Module 2

Combination Circuits

$0 \rightarrow \bar{A}$; $1 \rightarrow A$

Half Adder

Step 1:

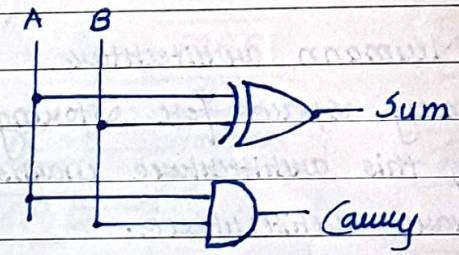
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Step 2:

Sum ⇒	$\bar{B}$	B	SOP
$\bar{A}$	0	1	$= \bar{A}B + A\bar{B}$
A	1	0	$= A \oplus B$

Carry ⇒			
	0	0	$= AB$
	0	1	

Step 3: Circuit diagram



Full Adder

Step 1:

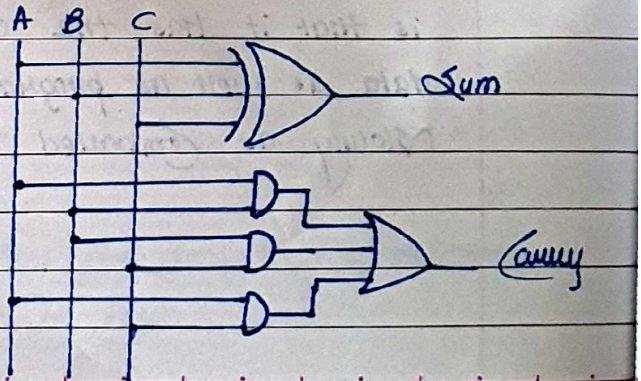
A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Step 2:

0	1	0	1	Sum = $A \oplus B \oplus C$
1	0	1	0	

0	0	1	0	Carry = $AB + AC + BC$
0	1	1	1	

Step 3:



Half Subtractor

Step 1 :

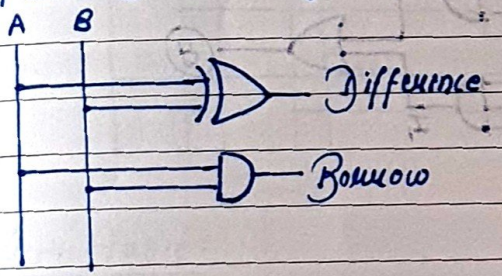
A	B	Diff.	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Step 2 :

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

Borrow = $\bar{A}B$

Step 3 : Circuit diagram



Full Subtractor

Step 1 :

A	B	C	Diff.	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

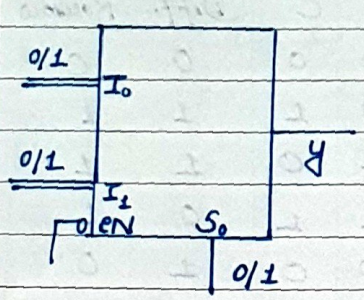
Step 2 :

Difference = $A \oplus B \oplus C$

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

Introduction to Multiplexer and Demultiplexer

Multiplexers ⇒ 2:1



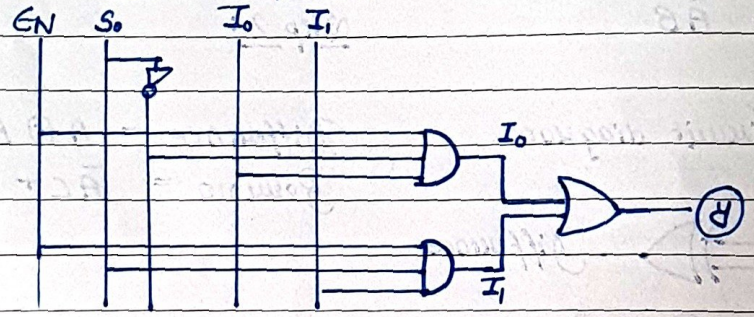
Multiple input → Single output

Output equation :

$y = I_0 \bar{S_0} + I_1 S_0$

$y = \in I_0 \bar{S_0} + \in I_0 S_0$

Circuit inside multiplexer :

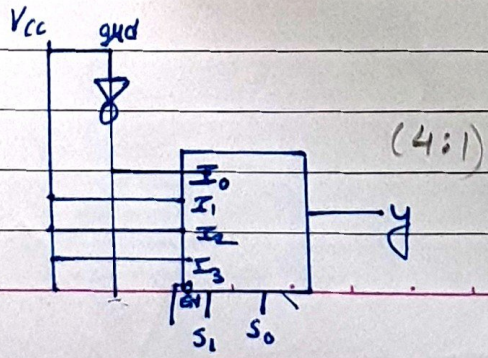


E	S_0	y
0	x	not enable
1	0	I_0
1	1	I_1

Ex. ⇒ $y = AB + \bar{A}B + A\bar{B}$

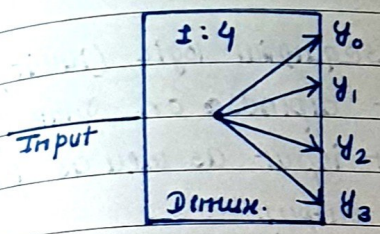
A	B	y	
0	0	0	max (0)
0	1	1	min (1)
1	0	1	
1	1	1	

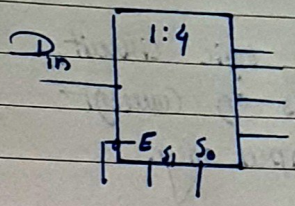
$f(AB) = \Sigma m(1,2,3)$



(4:1)

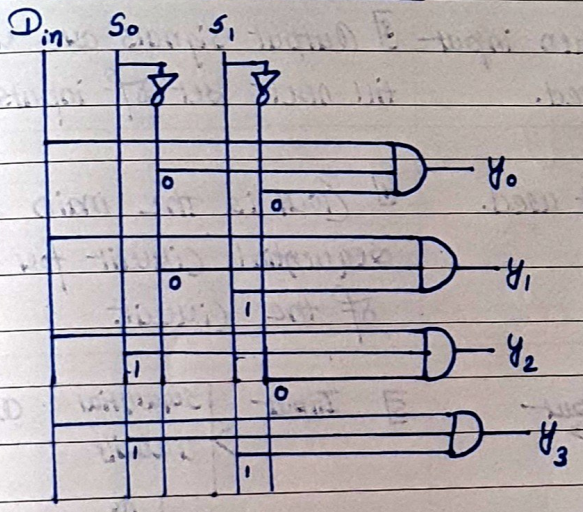
Demultiplexers





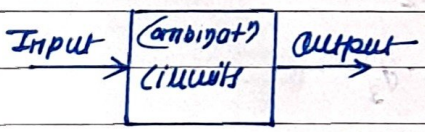
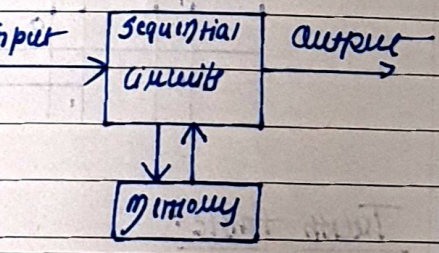
Input : Output
1 : 2
1 : 4

Circuit inside dimultiplexer :



Truth table :

D_{in}	S_1	S_0	y_0	y_1	y_2	y_3
D	0	0	D	x	x	x
D	0	1	x	D	x	x
D	1	0	x	x	D	x
D	1	1	x	x	x	D

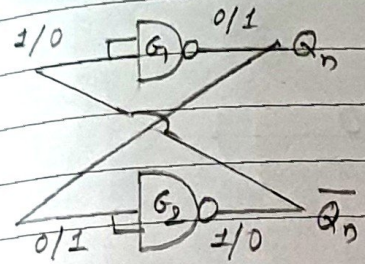
Combination Logic Circuit	Sequential Logic Circuit
1] In combination logic circuit, output depends on current state of input only.	1] In sequential logic circuit, output depends on state of the current input as well as previous state output.
2] No concept of memory is used.	2] To store previous state output, memory is required.
3] Outputs are lost when input signals are removed.	3] Output signals are retained till next set of inputs are applied.
4] Clock signals are not used.	4] Clock is the main requirement of sequential circuit for synchronisation of the circuit.
5] 	5] 
6] Ex. ⇒ Adders, multiplexer, demultiplexer	6] Ex. ⇒ Flip-flops, all registers, Counter, universal shift register, shift register

NAND Gate

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

Sequential Circuits

1-bit memory cell (latch)

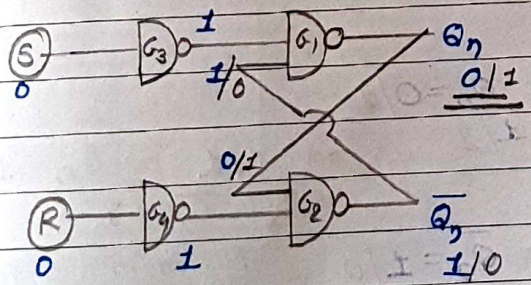


The above shown diagram is called 1-bit memory cell or latch because it is capable of storing 1 bit of information. It has 2 states Q_n and $\overline{Q_n}$ which are known as state 1 and state 0. The circuit is called as latch because it holds or latches output of previous operation. Basic flip-flops or memory elements can be obtained by universal gates NAND and NOR.

1] SR F/F

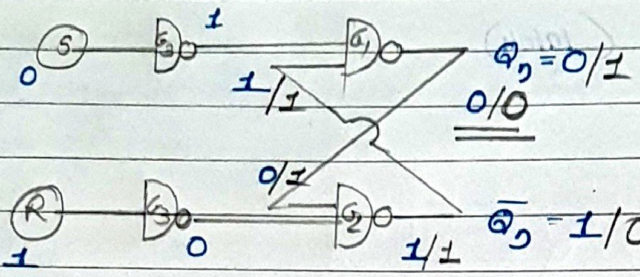
Case 1: $Q_n = 0$; $\overline{Q_n} = 1$; $S = R = 0$

$Q_n = 1$; $\overline{Q_n} = 0$



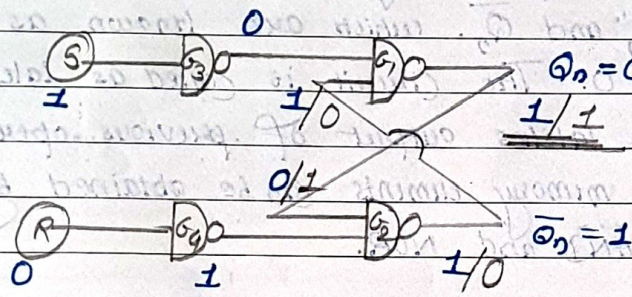
$\Rightarrow 0/1 \Rightarrow Q_n$

Case 2: $Q_n = 0$; $\overline{Q_n} = 1$; $S = 0$, $R = 1$
$Q_n = 1$; $\overline{Q_n} = 0$



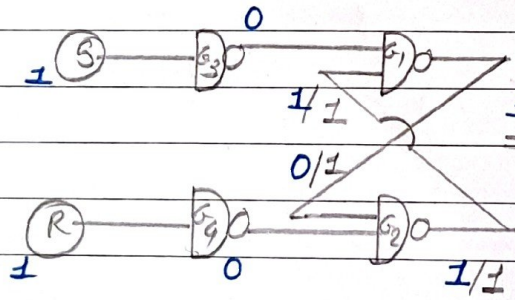
$\Rightarrow$ 0/0 $\Rightarrow$ Reset

Case 3: $Q_n = 0$; $\overline{Q_n} = 1$; $S = 1$, $R = 0$
$Q_n = 1$; $\overline{Q_n} = 0$



$\Rightarrow$ 1/1 $\Rightarrow$ Set

Case 4: $Q_n = 0$; $\overline{Q_n} = 1$; $S = 1$, $R = 1$
$Q_n = 1$; $\overline{Q_n} = 0$



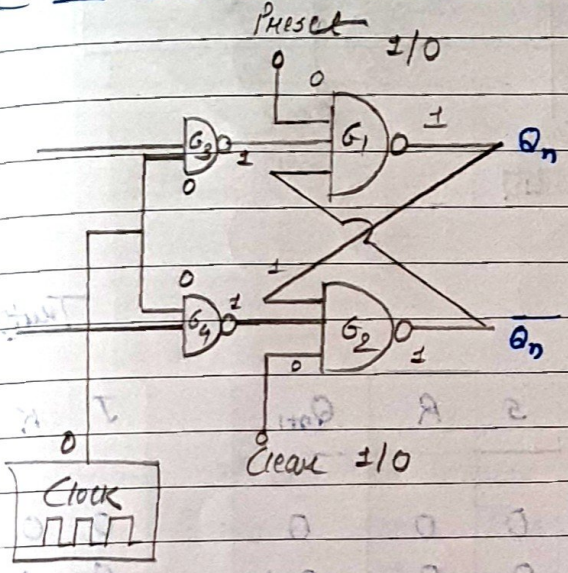
$\Rightarrow$ 1/1 $\therefore Q_n = 1$
But $\overline{Q_n} = 1$
This condition is false

Truth table of SR F/F:

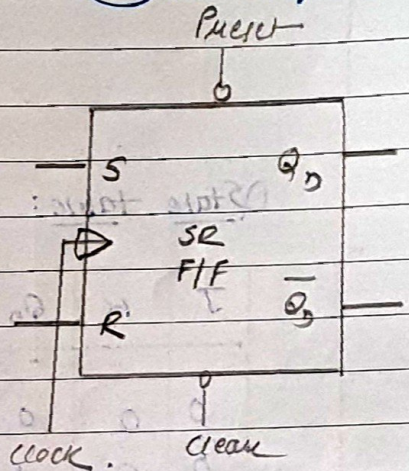
S	R	Q_{n+1}
0	0	Q_n
0	1	0 → Reset
1	0	1 → Set
1	1	Forbidden state.

In case 4, both the outputs will try to reach 1 which is conflicting the definition of 1-bit flip flop as the outputs should be complement of each. This is called as forbidden state. Thus, this is a drawback of SR flip-flop.

Clocked SR F/F with preset and clear pin



Block diagram



Truth table:

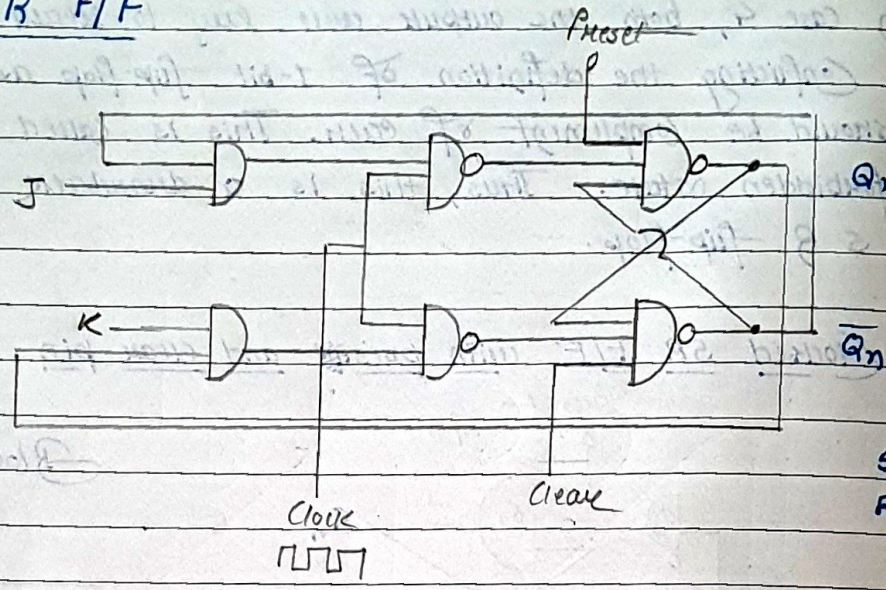
Clock	Preset	Clear	Condition	
0	X	X	F/F disable	⇒ irrespective of input, flip-flop is always disabled.
1	1	1	F/F normal operation	
1	0	1	F/F set condition	⇒ make output in set condition
1	1	0	F/F reset condition	⇒ clear previous state output

⇒ Continuous feedback change due to toggling condition at Q_n and $\overline{Q_n}$.
⇒ ∴ we cannot predict the output at end condition.
⇒ previous state output changes and thus feedback also changes.
∴ it keeps on toggling between 0 and 1.

Camlin Page 64
Date / /

Time period Δt of toggling is unpredictable ⇒ output is Q_n but at that time, we cannot predict Q_n is 0 or 1 as it is toggling

2] JK F/F



$S = J \cdot \overline{Q_n}$
$R = K \cdot Q_n$

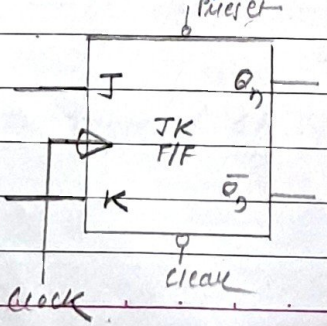
State table:

J	K	Q_n	$\overline{Q_n}$	S	R	Q_{n+1}
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	1	0	1	0	0	0
0	1	1	0	0	1	0
1	0	0	1	1	0	1
1	0	1	0	0	0	1
1	1	0	1	1	0	1
1	1	1	0	0	1	0

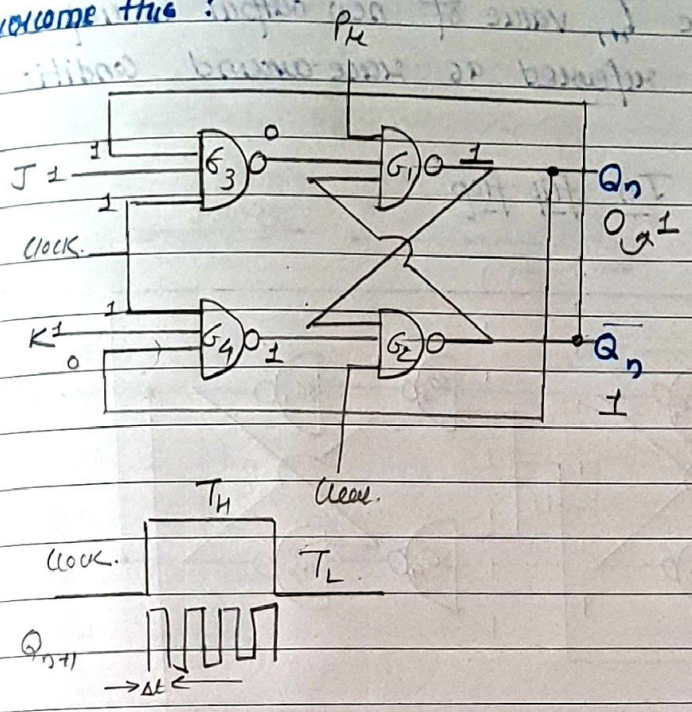
Truth table:

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

Block diagram of JK:



Que] What is race-around condition in JK flip-flop and how to overcome this ?



1] The difficulty of both the inputs S and R 1-1, SR flip-flop goes into forbidden state. This drawback, JK flip-flop has overcome using feedback connection from output to G_3 and G_4 gates.

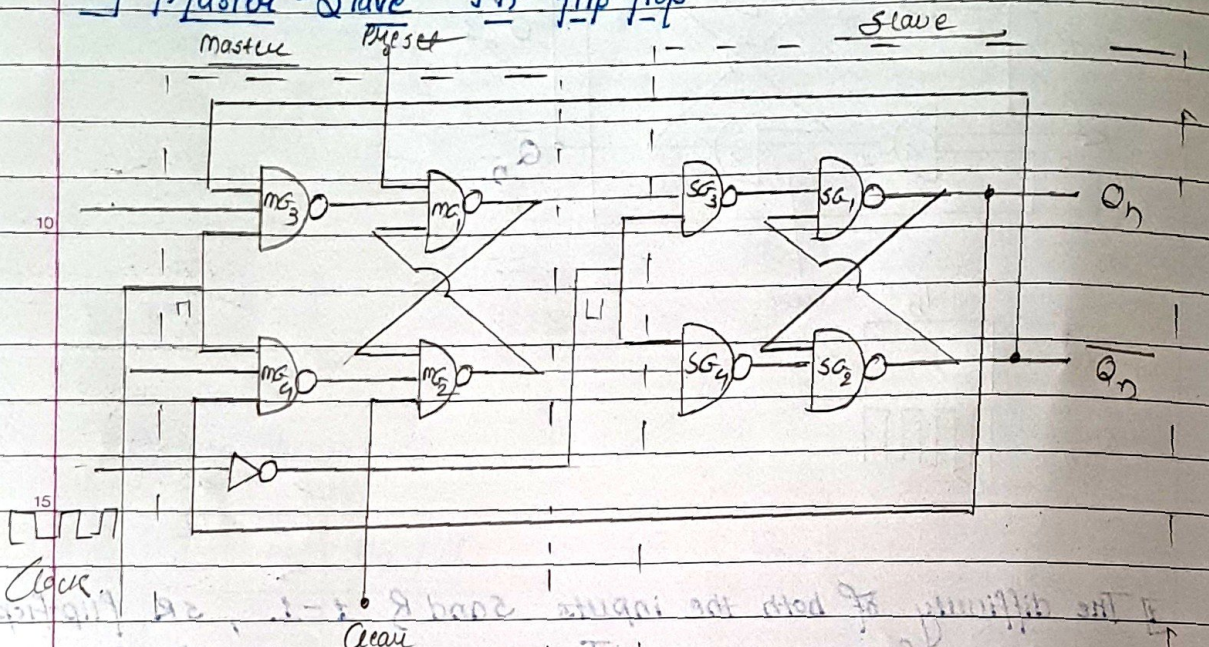
Race-around condition: [Continuously toggling condition]

1] Assume that input to G_3 and G_4 coming from the feedback do not change during the active-high clock pulse (which is not true because of the feedback connection).
Assume clock = J = K = 1 and if $Q_n = 0$; $\bar{Q}_n = 1$.
2] After a time interval [(Δt) i.e., propogation delay], to 2 NAND gates which are in series, output will change to logic 1. (Q_{n+1}).
3] After another time interval Δt, again output will change from 1 to 0. This process continues and hence, we conclude that for T_H (active high-time of clock signal) time period of

clock pulse, output will oscillate between logic 1 and logic 0.
4) At the end of the T_H, value of new output is unpredictable. This situation is referred as race-around condition.

5

3] Master-Slave JK flip-flop



20

⇒ 1] Master-Slave JK flip-flop is a cascade connection of 1 JK and 1 SR flip-flop with feedback connection from output of second flip-flop to input of first flip-flop.
2] When positive clock-pulse is applied to first flip-flop, it is inverted before applying it to second flip-flop.
3] When clock signal is 1 (active high), first flip-flop will be enabled and second one will be disabled and vice-versa.
4] Second flip-flop simply follows the output of the first flip-flop. Hence it is referred as slave and first one as master.
5] Therefore, entire configuration is known as 'Master-Slave' JK FF.

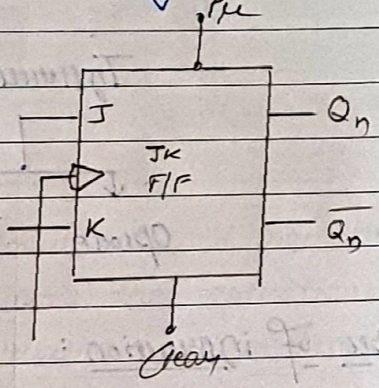
6] In this circuit, feedback input to MG_3 and MG_4 do not change during the active-high clock pulse. Hence output remains constant for one clock pulse.

7] Thus, the race-around condition does not exist.

Truth table:

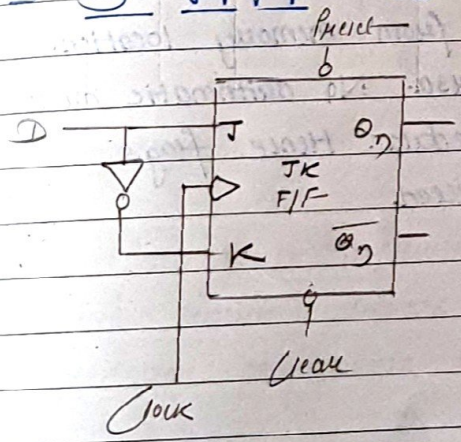
J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Block diagram:



4] D - Flip-flop (Delay / Data)

⇒ same output as D

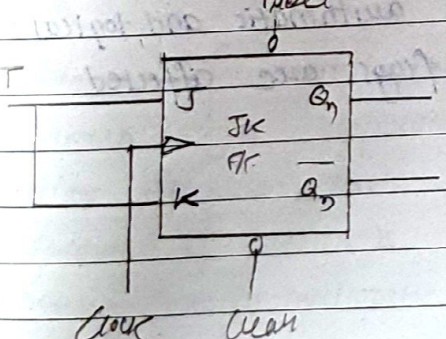


Truth table:

D	Q_{n+1}
0	0
1	1

5] T - Flip-flop (Toggle)

⇒ Previous state output.



Truth table:

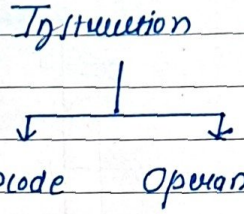
T	Q_{n+1}
0	Q_n
1	$\overline{Q_n}$

Module 3

Processor Organization and Architecture

Instructions

An instruction (word of processor) defines a task to be performed by the processor and stored in the memory.



Types of instruction :

① Data transfer instruction

These instructions transfer data from memory locations to processor (register) and vice-versa. No arithmatic or logical operations are performed on the data. Hence flags (status registers) are not affected.

ex. ⇒ MOV B

② Data processing instructions

These instructions perform all the arithmatic and logical operations on the data. Hence, flags are affected

ex ⇒ ADD B

③ Control instructions

They control the flow of the program. Hence, they are also called as Branch control instructions.
Their main job is to change the value of program counter.

⇒ JMP 2000H

Instruction format:

Every instruction has opcode. Additionally, it may have one or more operands. Opcode indicates the operations to be performed and hence, every instruction has a unique opcode. Operands indicate on which data operations are to be performed.

Opcode	Operand 1	operand ... n	Address of result	Address of next instruction

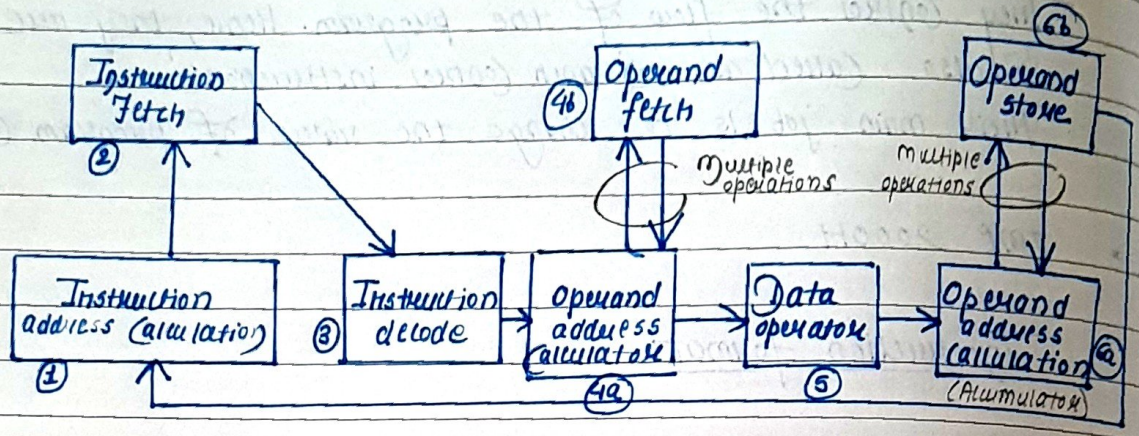
Drawback:

To fetch such instructions, we require more no. of machine cycles which increases the processor and decreases the speed and performance of the processor.

Solution:

Address of next instruction is replaced by Program Counter which is a seperate register.
After fetching each instruction, it is incremented by 1.
Address of result is replaced by Accumulator. It is used to store current arithmatic or logical operations' result.

Instruction Cycle (5m ⇒ Diagram and explanation)

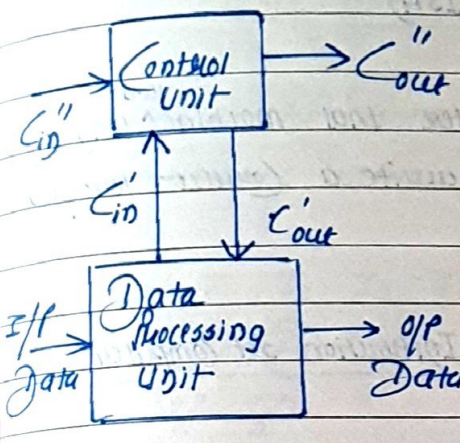


Instruction sequencing

Instruction sequencing is the method by which the instructions are selected for execution i.e., manner in which the control of the processor is transferred from one instruction to another. The simplest method of controlling the sequence in which the instructions which are being executed currently will specify the address of the next instruction. But this will increase the instruction length. & Also, the cost of the memory where instructions are stored.

Program counter contains the address of the instruction. Also, the address of (I+1) instruction can be determined by incrementing the content of PC.

Instruction interpretation



A control unit interpretes an instruction in order to determine the control signals to be issued. The control signals are transmitted from control unit to outside world via control lines. The above fig. shows the main control lines connected to typical control unit. 4 types of control lines are:

1] C'_{out} ⇒ These signals directly control the operation of the data processing unit. Main function of control unit is to generate C'_{out}.

2] C'_{in} ⇒ These signals enable data being processed to influence the control unit; allowing the data dependent decisions to be made. A frequent of C'_{in} indicates the occurence of unusual conditions such as errors. (underflow/overflow).

3] C''_{out} ⇒ These signals are transmitted to another control unit and may indicate status condition such as 'busy' or 'operation completed'.

4] C''_{in} ⇒ These signals are received from other control units. They typically indicate start signal. C''_{in} and C''_{out} primarily used to synchronize with other control units.

Tuesday

Camlin Page 72
Date 18 | 10 | 2022

Instructions can be of fixed and variable lengths.
The more restrictions we put on size & format of inst., lesser is the flexibility to the programmer. but processor design becomes simpler. This gives rise to 2 types of processor design i) RISC ii) CISC

Addressing mode ⇒ manner in which instructions are fetched.

Control Unit Design

Instruction Set Architecture (ISA)

ISA is a structure of the computer that machine (low) level language programmer must understand to write a correct program for that machine.

Note: [NOT for syllabus] Do study

	Reduce Instruction Set Computer (RISC)	Complex Instruction Set Computer (CISC)
1]	Size of the instruction is fixed.	1] Size of the instruction is variable.
2]	Format of instruction is fixed.	2] Format of inst is variable.
3]	No. of instructions is limited. It is less than 100.	3] Large no. of instructions are there, generally above 1000.
4]	Instructions are more register oriented.	4] Instructions are more memory oriented (memory-based).
5]	It will support simple addressing mode	5] It will support very advanced level addressing mode.
6]	More user registers are required. (as inst. are reg. based)	6] Less no. of user registers are required. (as mem-based).
7]	This supports Hard-wired control unit	7] Microprogrammed control unit is used.
8]	One machine cycle per instruction.	8] Complex instructions will require more machine cycles to complete the tasks.
9]	It supports higher degree of pipelining.	9] It supports lower degree of pipelining.
10]	Complexity in compiler designing.	10] Complexity in microprogram control unit designing.

① Hardwired Control Unit

a] State Table method:

The behaviour of control unit is represented in form of state table as shown:

States	I_1	I_2	I_3	I_4	I_5 ...	I_n
S_1	$S_{12}Z_{11}$	$S_{12}Z_{12}$ ⇒ o/p control signal to be generated [Cout]				
S_2	$S_{21}Z_{21}$	$S_{22}Z_{22}$				
S_3						
S_4						
S_5						
⋮	⋮	⋮	⋮	⋮	⋮	⋮
S_n						

The table shows what corresponding output should be generated when different inputs are applied at different states.
Let Cin and Cout denote input and output variables of the control unit. Each row in the state table corresponds to an internal state S_i of the control unit.
A state is determined by the information stored in the processors at that unit of time.
Each column denotes set of input signals applied to control unit.
The intersection of rows S_i and column I_j means when input I_j is applied to state S_i, we get S_{ij} and Z_{ij}. where Z_{ij} is a set of output signals to be activated (control signals) and S_{ij} indicates next state of control unit.
A circuit is constructed based on the state table.

Advantages:

1] This is the simplest method to implement hardwired control unit.

Disadvantages:

1] If the processor has large no. of states and input combinations (inst.), this method cannot be used as size of the table would become too large and circuit will become very difficult to implement.

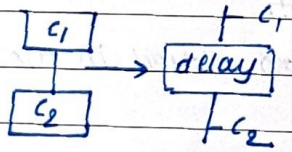
2] State table method tends to hide useful information like existence of the loops and repeated pattern. Even if two different states have same behaviour, will require seperate hardware for both.

3] Circuit design using this method tends to have random structure. and hence difficult to debug and maintain.

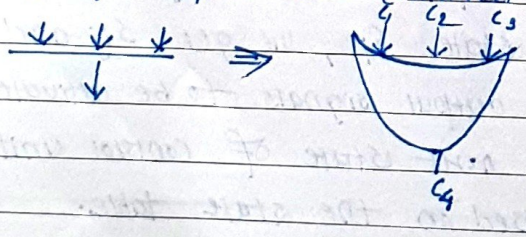
b] Delay element method:

Rules:

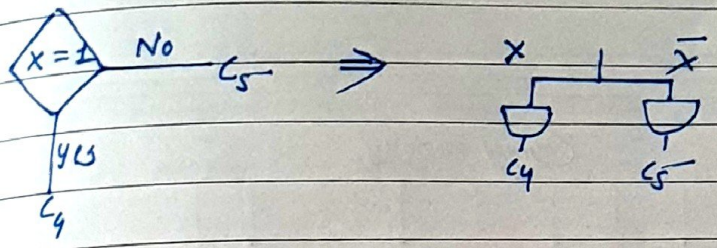
1] Between two successive steps D F/F

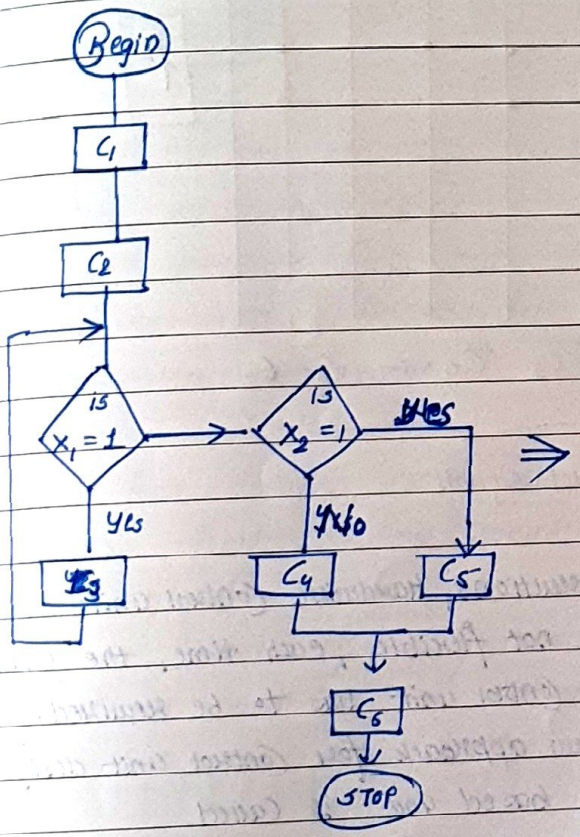


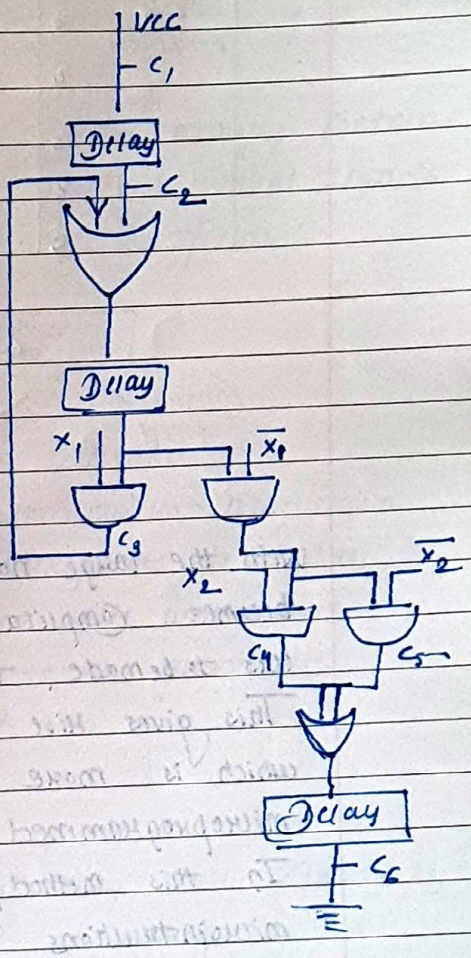
2] Multiple inputs are combined by OR gate.



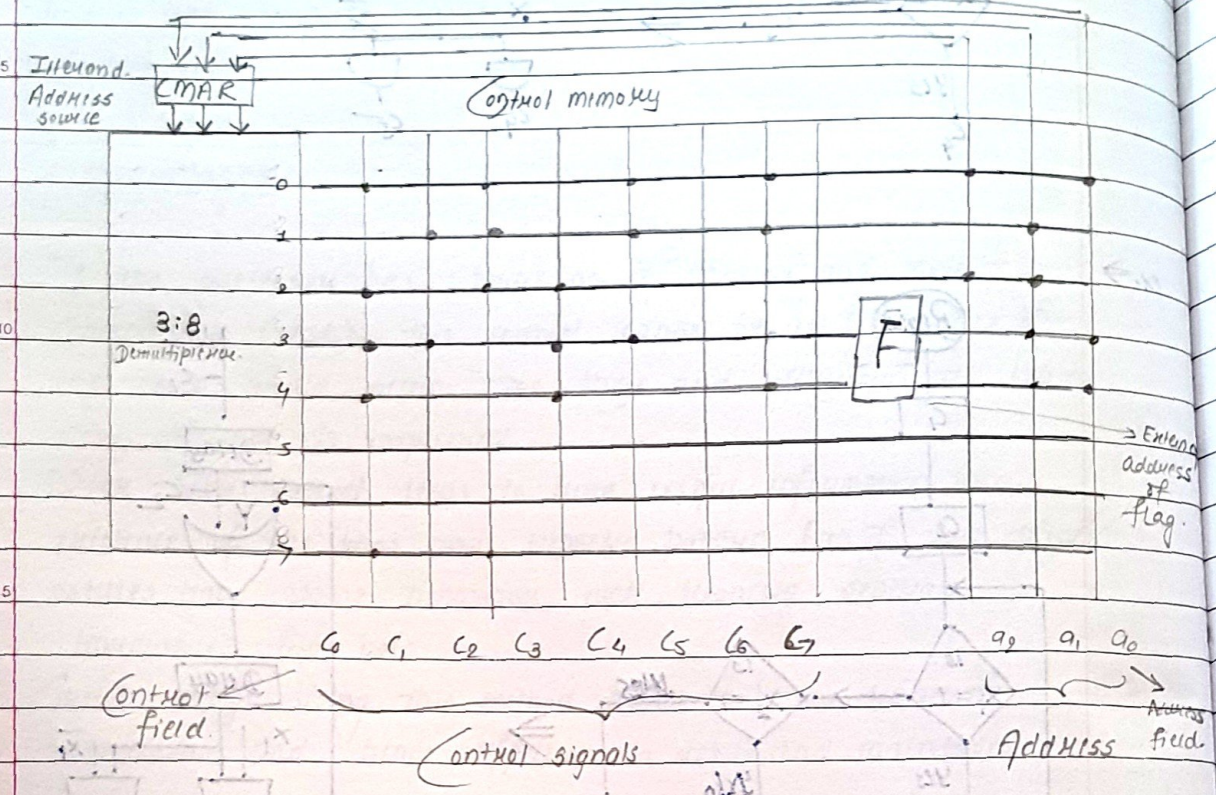
3] Conditional branch (Decision box)







② Microprogrammed Control Unit (Wilkes' design)



With the large no. of instructions, hardwire control unit becomes complicated and not flexible, each time, the change has to be made, the entire control unit has to be revised. This gives rise to a new approach for control unit design which is more software based and is called microprogrammed control unit.

In this method, for every instruction, there is a set of microinstructions and these indicate the control signals to be activated.

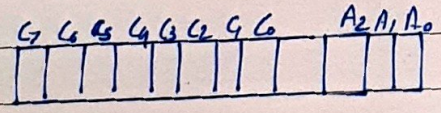
Micro-instructions are stored in ROM and it is called control memory. A set of related micro-instructions used for a single machine instruction is called micro-program.

Each micro-instruction specifies address of next micro-instruction. This is called microprogram instruction sequencing. Main advantage of microprogram control unit is that micro-instructions can be changed easily. Hence it is more flexible.
Time is wasted in fetching micro-instructions from control memory. Hence, it is slower in speed as compared to hardwired control unit.

Wilke's design:

Here, CPU has on-chip control memory. Control memory contains various micro-instructions which will specify control signals to be activated. Each micro-instruction has 2 fields:
i] Control field ii] Address field
Micro-instruction format ⇒ | Control field | Address field |

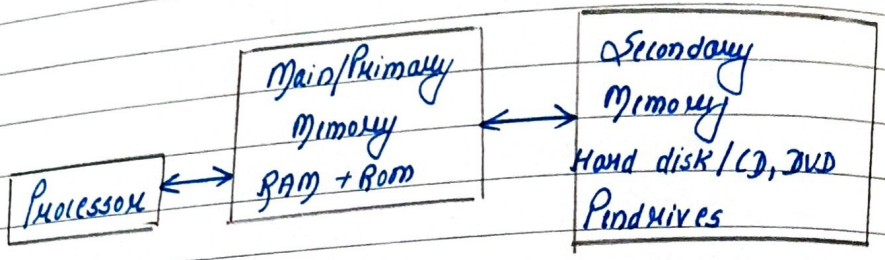
Control field specifies control signals to be activated.
Address field specifies address of next micro-instruction.
Each bit of control field directly corresponds to control signal, hence for any micro-instruction, bit which are logic 1 will be the control signals to be generated.



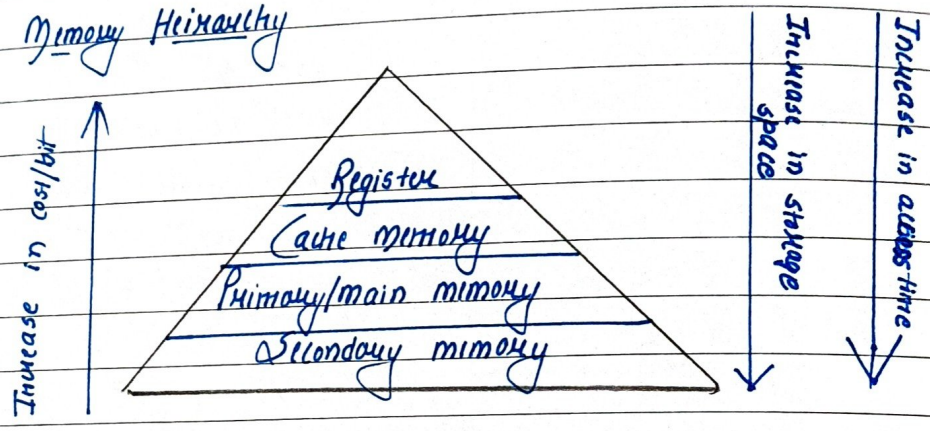
This allows us to change the control unit very easily (simply by adjusting the bits). Starting address of micro-program is produced by internal source to CMAR.
Based on this starting address, the micro-instruction is selected. Using this micro-inst, corresponding control signals are generated and also the address of next micro inst. is placed in CMAR.

Mod 5

Memory Organisation



Memory Hierarchy



Memory characteristics:

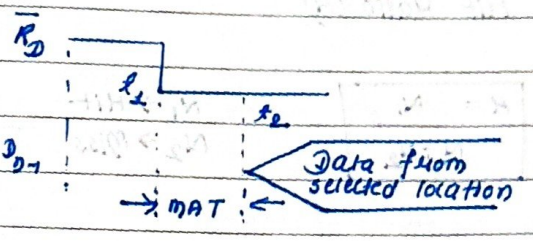
1] Location
- on-chip (registers, L_1 cache)
- internal (L_2, L_3 cache, main/primary)
- external (only connected)

2] Capacity → 2^n locations [1 location - 8 bit]

3] Access method

i] Sequential method ⇒ ex. → floppy disk, magnetic tape
ii] Direct access ⇒ ex. → CD, DVD
iii] Random access ⇒
iv] Associative access ⇒ [Content Addressible]

4] Performance



→ MAT
[memory access time]

→ MCT
[memory cycle time]

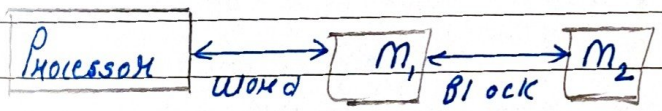
5] Physical types

i] Magnetic memories
ii] Semi-conductor memories [Flip-flop]
iii] Optical memories [DVD]

6] Physical Characteristics

i] Volatile (erasable)
ii] Non-volatile (non-erasable)

Performance characteristics of two level memory



C_1 → Cost/bit of M_1
S_1 → storage capacity of M_1
tA_1 → access time for 1×1

C_2 → cost/bit of M_2
S_2 → storage capacity of M_2
tA_2 → access time for $1 \times 1_2$

1] tA_1 ⇒ time required to

3] Hit Ratio (H)

$$H = \frac{N_1}{N_1 + N_2}$$

$N_1 \rightarrow$ Hit
$N_2 \rightarrow$ Miss

ex. ⇒ 10 → 8 Hit 2 Miss
$\therefore H = \frac{8}{10} = 0.8$

4] Avg. access time

$$tA = H.tA_1 + (1-H).tA_2$$

5] Avg. cost

$$\text{Avg. cost} = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2}$$

6] Avg. efficiency

$$E = \frac{tA_1}{tA}$$

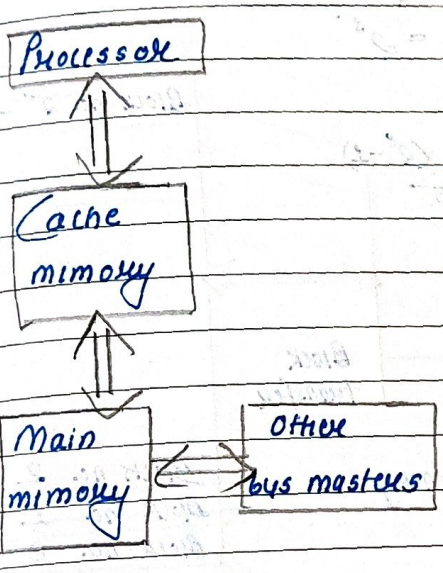
7] Speed Ratio

$$\text{Ratio} = \frac{tA_2}{tA_1}$$

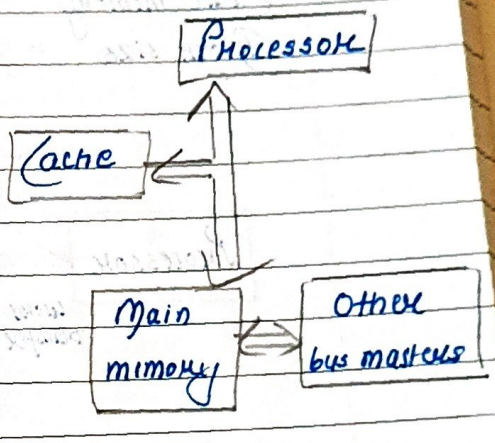
Cache mimory

1] Cache Architecture

① Look through cache



② Look aside cache



⇒ Look up pinalty
⇒ Conuuuency

Solution to cache inconsistency is cache write policies.

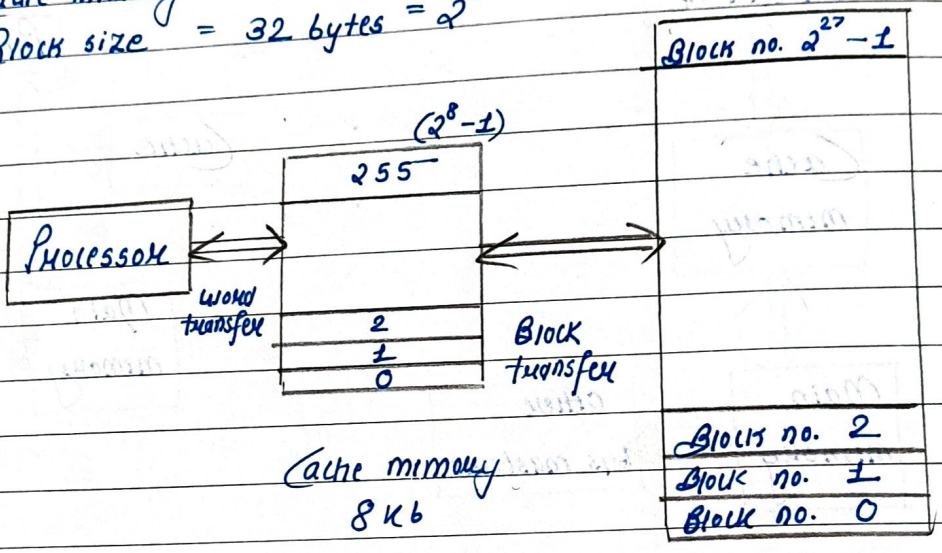
2] Cache write policies

① Write through policy
② Write back / buffered write / delayed write
③ Snoopy writes

Cache mapping techniques

ex. → Pentium processor.

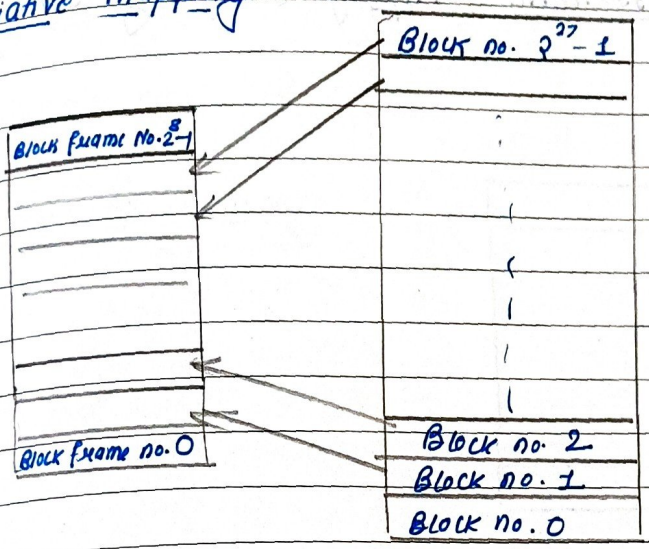
Main memory = 4 GB = 2^{32}
Cache memory = 8 KB = 2^{13}
Block size = 32 bytes = 2^5



No. of blocks in cache memory = $\frac{2^{13}}{2^5} = 2^8$

No. of blocks in main memory = $\frac{2^{32}}{2^5} = 2^{27}$

1] Associative mapping



4GB main memory

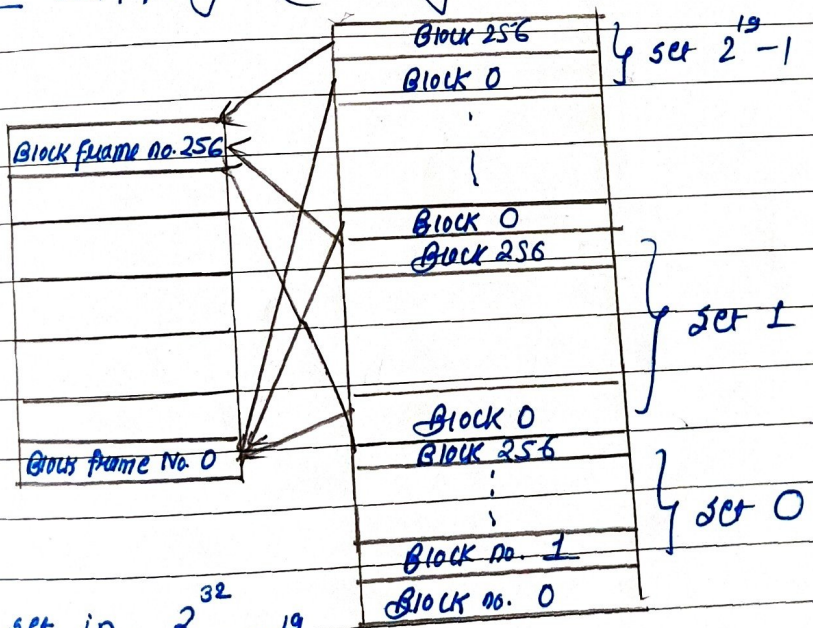
TAG

27 bits

27 bits 5 bit

Block no.

2] Direct mapping (One way set associative)



No. of set in main memory $= \frac{2^{32}}{2^{13}} = 2^{19}$