COMPUTER SYSTEM ARCHITECTURE

- **Decimal to Hexadecimal**

$(213)_{10}$ = 16 | 213 | 5 = $(D5)_{16}$
| 13 |

$(976)_{10}$ = $(3D0)_H$

- **Hexadecimal to Decimal**

$(D5)_H$ = $D \times 16^1 + 5 \times 16^0$ = $(213)_{10}$

- **Hexadecimal to Binary**

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

$(2A6)_H = (0010\ 1010\ 0110)_2$

$(9B7D)_H = (1001\ 1011\ 0111\ 1101)_2$

- **Binary to HexaDecimal**

$(1011\ 0100\ 0111\ 0111)_2 = (B477)_H$

- **Decimal to Octal**

$(467)_{10} = 8 \overline{)467}\ 3,\ \overline{)58}\ 2,\ 7 = (723)_8$

- **Octal to Hexadecimal**

$(7643)_8 = (111110100011)_2$

$(111110100011) = (FA3)_H$

$(A90D)_H = (1010\ 1001\ 0000\ 1101)_2 = (124415)_8$

Summary

- **Decimal to any Number System**

(a) Divide by the base 2 for binary, 8 for octal, & 16 for Hexa if no. is real

(b) Multiply by base 2 for binary, 8 for octal & 16 for Hexa if no. is fractional

- From any to Decimal

Multiply by powers of base

binary = 2^3 2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3}

real fractional

8 for octal & 16 for Hexa

Binary to Octal

Combination of three bits starting from LSB

Octal to Binary

Each digit into respective three bits

Binary to Hexa

Combination of four bits from LSB

Hexa to Binary

Each digit into four bits.

Octal to Hexa

First convert octal to Binary then Binary to Hexa

Hexa to Octal

Hexa → Binary → Octal

Logic Gates

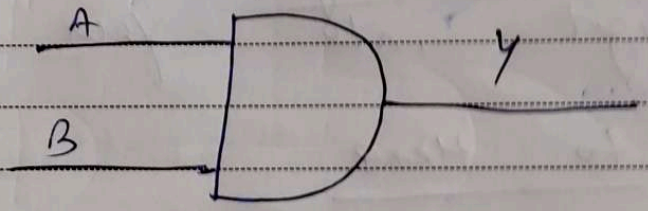
Logic gates are the basic building blocks of any digital circuit.

Types

Basic Gates	Universal Gates	Exclusive Gates
1) AND	1) NAND	1) EX-OR
2) OR	2) NOR	2) EX-NOR
3) NOT		

1) AND Gate

Symbol:



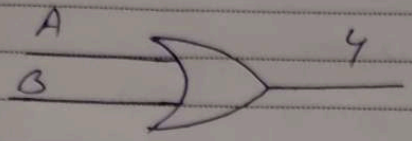
Equation : $Y = A \cdot B$

T.T

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

If any ~~output~~ input is zero than output is zero, ~~when~~ both 1 then only output 1.

2) OR Gate

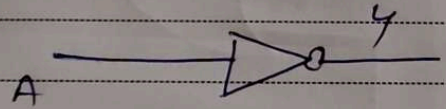


$Y = A + B$

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

If any input is 1 then output 1 else output is zero

3) NOT Gate



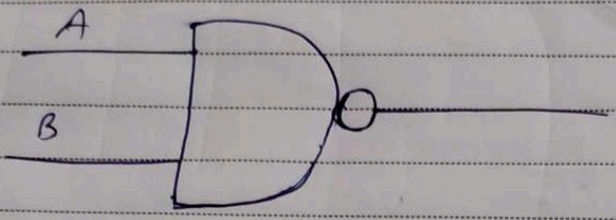
$Y = \overline{A}$

A	Y
0	1
1	0

NOT gate acts as an inverter if input is high output is one & vice versa

4) NAND Gate

NAND = NOT & AND



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

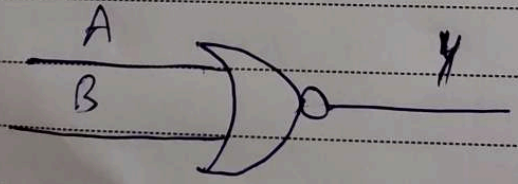
T.T =

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

If any input is low output is high otherwise low.

5) NOR Gate

NOT + OR



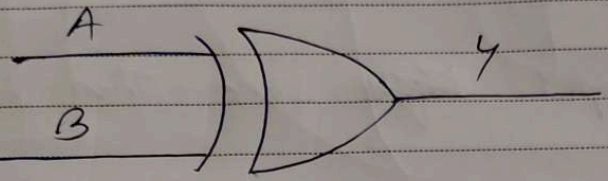
$Y = \overline{A+B} = \bar{A} \cdot \bar{B}$

TT =

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

If any input low then low. otherwise output is high.

6) EX - OR

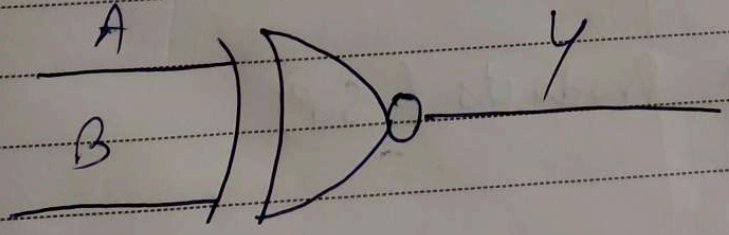


$Y = \bar{A}B + A\bar{B}$ or $A \oplus B$

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

If both input same output is low otherwise output is high when both output different.

7) EX - NOR

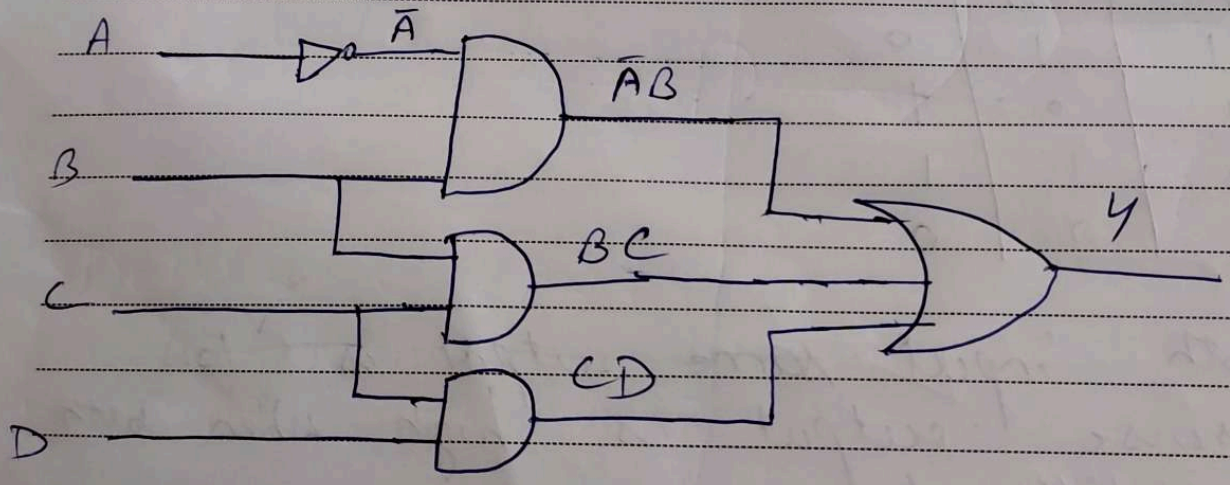


$$Y = \overline{A \oplus B} = A \odot B = \bar{A}\bar{B} + AB$$

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

When both input ~~Right~~ same Then High otherwise low

$$\Rightarrow Y = \bar{A}B + CD + BC$$



$$\Rightarrow ① Y = A\bar{B}C + ACD + B\bar{D}$$

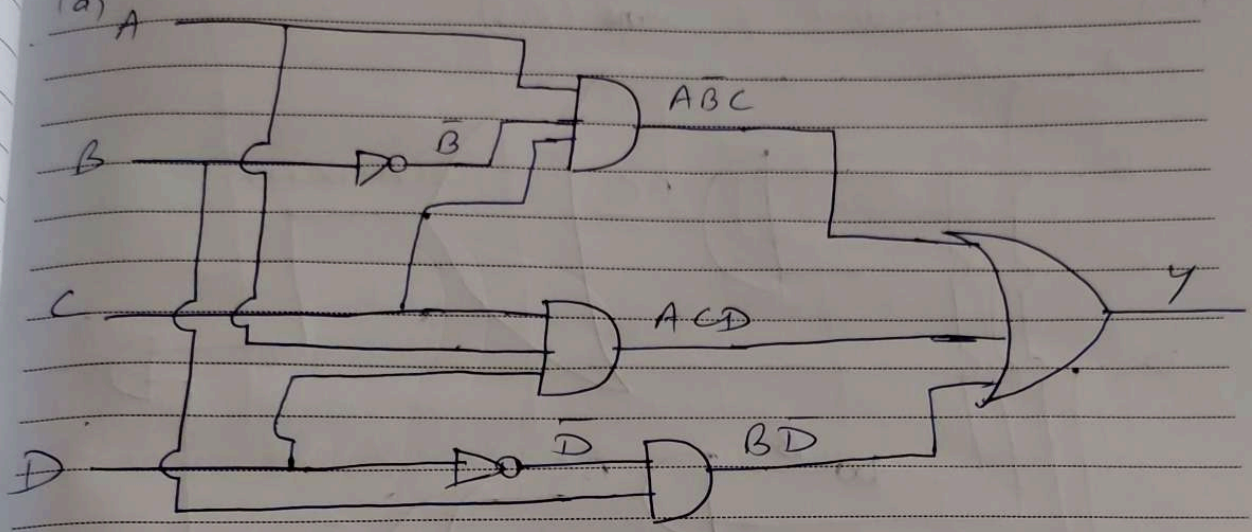
$$② Y = AC + \bar{B}CD + ABC$$

These are Sum of Products (SOP)

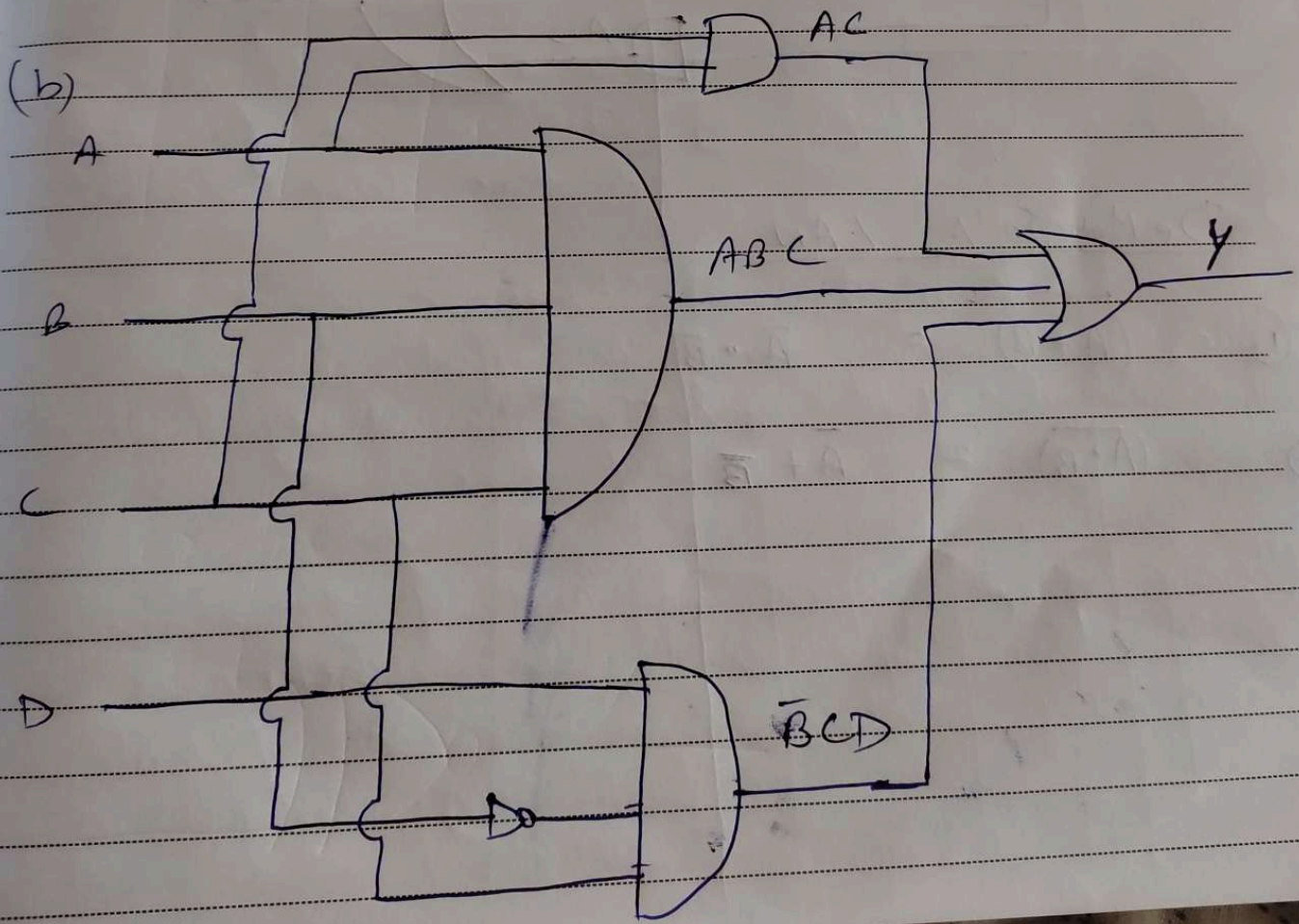
Sol:
(a) A
B
C
D
(1

Sol:

(a)

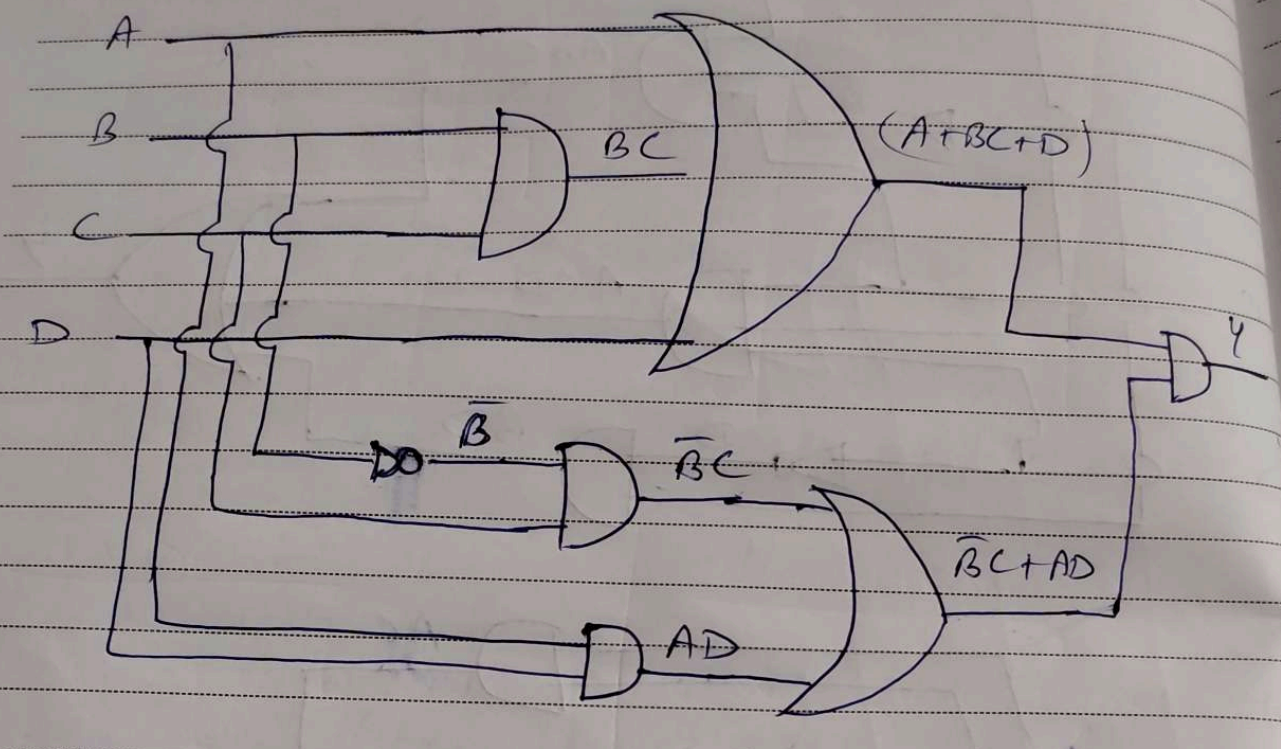


(b)



Ques: $Y = (A + BC + D)(\bar{B}C + AD)$

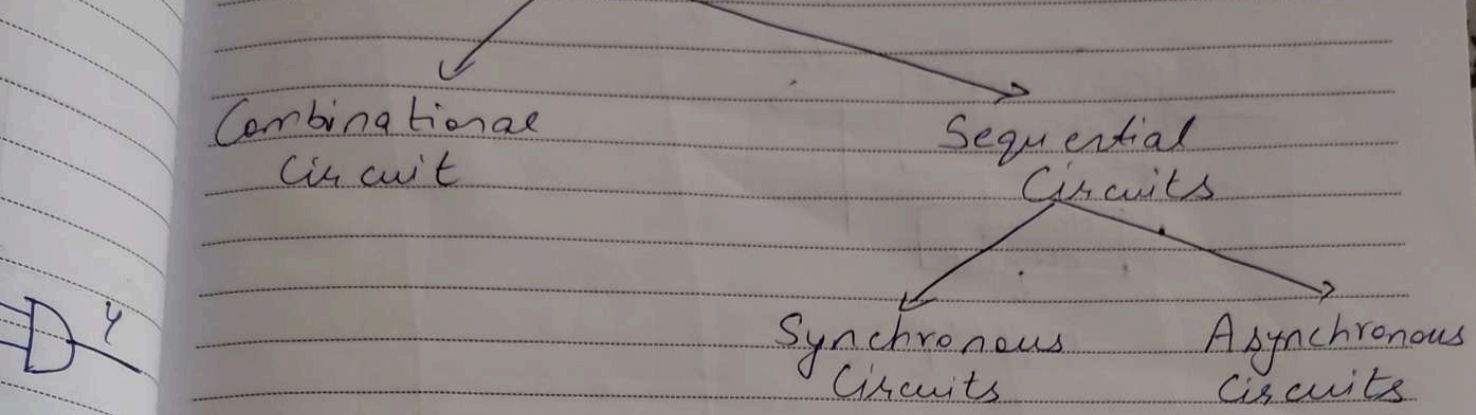
This is Product of Sums (POS)



DeMorgan's LAN

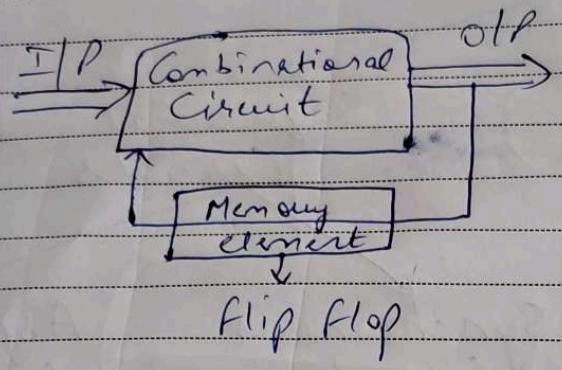
1. $(A + B) = \bar{A} \cdot \bar{B}$
2. $\overline{(A \cdot B)} = \bar{A} + \bar{B}$

Digital Circuits



Combinational	Sequential
1) In this type of circuits output variables at any instant of time are dependent only on the present input variables	In this output variables at any instant of time are dependent not only on the present input variables but also on the present state i.e. on the past history of the system
2) Memory unit is not required	Memory unit is required to store past history of input variables
3) Faster because the delay b/w the input & the output is due to the propagation delay of gates only	Slower than combinational circuits

4) Easy to design	Comparatively harder to design

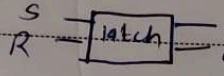


Clock: It is a continuous pulse used to synchronise all the units of the system

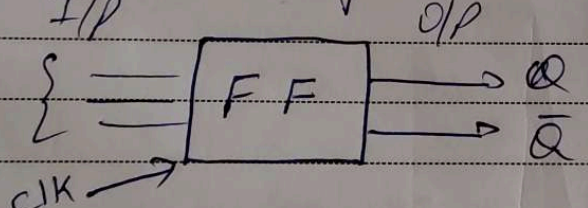
★ Synchronous	Asynchronous
1) Memory elements are clocked flip flops	Memory elements are either unclocked flip flops or time delay elements
2) The change in input signal can effect memory elements upon activation of clock signal	Change in input signal can effect memory elements at any instant of time
3) Maximum operating speed of the clock depends on time delays involved	Because of the absence of clock asynchronous circuits can operate faster than synchronous circuits

4) Easier to design | More difficult to design

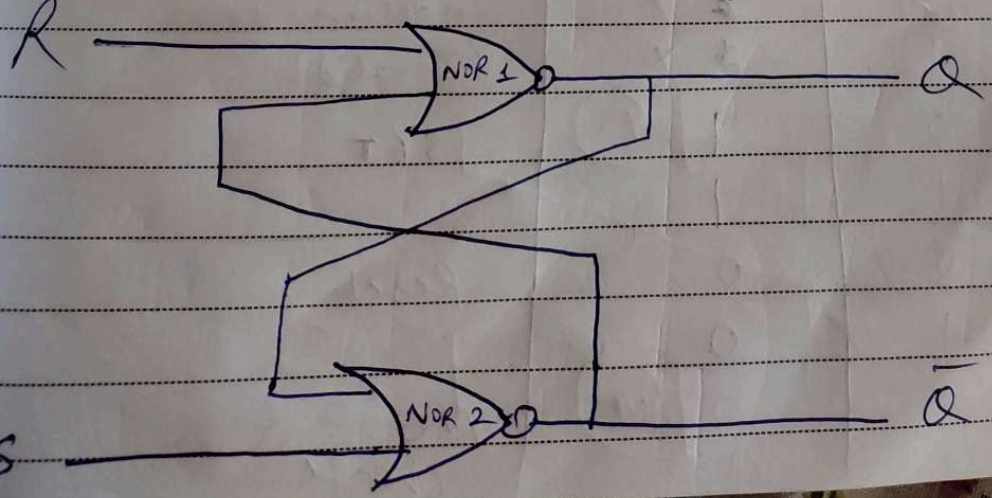
Latch: It refers to non clocked flip flops as these flip flops latch onto 1 or 0 immediately after receiving the input pulse SET or RESET. Latch doesn't depend on clock for their operation



Flip Flop: It is a bi-stable device having two stable states, it can remain in either of the state indefinetly. It's state can be changed by applying a proper triggering pulse. It is also called a binary or 1-bit memory



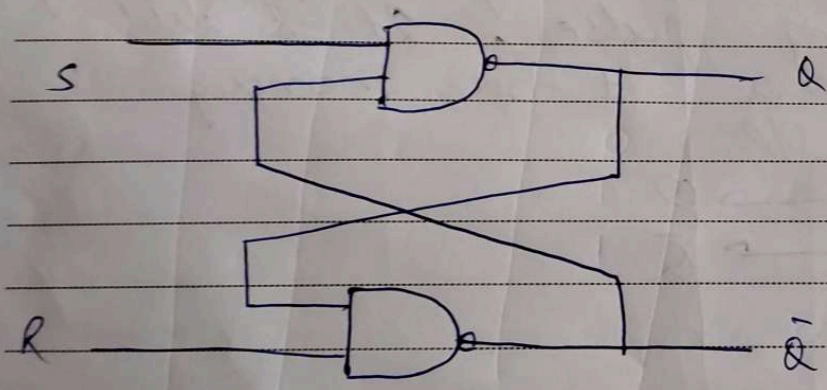
SR Latch



T.T

S	R	Present State(Q_n)	Next State(Q_{n+1})	Status
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	SET
1	0	1	1	
1	1	0	X	Indeterminate (Invalid)
1	1	1	X	

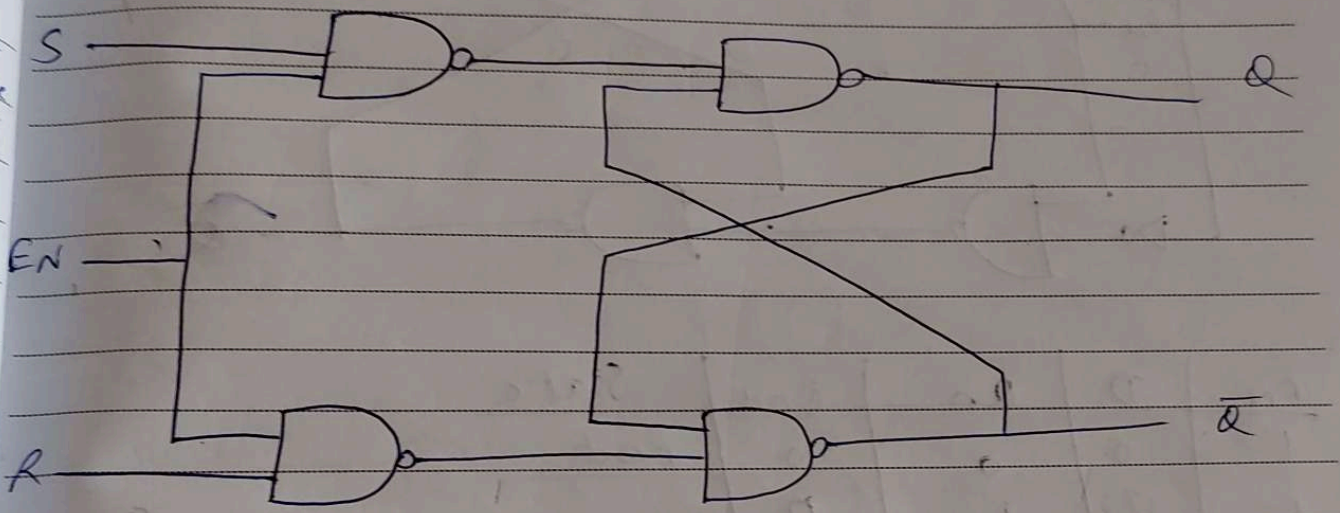
* SR Latch using NAND Gate



S	R	Q_n	Q_{n+1}	States
0	0	0	X	Indeterminate (Invalid)
0	0	1	X	
0	1	0	1	SET
0	1	1	1	
1	0	0	0	Reset
1	0	1	0	
1	1	0	0	NO change
1	1	1	1	

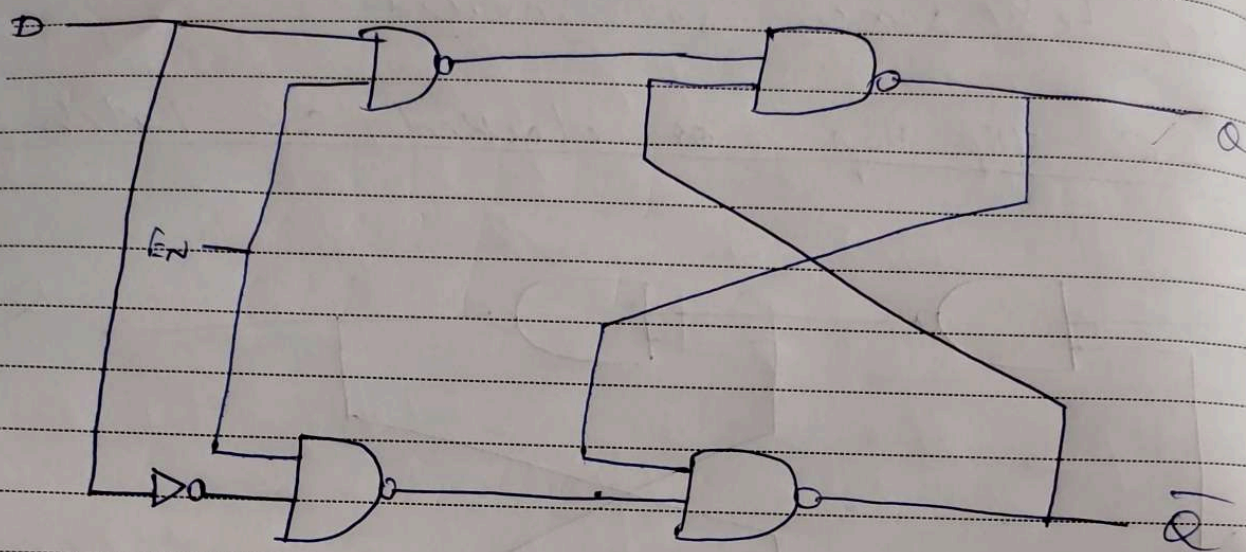
Present state will be ignored as output will always be 1 as when one input is zero output of nand is 1 & Q & $\bar{Q}$ cannot be same so invalid

SR Flip flop or clocked SR Latch



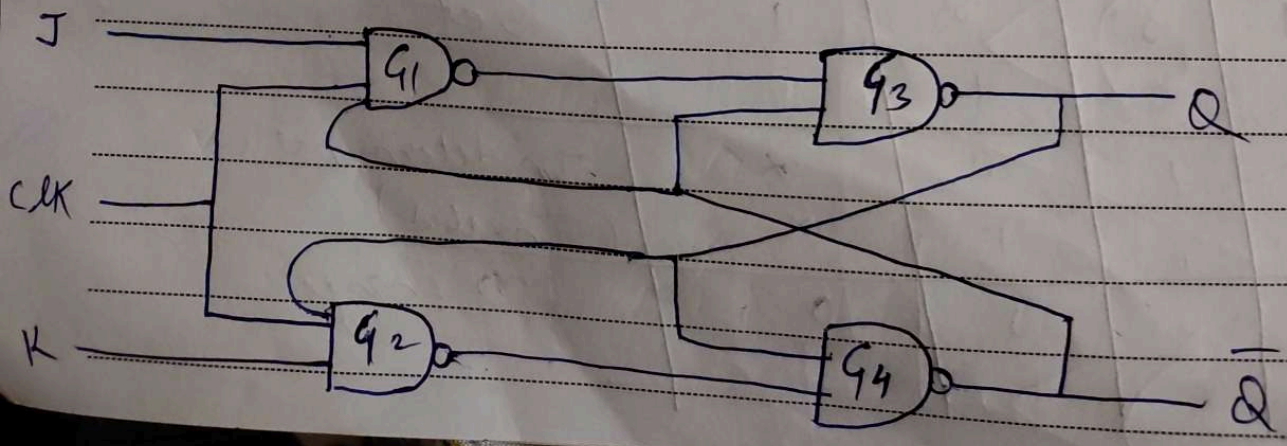
EN	S	R	Q_n	Q_{n+1}	State
1	0	0	0	0	No change
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	x	Indeterminate (Invalid)
1	1	1	1	x	
0	x	x	0	0	No change
0	x	x	1	1	

D Flip Flop



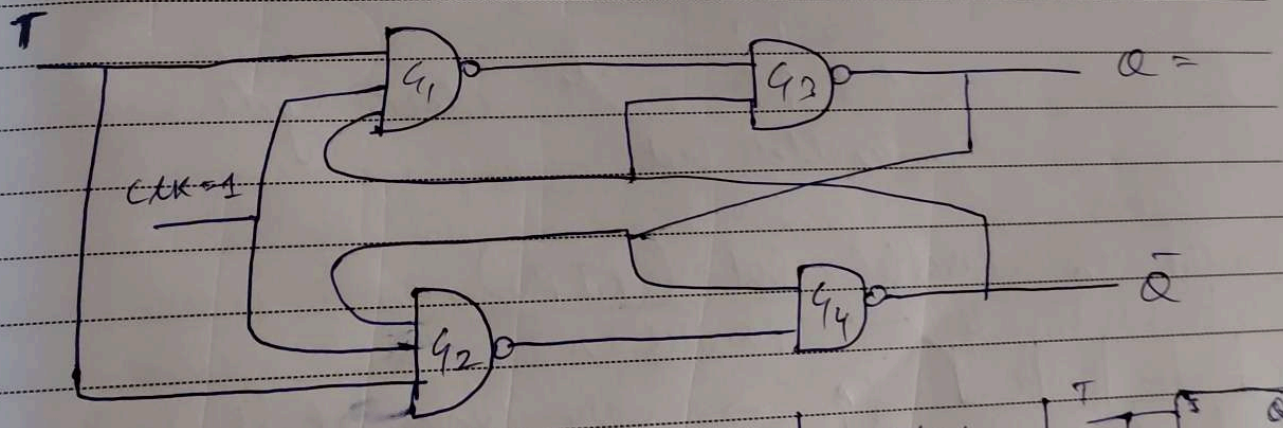
EN	D	Q_n	Q_{n+1}	State
1	0	0	0	Reset
1	0	1	0	
1	1	0	1	Set
1	1	1	1	
0	x	0	0	No change
0	x	1	1	

J-K Flip Flop

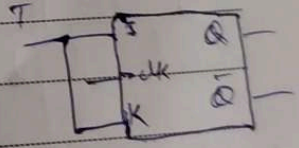


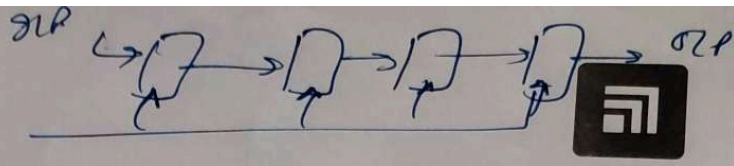
Clk	J	K	Q_n	Q_{n+1}	State
↑	0	0	0	0	No change
↑	0	0	1	1	
↑	0	1	0	0	Reset
↑	0	1	1	0	
↑	1	0	0	1	Set
↑	1	0	1	1	
↑	1	1	0	1	Toggle
↑	1	1	1	0	
0	X	X	0	0	No change
0	X	X	1	1	

★ T - Flip Flop



clk	T	Q_n	Q_{n+1}	state
↑	0	0	0	No change
↑	0	1	1	
↑	1	0	1	Toggle
↑	1	1	0	
0	X	0	0	No change
0	X	1	1	

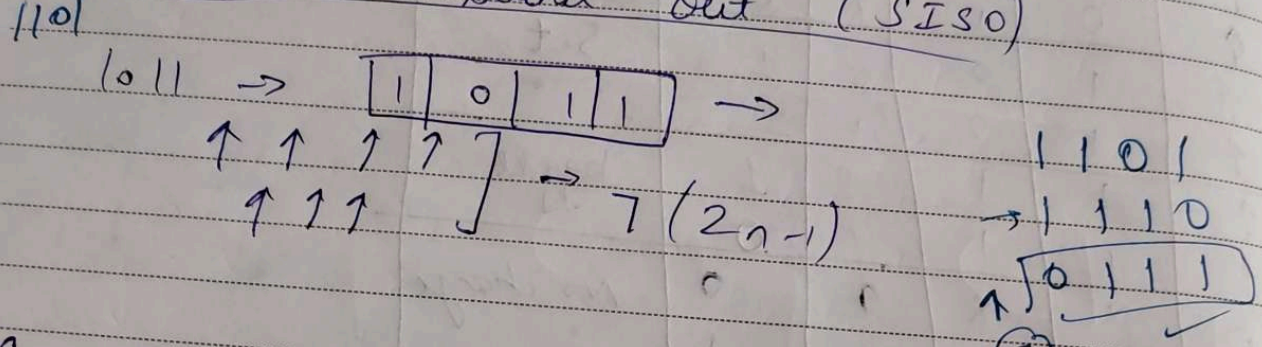




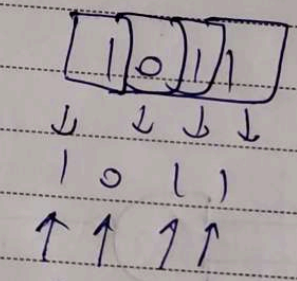
Combination of flip flops ⇒ Registers

Techniques:

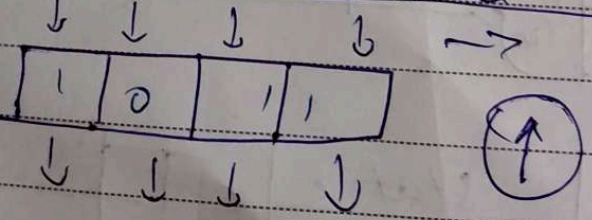
(a) Serial in Serial Out (SISO)



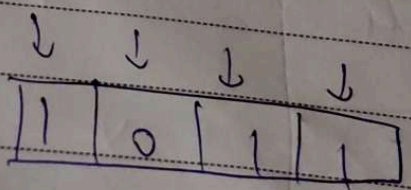
(b) Serial In Parallel Out (SIPO)



(c) Parallel In Parallel Out (PIPO)



(d) Parallel In Serial Out (PISO)



Digital Computers

Computer System

Hardware: Electronic & electromechanical devices comprising the physical entity of the system

Software: Instructions & data that computer/processor manipulates to perform data processing task.

Program: Sequence of instructions written to perform a specific task.

Database: Data that are to be manipulated by the program.

Program

Software Program (system program OS compiles)	Application Program (Written by the user)
↓	Are written by the user for the purpose of solving a particular problem
Make more effective use of computer & provide an environment for the user to work efficiently.	eg: all high level languages.
↓	
These are included in system software package for eg: OS, compiler	

* Computer Architecture

Instruction set, instruction format architecture is concerned with the specific instruction set & instruction format supported by the processor. It includes instruction set, instruction format, specific registers, and their roles, techniques for accessing data stored in a memory & the way in which I/O operations are performed. Thus computer architecture is the external view of a comp i.e. essential to be understood by the user who is to program a computer using machine or assembly language.

* Computer Organisation: The way hardware components operate & how they are connected together to form a computer system. It helps to understand the internal operations that are carried out by a computer while the program is being executed. It gives the internal view of the computer.

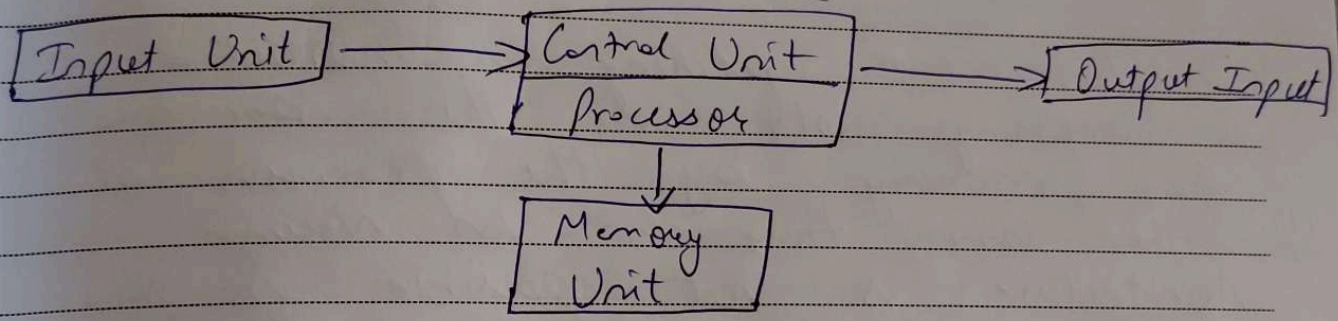
↓ Application Program ↑

↓ System Program ↑

Architecture / Organisation — Interface b/w the hardware & software.

* Computer Design: Concerned with the hardware design of the computer once the architecture & other specs of the computer are formulated then it is the task of the computer designer to develop the hardware of the system. What hardware to be used & how they are interconnected. Also known as computer implementation

Von Neumann Architecture



In early days of computing one instruction at a time of a program was executed with an operator setting up each instruction & also initiating the execution of each instruction (by flipping some switches on a switch board that will set up the required data & control part). Then in 1945 John Von Neumann proposed that instructions can be encoded & stored in the memory just like data. During the execution of program the stored instruction can be fetched from memory & then decoded to set up the necessary data path & generate control signal.

Advantages: (i) Eliminated the need of operator
(ii) Execution became faster as human intervention is removed.

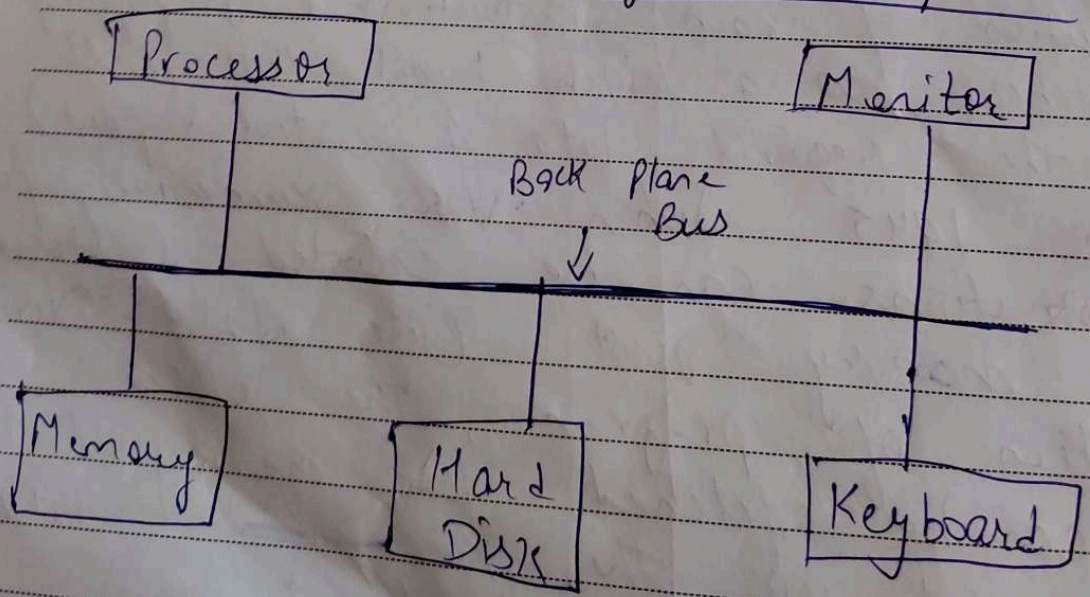
Bottle Neck of Von Neuman Architecture

For faster computation there is a bottle neck in Von Neuman Architecture

In Von Neuman Architecture a single connection exists b/w memory and processor. At any time only one memory access can occur (instruction can be fixed or data item can be accessed)

Therefore both fetching an instruction & accessing data over the same bus from memory by the processor at the same time in Von Neuman Architecture is not possible. So this is bottle neck in achieving high performance competition.

Basic Organisation of A Computer



① Back Plane Bus: A single bus interconnecting all the components of a computer

② Processor: It is responsible for fetching an instruction stored in the memory & executing it.

It contains ALU for manipulating data, a number of resistors for storing data and a control unit. Control Unit generates necessary control signals for fetching & executing the signals & storing the results. ~~Also called as RAM~~

Main Memory: Also called as RAM because CPU can access any location in the memory at random & retrieve/store the binary information at that memory location.

Monitor: It is a visual display unit & it serves as the main/primary output unit.

Peripheral Devices: These are the I/O devices that are connected to the computer eg: Keyboard, printer, scanner, camera. CPU can provide output to these output devices & collect input from the input devices over back plane bus.

⇒ Back Plane Bus is partitioned further in three type of buses :- Control Bus, Address Bus & Data Bus.

- Buses used to carry control signals are called as control bus
- Buses that carry data are termed as data bus.
- Buses that carry address are termed as Address bus

Generation of Computers

1) First Generation Computers (1941 - 1956)

First Gen Computers are characterised by their use of vaccum tubes for computation & magnetic drums for storing the data

2) Second Generation Computers (1956 - 1963)

With the invention of transistor in 1947 by Schockley, Braittain & Bardeene vaccum tubes were replaced by transistors

3) Third Generation Computers (1964-1971)

Integrated Circuit (IC) created in 1958 by Jack Kilby were used. Third Gen is much more compact & multi-programming came into existence

4) Fourth Generation Computers (1971 - present)

VLSI, ULSI were used & structure became more compact & it increased power efficiency & speed.

5) Fifth Generation Computers (Present & beyond)

Based on Machine learning & Artificial Intelligence and are still in development stage

Instruction Code

An instruction code is a group of bits that instruct the computer to perform a specific operation. There are two main parts of instruction code operational code (op code) & operand.

Op Code/Operational Code: It is the group of bits that define operations such as

add, sub, multiply, shift & complinent

No. of bits required for operation code depends upon the total number of operations available in the computer.

For n bits 2^n distinct operations can be performed.

For eg: if $n=6$ then $2^6=64$ operations

When op code is decoded by the control unit the computer issues control signals to read an operand from the memory & add the operand to the processor registered

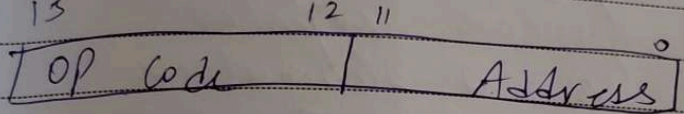
Operand: It is the data on which the operation is to be performed. Operand are specified in the resistors or by memory words, from where the data is to be retrieved or stored.
Memory words specify address of the memory

Binary code of K bits will specify 2^K resistors

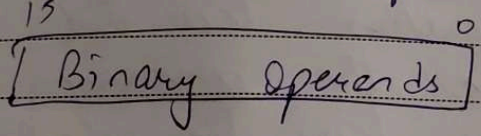
Instruction code format are to be designed

by the computer designer who specify the computer architecture.

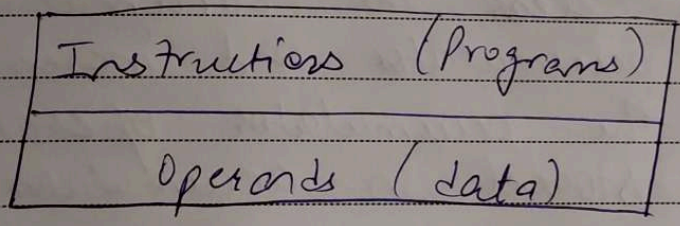
Op code also known as Macro operation and the operations that are initiated by the control unit are called as micro operations.



(a) Instruction format



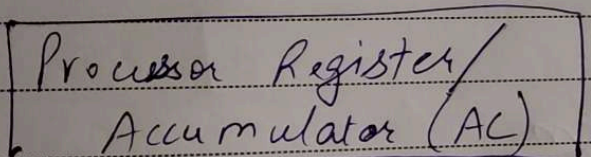
(b)



Memory 4096 X16

stored Program Organization

(c)



Instructions are stored in one section of memory & data in another From the instruction format 12 bit address line can access 4096 bits of memory. Instruction code consist of 16 bits, 12 bits for address of operand &

4 bits for op code. Control Unit reads a 16 bit instruction from the program portion of the memory, uses 12 bits to read the address of the instruction, to read a 16 bit operand from the data portion of the memory & then executes the operation specified by the op code. Computers having a single process register usually assign it as a accumulator The operation is performed with the memory operand & the content of accumulator. If the operation does not need an operand from the memory the rest of the bits can be used for some other purpose for eg:- operations like clear the accumulator, compliment the accumulator and increment or decrement the accumulator operates on the data stored in accumulator register.

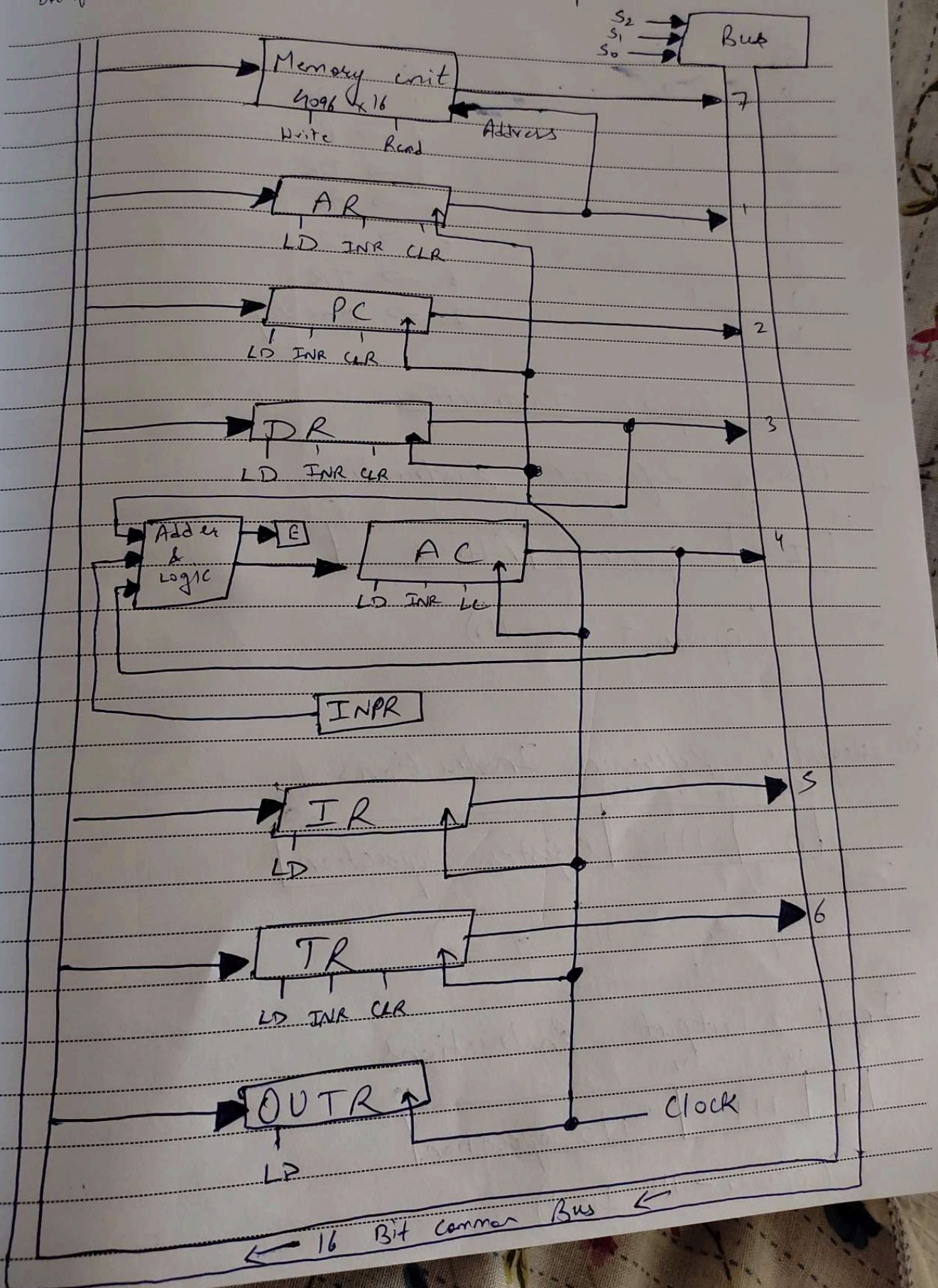
Common Bus System

It is a 16 bit system where all the transistors are connected and provided with unique code so that every register or transistor can be accessed using the bus

LD = Load
INR = increment
CLR = clear
E = extended bit of accumulator

AR = address register
PC = program counter
DR = data register
INPR = input "

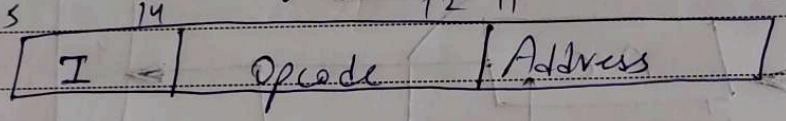
AC = accumulator
IR = instruction register
TR = temp register
OUTR = output "



S_2	S_1	S_0	
0	0	0	0
0	0	1	1 → AR
0	1	0	2 → PC
0	1	1	3 → DR
1	0	0	4 → AC
1	0	1	5 → IR
1	1	0	6 → TR
1	1	1	7 → Memory

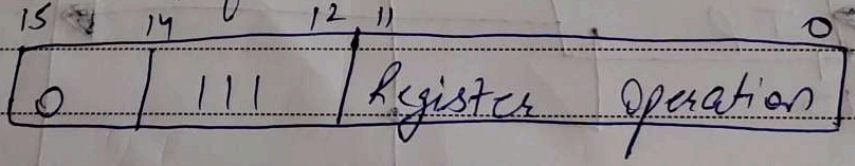
Basic Computer Instructions

(a) Memory Reference Instructions



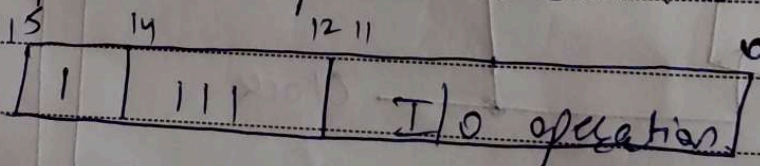
(Opcode - 000 - 110)

(b) Register Reference Instructions



(Opcode - 111, I = 0)

(c) Input - Output Instructions



(opcode - 111, I = 1)

AND

It will perform AND operation b/w the contents of memory word and accumulator

ADD

It will add the content of memory word with accumulator

LDA (Load Accumulator)

Load memory word to accumulator

STA

Store the contents of accumulator in memory

BUN (Branch unconditionally)

BSA: Branch and Save the return address

ISZ: increment & skip if zero

CLA: Clear the contents of accumulator

CLE: Extended bit of accumulator

CMA: Compliment the content of accumulator.

CME: Compliment the content of extended bit of accumulator

CIR: Circulate right the content of accumulator and E

CIL: Circulate left accumulator & E

INC: Increment of the content of Accumulator

SPA: Skip next instruction if accumulator is positive.

SNA- Skip next instruction if accumulator is -ve

SZA - Skip next instruction if accumulator is zero

SZE: Skip next instruction if extended accumulator is zero

HLT: Halt the computer

INP: input character to accumulator

OUT: output character from accumulator

SKI: Skip on input flag

SKO : Skip on output flag

ION : interrupt on

IOF = interrupt off

Types of Instructions

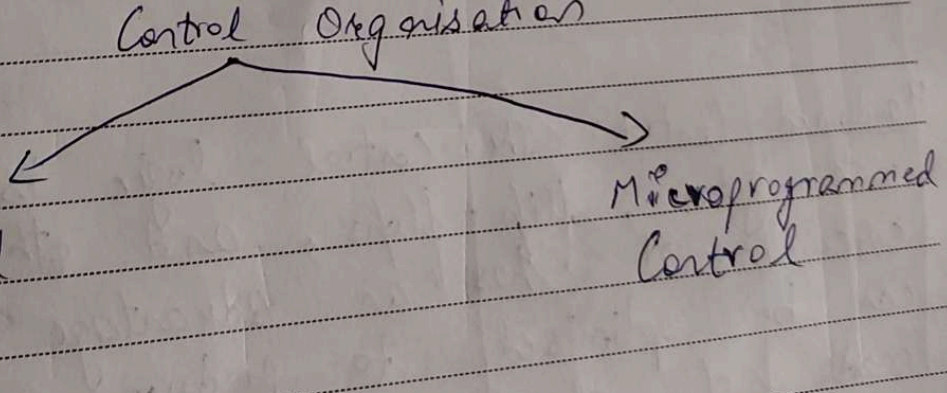
① Arithmetic, logical & shift instructions

② Instructions for moving information between memory & processor register

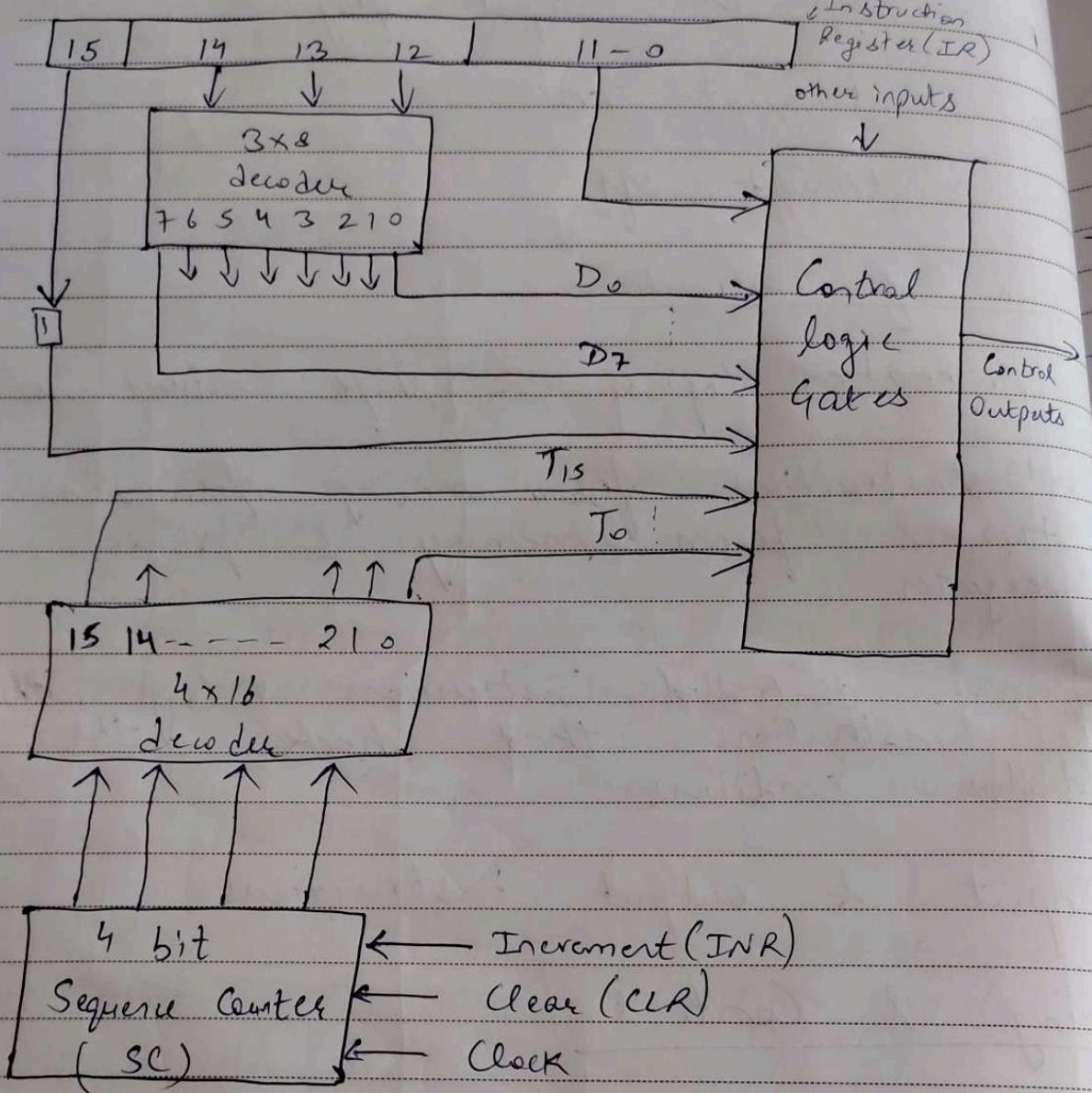
③ Program controlled instructions along with the instructions that check the status conditions

④ Input & output instructions

Timing & Control



Control Unit of Basic Computer



Hardwired Control: The control logic is implemented with gates, flip-flops, and other digital circuit. It has an advantage that it can be optimised to produce a fast mode of operation, any change or

modification requires change in the hardware (wiring among various components)

Microprogrammed Control: In microprogrammed control the control information is stored in a controlled memory. The control memory is programmed to initiate the required sequence of micro-operations. Any change or modification can be done by updating the microprogram in control memory

* Block Diagram of Control Unit Consists of two decoders, one sequence counter & number of control logic Gates.

IR: An instruction read from memory is placed in instruction Register

15th bit → Mode (I)
14 – 12 → op code
0 – 11 → Address

12 - 14 bit i.e. op code line or op code bits are connected to 3x8 decoder which will generate D0 to D7 outputs / decimal equivalent of the binary value of the corresponding operation code

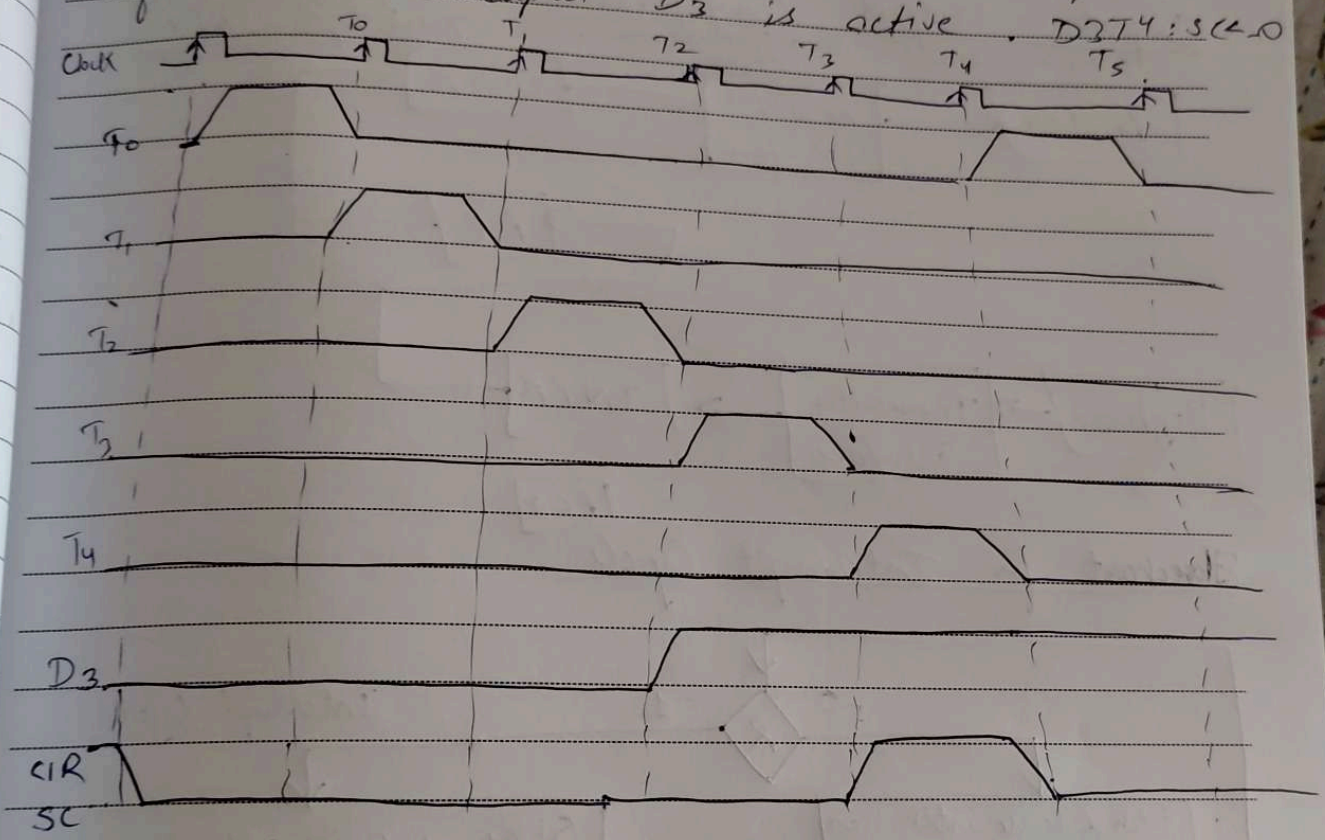
15th bit i.e. I bit i.e. Mode is transferred to the flip-flops designated by the symbol I

Address line 0-11 applied to control logic gates

4 bit Sequence Counter: It will count from 0 to 15 the output of the counter are decoded into 16 timing signals T0 through T15. Sequence counter can be incremented and cleared synchronously Most of the time SC is incremented to provide sequence of timing signals out of 4x16 decoder, Once in a while the counter is cleared to zero causing the next active timing ~~cycle~~ signal to be T0

Eg: Consider a case sequence counter is incremented to provide timing signals T0, T1, T2, T3 and T4 in sequence, At time T4 SC is cleared to zero if decoded output D3 is Active. So

As an example SC is incremented to provide signals to T_0, T_1, T_2, T_3, T_4 in sequence. At time T_4, SC is cleared to 0 if decoder output D_3 is active. $D_3T_4 : SC \leftarrow 0$



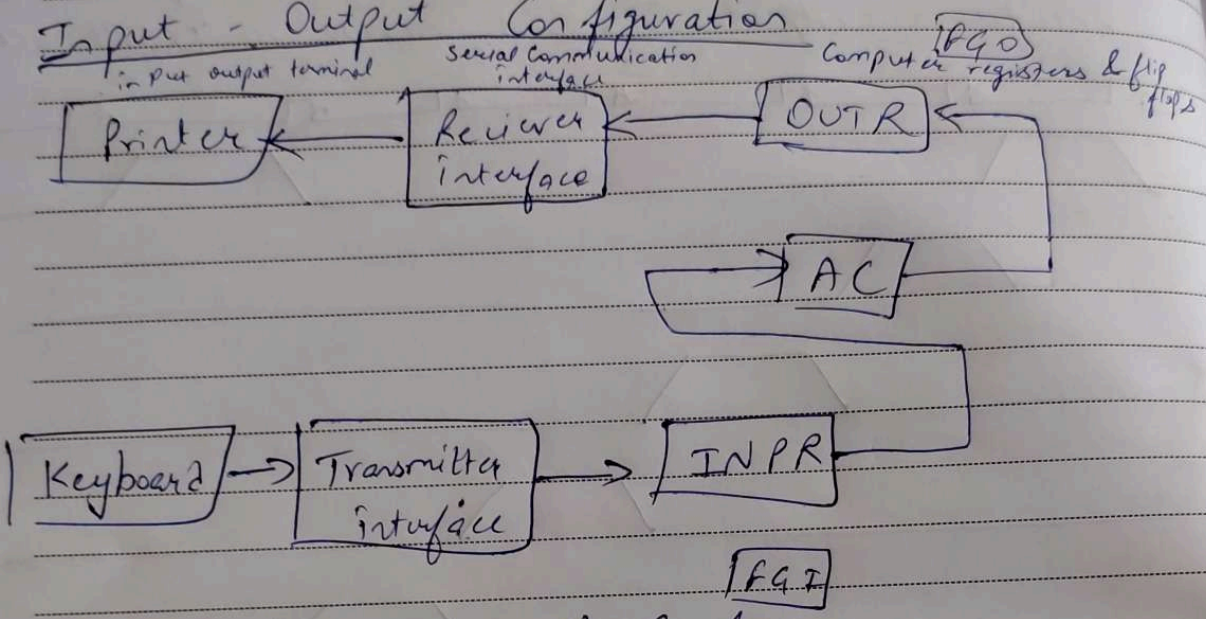
Instruction Cycle

Each instruction cycle consists of following phases

1. Fetch an instruction from memory
2. Decode the instruction
3. Read the effective address from memory if the instruction has an indirect address
4. Execute the instruction

This process continues indefinitely unless a Halt instruction is encountered.

Input - Output Configuration



Flowchart for Interrupt Cycle