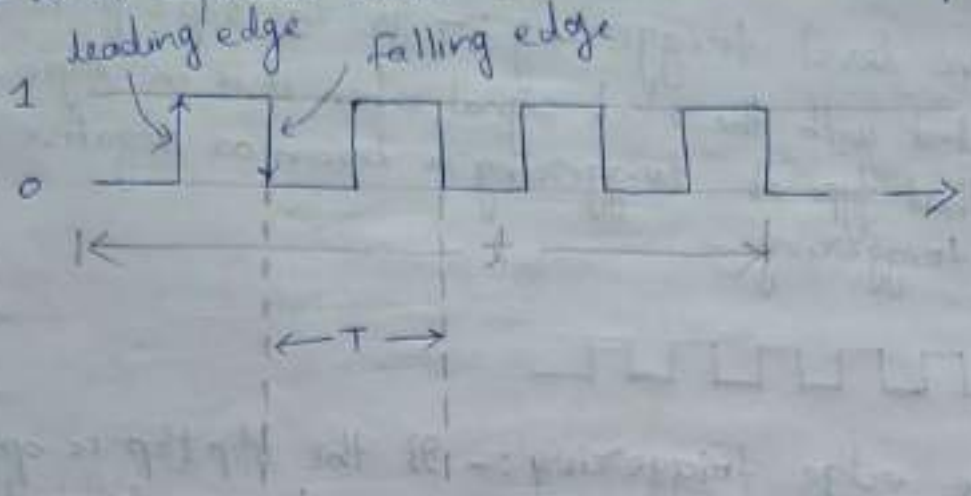
Sequential logic circuit

- In sequential logic circuit, the present output depends on the present input as well as past output.
- It is combinational circuit with memory, and feedback.

- Examples are :- flip flop, registers, counter.
- sequential logic circuit is two types.
  1. synchronous
  2. Asynchronous.
- synchronous sequential circuit runs with same clock pulse.
- Asynchronous sequential circuit runs with different clock pulses.

clock

- clock is the signal that control the outputs of the sequential circuit.
- The sequential circuit responds the inputs only when clock signal is present at the input.
- It is a square wave with constant frequency.

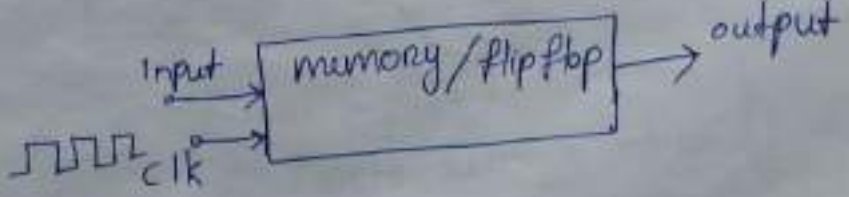


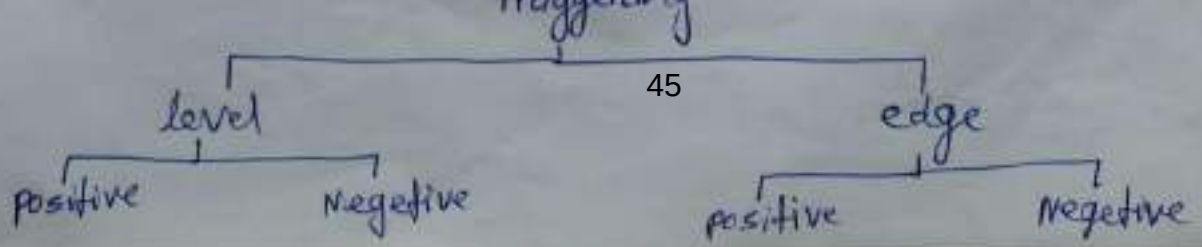
- T = time period, $f = \frac{1}{T}$
- clock signal has duty cycle to 50%.
- Duty cycle = $\frac{\text{Time for which signal is high}}{\text{Total time}}$

$= \frac{t/2}{t} = \frac{1}{2} = 50\%$

- Time duration of high level = Time duration of low level.

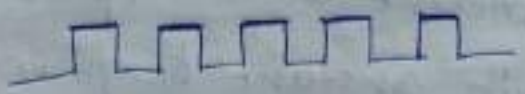
Triggering Methods



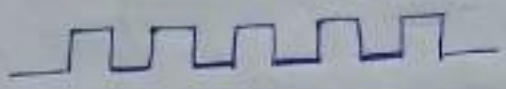


(3)

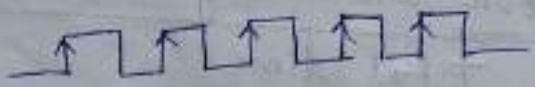
1) Positive level triggering :- If the flipflop is operated with the clock signal when it is in logic high, then that type of triggering is known as positive level triggering



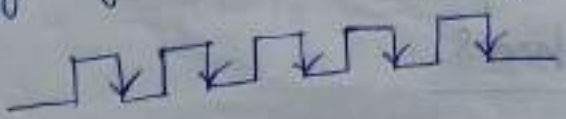
2) Negative level triggering :- If the flip flop is operated with the clock signal when it is in logic low then that type of triggering is known as Negative level triggering.



3) Positive edge triggering :- If the flip flop is operated with the clock signal that is transitioning from logic low to logic high, then that type of triggering is known as positive edge triggering.

4) 

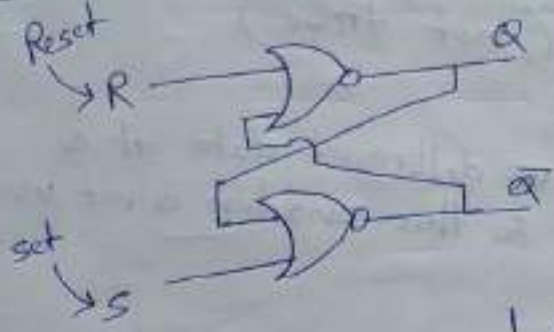
4) Negative edge triggering :- If the flip flop is operated with the clock signal that is transitioning from logic high to logic low, then that type of triggering is known as negative edge triggering.



SR latch

- The basic storage element is called latch.
- It stores single bit number i.e '1' or '0'. until other I/P are
- It is of two types.
  1. NOR SR latch
  2. NAND SR latch.

NOR SR latch



- 'R' represents the word Reset.
- 'S' represents the word set.

when $R = 1$, $Q = 0$
when $S = 1$, $Q = 1$

truth table of NOR gate

A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Case =1
when $S = 1$, $R = 0$
then $Q = 1$ $\bar{Q} = 0$

when $S = 0$, $R = 0$ (no inputs)
then $Q = 1$ $\bar{Q} = 0$ (data remain stored)

Case - II

when S=0, R=1

then Q = 0 $\bar{Q}$ = 1

when S=0, R=0 (no inputs)

then Q = 0 $\bar{Q}$ = 1 (data remain stored)

Case - III

when S = 1, R = 1

then Q = 0, $\bar{Q}$ = 0 (not true)

If S=0, R=0

then if Q=0, $\bar{Q}$=1

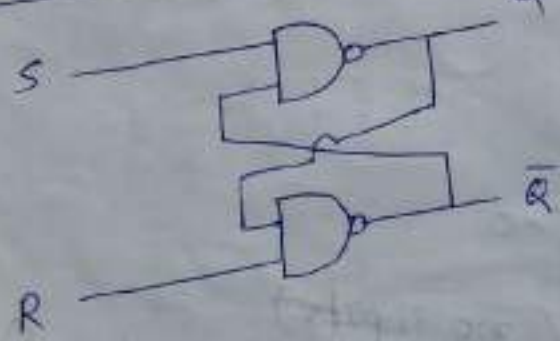
if $\bar{Q}$=0, Q=1

[Give different data at Q so this condition is not used]

Truth table for SR latch with NOR gates

S	R	Q $\bar{Q}$
0	0	Same as before
0	1	0 1
1	0	1 0
1	1	Not used

NAND SR latch

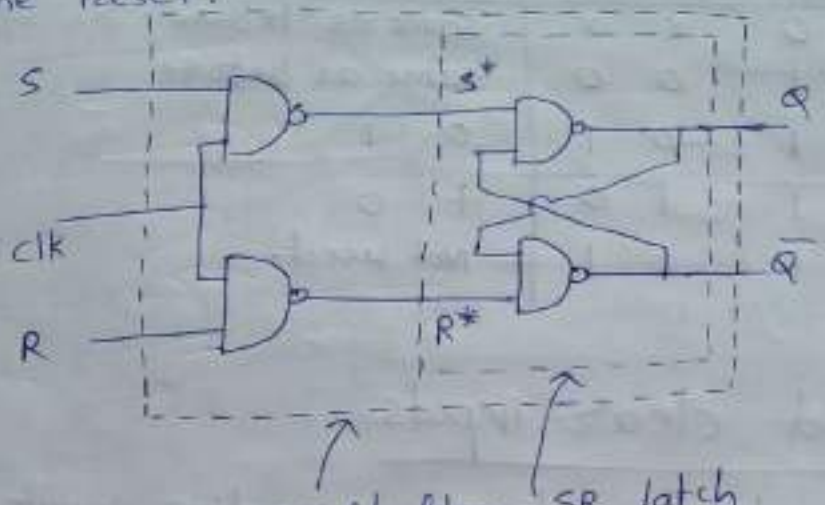


truth table

S	R	Q $\bar{Q}$
0	0	Not used
0	1	1 0
1	0	0 1
1	1	same as before

S R flip flop

- This is a one bit memory device.
- It's output is controlled by clock signal.
- 'S' stands for 'set' and 'R' stands for 'reset'
- 'set' means output = 1 ; 'Reset' means output = 0
- when S=1, device become set; when R=1, device become reset.



Truth table of SR latch

S^*	R^*	Q	$\bar{Q}$
0	0	Not used	
0	1	1	0
1	0	0	1
1	1	same as before	

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

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

- It has three inputs S, R & clk. It has two outputs Q & $\bar{Q}$.
- when clk = 0, then $S^* = 1$ & $R^* = 1$, no matter what is the value of S & R, Q & $\bar{Q}$ will be "same as before."
- when clk = 1, then $S^* = \bar{S}$ & $R^* = \bar{R}$
- when clk = 1, S=0, R=0, then $S^* = 1$, $R^* = 1$, Q & $\bar{Q}$ will be "same as before".

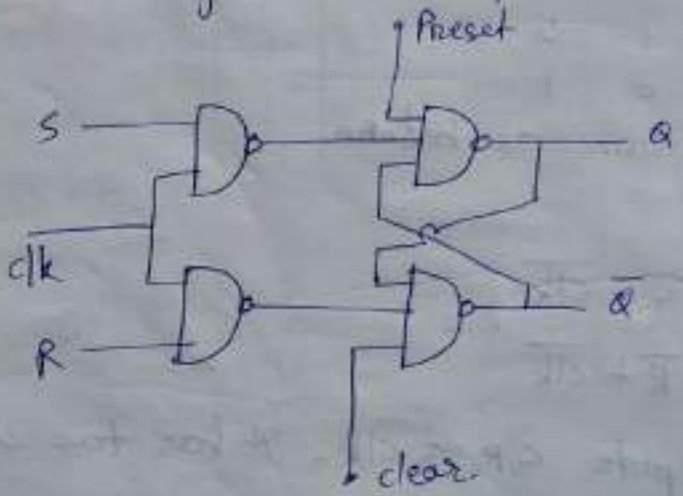
- when clk=1, S=0, R=1, then $S^*=1$, $R^*=0$, $Q=0$, $\bar{Q}=1$
- when clk=1, S=1, R=0, then $S^*=0$, $R^*=1$, $Q=1$, $\bar{Q}=0$
- when clk=1, S=1, R=1, then $S^*=0$, $R^*=0$, Q & $\bar{Q}$ become "not used."

Truth table of SR flip flop

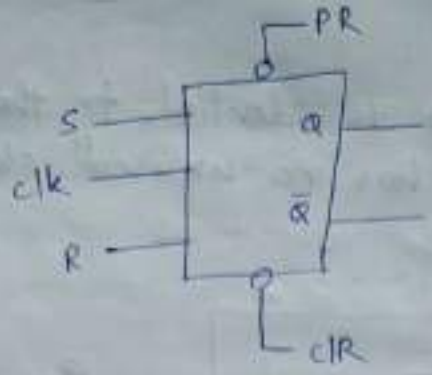
clk	S	R	Q	$\bar{Q}$
0	X	X	same as before	
1	0	0	same as before	
1	0	1	0	1
1	1	0	1	0
1	1	1	Not used.	

Preset and clear inputs

- Preset and clear inputs are overriding inputs.
- These inputs does not work with clock signal so these are also called asynchronous input.



- when Present & clear inputs are active then the inputs & clock signal has no control over the output of flip flop so these inputs are called overriding inputs.
- when Preset = 0, then Q = 1, $\bar{Q}=0$
- when clear = 0, then Q = 0, $\bar{Q}=1$



- when PR=0 & clR=0 then $Q = \bar{R}$ So this condition is not used.
- when PR=0, clR=1 then Q=1
- when PR=1, clR=0, then Q=0
- when PR=1, clR=1, then no effect of PR & clR inputs on flipflop & the flip flop functions normally depending of S, R & clk inputs.

Truth table

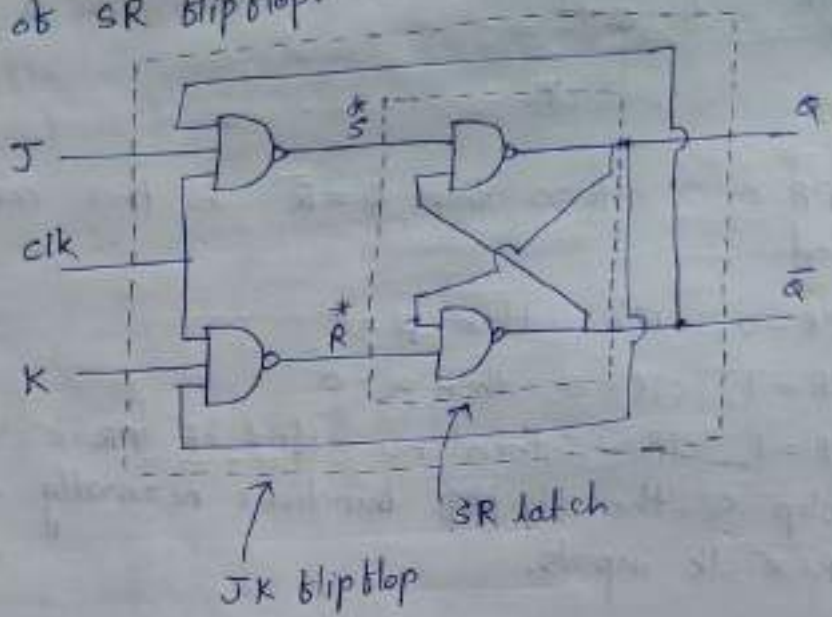
PR	clR	Q
0	0	not used
0	1	1
1	0	0
1	1	ff will function normally.

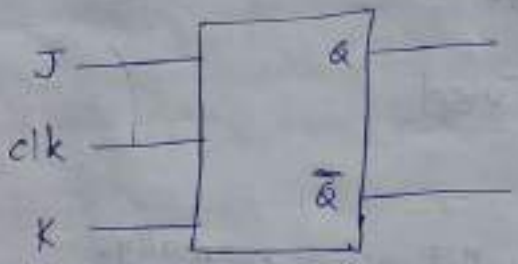
- Bobble at the inputs of Preset & clear indicate that these are active low signals.

9

J-K flip flop

- The function of JK flip flop is identical to that of SR flip flop except that it has no 'unused' state. like that of SR flip flop.





- It has three inputs J, K, clk & two outputs Q & $\bar{Q}$.

- Truth table of SR latch

S^*	R^*	Q	$\bar{Q}$
0	0	Not used	
0	1	1	0
1	0	0	1
1	1	same as before	

- when clk=0 then $S^*=1$, $R^*=1$ so no change in the outputs. whatever may be the value of J & K.

- when clk=1, J=0, K=0, then $S^*=1$, $R^*=1$ so no change in the output.

- when clk=1, J=0, K=1, let $Q_n=0$ & $\overline{Q_n}=1$ then

$S^*=1$, $R^*=1$, Q_{n+1} = no change = 0

$\overline{Q_{n+1}}=1$

let $Q_n=1$, & $\overline{Q_n}=0$ then

$S^*=1$, $R^*=0$, $Q_{n+1}=0$

$\overline{Q_{n+1}}=1$

- when clk=1, J=1, K=0, let $Q_n=0$ & $\overline{Q_n}=1$ then

$S^*=0$, $R^*=1$, $Q_{n+1}=1$, $\overline{Q_{n+1}}=0$

let $Q_n=1$, $\overline{Q_n}=0$ then

$S^*=0$, $R^*=1$, $Q_{n+1}=1$, $\overline{Q_{n+1}}=0$

- when clk=1, J=1, K=1, let $Q_n=0$, $\overline{Q_n}=1$ then

$S^*=0$, $R^*=1$, $Q_{n+1}=1$, $\overline{Q_{n+1}}=0$

let $Q_n=1$, $\overline{Q_n}=0$ then

$S^*=1$, $R^*=0$, $Q_{n+1}=0$, $\overline{Q_{n+1}}=1$

Here pattern of Q_{n+1} & $\overline{Q_{n+1}}$ will be as bellow,

Q_{n+1} = 0, 1, 0, 1, 0 1

$\overline{Q_{n+1}}$ = 1, 0, 1, 0, 1, 0

means output become tagging. And this condition is called race around condition.

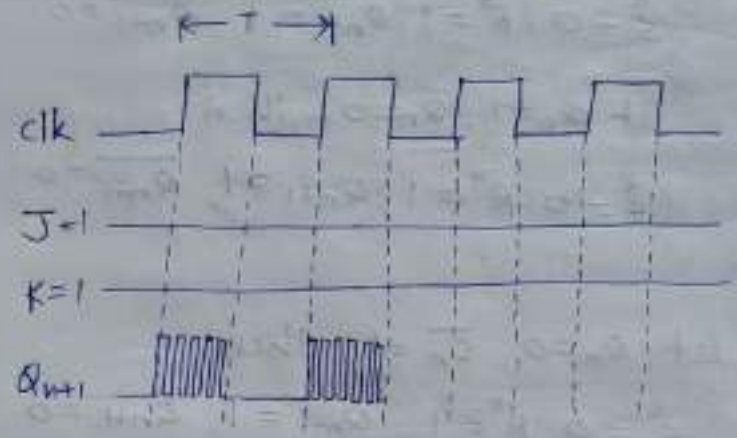
Truth table of JK flip flop

clk	J	K	Q_n	Q_{n+1}	state
0	x	x	x	No change (Q_n)	
1	0	0	x	No change (Q_n)	
1	0	1	x	0	Reset
1	1	0	x	1	set
1	1	1	x	$\overline{Q_n}$	toggle

Race Around condition

- For J-K flip flop, if J=K=1 and if clock=1 for a long period of time, then Q output will toggle as long as clock is high, which makes the output of the flip flop unstable or uncertain. This problem is called race around condition.

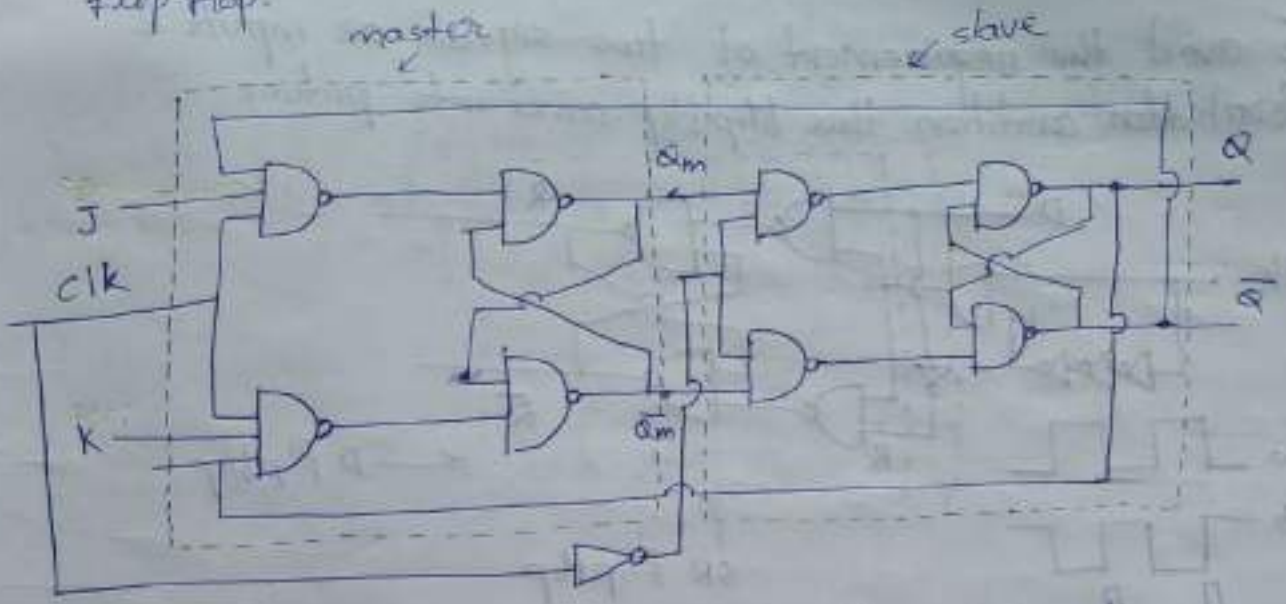
- Below is the wave diagram of Race around condition



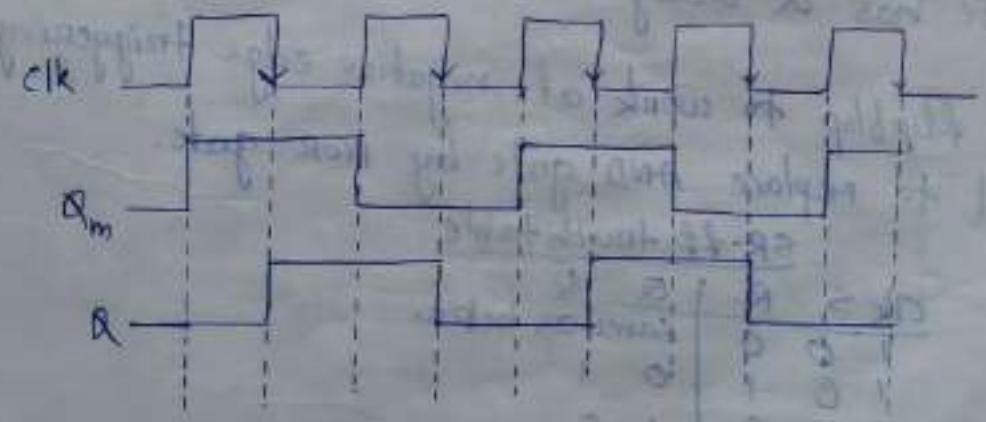
- If delay of FF < T/2 time then we get Q_{n+1}
- If delay of FF > T/2 time then $Q_{n+1}=0$
- Race around condition is uncontroled toggling.
- If toggling can be controled then this can be used in sequential circuit called counter.
- Conditions to overcome Race Around condition
  1. $\frac{T}{2}$ < propogation delay of FF
  2. Using edge trigging
  3. Master-slave

Master slave JK flip flop

- It is used to avoid race around condition occurs in J-K flip flop when J=K=1 with clock presence.
- It's operation is same as the negative edged triggering flip flop.

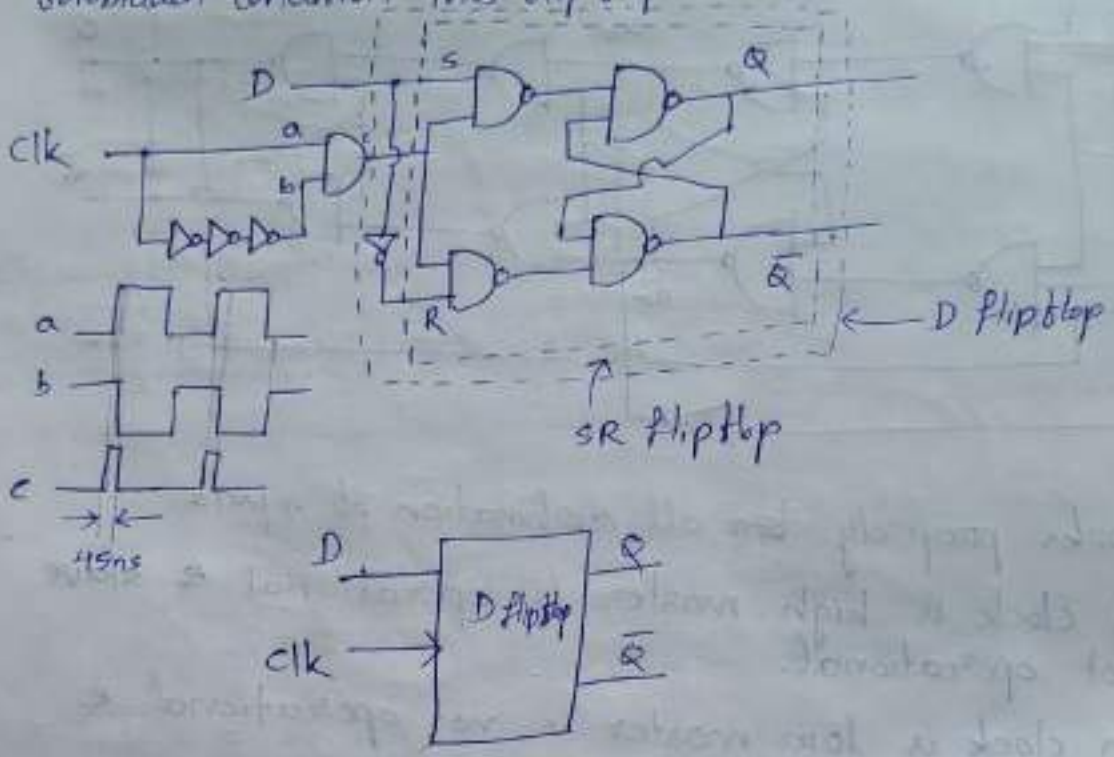


- It works properly for all combination of inputs.
- When clock is high master is operational & slave is not operational.
- when clock is low master is not operational & slave is operational.
- when K=J=1 for high clock signal instead of toggling continuously it will toggle once for a single clock cycle.
- When J=K=1 the waveform of Qm & Q are shown below.



Edge triggered D flip flop

- D flipflop is the modified form of SR flipflop.
- D stands for data.
- To avoid the requirement of two signals as inputs & forbidden condition this flipflop comes into picture.



- Above figure is the positive edge triggered D flip flop.
- The clock signal become clock pulse for a very small time i.e 45 ns using ~~three~~ Not gates & a AND gate.
- Each NOT gate has a delay of 15 ns.
- To make the flipflop to work at negative edge triggering it is required to replace AND gate by NOR gate.

SR-ff truth table

Clk	S	R	Q $\bar{Q}$
1	0	0	same as before
1	0	1	0 1
1	1	0	1 0
1	1	1	Not used
0	x	x	same as before

Case-1
when D=0 then S=0, R=1 So Q=0, $\bar{Q}$=1

Case-2
when D=1 then S=1, R=0 So Q=1, $\bar{Q}$=0

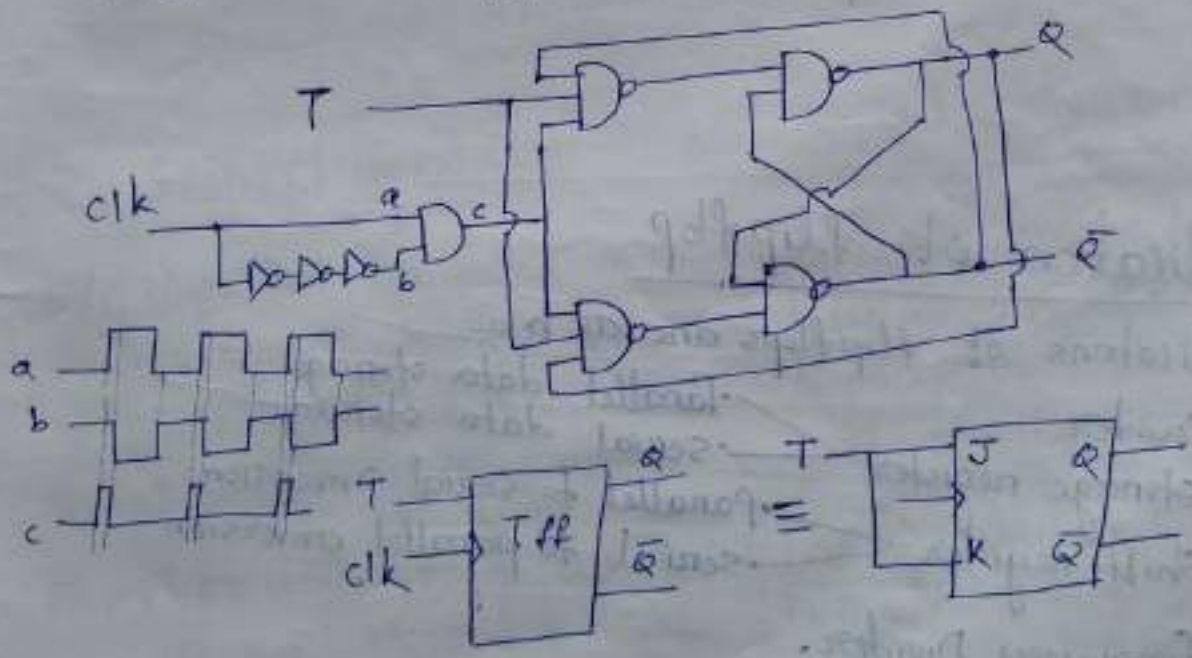
Case-3
when clk=0, ff is not functioning. what ever may be the value at D; output does not change.

Truth table of D ff

Clk	D	Qn+1
0	x	Qn
1	0	0
1	1	1

Edge triggered T flip flop

- When the two inputs of clocked JK flipflop are shorted to make one input then the flip flop is called T-flipflop.
- T stands for Toggle



- Above figure is the positive edge triggered T flip flop.
- The duration of clock pulse is 45 ns using three NOT gates & a AND gate.

- Each NOT gate has a delay of 15 ns.
- To make the flipflop to work in negetive edge triggering it is required to replace AND gat with NOR gate.

Case-1
when clk = 0, then flip flop does not respond T. Q & $\bar{Q}$ remain same as before.

Case-2
when clk = 1, T = 0 then J = 0, k = 0, Q & $\bar{Q}$ remain same as before.

Case-3
when clk = 1, T = 1 then J = K = 1, Q become in toggle mode.

clk	T	Q_n	Q_{n+1}	state
0	x	x	Q_n	No change
1	0	x	Q_n	No change
1	1	x	$\bar{Q}_n$	Toggle

Application of flip flop

Applications of flip flops are as below

- Counter
- storage register
  - Parallel data storage
  - Serial data storage
- Shift register
  - Parallel to Serial Conversion
  - serial to parallel conversion
- Freqwency Divider.

COUNTER

- Counter is a sequential circuit used to count number of pulses.
- It is also used as frequency divider.
- Counter contains set of flip flop.
- A n-bit counter requires n number of flipflop. and can count 2^n states.
- Each state frequency = $\frac{\text{clock frequency}}{2^n}$ or $T_s = 2^n T_c$
- Depending on the application of clock pulse counter is of two types
  - Asynchronous / Ripple counter
  - synchronous counter.
- Depending on progression of counting the counter is three types
  - up counter
  - Down counter
  - UP/Down counter.

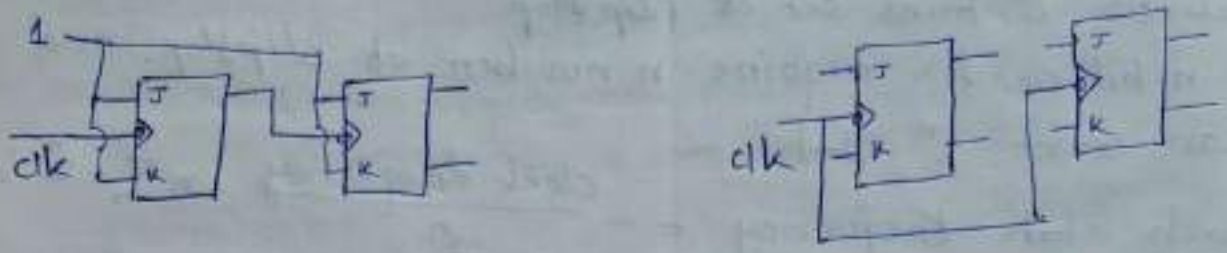
Asynchronous

- Flipflops are connected in such a way that the o/p of first flipflop drives the clock of next flip flop.
- Flip Flops are not clocked simultaneously
- Circuit is simple for more number of states.

synchronous

1. There is no connection between o/p of first flipflop and clock of next flipflop
2. Flip Flops are clocked simultaneously
3. Circuit becomes complicated as number of states increases.

- Speed is slow as clock is propagated through number of stages.
- speed is high as clock is given at a same time.



Modulus of a counter

- The number of states passed by the counter before going to its initial state is called modulus of counter.

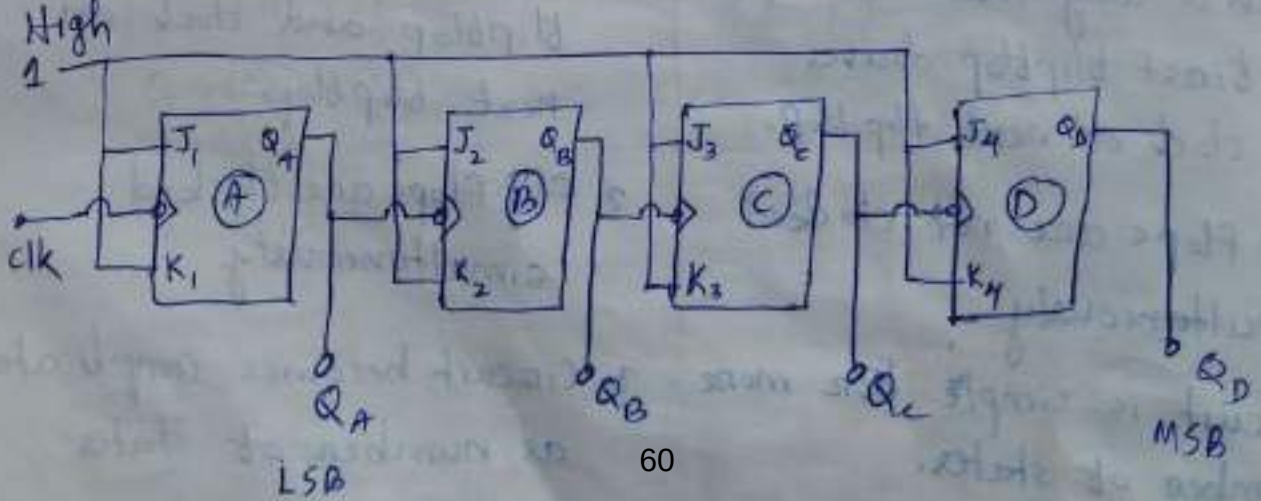
$$\text{modulus} = 2^n$$

where n = no. of bits a counter has

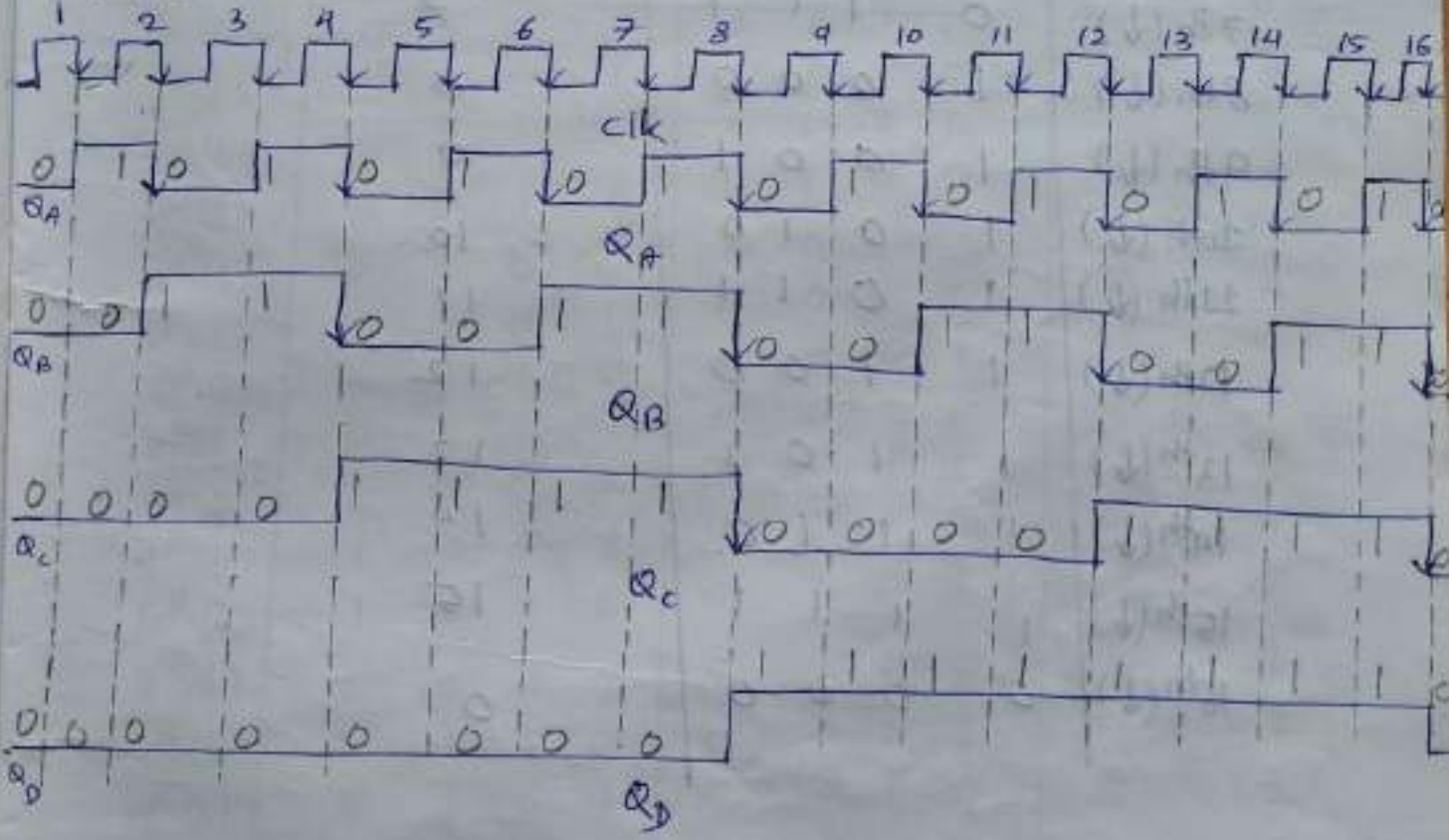
Ex:- 2 bit counter has modulus $= 2^2 = 4$

4-bit asynchronous up counter

- 4 bit asynchronous up counter consist of 4 ff which count from 0000 to 1111.



- All the ffs are negative edge teigged & connected in toggle mode.
- Output Q_A of (A) ff is given as clock pulse to the next (B) Ff. (B) ff o/p Q_B is applied as clock pulse to the (C) ff & the o/p Q_C is applied as clock pulse to the (D) ff.
- outputs of counter are $Q_D\ Q_C\ Q_B\ Q_A$.
- Last ff output i.e Q_D is the MSB & 1st FF output i.e Q_A is the LSB.
- Each ff toggle when negative edge of clock pulse occure.
- modulus of this counter $= 2^4 = 16$. means it has 16 nos of states i.e 0000 to 1111
- maximum count $= 2^n - 1 = 2^4 - 1 = 15$



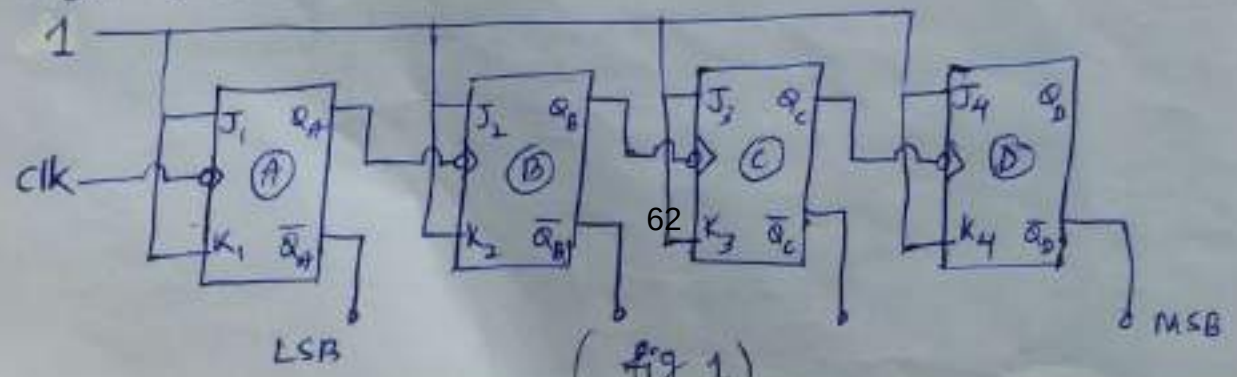
(timing diagram)

- By looking the timing diagram waveform we can make a table which shows how the counter counts.

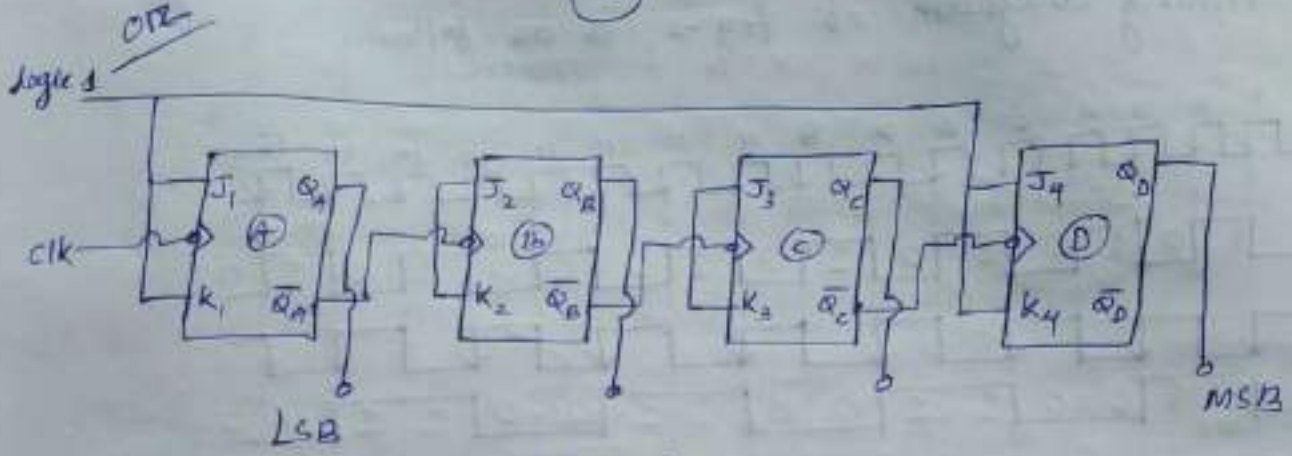
Clk	Q_D	Q_C	Q_B	Q_A	Decimal equivalent
Initially	0	0	0	0	0
1st (↓)	0	0	0	1	1
2nd (↓)	0	0	1	0	2
3rd (↓)	0	0	1	1	3
4th (↓)	0	1	0	0	4
5th (↓)	0	1	0	1	5
6th (↓)	0	1	1	0	6
7th (↓)	0	1	1	1	7
8th (↓)	1	0	0	0	8
9th (↓)	1	0	0	1	9
10th (↓)	1	0	1	0	10
11th (↓)	1	0	1	1	11
12th (↓)	1	1	0	0	12
13th (↓)	1	1	0	1	13
14th (↓)	1	1	1	0	14
15th (↓)	1	1	1	1	15
16th (↓)	0	0	0	0	0

4-bit asynchronous decon counter

- It count from 1111 to 0000.



(fig 1)

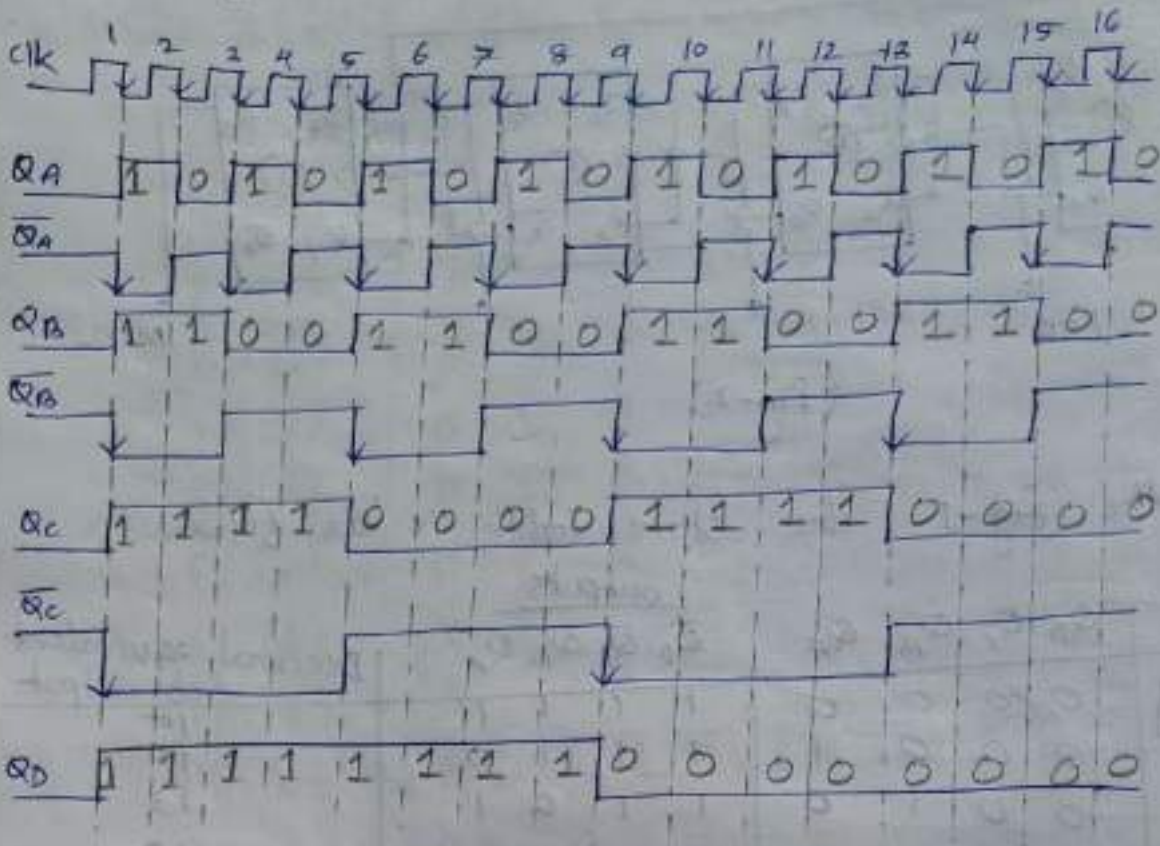


(fig-2)

- From the (fig-1) the state table is as follow

clk	outputs Q_D	Q_C	Q_B	Q_A	$\overline{Q_D}$	$\overline{Q_C}$	$\overline{Q_B}$	$\overline{Q_A}$	Decimal equivalent of output
Initially	0	0	0	0	1	1	1	1	15
1st (↓)	0	0	0	1	1	1	1	0	14
2nd (↓)	0	0	1	0	1	1	0	1	13
3rd (↓)	0	0	1	1	1	1	0	0	12
4th (↓)	0	1	0	0	1	0	1	1	11
5th (↓)	0	1	0	1	1	0	1	0	10
6th (↓)	0	1	1	0	1	0	0	1	9
7th (↓)	0	1	1	1	1	0	0	0	8
8th (↓)	1	0	0	0	0	1	1	1	7
9th (↓)	1	0	0	1	0	1	1	0	6
10th (↓)	1	0	1	0	0	1	0	1	5
11th (↓)	1	0	1	1	0	1	0	0	4
12th (↓)	1	1	0	0	0	0	1	1	3
13th (↓)	1	1	0	1	0	0	1	0	2
14th (↓)	1	1	1	0	0	0	0	1	1
15th (↓)	1	1	1	1	0	0	0	0	0
16th (↓)	0	0	0	0	1	1	1	1	15

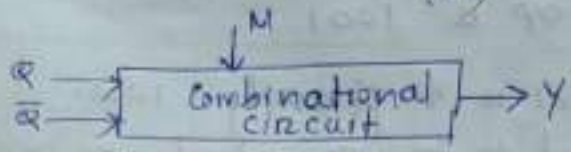
- Timing diagram of fig-2 is as follow.



- outputs of fig-2 are $Q_D Q_C Q_B Q_A$. Q_D is the MSB & Q_A is the LSB.
- If we see the output states of fig-2 from timing diagram & compare the output states of fig-1 from table then we can see the o/p of fig-2 is same as o/p of fig-1.
- Hence the function of fig-1 & fig-2 circuit is same i.e down counting from 1111 to 0000.

4-bit asynchronous up/down counter

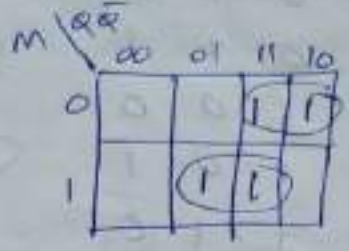
- This counter can work as up counter or can be used as down counter
- A mode control input (M) is used to select either up or down mode.
- A combinational circuit is required between each pair of flip flop.



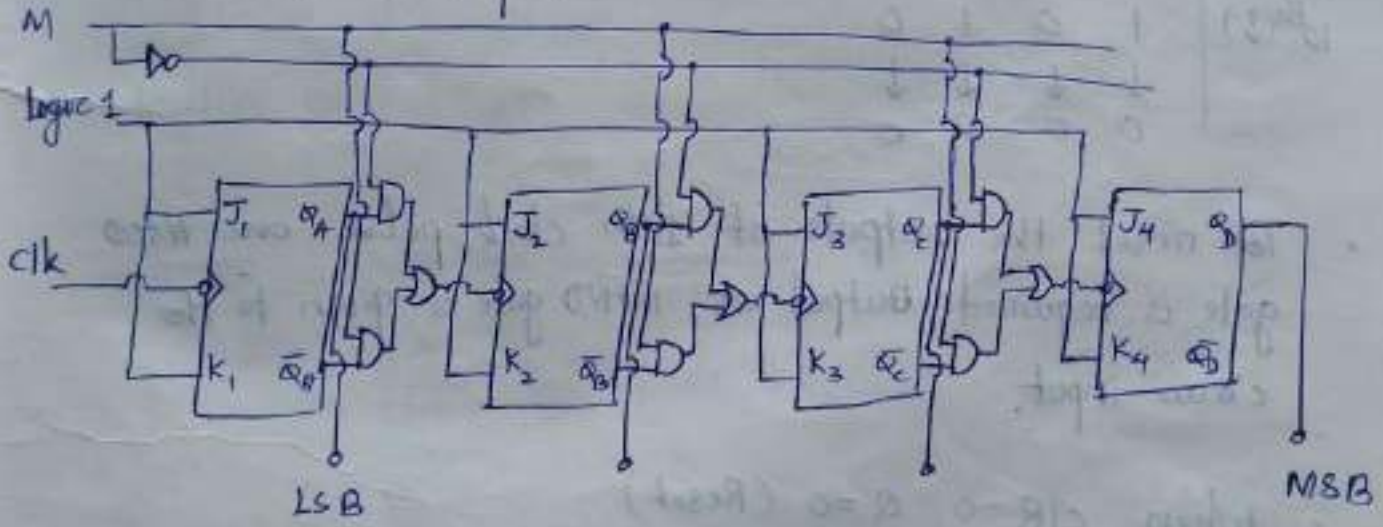
Y is the clock pulse to the next flipflop.
Q & $\bar{Q}$ are the outputs of previous flipflop.

- Let M=0 then up counter, $Y = Q$
  M=1 then down counter, $Y = \bar{Q}$

M	Q	$\bar{Q}$	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



$Y = \bar{M}Q + M\bar{Q}$



Asynchronous decade (BCD) counter / Mod-10 counter

- This counter count from 0000 to 1001.
- As o/p is in 4 bit it requires 4 flip flops to construct.
- Normal operation takes place up to 9th clock.
  means for 1st clock pulse o/p is 0001
  for 2nd clock pulse o/p is 0010

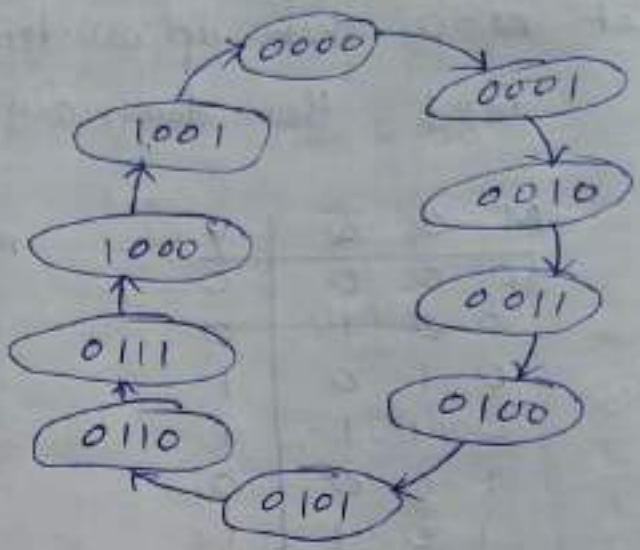
for 9th clock pulse o/p is 1001

- For 10th clock pulse o/p is temporarily 1010.
  Immidiately it is forced to become reset to 0000.

state table

clk	Q_D	Q_C	Q_B	Q_A
Initially	0	0	0	0
1st(↓)	0	0	0	1
2nd(↓)	0	0	1	0
3rd(↓)	0	0	1	1
4th(↓)	0	1	0	0
5th(↓)	0	1	0	1
6th(↓)	0	1	1	0
7th(↓)	0	1	1	1
8th(↓)	1	0	0	0
9th(↓)	1	0	0	1
10th(↓)	1	0	1	0
	↓	↓	↓	↓
	0	0	0	0

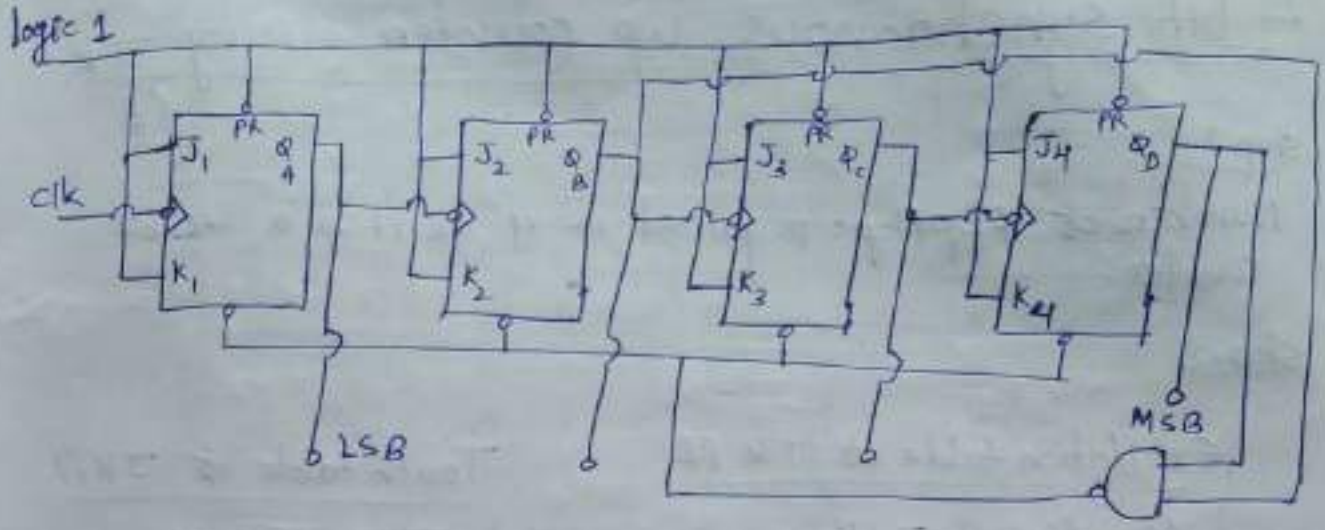
state digram



- To reset the outputs at 10th clock pulse one NAND gate is required. Output of NAND gate is given to the clear input.

- When CLR = 0, Q = 0 (Reset)
  PR = 0, Q = 1 (set)

  CLR = 1, no effect on FF
  PR = 1, no effect on FF



Notes

1. Negative edge triggered → Q is clock ; up counter
   Positive edge triggered → $\bar{Q}$ is clock ; up counter
   Negative edge triggered → $\bar{Q}$ is clock ; down counter
   Positive edge triggered → Q is clock ; down counter

2. 

   Then the total mode is = MN

Designing of synchronous counter

steps

1. Decide the number of flipflops & type of flip flop.
2. Excitation table of flipflop.
3. State diagram & circuit excitation table
4. obtain simplified equation using K-map
5. Draw the logic diagram.

4 bit synchronous up counter using JK ff

step-1

Number of flipflops required is 4 as it is a 4-bit counter.

step-2

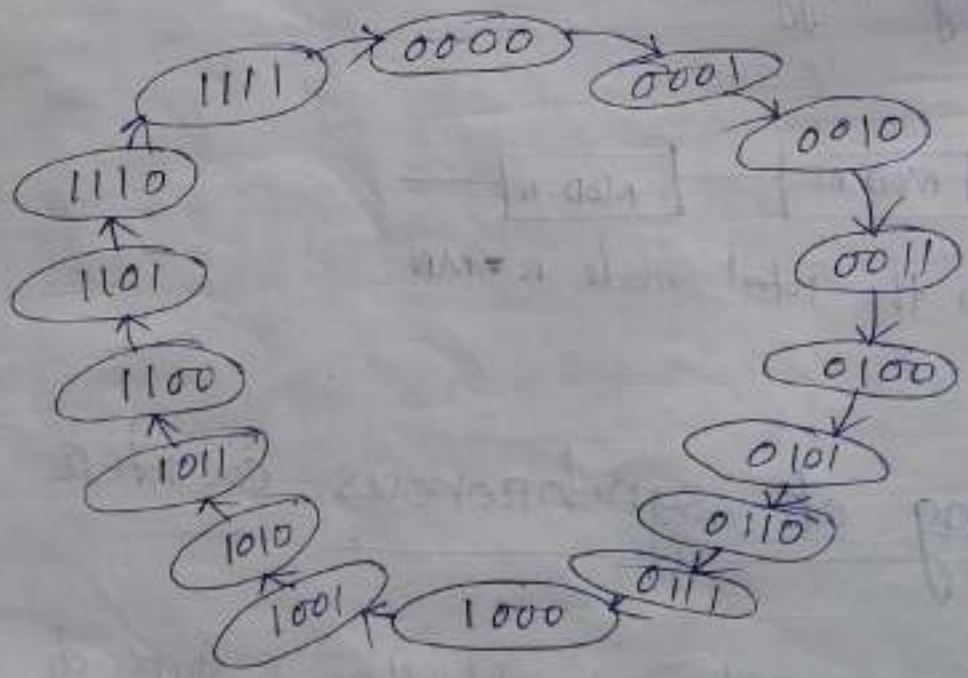
Excitation table of JK ff

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

Truth table of JK ff

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

step-3

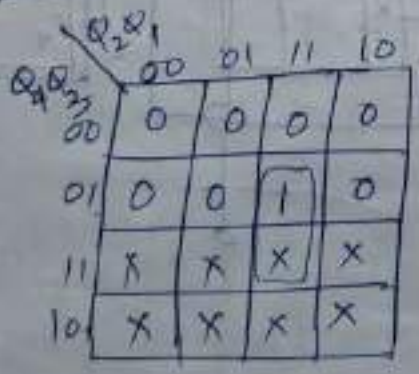


(11) circuit excitation table

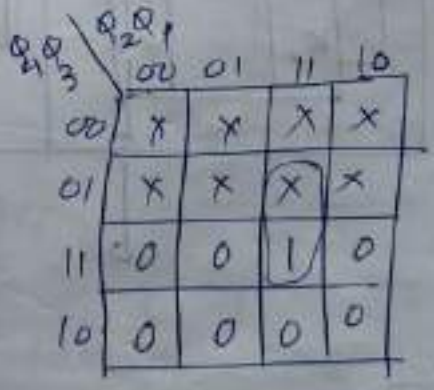
Q_4	Q_3	Q_2	Q_1	Q_4^+	Q_3^+	Q_2^+	Q_1^+	J_4K_4	J_3K_3	J_2K_2	J_1K_1
0	0	0	0	0	0	0	1	0x	0x	0x	1x
0	0	0	1	0	0	1	0	0x	0x	1x	x1
0	0	1	0	0	0	1	1	0x	0x	x0	1x
0	0	1	1	0	1	0	0	0x	1x	x1	x1
0	1	0	0	0	1	0	1	0x	x0	0x	1x
0	1	0	1	0	1	1	0	0x	x0	1x	x1
0	1	1	0	0	1	1	1	0x	x0	x0	1x
0	1	1	1	1	0	0	0	1x	x1	x1	x1
1	0	0	0	1	0	0	1	x0	0x	0x	1x
1	0	0	1	1	0	1	0	x0	0x	1x	x1
1	0	1	0	1	0	1	1	x0	0x	x0	1x
1	0	1	1	1	1	0	0	x0	1x	x1	x1
1	1	0	0	1	1	0	1	x0	x0	0x	1x
1	1	0	1	1	1	1	0	x0	x0	1x	x1
1	1	1	0	1	1	1	1	x0	x0	x0	1x
1	1	1	1	0	0	0	0	x1	x1	x1	x1

Here Q_1, Q_2, Q_3, Q_4 are the present states, $Q_1^+, Q_2^+, Q_3^+, Q_4^+$ are the next states of ffs.

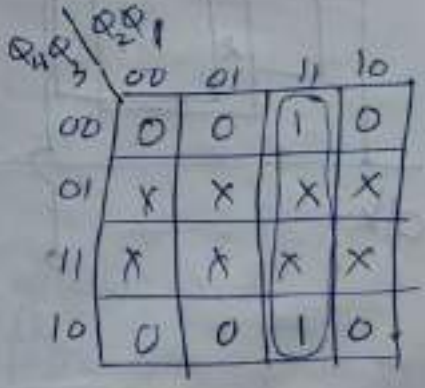
Step-4



$J_4 = Q_1Q_2Q_3$



$K_4 = Q_1Q_2Q_3$



$J_3 = Q_1Q_2$

(12)

Q_4Q_3 \ Q_2Q_1	00	01	11	10
00	X	X	X	X
01	0	0	1	0
11	0	0	1	0
10	X	X	X	X

$K_3 = Q_1Q_2$

Q_4Q_3 \ Q_2Q_1	00	01	11	10
00	0	1	X	X
01	0	1	X	X
11	0	1	X	X
10	0	1	X	X

$J_2 = Q_1$

Q_4Q_3 \ Q_2Q_1	00	01	11	10
00	X	X	1	0
01	X	X	1	0
11	X	X	1	0
10	X	X	1	0

$K_2 = Q_1$

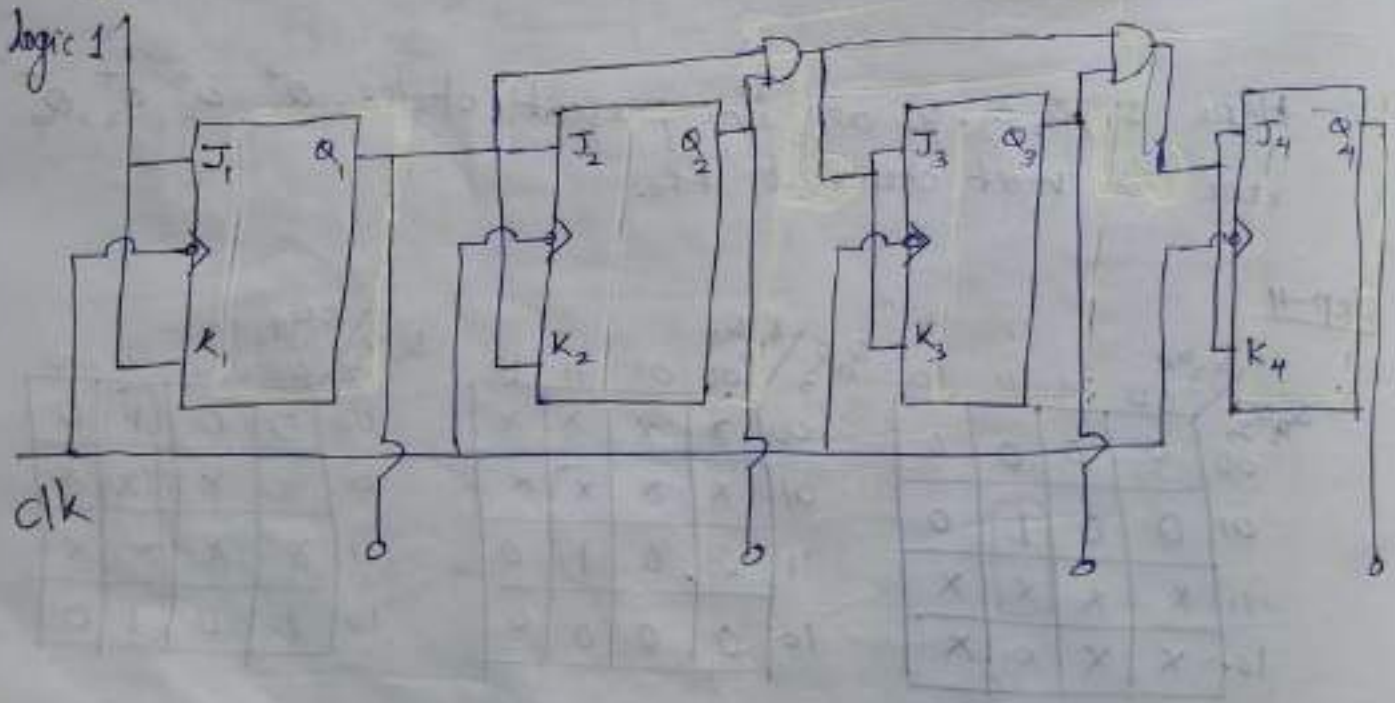
Q_4Q_3 \ Q_2Q_1	00	01	11	10
00	1	X	X	1
01	1	X	X	1
11	1	X	X	1
10	1	X	X	1

$J_1 = 1$

Q_4Q_3 \ Q_2Q_1	00	01	11	10
00	X	1	1	X
01	X	1	1	X
11	X	1	1	X
10	X	1	1	X

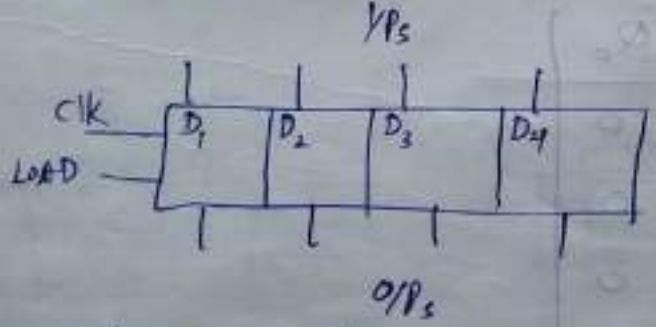
$K_1 = 1$

step-5



Registers

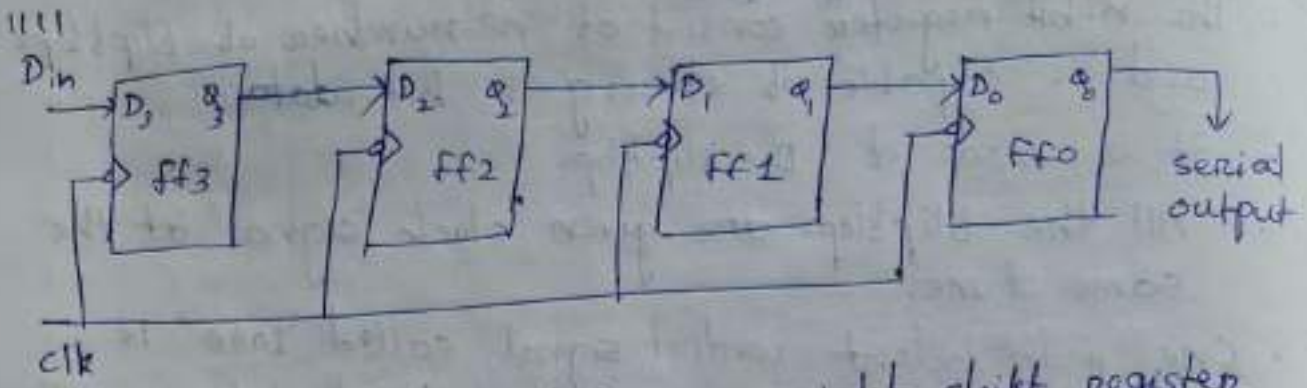
- Register is a storing device used to store more than one bit.
- It is made of group of flipflop.
- The n-bit register consist of 'n' number of flipflops and is capable of storing 'n' bit' data.
- It is made of D-flipflop.
- All the flipflops are given "clock signal" at the same time.
- One indipendent control signal called 'LOAD' is given to register to hold the stored data even if nos of clock signal passes.
- LOAD is two types
  1. synchronous i.e clock ↑ LOAD ↑
  2. Asynchronous i.e LOAD ↑
- when LOAD is ↑ (high) ff will allowed to store. when LOAD is ↓ (low) ff is not allowed to change the O/P.



- Depending on the way inputs are given and the way outputs are taken, registers are four types
  1. SISO (Serial IN Serial OUT)
  2. SIPO (Serial IN Parallel OUT)
  3. PIPO (Parallel IN Parallel OUT)
  4. PISO (Parallel IN Serial OUT)

SISO shift register

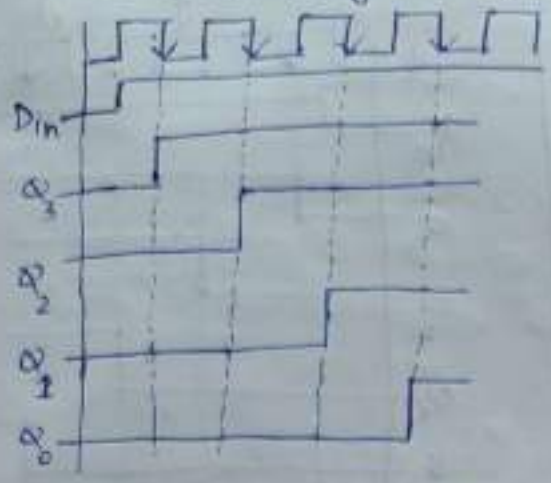
- This is a sequencial circuit used for data storage, transmission, arithmetic & logic operation



- Above figure is 4-bit SISO right shift register
- Data to be stored is given to the Din input of FF3.
- Bits of data are shifted one by one from one FF to next FF.
- Serial output is taken from last flipflop FF0.
- Let the data we want to store is 1111.
- What will be the outputs $Q_3 Q_2 Q_1 Q_0$ when clock pulses are applied is listed below.

clk	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1st (↓)	1	0	0	0
2nd (↓)	1	1	0	0
3rd (↓)	1	1	1	0
4th (↓)	1	1	1	1

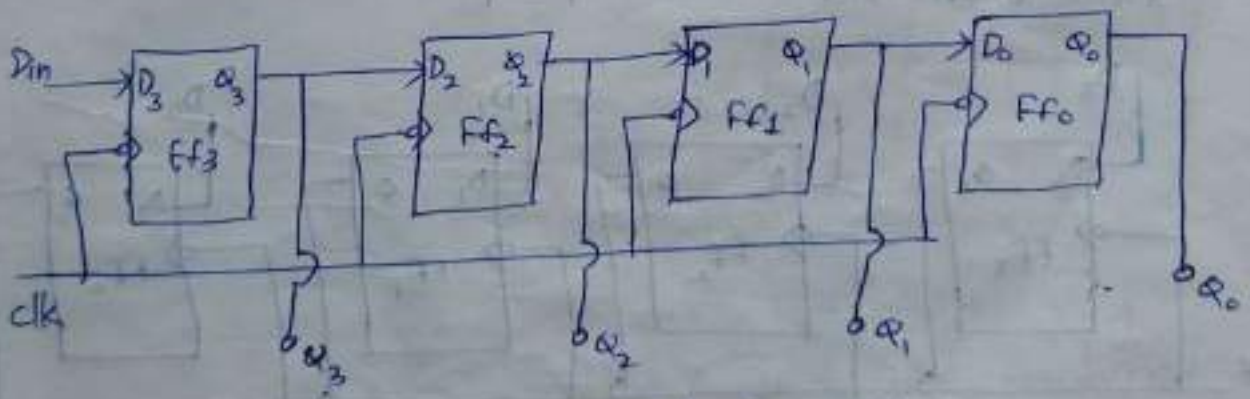
timing diagram



- It requires four clock pulse to store four bit data.
- It requires more three clock pulse to get all the bits of data at serial out.
- So it requires total 7 clock pulse to store & retrive data in serial mannner.

SIPO shift register

- In SIPO shift registers, the data is stored into the register serially while it is retrieved from it in parallel manner.
- This register is same as SISO. but here outputs are taken from all the flipflop. i.e $Q_3\ Q_2\ Q_1\ Q_0$.

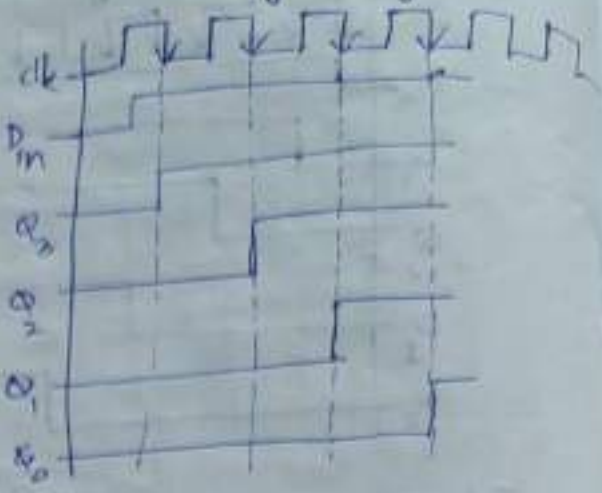


: Let the data to be stored is 1111.

- What will be the outputs $Q_3\ Q_2\ Q_1\ Q_0$ when clock pulses are applied is listed below.

clk	Q_3	Q_2	Q_1	Q_0
Initially	0	0	0	0
1st (↓)	1	0	0	0
2nd (↓)	1	1	0	0
3rd (↓)	1	1	1	0
4th (↓)	1	1	1	1

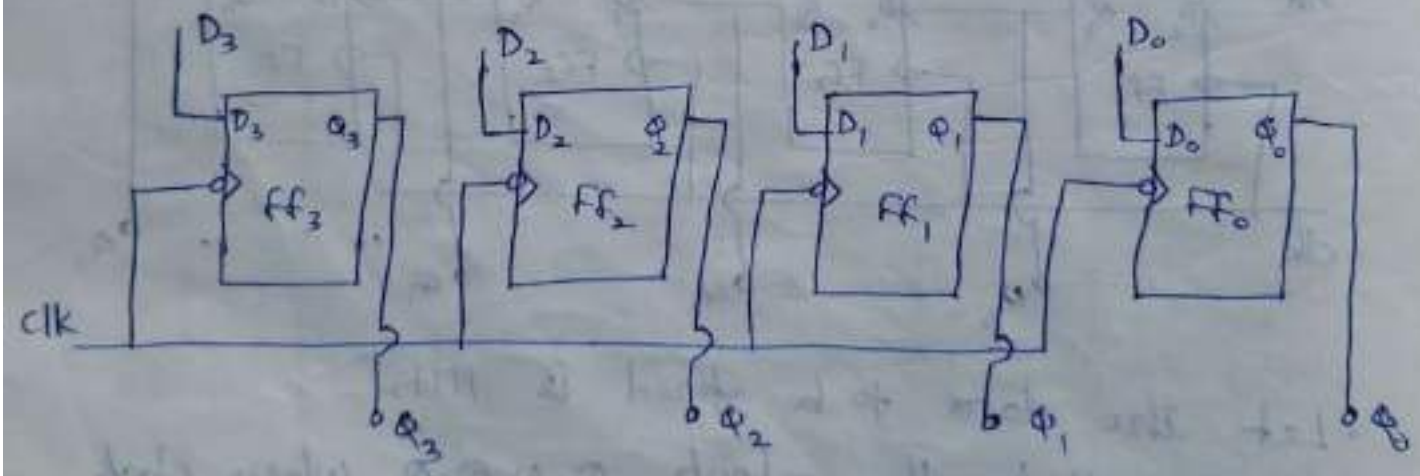
Timing diagram



- It requires four clock pulse to store four bit data.
- But to retrive no clock pulse require as data bits are already available at Q_3 Q_2 Q_1 Q_0.

PIPO shift register

- In this register data are stored in parallel manner i.e all the bits of data are stored at the same time to the individual flipflop.
- Data can also be retrived in parallel manner i.e all the bits of data can be retrived at the same time from individual flipflop.

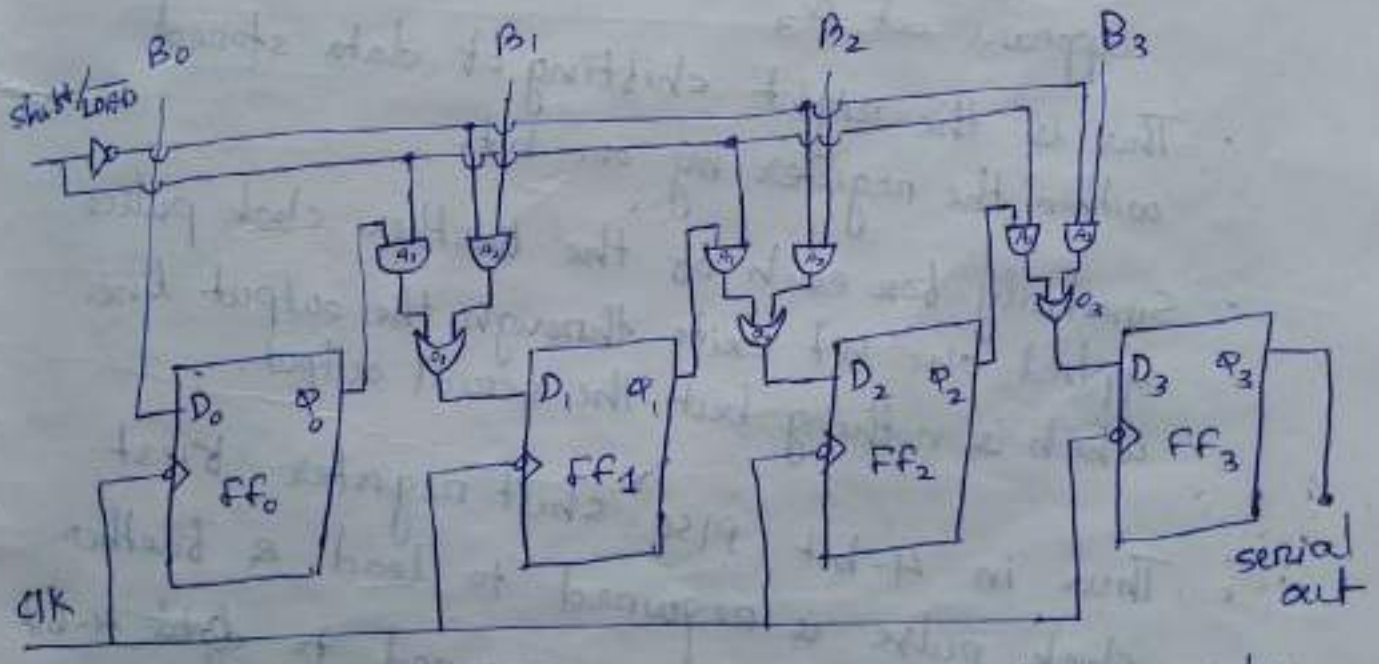


- D_3 D_2 D_1 D_0 are the data inputs Q_3 Q_2 Q_1 Q_0 are the data output.

- When negative edge of clock pulse come then $D = Q$
- It requires one clock pulse to store & retrive data.

PISO shift register

- In this register data are stored in parallel manner.
- So it needs 4 input lines for data storage of 4 bit PISO shift register.
- Here data can be retrived in serial manner through single output line.
- So shifting of bits of data needed from one flip flop to other.



- This is the figure of 4-bit PISO shift register.
- It has a control line 'shift/$\overline{LOAD}$' and combinational circuit of AND and OR gates.
- When 'shift/$\overline{LOAD}$' line is made low, A_2 AND gates of all combinational circuits become active while A_1 gates become inactive.

- Thus the bits of the input data appears as inputs to the gates A_2 are passed on as the outputs of OR gates at each individual.
- This causes the individual bits of the data to be loaded into respective flip flops at the appearance of first negative edge of clock.
- When 'shift/$\overline{LOAD}$' line is made high, A_1 AND gates of all combinational circuit become active & A_2 AND gates become inactive.
- This causes output bit of each flipflop to appear at the output of the OR gate
- When -ve edge of 2nd clock appear then Q_0 appears at Q_1, Q_1 appears at Q_2, Q_2 appears at Q_3
- This is the right shifting of data stored within the register by one bit.
- Similarly for each of the further clock pulses applied, one bit exits through the output line. which is nothing but the serial output.
- Thus in 4-bit PISO shift register first clock pulse is required to load & further three clock pulses are required to get 4-bit data at the output line.

Microprocessor

(1)

Introduction to Microprocessor

- Microprocessor is a central processing unit (CPU) on a single chip that contains milions of transistors.
- It is a multipurpose, clock driven, register based, digital-integrated circuit which accepts binary data as input, processes it according to instructions stored in its memory and provides results as output.
- Microprocessor operations include adding, subtracting, comparing two numbers and fetching numbers from one area to another.
- This is also known as the brain of the computer system.

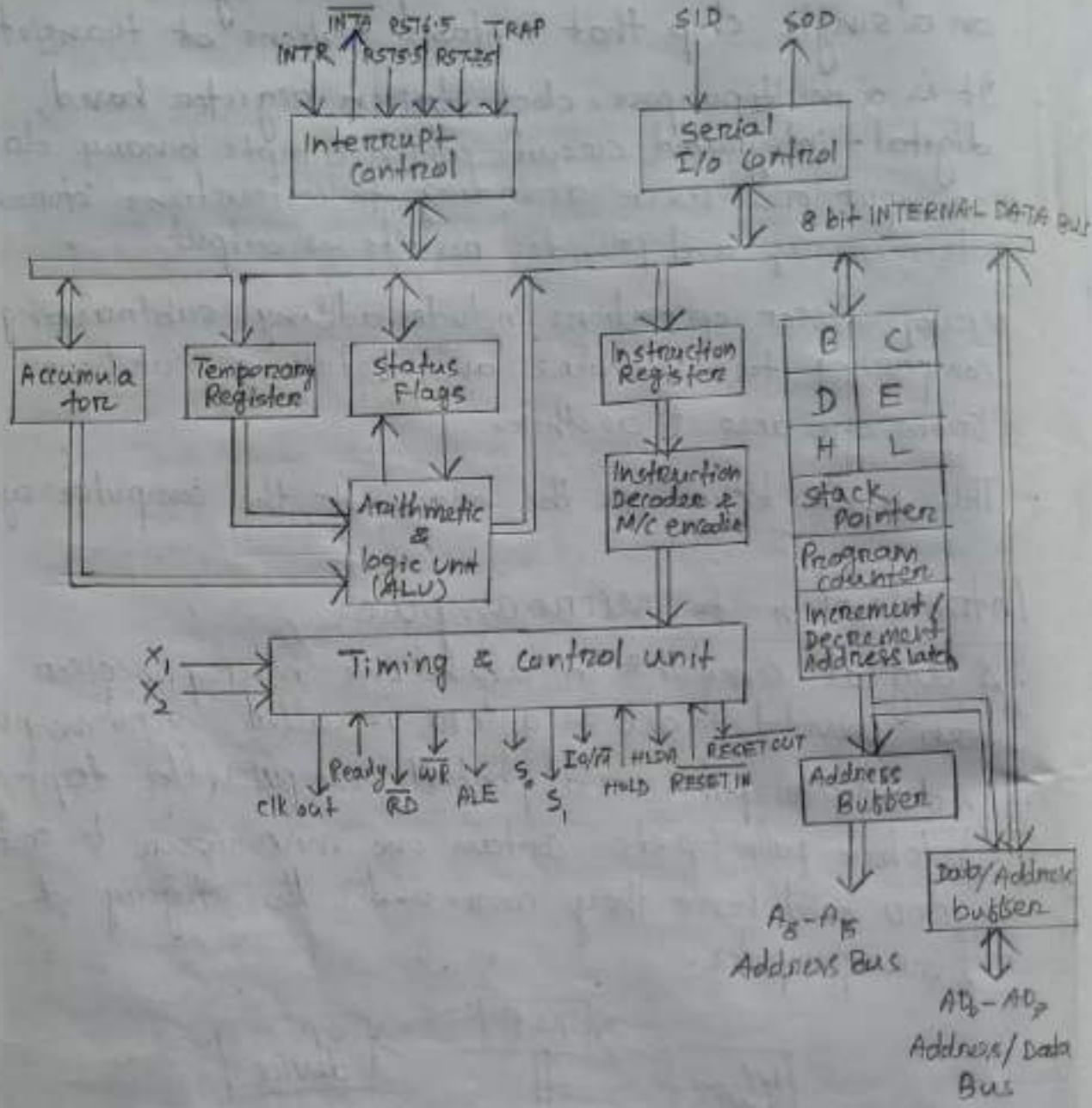
Introduction to Microcomputer

- A digital computer in which one microprocessor has been provided to act as a CPU, is called micro computer.
- A desktop computer and portable computers like laptop, notebook, palmtop etc contain one microcessor to act as CPU and hence they come under the category of microcomputer.

Architecture of Microprocessor

- It is an 8-bit microprocessor.
- It is manufactured with N-mos technology
- It has 16-bit address bus (line) and hence can address up to $2^{16} = 216 = 65536$ bytes or 64KB memory capacity.
- It supports external interrupt request.

- It is enclosed with 40 pins DIP
- It requires a +5V power supply and operates at at 3.2 MHz single phase clock.



- It consists of three main sections.
  1. Register unit.
  2. Timing and control unit
  3. Arithmetic and logic unit.

Register unit:

General purpose register:-

- It has six general purpose register to store 8-bit

data temporarily during execution of program. (3)
. These registers are named as B, C, D, E, H and L
. These registers are 8-bits but whenever the microprocessor has to handle 16-bit data, these registers can be combined as register pairs - BC, DE & HL.

Program counter (PC) :-
. It is a 16 bit special-purpose register.
. Microprocessor uses this register to sequence the execution of the instructions.
. The function of program counter is to point to the memory address from which the next byte is to be fetched.
. When a byte of instruction is being fetched, the program counter is incremented by one to point to the next memory location.

Stack pointer (SP) :-
. It is a 16 bit special-purpose register.
. This is also a memory pointer.
. It points to a memory location in R/W memory, called the stack.
. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

Address Buffer register & Data/Address Buffer register:
. These register hold the address/data, received from PC/internal data bus and then load the external address and data buses.
. These register actually behave as the buffer stage between the microprocessor and external system buses.

Control unit
. Control unit generates signals within microprocessor to

(4)

Carry out the instruction, which has been decoded.

- The control unit itself consists of three parts; i.e instruction register (IR), instruction decoder, machine cycle encoder and timing and control unit.

Intruction Register:-

- This register holds the machine code of the instruction.
- When microprocessor executes a program it reads the opcode from the memory, this opcode is stored in the Intruction Register.

Instruction Decoder & Machine cycle encoder:-

- The IR sends the machine code to this unit.
- This unit, as its name suggests, decoder the opcode and finds out what is to be done in response of the coming opcode and how many machine cycles are required to execute this instruction.

Timing & control unit:-

- It causes certain connections between blocks of the microprocessor to be opened or closed, so that the data goes where it is required and the ALU operation occur.

Arithmetic & Logic unit:-

- The ALU performs the arithmetic & logic operations such as 'add', 'subtract', 'AND', 'OR', etc.
- ALU uses temporary register and from accumulator to perform the arithmetic operations and alway stores the result in accumulator.
- ALU consists of accumulator, flag register & temporary register.

Accumulator:-

- This is an 8-bit register that is a part of ALU.
- It hold one of the operands of an arithmetic & logic operation.

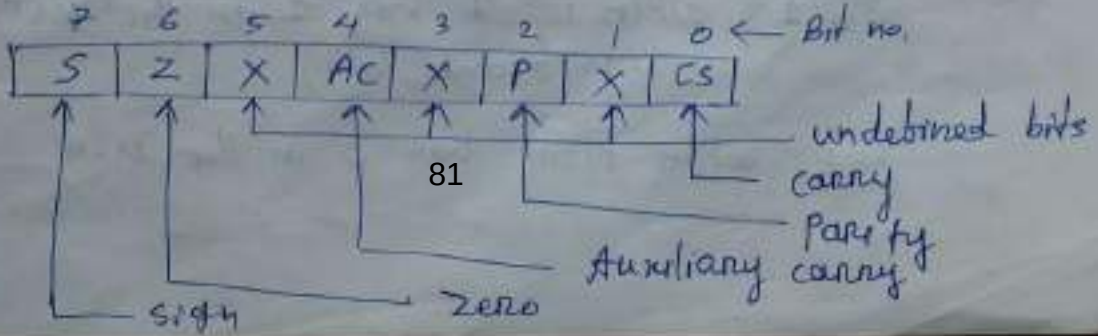
- The result of an arithmetic & logic operation is stored in the accumulator.
- It is also identified as register A.

Flags register:-
- This flags register contains five flip flops.
- The flipflops are set or reset according to the conditions arise during an arithmetic & logic operation.
- These flags registers are zero (Z), carry (CS), sign (S), parity (P) & auxiliary carry (AC).

Interrupt control :-
- The interrupt control unit has 5 interrupt inputs TRAP, RST 7.5, RST 6.5, RST 5.5 & INTR and one acknowledge signal INTA.

serial IO control :-
- IO control provides two lines, SOD and SID for serial communication.
- The serial output data (SOD) line is used to send data serially and serial input data line (SID) is used to receive data serially.

Flags Register
- It consists of 5 flipflop which changes its states according to the result stored in an accumulator after arithmetic or logic operation.
- It is also known as status registers.
- The five status flags of Intel 8085 are Carry (CS), Parity (P), Auxiliary carry (AC), Zero (Z), Sign (S)



(6) Carry flag (CS) :-

- After the execution of an arithmetic operation

CS = 1 if carry out from MSB (most significant carry bit) on addition or there will be borrow into the MSB on subtraction.

CS = 0 , otherwise

- CS is also affected by shift and rotate instructions.

Parity flag (P) :-

P = 1 , if even number of one bits in result.

P = 0 , if odd number of one bits in result

Auxiliary carry flag (AC) :-

AC = 1 , if carry out from bit 3 on addition or there is borrow into the bit 3 on subtraction

AC = 0 , otherwise

Ex :- ADD CB and E9

	7 6 5 4	3 2 1 0 ← Bit no.
CB =	1 1 0 0	1 0 1 1
E9 =	1 1 1 0	1 0 0 1
01 B4 =	1 1 0 1	0 1 0 0

Zero flag (Z) :-

Z = 1 , for a zero result

Z = 0 , for a non zero result

Sign flag (S) :-

S = 1 , when result has 1 in the MSB (most significant bit)

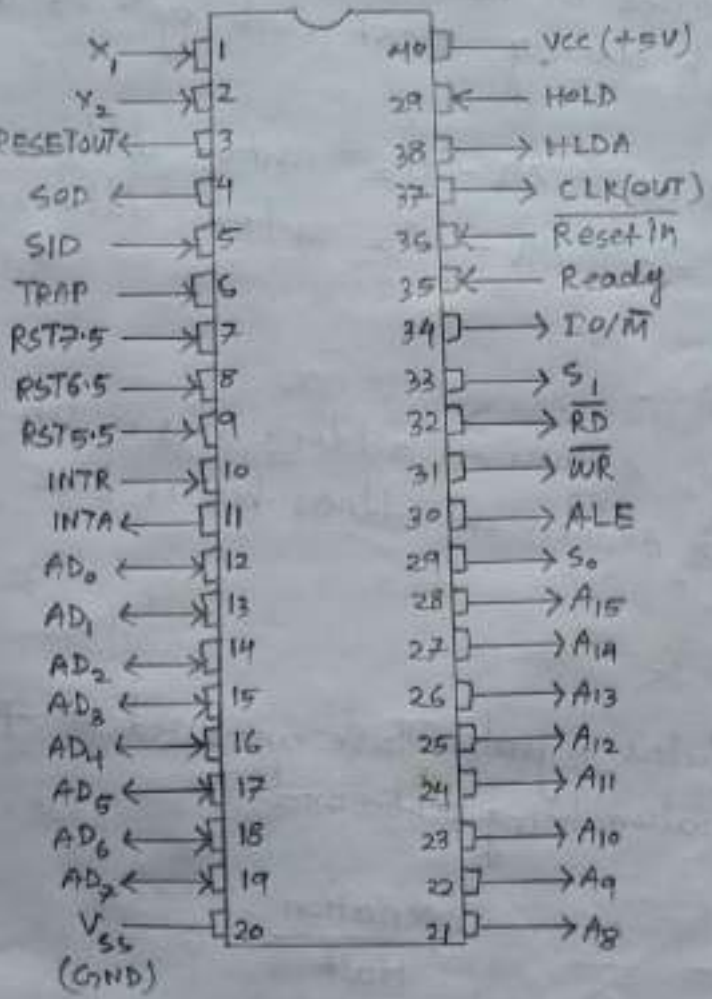
S = 0 , when result has 0 in the MSB

PSW (Program status word)

- The combination of five bits of flag register & three undefined bits is called program status word.
- PSW and accumulator are treated as a 16-bit unit for stack operation.
- Accumulator is higher order register & PSW is the lower order register.

Pin Description of Intel 8085

- Intel 8085 is a 40 pins IC.



A8 - A15 (output):-

- These are address bus and are used for the most significant bits of the memory address or 8 bits of I/O address.

(8)

$AD_0 - AD_7$ (input/output) :-

- These are time multiplexed address / data bus.
- They are used for the least significant 8 bits of the memory or I/O address during the first clock cycle of a machine cycle.
- Again they are used for data during second and third clock cycles.

ALE (output) :-

- It is an "address latch enable" signal.
- This signal is connected to the enable input of address latch circuit.
- When this signal is high the lower 8 bits of address stored in address latch.
- It goes high during first clock cycle of machine cycle.
- when ALE = 1, $AD_0 - AD_7$ contain address
  = 0, $AD_0 - AD_7$ contain data

$IO/\overline{M}$ (output) :-

- $IO/\overline{M} = 1$, Address on address bus is for I/O device.
  $= 0$, Address on address bus is for memory location.

S_0, S_1 (output) :-

- S_0, S_1 are status signals. These are used to specify the kind of operation being performed.

S_1	S_0	operation
0	0	Halt
0	1	write
1	0	Read
1	1	Fetch

$\overline{RD}$ (output) :-

when $\overline{RD} = 0$, read operation occurs.

$\overline{WR}$ (output) :-
when $\overline{WR}$ =0, write operation occurs.

Ready (input) :-
- It is used by the mp to sense whether a peripheral is ready to transfer data or not.

Ready = 1, peripheral is ready to send or receive data
=0, wait state; mp waits till it goes high.

Hold (input) :-
- This signal is used by the external devices to request the microprocessor for using the buses to transfer data from I/O device to the memory or vice-versa

HLDA (output) :-
- This is 'Hold' acknowledgement.
- It indicates that the CPU has received the 'Hold'.

HLDA = 1, microprocessor release address bus, data bus, $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$

=0, microprocessor take over address bus, data bus, $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$

INTR (input) :-
- It is an interrupt signal used by I/O devices to transfer data.
- It has lower priority
- INTR = 1, Program counter will not be allowed to increase.

$\overline{INTA}$ (output) :-
- It is an interrupt acknowledgement sent by microprocessor after 'INTR' is received.

RST 7.5, 6.5, 5.5 (inputs) :-
- These are maskable interrupt.
- When an interrupt is recognised, the next instruction is executed from a fixed location in the memory.

(10)

Interrupt	Location from which next instruction is picked up
TRAP	0024
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C

TRAP (Inputs) :-
- This is non-maskable interrupt.
- It is the highest priority interrupt.
- It is used for power failure and emergency shutdown.

$\overline{\text{Reset IN}}$ (Input) :-
$\overline{\text{Reset IN}}$ = 0, reset the microprocessor.
- This input forces
  (i) Program count is reset
  (ii) IR is reset
  (iii) All interrupts except TRAP are disabled and masked.
  (iv) SOD line is forced to 0.
  (v) Data, address & control bus are in tri-state.
- microprocessor remains in Reset sate until the $\overline{\text{Reset IN}}$ goes high.

Reset OUT (output) :-
- It indicate that the CPU is being reset.
- This reset all the peripheral devices.

X_1, X_2 (input) :-
- A crystal oscillator is connected at these two pins.
- This frequency is internally devided by 2.

CLK (output) :-
- It is a clock output for peripheral devices interfaced with the microprocessor.

SID (input) :-
- The data on this line is loaded into the 7th bit of the accumulator when "RIM" instruction is executed.

SOD (output) :-
- The 7th bit of the accumulator is output on SOD line when "SIM" instruction is executed.

Vcc :- +5 volt supply

Vss :- Ground

NOTES

- The order of priority of interrupts is as follows:
  - TRAP (highest priority)
  - RST 7.5
  - RST 6.5
  - RST 5.5
  - INTR (lowest priority)

Stack

- The stack is a sequence of memory location set aside by a programmer to store/retrive the contents of accumulator, flags, program counter and general-purpose registers during the execution of a program.
- Any part of the memory can be used as stack.
- stack works on Last In First out (LIFO) principle.

Stack instruction:-

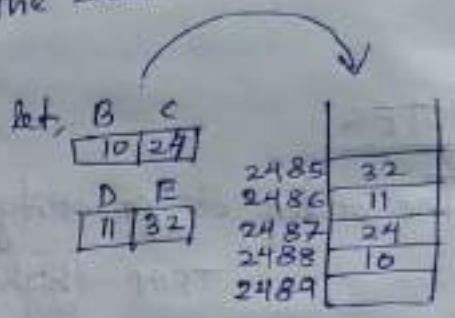
① LXI SP
- This will initialize the stack pointer & define the stack location in memory.

Ex- LXI 2489

(12) (ii) PUSH RP

- RP = Register Pair
- When this instruction is executed the contents of the register pair copied into the stack.
- First, the stack pointer register is decremented and the content of the higher order register (B/D/H/A) is copied into the location.
- Then the stack pointer register is decremented again and the contents of the lower order register (C/E/L/Flags) is copied to the location.

Ex:- LXI 2489
PUSH B
PUSH D

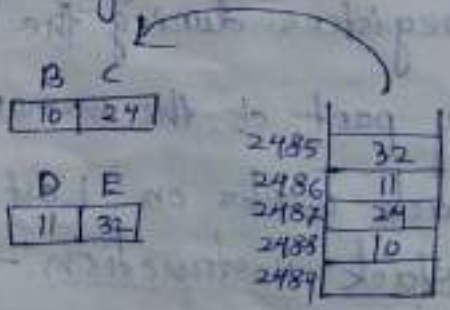


(After PUSH operation)

(iii) POP RP

- When this instruction is executed the contents of the memory location pointed out by the stack pointer register is copied to the lower order register (C/E/L/Flags).
- Then the stack pointer is incremented by 1 and the contents of that memory location is copied to the higher order register (B/D/H/A).
- The stak pointer register is again incremented by 1.

Ex:- POP B
POP B



Stack top :-

This is the top most location in stack.

Interrupts

- Interrupt is the mechanism by which the control of processor is transferred from its current program execution to another program having higher priority.
- The interrupt signal may be given to the processor by external peripheral device.
- The program that is executed after interrupt signal is received is called interrupt service routine (ISR)
- After execution of ISR the processor must return to the interrupted program.
- Interrupts can be classified into various categories based on different parameters :-

(i) Hardware & software Interrupts
(ii) Vectored & Non-vectored interrupts
(iii) Maskable & Non-maskable interrupts

Hardware Interrupts

- When microprocessor receive interrupt signals through pins of microprocessor, they are known as hardware interrupt.
- There are 5 hardware interrupts in 8085 microprocessor. i.e INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Software Interrupts

- These interrupts are inserted in between the program.
- These are the instructions of microprocessor.
- There are 8 software interrupts in Intel 8085.
- They are RST0, RST1, RST2, RST3, RST4, RST5, RST6, RST7.

Vectored Interrupts

- Vectored Interrupts are those which have fixed vector address (starting address of ISR)
- Vector address can be calculated by the formula 8 * type.

(14)

Interrupt	Vector Address
TRAP (RST 4.5)	0024 H
RST 5.5	002C H
RST 6.5	0034 H
RST 7.5	003C H
RST 0	0000 H
RST 1	0008 H
RST 2	0010 H
RST 3	0018 H
RST 4	0020 H
RST 5	0028 H
RST 6	0030 H
RST 7	0038 H

Non-Vectored Interrupts:

- Non-vectored interrupts are those in which vector address is not predefined.
- The interrupting device gives the address of ISR.
- INTR is the only non-vectored interrupt.

Maskable Interrupts

- Maskable Interrupts are those which can be blocked by microprocessor.
- These are INTR, RST 7.5, RST 6.5, RST 5.5

Non-Maskable Interrupts

- Non-Maskable interrupts can not be blocked.
- TRAT is non-maskable interrupt
- This interrupt is used in power failure condition.

Opcode

- Opcode is a part of the instruction that tells the processor what should be done.
- This is the first part of the instruction.

Operand

- Operand is a part of the instruction that contains the data to be acted on or the register or the memory location of the data.
- This is the second part of the instruction.

Ex:- LXI H, 2500H

opcode — operand

ADD B

opcode — operand

Instruction word size

- According to word size instructions are three types
  1. 1 byte
  2. 2 byte
  3. 3 byte

1 byte Instruction :-

- It includes the opcode and operand in the same byte.

Ex:- MOV A,B , MOV A,M , ADD B , RAL

2 byte Instruction :-

- First byte specifies the opcode & second byte specifies the operand.
- Second byte is either 8 bit data or I/O address.

Ex:- MVI B, 05
IN 01

3 byte Instruction:-
- First byte of the instruction is opcode.
- 2nd and 3rd byte are either 16-bit data or 16-bit address.

Ex:- LXI H, 2400H
LDA 2500H

Instruction set

- According to the function perform by an instruction, instructions are grouped as follow
  (i) Data transfer group
  (ii) Arithmetic group
  (iii) Logic group
  (iv) Branch group
  (v) stack, I/O & Machine control group.

(i) Data transfer group:-

1. MOV r_1, r_2 ; contents of register r_2 are copied into the register r_1. This is a 1 byte instruction.
   Ex:- MOV A, B

2. MOV r, M ; Contents of memory location (whose address is in H-L pair) are copied into the register r. This is a 1 byte instruction.
   Ex:- MOV B, M

3. MOV M, r ; contents of register are copied into the memory location (whose address is in H-L pair)
This is a 1 byte instruction
Ex:- MOV M, C

4. MVI r, data ; Load the data to register.
This is a 2 byte instruction.
1st byte is the opcode. 2nd byte is the data (operand)
Ex:- MVI A, 05

5. MVI M, data ; The data specified by the operand is loaded to the memory location (whose address is in H-L pair)
This is a 2 byte instruction.
Ex:- MVI M, 08

6. LXI RP, data 16 bit ; Higher order 8-bit data is loaded in higher order register. Lower order 8-bit data is loaded in lower order register.
This is a 3 type instruction
1st byte opcode. 2nd & 3rd bytes are operand.
Ex:- LXI B, 12A0 ; [B] ← 12
[C] ← A0

7. LDA Address ; contents of memory location (whose address is specified by operand) is copied to the Accumulator.
This is a 3 byte instruction.
Ex:- LDA 2400

(8)

8. STA address ; Contains of Accumulator are copied to the memory location whose address is specified by operand.
This is a 3 byte instruction.
Ex:- STA 2000

9. LHLD address ; Contains of memory location (whose address is specified by operand) are copied into the register L. The contains of next memory location (incremented by 1) are copied into the register H.

Ex:- LHLD 2500 ; [L] ← [2500]
[H] ← [2501]

10. SHLD address ; The contents of register L are stored into the memory location specified by operand. The contents of H register are stored into the next memory location.
This is a 3 byte instruction.

Ex:- SHLD 2500 ; [2500] ← [L]
[2501] ← [H]

11. LDAX RP ; copy the content of memory location (whose address is in register pair) into the accumulator
This is a 1 byte instruction

Ex:- LDAX B

12. STAX RP ; copy the content of Accumulator into the memory location (whose address is in register pair)

Ex:- STAX D

13. XCHG ; The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.

(ii) Arithmetic group :-

1. ADD R ; Contains of register is added with the contains of accumulator, and the result is stored in accumulator. This is a 1 byte instruction
Ex:- ADD B

2. ADD M ; Contains of memory location (whose address is in H-L pair) is added with the contains of accumulator, and the result is stored in accumulator.
This is a 1 byte instruction.
Ex:- ADD M

3. ADC R ; Contains of register is added with the contains of accumulator and contains of carry flag, the result is stored in accumulator.
Ex:- ADC B

4. ADC M ; Contains of memory location (whose address is in H-L pair) is added with the contains of accumulator and contains of carry flag, the result is stored in accumulator.
Ex:- ADC M

5. ADI data ; Data specified in operand is added with the contains of accumulator and the result is stored in accumulator.
This is a 2 byte instruction
Ex:- ADI 05

6. ACI data ; Data specified in operand is added with the contains of accumulator and contain of carry flag, the result is stored in accumulator.
This is a 2 byte instruction.
Ex:- ACI 08

7. DAD RP ; Contains of register pair is added with the contains of H-L pair, and the result is stored in H-L pair
This is a 1 byte instruction.
Ex:- DAD B

8. SUB R ; Contains of register is subtracted from contains of accumulator, and the result is stored in accumulator
This is a 1 byte instruction
Ex:- SUB B

9. SUB M ; Contains of memory location (whose address is in H-L pair) is subtracted from contains of accumulator, and the result is stored in accumulator.
This is a 1 byte instruction.

Ex:- SUB M

10. SBB M ; Contains of memory location (whose address is in H-L pair) and contain of carry flag are subtracted from contains of accumulator, result is stored in accumulator.
This is a 1 byte instruction.
Ex:- SBB M

11. SBB R ; Contains of register is subtracted from contains of accumulator, and result is stored in accumulator.
This is a 1 byte instruction.
Ex:- SBB C

12. SUI data ; Data specified in operand is subtracted from contains of accumulator and result is stored in accumulator.
This is a 2 byte instruction.
Ex:- SUI A2

13. SBI data ; Data specified in operand and contain of carry flag are subtracted from contains of accumulator, result is stored in accumulator.
This is a 2 byte instruction
Ex:- SBI 82

14. INR R ; Contains of register is incremented by 1.
This is a 1 byte instruction
Ex:- INR B

15. INR M ; Contains of memory location (whose address is in H-L pair) is incremented by 1.

16. DCR R ; Contains of register is decremented by 1.
ex:- DCR B

17. DCR M ; Contains of memory location (whose address is in H-L pair) is decremented by 1.

18. INX RP ; Contains of register pair specified in operand is incremented by 1. ex - INX H

19. DCX RP ; Contains of register pair is decremented by 1.
Ex:- DCX B

20. DAA ; Contains of Accumulator is converted from Hexadecimal to BCD (Binary coded decimal).

(III) Logical Group :-

1. ANA R ; Contains of register are ANDed with the contains of accumulator, result is stored in accumulator. This is a 1 byte instruction.
Ex:- ANA C

2. ANA M ; Contains of memory location (whose address in in H-L pair) are ANDed with the contains of accumulator, result is stored in accumulator.

This is a 1 byte instruction.

3. ANI data ; Data specified by operand is ANDed with the contains of accumulator, and result is stored in accumulator.
This is a two byte instruction.
Ex:- ANI 05

4. ORA R ; Contains of register are ORed with the contains of accumulator, and result is stored in accumulator
This is 1 byte instruction.
Ex:- ORA D

5. ORA M ; Contains of memory location (whose address is in H-L pair) are ORed with the contains of accumulator, and result is stored in accumulator.
This is 1 byte instruction.
Ex:- ORA M

6. ORI data ; Data specified by operand is ORed with the contains of accumulator, and result is stored in accumulator.
This is a byte instruction.
Ex:- ORI 32

7. XRA R ; Contains of register are XORed with the contains of accumulator, and result is stored in accumulator.
This is 1 byte instruction.
Ex:- XRA B

8. XRA M ; Contains of memory location (whose address is in H-L pair) are XORed with the contains of accumulator, and result is stored in accumulator
This is 1 byte instruction.

(24)

9. XRI data ; Data specified by operand is XORed with the contains of accumulator, and result is stored in accumulator. This is 2 byte instruction.
Ex:- XRI 42

10. CMA ; Contains of accumulator are complemented.
This is 1 byte instruction.

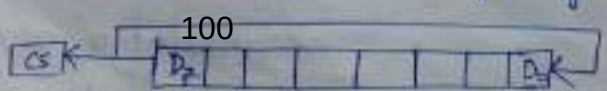
11. CMC ; Contain of carry flag is complemented.

12. STC ; Contain of carry flag becomes 1 (set)

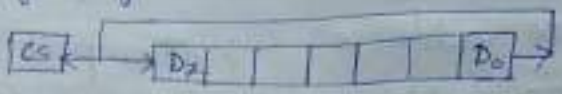
13. CMP R ; Contains of register are compared with accumulator contains.
In this process the content of register is subtracted from content of accumulator.
Status flags are set according to the result of the subtraction.
But result is discarded.
Content of accumulator remain uncharge.

14. CPI data ; Data specified by operand is compared with accumulator content.

15. RLC ; Each binary bit of the accumulator is rotated left by one position.
Bit D_7 is placed in the position of D_0 as well as in carry flag.

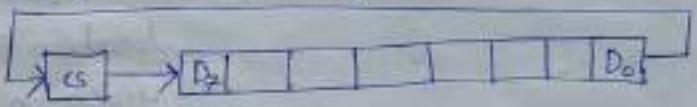


16. RRC ; Each binary bit of the accumulator is rotated right by one position. Bit D_0 is placed in the position of D_7 as well as in carry flag.



17. RAL ; Each binary bit of the accumulator is rotated left by one position through carry flag.

18. RAR ; Each binary bit of the accumulator is rotated right by one position through carry flag.



(iv) Branch Group :-

- There are two types of branch instructions
  - (i) Unconditional
  - (ii) Conditional

1. Unconditional jump

JMP address ; The program sequence is transferred to the memory location specified by the address in the operand

Ex :- JMP 2034

2. Conditional jump

JC address ; Jump if there is carry

JNC address ; Jump if there is no carry

```
JZ address      ; Jump if the result is zero.
JNZ address     ; Jump if the result is not zero.
JP address      ; Jump if the result is plus.
JM address      ; Jump if the result is minus.
JPE address     ; Jump if the result has even
                  parity
JPO address     ; Jump if the result has odd
                  parity.
```

3. Unconditional subroutine call

```
CALL address    ; The program sequence is
                  transferred to the memory
                  location specified by address
                  given in operand.
                  Before the transfer, the address
                  of the next instruction after CALL
                  (i.e the contents of program
                  counter) is stored into the
                  stack. Higher order 8-bit of PC
                  are stored first. Then lower order
                  8-bit of PC are stored.
```

4. Conditional subroutine call

```
CC address      ; Call subroutine if carry occurs
CNC address     ; Call subroutine if no carry.
CZ address      ; Call subroutine if the result
                  is zero
CNZ address     ; Call subroutine if the result
                  is not zero.
CP address      ; Call subroutine if the result
                  is plus.
CM address      ; Call subroutine if the result
                  is minus
CPE address     ; Call subroutine if the result has
                  even parity.
```

(27) CPO address ; Call subroutine if the result has odd parity.

5. Return from subroutine unconditionally

RET ; The program sequence is transferred from the subroutine to the calling program.
The two bytes from the top of the stack are copied into the program counter.

6. Return from subroutine conditionally

RC ; Return from subroutine if carry occurs
RNC ; Return from subroutine if no carry
RZ ; Return from subroutine if the result is zero
RNZ ; Return from subroutine if the result is not zero.
RP ; Return from subroutine if the result is plus.
RM ; Return from subroutine if the result is minus.
RPE ; Return from subroutine if the result has even parity.
RPO ; Return from subroutine if the result has odd parity.

7. Restart

RST n ; RST instruction is equivalent to 1 byte call instruction.
These are the interrupt instructions.

Instruction	Restart Address
RST 0	0000 H
RST 1	0008 H
RST 2	0010 H
RST 3	0018 H
RST 4	0020 H
RST 5	0028 H
RST 6	0030 H
RST 7	0038 H

8. PCHL ; The program sequence is transfered to the memory location (whose address is in H-L pair) The contents of registers H and L are copied into the program counter. The contents of H are placed as the higher order byte and the contents of L as the lower order byte.

(V) stack, I/O and Machine control Group

stack Instructions:-

1. PUSH RP ; Content of register pair are copied into the stack.
   Ex:- PUSH B
2. PUSH PSW ; Content of accumulator & program status word (PSW) are copied into the stack. First content of accumulator are stored then content of PSW are stored.
3. POP RP ; The contents of the memory location pointed out by the stack pointer register are copied to the lower

(24)

order register of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the higher order register. SP again incremented by 1

Ex :- POP B

4. POP PSW ; The contents of the memory location pointed by stack pointer are copied to the PSW. The stack pointer is incremented by 1 and the contents of that memory location are copied to the accumulator. S.P again incremented by 1.

5. XTHL ; Exchange stack top with HL i.e [L] ↔ [SP] H ↔ [SP+1]

6. SPHL ; move the contents of H-L pair to stack pointer

I/O instructions :-

1. IN 8bit port address ; Contents of input port (whose address is in operand) are loaded into the accumulator.

2. OUT 8bit port address ; Contents of accumulator are copied into the output port specified by operand.

Machine control instructions :-

1. HLT ; CPU finished its current
2. EI ; Interrupt is enabled.
3. DI ; Interrupt is disabled.
4. SIM ; set Interrupt Mask
5. RIM ; Real Interrupt Mask
6. NOP ; No operation

Addressing Modes

- Addressing modes are the different ways the microprocessor access data.
- The addressing modes are
  1. Immediate addressing
  2. Register Addressing
  3. Direct Addressing
  4. Register indirect Addressing
  5. Implicit Addressing.

Immediate Addressing mode :-

- In this mode 8 or 16 bit data can be specified as a part of instruction

```
Ex:- MVI A, 05    ; 05 is copied into accumulator
     ADI 06       ; 06 is added with accumulator & result stores in Accumulator
     LXI H, 25AB  ; 25AB is copied into H-L pair.
                  [H] ← 25, [L] ← AB
```

Register Addressing mode :-

- In this mode data transfers from source register to destination register, addition, subtraction operations
- Name of registers (source & destination) are mentions in instruction.

```
Ex:- MOV A, B ; copy the content of B register to Accumulator.
```

(31)

ADD D ; Add content of register D with content of Accumulator & result is stored in Accumulator.

SUB C

Direct Addressing mode :-

. In this mode of addressing 16 bit address of data is specified in the instruction.

Ex :- LDA 2400 ; copy data from 2400 memory location to accumulator

STA 2A3B ; copy data from accumulator to memory location 243B

LHLD 7203

SHLD 4321

Register indirect Addressing mode

. In this mode of addressing data is in memory location whose address is specified by a register pair.

Ex :- MOV A,M ; Content of memory location is copied to the accumulator

ADC M ; Contains of memory location (whose address is in H-L pair) is added with the contains of accumulator and contains of carry flage, the result is stored in Accumulator.

STAX B ; Copy the content of Accumulator into the memory location (whose address is in register pair B-C)

LDAX D

(32)

Implicit Addressing mode :-

- This mode does not require any operand address
- The operation mentioned by instruction is performed on the content of the accumulator

```
Ex:- CMA  ; Contents of accumulator are complemented.
     RAL  ; Rotate the contents of accumulator left through carry.
     RAR  ; Rotate the contents of accumulator right through carry.
     STC  ; Contain of carry flage becomes set.
```

Instruction Cycle

- The necessary steps that a CPU carries out to fetch an instruction & necessary data from the memory and execute it constitute an instruction cycle.

Instruction cycle = Fetch cycle + execution cycle.

Fetch cycle

- The necessary steps which are carried out to fetch an opcode from the memory constitute a fetch cycle.
- Fetch cycle consist of "four clock pulses"

Execution cycle

- The necessary steps which are carried out to get data, if any from the memory and to perform the specific operation specified in an instruction, constitute an execution cycle.
- The requirement of no of clock pulses is not fix for execution cycle. It varies depends on the type of instruction to be executed.

Fetch operation

- The program counter (PC) keeps the memory address of the next instruction to be executed.
- In the beginning of the fetch cycle the content of the PC, which is the address of the memory location where opcode is available, is sent to the memory.
- The memory places the opcode on the data bus so as to transfer it to the CPU.

Execute operation

- The opcode fetched from the memory goes to the data/address buffer and then to Instruction register, IR.
- From the IR it goes to the decoder circuit which decode the instruction.
- After the instruction is decoded, execution begin.
- If the operand is in the general purpose register, execution is immediately performed. Here one clock cycle is required for decoding & execution.
- If an instruction contain data or operand address which are still in the memory, CPU has to perform read operation to get desired data.
- After receiving the data it performs execute operation.

(34)

Machine cycle

- The necessary steps carried out to perform a fetch, a read or a write operation constitute a machine cycle.
- An instruction cycle consists of several machine cycle.
- Opcode of an instruction is fetched in the first machine cycle.

State or T-state

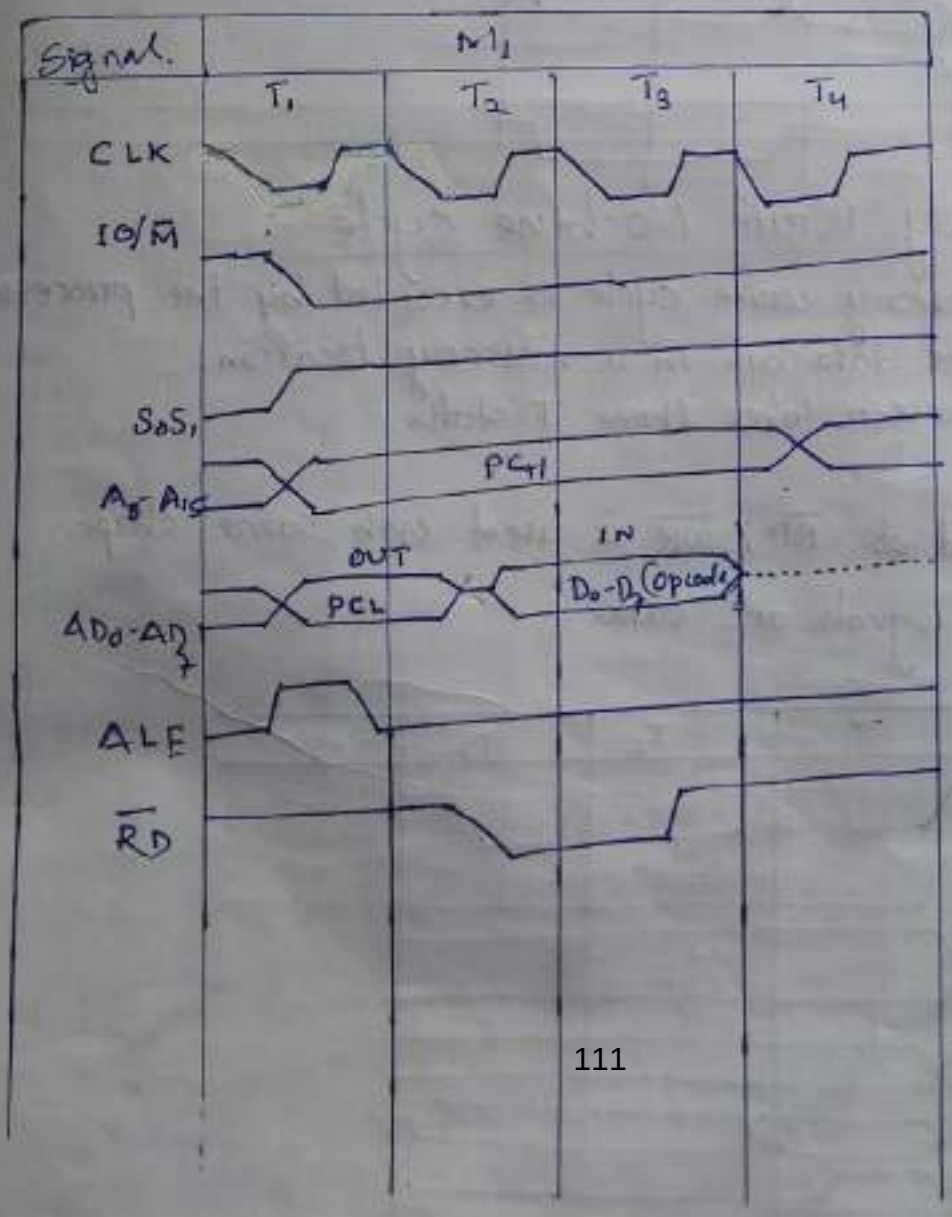
- One subdivision of an operation performed in one-clock cycle is called a state or T state.

Timing Diagram for opcode fetch cycle

- Graphical representation of an machine cycle is called timing diagram.
- For opcode fetch 4 clock cycles are required i.e T_1, T_2, T_3, T_4.

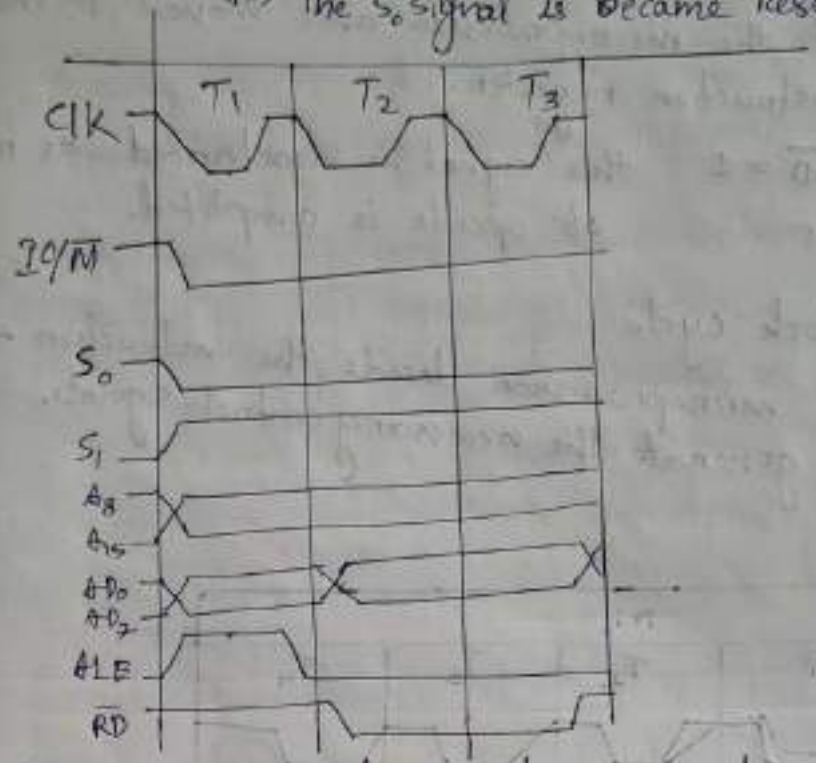
- In T_1 clock cycle
  - i. The content of program counter is placed in the address bus. $A_{15} - A_8$ contains the higher order bits & $AD_7 - AD_0$ contains the lower order bits.
  - ii. $IO/\overline{M} = 0$, indicate that this is a memory operation
  - iii. $S_0 = 1$, $S_1 = 1$, indicate that this is a opcode fetch operation
  - iv. ALE = 1, indicate that $AD_0 - AD_7$ act as address bus.

- In T_2 clock cycle
  - i. ALE = 0, indicate that $AD_0 - AD_7$ act as data bus
  - ii. $\overline{RD} = 0$, This signal make memory to load opcode into the data bus.
- In T_3 clock cycle
  - i. The opcode available on the data bus enters into the microprocessor and moved to the instruction register.
  - ii. $\overline{RD} = 1$, this signal is deactivated as read of opcode is completed.
- In T_4 clock cycle
  - i. The microprocessor decode the instruction & generate the necessary control signals.



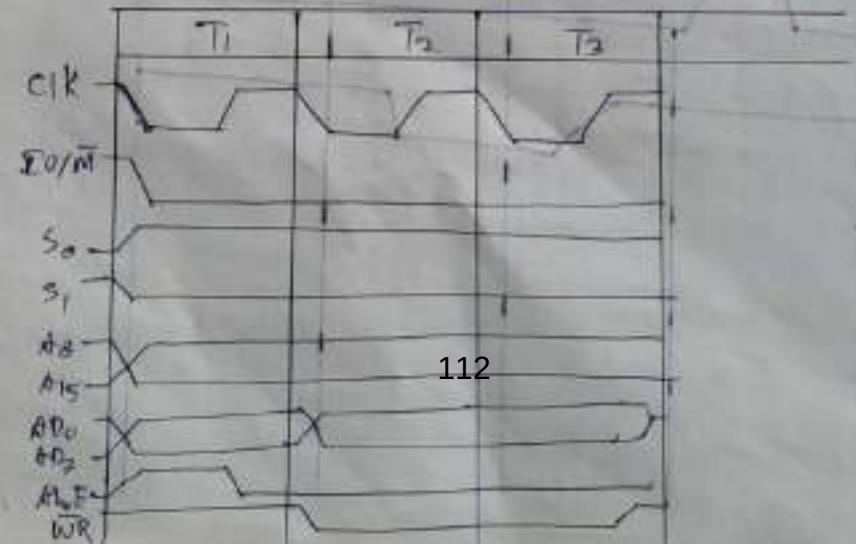
Memory Read Machine Cycle

- The memory read cycle is executed by the processor to read a data byte from memory.
- The machine cycle is exactly same to opcode fetch except: (i) It has three T-states
  (ii) The S_0 signal is became reset. i.e 0.



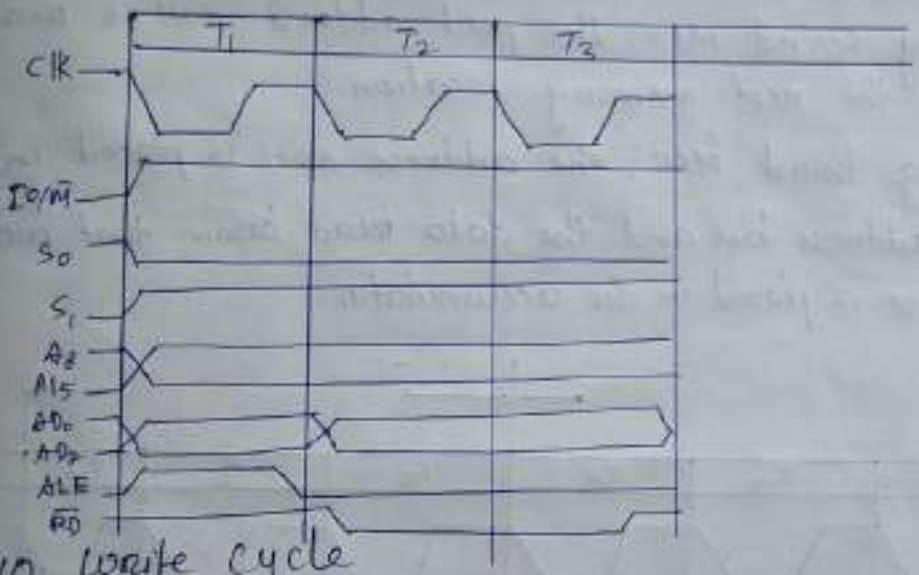
Memory Write Machine Cycle

- The memory write cycle is executed by the processor to write data byte in a memory location.
- The processor takes three T states
- $S_0 = 1, S_1 = 0$
- Instead of $\overline{RD}$; $\overline{WR}$ is used with same shape.
- Other signals are same.



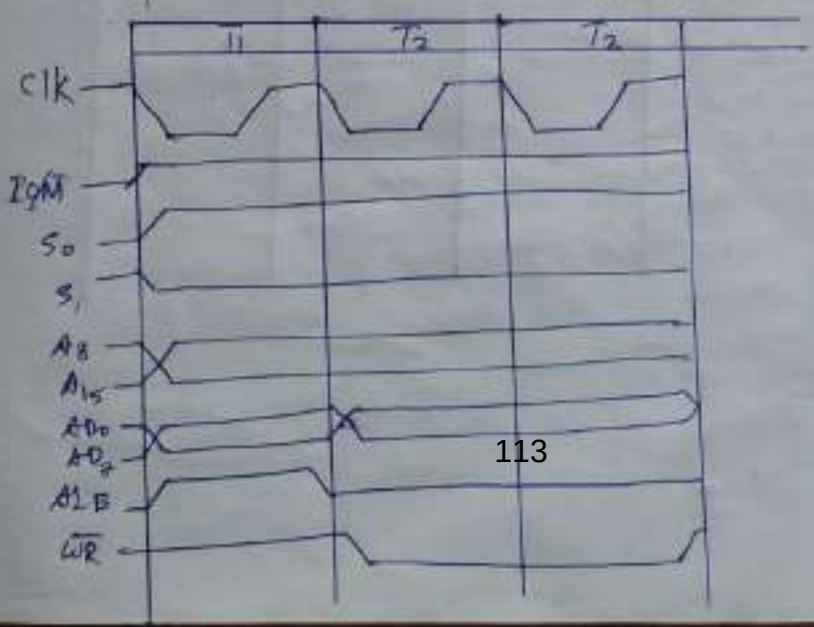
I/O Read cycle

- The I/O read cycle is executed by the processor to read a data byte from an input port.
- The 8-bit port address is placed both in the lower and higher order address bus.
- The processor takes three T-states to execute this machine cycle (M/C)
- $S_0 = 0$, $S_1 = 1$; indicate read operation
- $IO/\overline{M} = 1$; indicate I/O operation
- others are say as Memory read M/C.



I/O write cycle

- The I/O write cycle is executed by the processor to write a data byte to output port.
- It take 3 T states.
- $S_0 = 1$, $S_1 = 0$
- $IO/\overline{M} = 1$
- Others are same as memory write M/C.

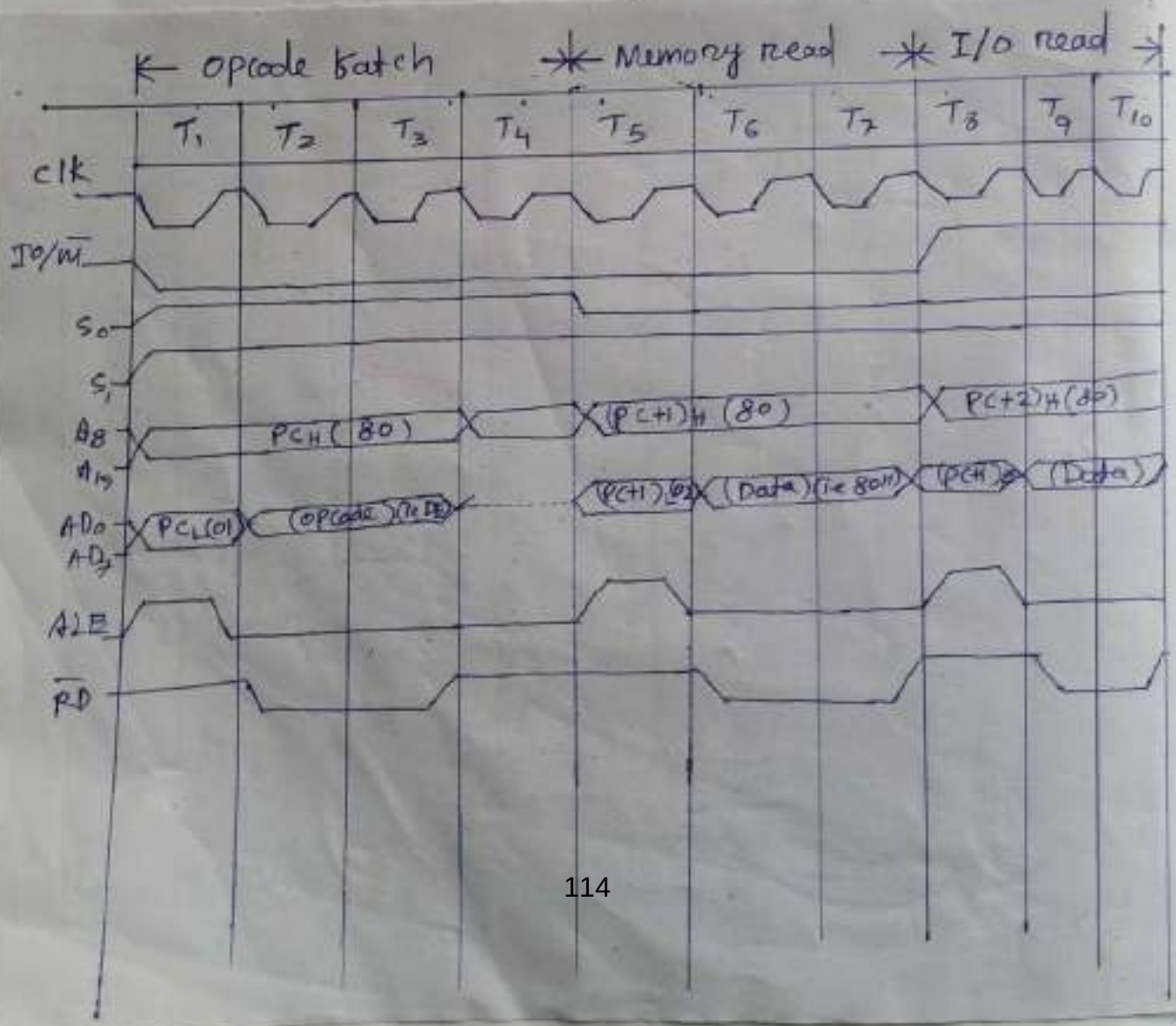


Timing diagram for 'IN 80H'

Let, the instruction 'IN 80H' is in memory location 8001 & 8002 as it is a 2 bytes instruction

Address	Mnemonics	opcode
8001	IN 80H	DB
8002		80

- During the first M/C, the opcode DB is fetched from the memory, placed in the instruction register and decoded.
- During second M/C, the port address 80H is read from the next memory location.
- During third M/C, the address 80H is placed in the address bus and the data read from that port address is placed in the accumulator.



Counter & Time delay

- Counter is a subroutine. This is designed simply by loading required number into a register & using INR or DNR instruction.
- Loop is established to increment or decrement the content of register till it reaches to final number.
- Time delay is a subroutine designed to create specific delay.
- A register is loaded with a number depending on the time delay required and then register is decremented until it reaches zero by setting up a loop with conditional jump instruction.

Time delay using one register

Label	Opcode	operand	comments		
	MVI	C, FFH	; load FF into register C	7T	1X7T
Repeat	DCR	C	; Decrement C	4T	255X4T
	JNZ	Repeat	; Jump back to Decrement C if C≠0	10T/7T	(254X10T)+(1X7T)
	RET		; Return to main program	10T	1X10T
				Total T states	= 3584 T

- JNZ instruction requires 7 T states if condition is not satisfied otherwise it requires 10 T states.
- 255 is the decimal value of hexadecimal no. FFH.
- If 2MHz is the external clock frequency of 8085 then internal clock frequency is half of external clock frequency i.e $\frac{2MHz}{2} = 1MHz$

· Time for one T state = $\frac{1}{F} = \frac{1}{1MHz} = 2\mu s$

· Time required for entire program

$= 3584 \times 2\ \mu s$

$=$

Time delay using two registers / nested Loop

Label	opcode	operand	comments	states per execution	No. of times executed
	MVI	B, FFH	; Load FF into B	7	1
Loop	MVI	C, FFH	; Load FF into C	7	255
Repeat	DCR	C	; Decrement C	4	255 x255
	JNZ	Repeat	; Jump back to Decrement C if C ≠0	10/7	(254 +1) x255
	DCR	B	; Decrement B	4	255
	JNZ	Loop	; If B ≠0, then go back to Loop	10/7	254 +1
	RET		; Return to main program	10	1

Total T-states required for this program

$= 7\times1 + 7\times255 + 4\times255\times255 + (254\times10 + 7\times1)\times255$
$+ 4\times255 + (254\times10 + 7\times1) + 10\times1$

$= 914954$ states

If internal clock frequency is 1 MHz then,

Time for one T state = $\frac{1}{F} = \frac{1}{1MHz} = 2\mu s$

· Time delay for entire program

$= 914954 \times 2\ \mu s$

$=$

Time delay using Register pair

Label	opcode	operand	comments	states per execution	No. of time executed
	LXI	D, FFFF	; load FFFF in register pair D-E	10	1
Repeat	DCX	D	; Decrement the content of D-E	6	
	MOV	A, D	; Move content of D to A	4	
	ORA	E	; Do OR operation of A content with E content	4	
	JNZ	Repeat	; check if A≠0 then go back to Repeat	10/7	
	RET		; Return to main program	10	

Memory Mapping

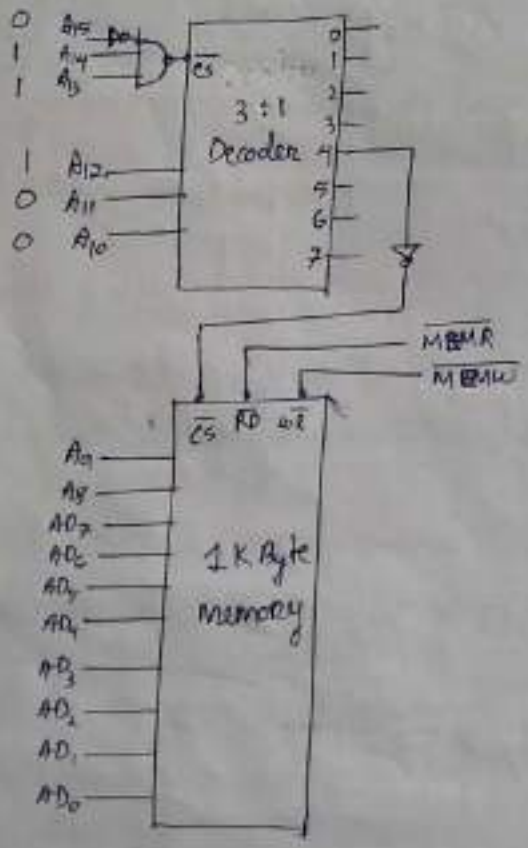
- It is the process of assigning address range to each memory chip in a micro computer.
- If a memory chip has a capacity of 2^n Byte then required address line for that memory chip is n line.

ex:- for 1K Byte memory chip

$1\,K\,Byte = 2^{10}$ Byte

$\therefore$ address line = 10 lines.

- Let's consider 1K Byte memory chip is connected to the microprocessor.



- 10 address lines A_0 to A_9 are used to address location inside the memory. A_{10} to A_{15} are used for chip select.
- As per the diagram the range of memory address can be assigned is as follow:

43

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	In Hexadecimal
0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	7000
0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	7001
0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	7002
0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	7003
0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	7004
0	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	73FF

I/O Mapping

- It is the technic to assign address to the input output devices.
- This technic is of two types.
  1. Memory mapped I/O
  2. I/O mapped I/O

	Memory Mapped I/O	I/O Mapped I/O
	I/O devices are treated as memory.	I/O devices are treated as I/O
1. No.s of Address lines for memory	16	16
2. No.s of Address lines for I/O	16	8
3. Control signal for memory	$\overline{MEMR}$, $\overline{MEMW}$	$\overline{MEMR}$, $\overline{MEMW}$
4. Control signal for I/O	$\overline{MEMR}$, $\overline{MEMW}$	$\overline{IOR}$, $\overline{IOW}$
5. Addressing capacity of memory	$2^{16} = 64K$	$2^{16} = 64K$
6. Addressing capacity of I/O	Part of 64K is used for I/O addressing	$2^8 = 256$
7. Instructions	Instructions used for memory are also used for I/O ex:- STA 2000H LDA 2500H MOV A,M	Separate instructions are used for I/O ex:- IN 08H OUT 05H

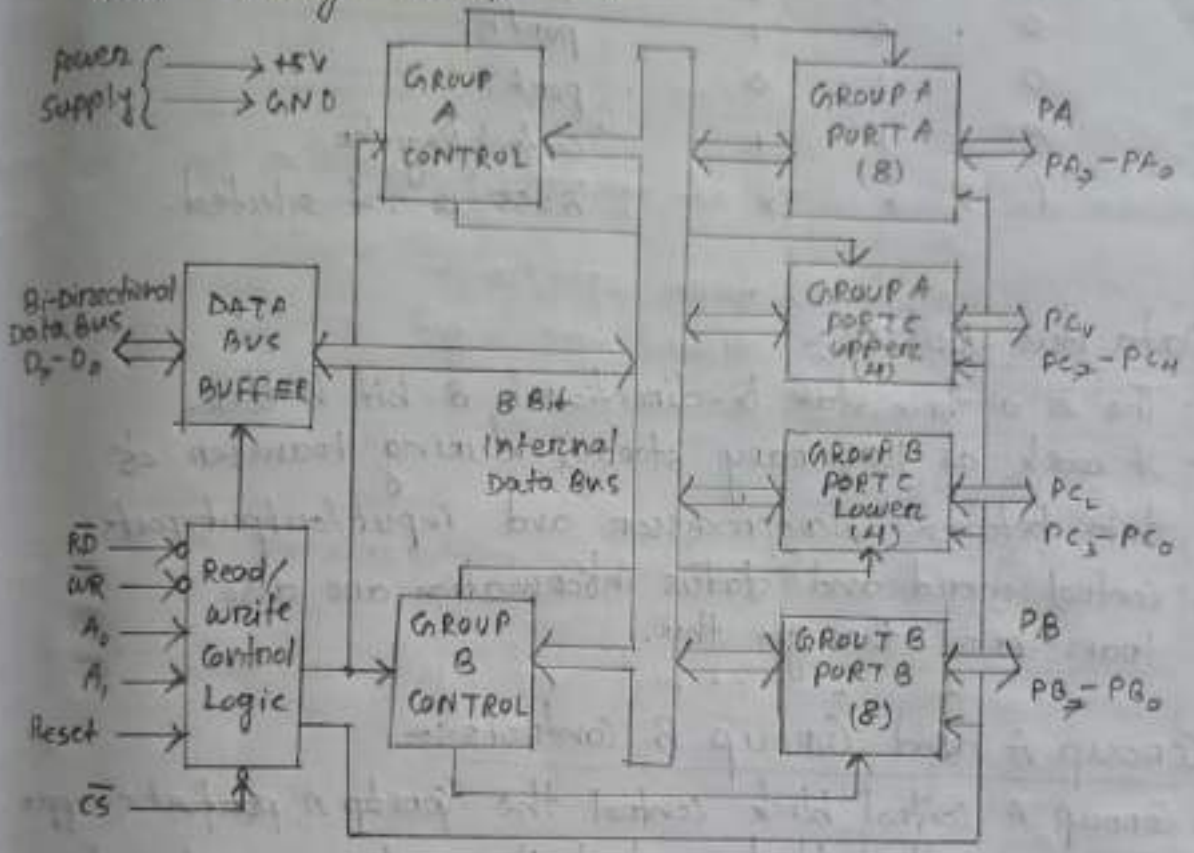
44

	Memory Mapped I/O	I/O Mapped I/O
Advantages	1. IO/$\overline{M}$ signal is not required as I/O & memory are considered same. 2. Arithmatic & logic operation can be performed directly on I/O data. 3. Same Instructions can be used for data transfer	1. Maximum memory capacity is used 2. Interfacing of I/O & memory is less complex
Disadvantages	1. Interfacing of I/O & memory is complex 2. Memory Capacity is devided between memory & I/O	1. Separate Instruction required for I/O 2. Arithmatic & logic operation can not be performed directly on I/O data 3. 4 control signals are required

(45)

Block diagram of the Programmable Peripheral Interface (Intel 8255)

- This is a 40 pins IC used as interface between microprocessor and peripheral device (input or output device)
- This IC can be programmed as input interface or an output interface.
- To program the IC; user writes control word into this IC 8255 through microprocessor.



Read write control logic :-

$\overline{RD}$:- This is an active low signal. When signal is low microprocessor reads data from selected input/output port of 8255.

$\overline{WR}$:- This is an active low signal. When signal is low microprocessor writes data or control word into I/O port or control register.

RESET :- This is an active high signal. When signal is high control register become clear or zero.

(46)

$\overline{CS}$:- This is active low signal. When signal is low 8255 IC is selected.

A_0, A_1 :- A_0 & A_1 are 2 LSB address line from microprocessor. Depending of signal at A_0 & A_1 the ports are selected. or control register is selected.

$\overline{CS}$	A_1	A_0	selected
0	0	0	port A
0	0	1	port B
0	1	0	port C
0	1	1	Control Register
1	x	x	8255 is not selected.

Data Bus Buffer :-

- This is a three state bi-directional 8-bit buffer.
- It work as temporary storage during transfer of data between microprocessor and input/output ports.
- Control word and status information are also transferred through this.

Group A and Group B controls :-

- Group A control block control the port A & port C upper
- Group B control block control the port B & port C lower.
- CPU outputs a control word to 8255 for programming the ports.
- Each of the control block (Group A and Group B) accepts "commands" from Read/Write control logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

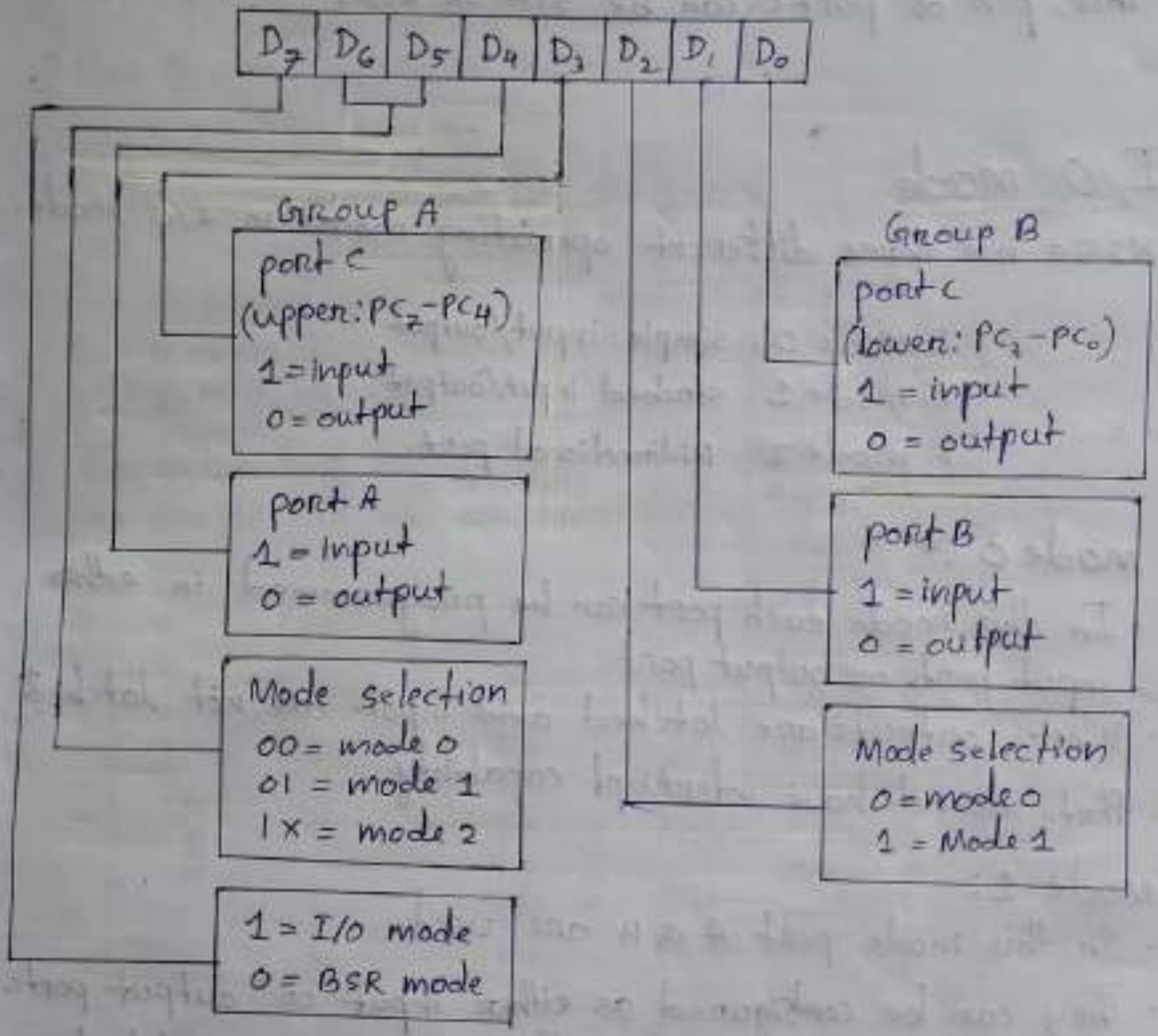
port A, B and C :-

- The 8255 contains three 8 bit ports (A, B and C)
- The port C can be divided into two 4-bit ports i.e port C upper (PC_7 PC_6 PC_5 PC_4), port C lower (PC_3, PC_2, PC_1, PC_0)

(47) Each port contains one 8-bit data output buffer/latch & one 8-bit data input buffer/latch.

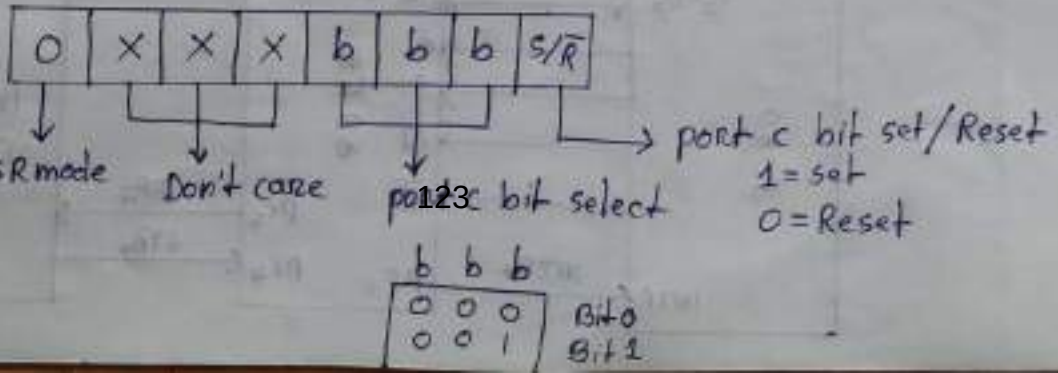
Control word

- The contents of the control register are called the control word that specifies the input/output functions of each port.



BSR Mode

- This mode is for port C only.



b	b	b	
0	1	0	Bit 2
0	1	1	Bit 3
1	0	0	Bit 4
1	0	1	Bit 5
1	1	0	Bit 6
1	1	1	Bit 7

- The pin of port C can be set or reset in this mode.

I/O mode

- 8255 has three different operating modes in I/O mode.

1. Mode 0, simple input/output
2. mode 1, strobed input/output
3. mode 2, Bidirectional port.

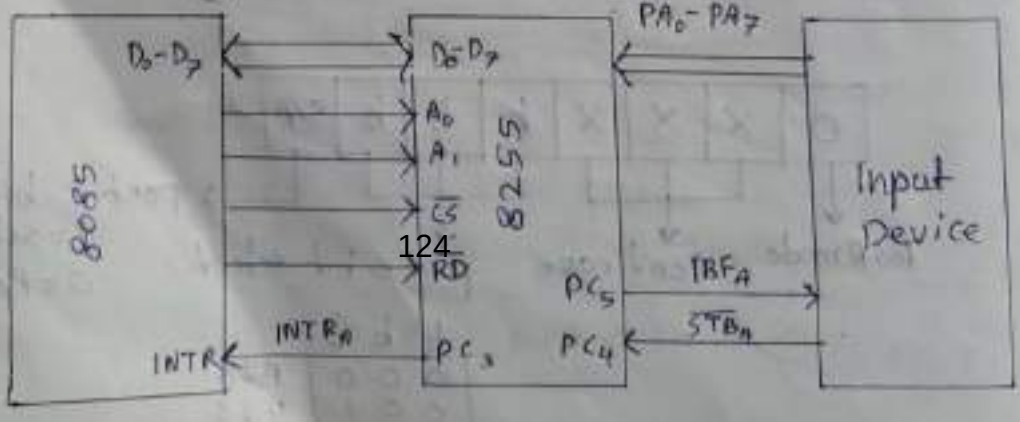
Mode 0:

- In this mode each port can be programmed in either input port or output port.
- Where outputs are latched and inputs are not latched
- Ports do not have interrupt capability.

Mode 1:

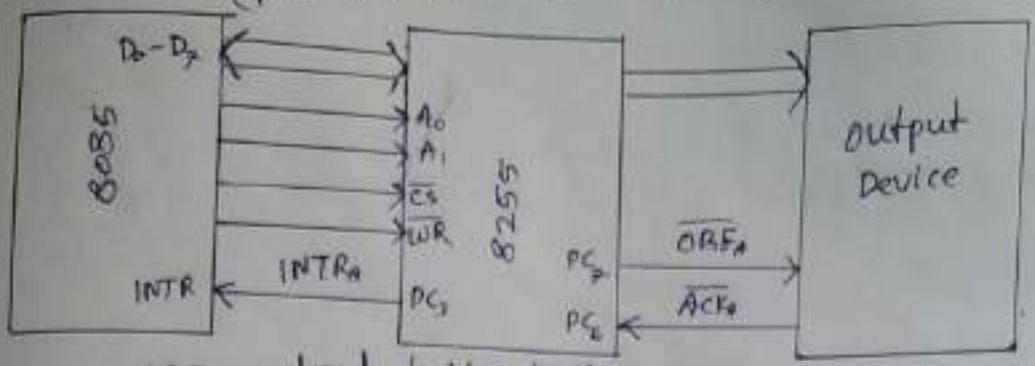
- In this mode port A & B are used.
- They can be configured as either input or output ports
- Each port uses three lines from port C as handshake signals. Other lines are work as I/O pins.
- Inputs & outputs are latched.

(port A mode 1 input)



IBF = input buffer full
STB = strobe signal

(port A : mode 1 output)



OBF = output buffer full

- Port B work same as port A.

Mode 2:

- In this mode, port A can be configured as the bidirectional port.
- Port B works as either in mode 0 or mode 1.
- Port A uses five signals from port C as handshake signals for data transfer.
- The remaining three signals from port C can be used either as simple I/O or as handshake for port B.

(port A, mode 2 input/output)