MSB

Number System

Conversion, Arithmetic operations, Complements, Codes

Weighted Binary Codes

- Binary ⇄ Octal

Octal No.	Binary (421 BCD)
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

- Binary ⇄ Hexadecimal

Hexadecimal No.	Binary (8421 BCD)
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

Non-weighted Binary Codes → Gray Code, Excess-3

- Excess-3 Code

Add +3 to each digit of Decimal Number. Ex-> 43.13, in decimal => 43.13 + 33.33 => 76.46 => 1010 1001 . 0111 1100

Decimal Digit	8421 BCD Code	Excess 3 Code
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1

COA Notes By Garvit Singh

__/__/____

7	0111	1010
8	1000	1011
9	1001	1100

- Gray Code (4 bit)

Only 1 bit changes at once. Useful in error checking. If more than 1 bit has changed at some step, then data must be flawed.

Decimal	8421 BCD Code	Gray Code
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

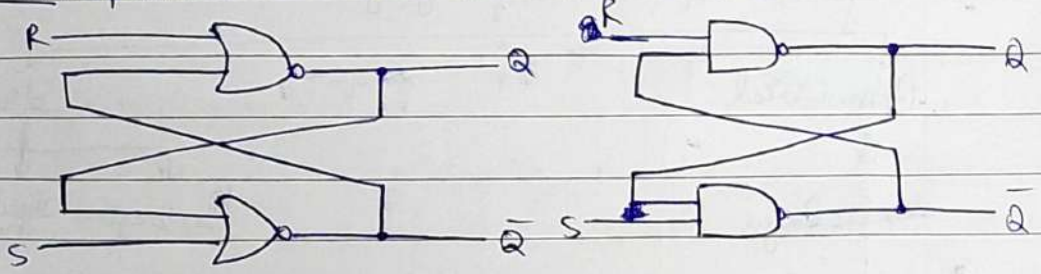
COA Notes By Garvit Singh

- Gray Code (3 bit)

Decimal	424 BCD Code	Gray Code
0	0 0 0	0 0 0
1	0 0 1	0 0 1
2	0 1 0	0 1 1
3	0 1 1	0 1 0
4	1 0 0	1 1 0
5	1 0 1	1 1 1
6	1 1 0	1 0 1
7	1 1 1	1 0 0

COA Notes By Garvit Singh

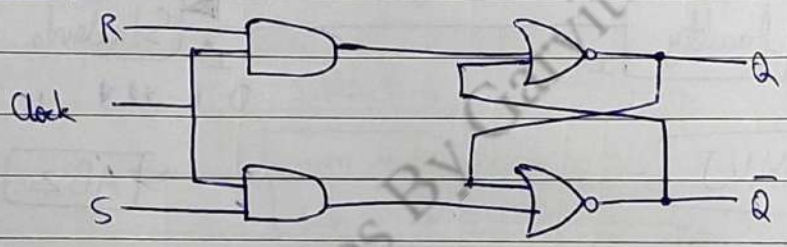
- **SR latch implemented with NOR Gates & NAND Gates**

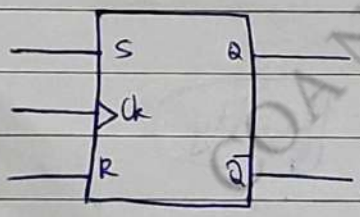


Characteristic Table

S	R	Q_{n+1}	
0	0	Q_n	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	—	Indeterminate

- **SR Flip Flop**

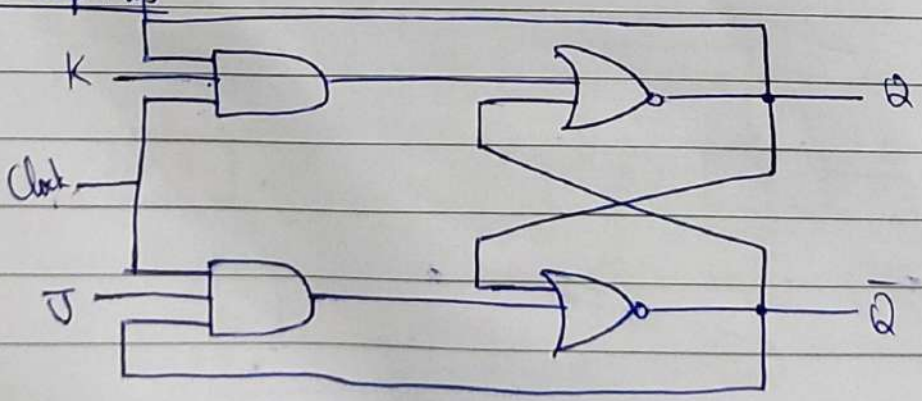




Truth Table

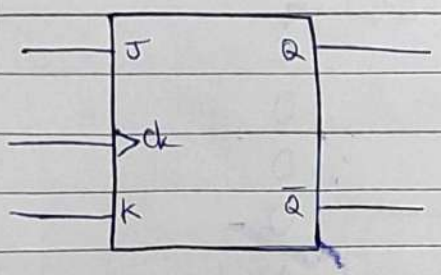
S	R	Q_{n+1}	
0	0	Q_n	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	—	Indeterminate

- **JK Flip Flop**



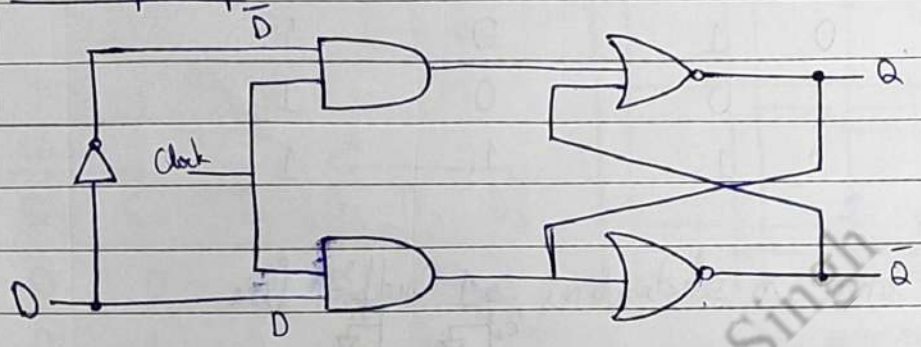
COA Notes By Gurjit Singh

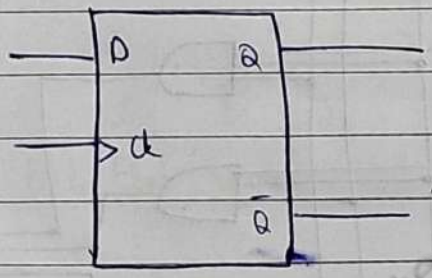
__/__/____



J	K	Q_{n+1}	
0	0	Q_n	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	$\bar{Q}_n$	Complement

- **D Flip Flop**

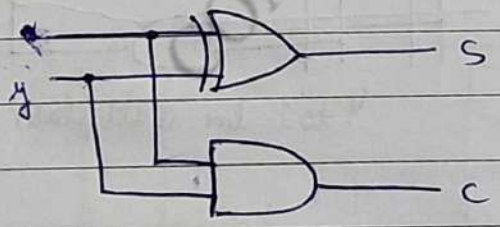




D	Q_{n+1}	
0	0	Clear to 0
1	1	Set to 1

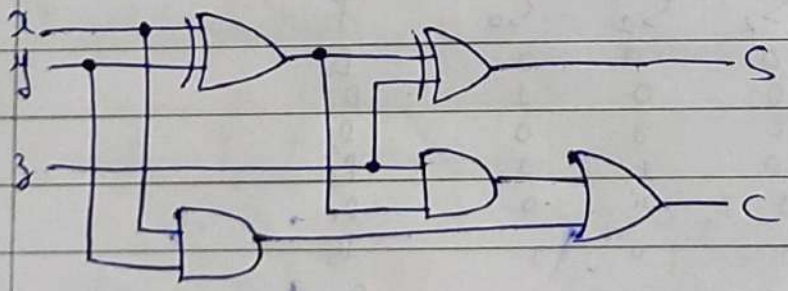
OOA Notes By Garvit Singh

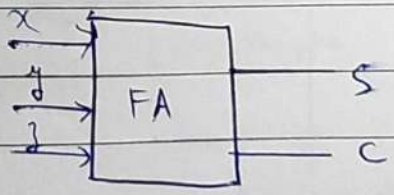
- **Half Adder**



x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- **Full Adder**



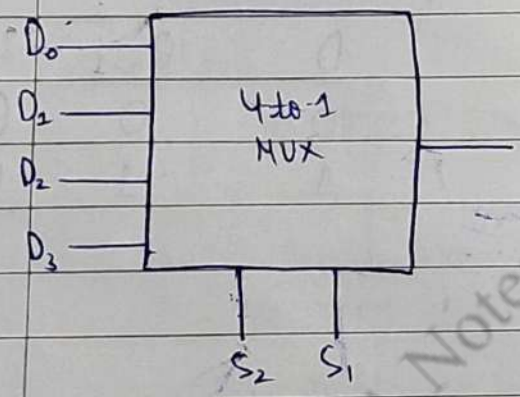


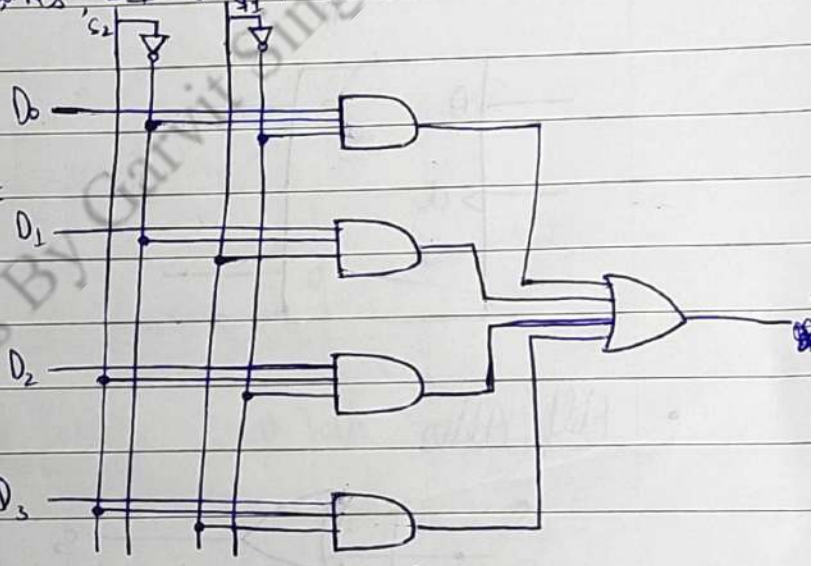
Truth Table

Inputs			Output	
x	y	z	S	C
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

- Multiplexers

2^n input lines, n selection lines, 1 output line





4 to 1 line multiplexer

Selection lines		Output
S_2	S_1	F
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

For 8×1 MUX:

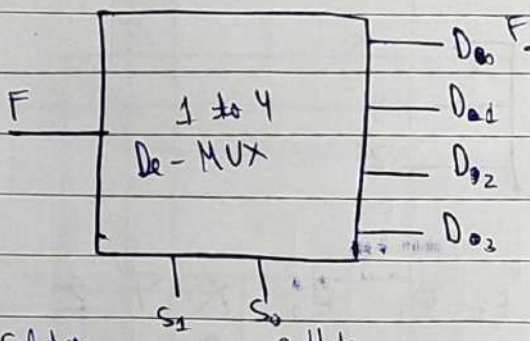
S_2	S_1	S_0	F
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

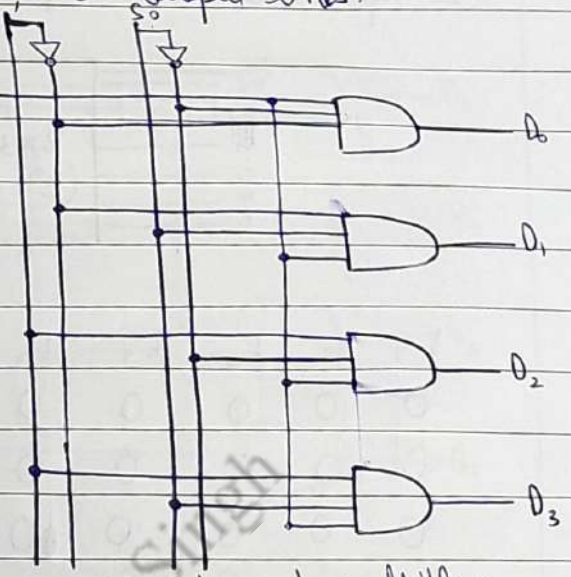
COA Notes By Gurvir Singh

__/__/____

- **De-Multiplexer**

1 input line, n selector lines, 2^n output lines.





Selectors		outputs			
S_1	S_0	D_0	D_1	D_2	D_3
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

1 to 4 line de-multiplexer

Using SOP form, find values of $D_0, D_1 \ldots$

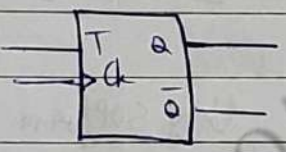
$D_0 = S_1 S_0$

$D_1 = S_1 \bar{S_0}$

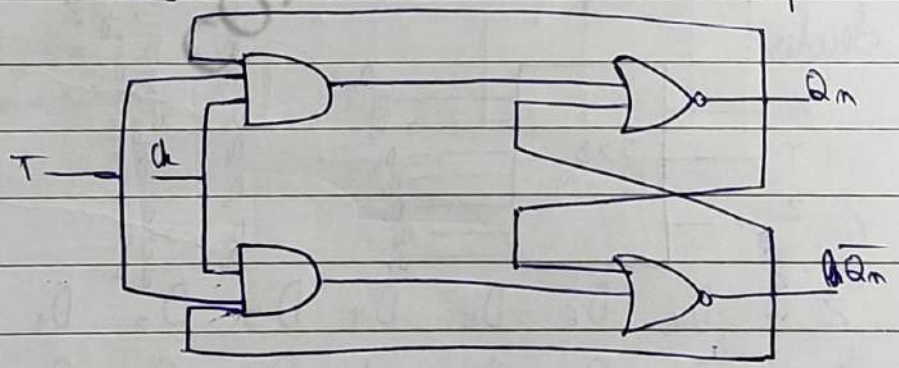
$D_2 = \bar{S_1} S_0$

$D_3 = \bar{S_1} \bar{S_0}$

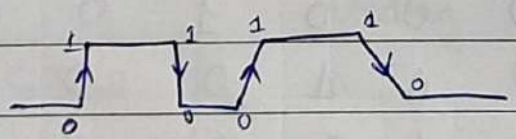
- **T Flip Flop (Toggle)** → Used in counters



T	$Q(n+1)$	
0	Q_n	No change
1	$\bar{Q_n}$	Complement



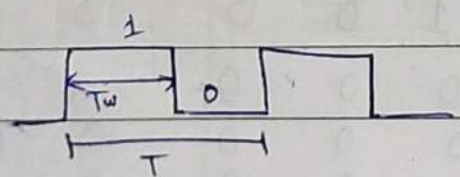
- **Edge & Level Triggers**



0 → 1 (Positive edge)

1 → 0 (Negative edge)

] Edge triggers



1 – high level

0 – low level

$T_w = T/2$

$T = 2 \times T_w$

] Level Triggers

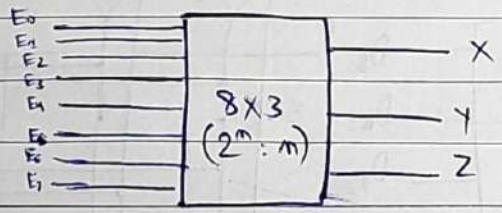
OOA Notes By Gaurav Singh

__/__/____

~~Using SOP form~~

- **Encoder** $2^n : n$

  2^n input lines, n output lines

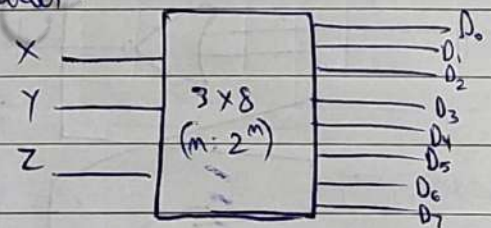


E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0	X	Y	Z
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

- **Decoder** $n : 2^n$

  n input lines, 2^n output lines

For 3×8 decoder:



Using SOP form:

$D_0 = \bar{x}\bar{y}\bar{z}$

$D_1 = \bar{x}\bar{y}z$

$D_2 = \bar{x}y\bar{z}$

$D_3 = \bar{x}yz$

$D_4 = x\bar{y}\bar{z}$

$D_5 = x\bar{y}z$

$D_6 = xy\bar{z}$

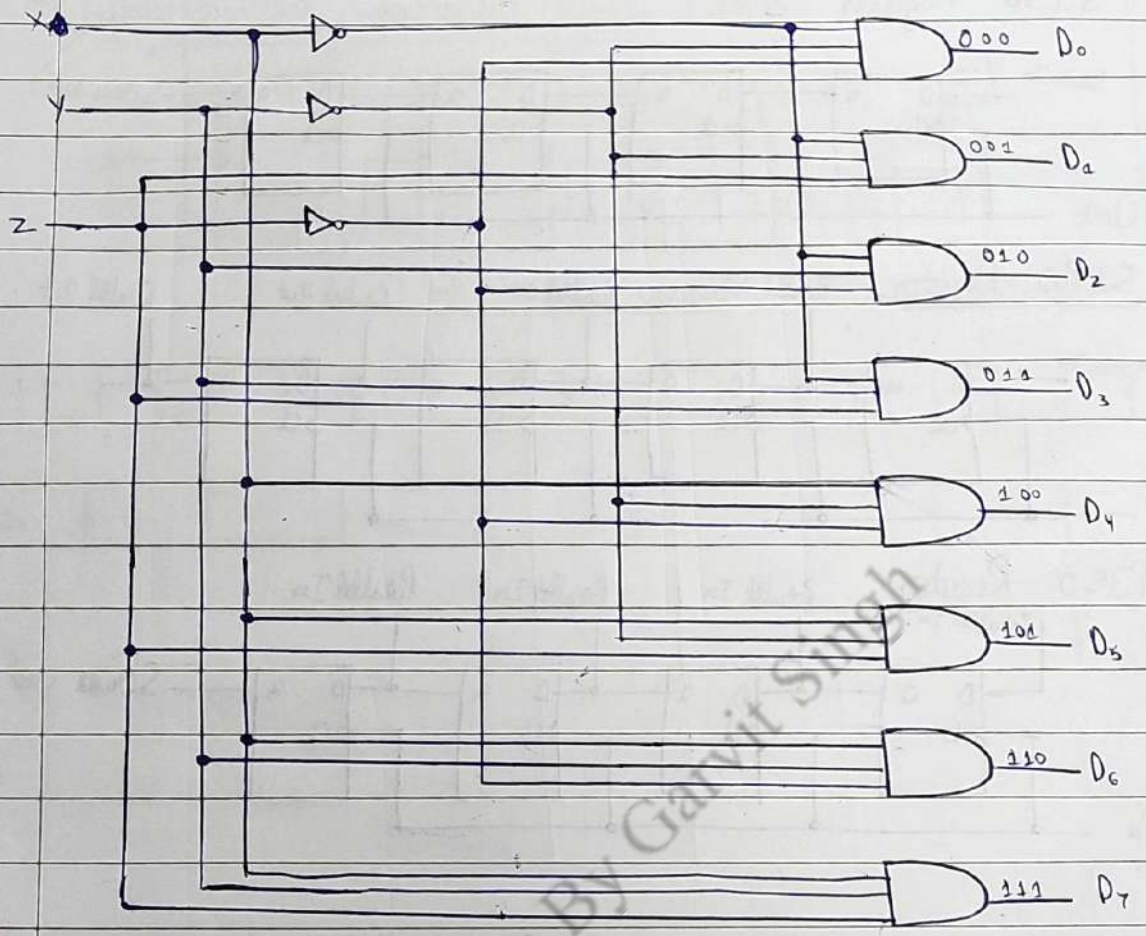
$D_7 = xyz$

X	Y	Z	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_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	1	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

CO Notes By Garvit Singh

__/__/____

Decoder Implementation using logic gates



Decoder with 3 inputs and $2^3 = 8$ outputs

COA Notes By Ganvir Singh

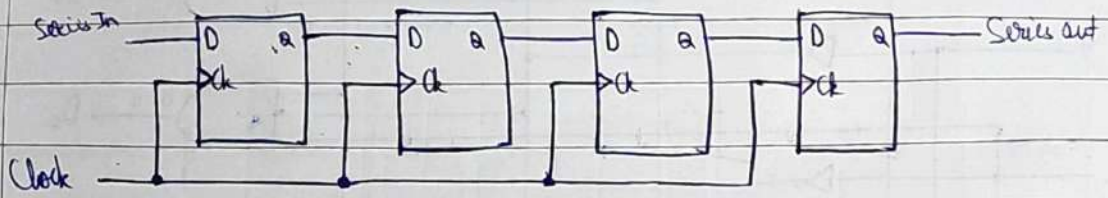
- **Registers**

Based on Application: Shift Register, Storage Register

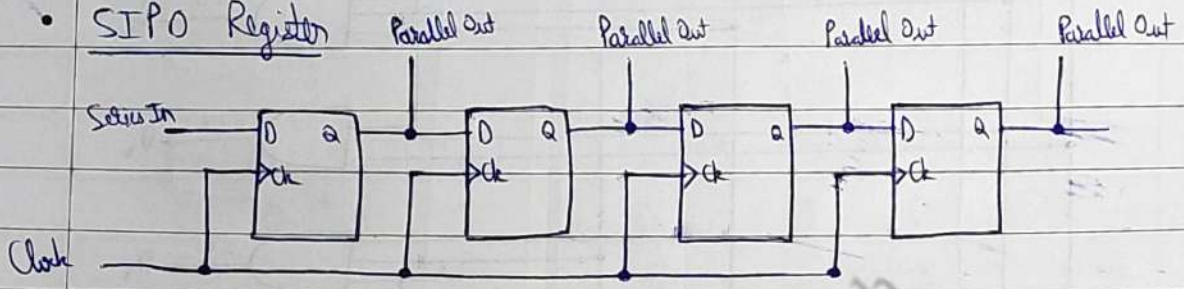
Shift Registers: (i) Series In Series Out (SISO)
(ii) Parallel In Parallel Out (PIPO)
(iii) Parallel In Serial Out (PISO)
(iv) Series In Parallel Out (SIPO)

Mode	Clocks needed for n bit shift register		
	Loading	Reading	Total
SISO	n	n-1	2n-1
SIPO	n	0	n
PISO	1	n-1	n
PIPO	1	0	1

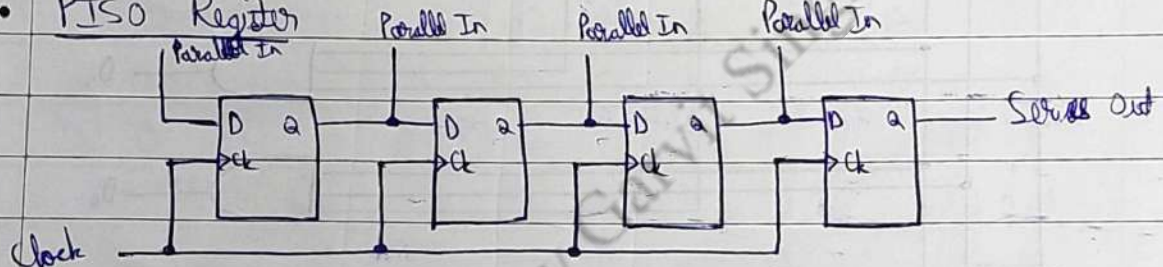
- **SISO Register**



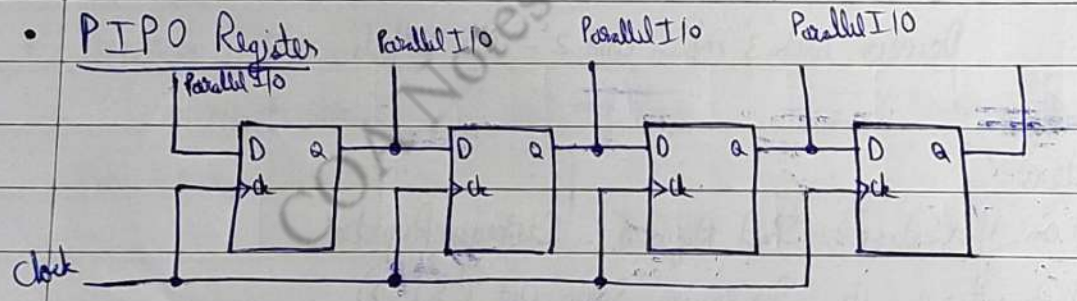
- **SIPO Register**



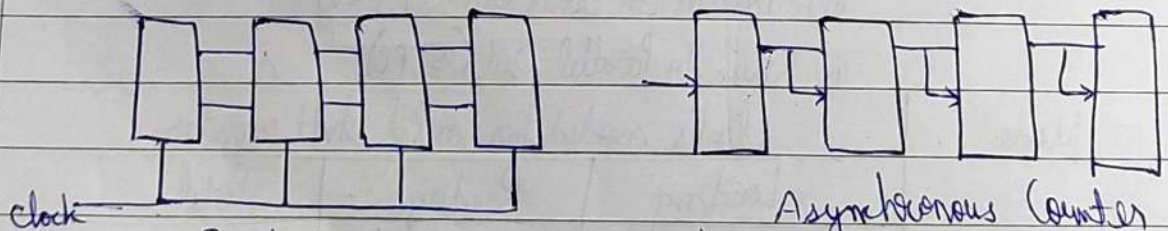
- **PISO Register**



- **PIPO Register**



- **Counters**: Synchronous & Asynchronous



Synchronous Counter

Asynchronous Counter
aka ripple counter

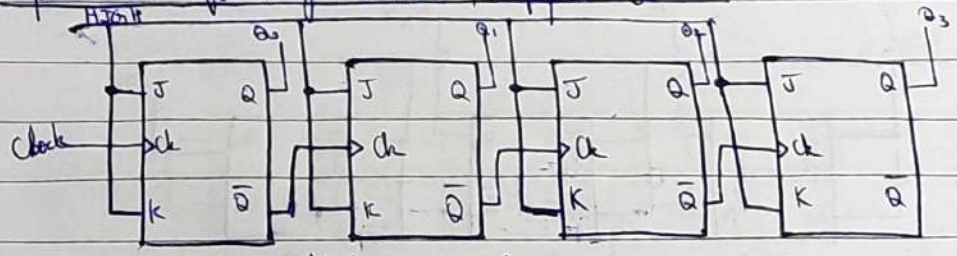
Up Counter ⇒ 0 – 1 – 2 – 3 – 4

Down Counter ⇒ 4 – 3 – 2 – 1 – 0

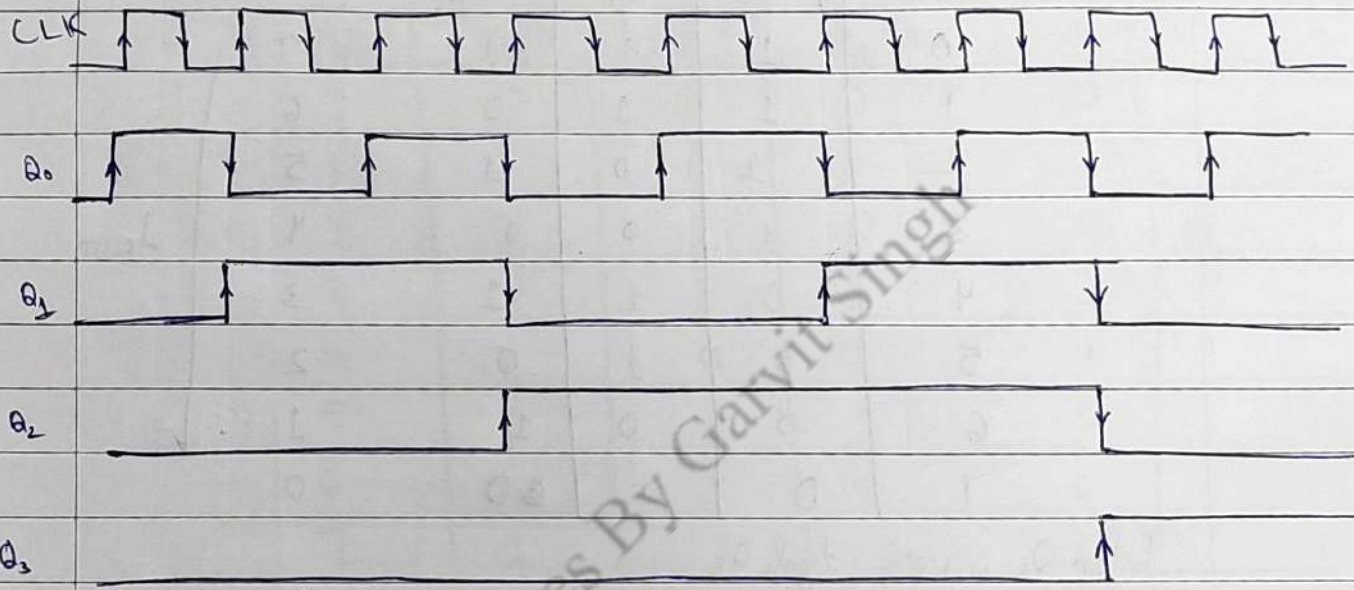
Class Notes By Gavi Singh

__/__/____

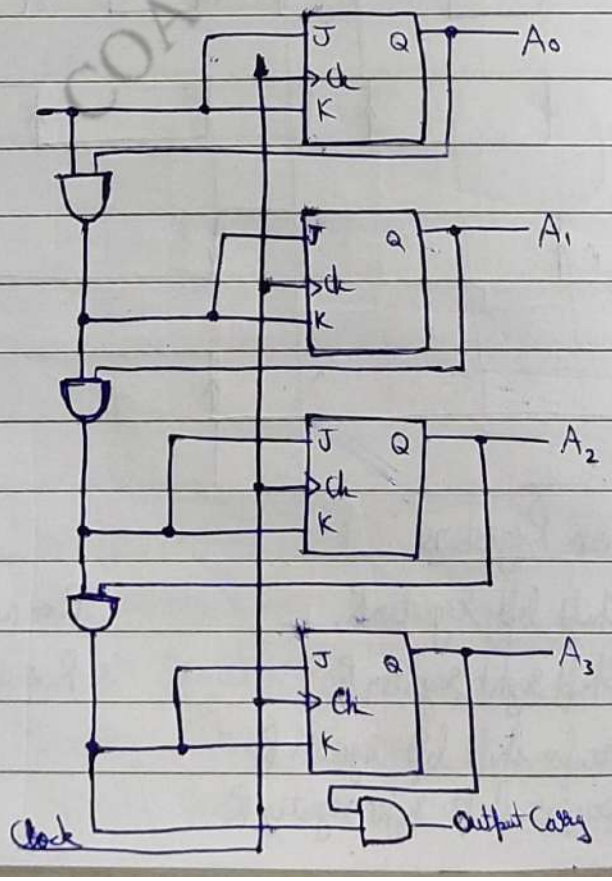
- **Representation for Asynchronous / Ripple Counter**



4 bit ripple counter



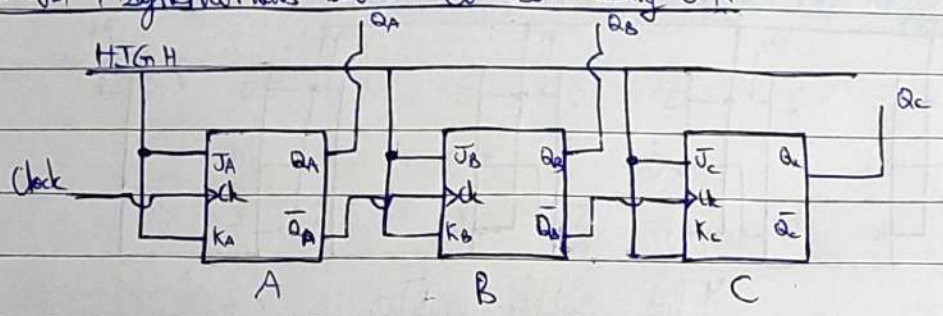
- **Representation for Synchronous Counter**



COA Notes By Garvit Singh

__/__/____

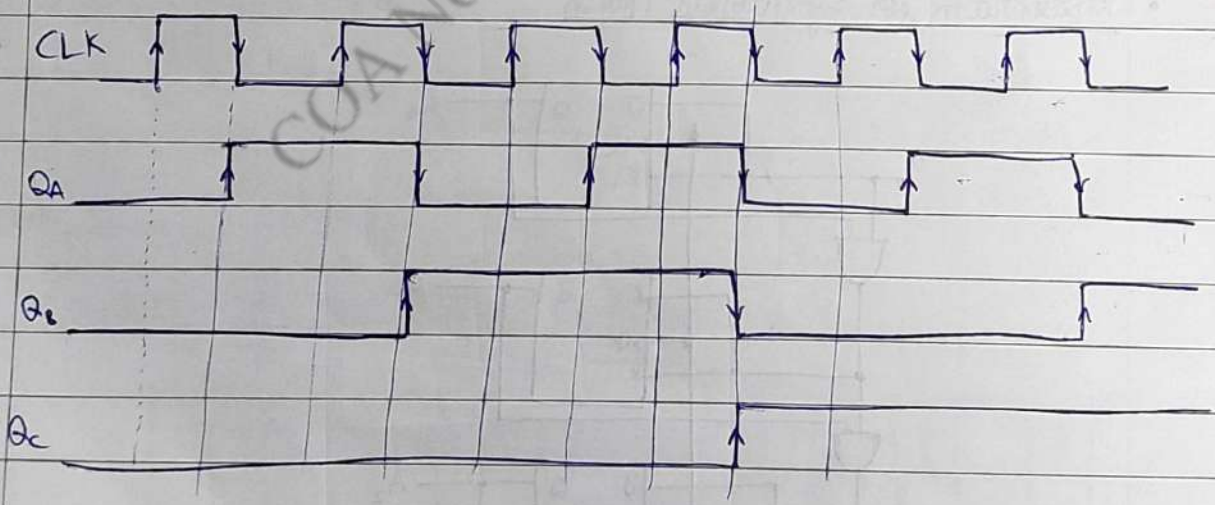
- **3 bit Asynchronous Down Counter using JK**



State	Q_C	Q_B	Q_A	
0	1	1	1	7
1	1	1	0	6
2	1	0	1	5
3	1	0	0	4
4	0	1	1	3
5	0	1	0	2
6	0	0	1	1
7	0	0	0	0

down

When Q_A is +ve, toggle Q_B



COA Notes By Gauri Singh

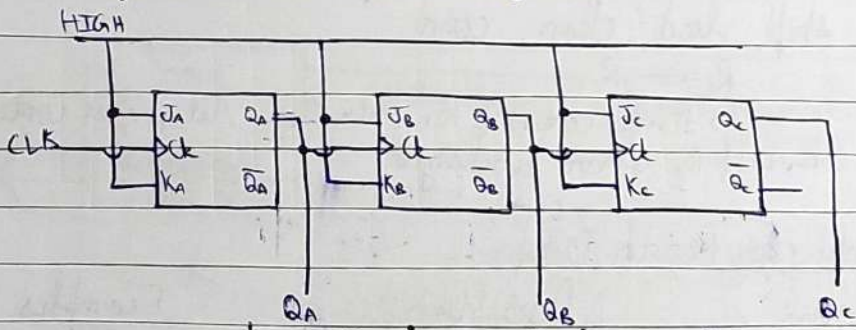
- **Shift Microoperations on Registers**

R ← shl R	Shift left register R	R ← ashl R	Arithmetic shift left R
R ← shr R	Shift right register R	R ← ashr R	Arithmetic shift right R
R ← cil R	Circular shift left register R		
R ← cir R	Circular shift right register R		

__/__/____

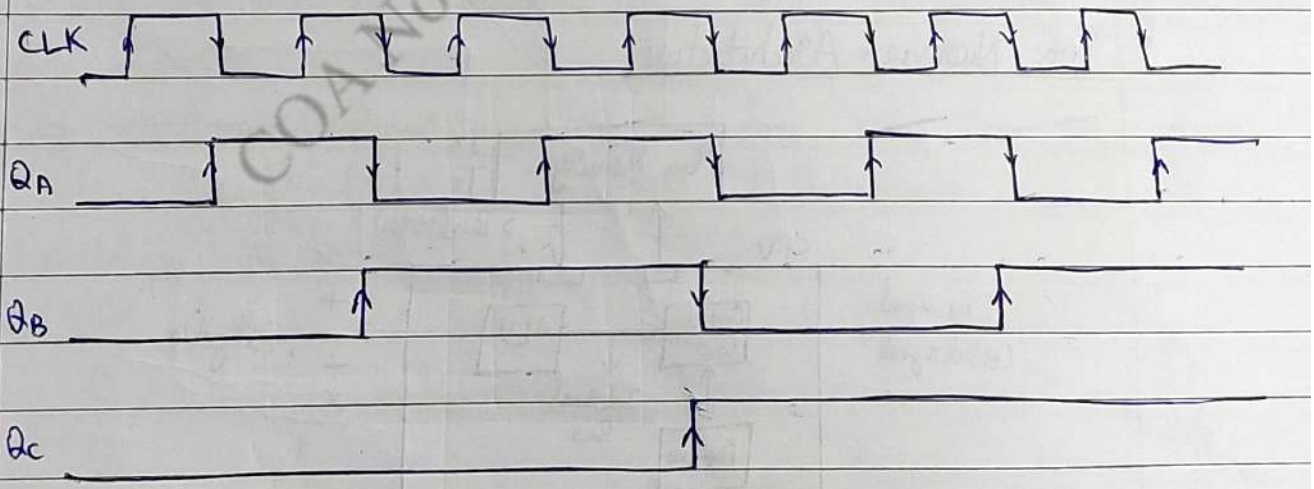
Up

- 3 bit Asynchronous Counter using JK



State	Q_C	Q_B	Q_A	
0	0	0	0	0
1	0	0	1	1
2	0	1	0	2
3	0	1	1	3
4	1	0	0	4
5	1	0	1	5
6	1	1	0	6
7	1	1	1	7

Up



COA Notes By Garvit Singh

- Register Transfer Language (RTL) and Microoperations

1) Register: Store information

RI : Processor Register
MAR : Memory Address Register
MDR : Memory data

IR : Instruction Register
PC : Program Counter
SR : Status Register

__/__/____

2) Microoperations : operation performed on information stored in registers.
Ex→ shift, load, clear, count.

$R_2 \leftarrow R_1$

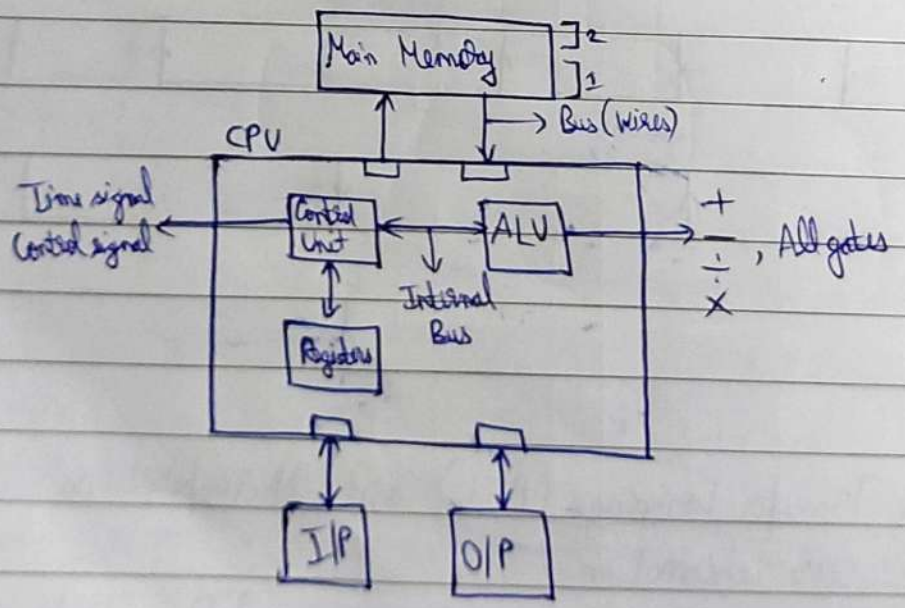
↳ transfer content of R_1 into R_2. All previous content on R_2 will be erased. Contents of R_1 remain unchanged.

- Basic Symbols for Register Transfer

Symbol	Description	Examples
Letters & Numbers	Denotes a register	MAR, R2
Parenthesis ( )	Denotes a part of register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	$R_2 \leftarrow R_1$
Comma ,	Separates two microoperations	$R_2 \leftarrow R_1$, $R_1 \leftarrow R_2$
Colon :	Denotes conditional operations	$P: R_2 \leftarrow R_1$ if P=1
Naming operator :=	Denotes another name for an already existing register/alias	$R_a := R_1$

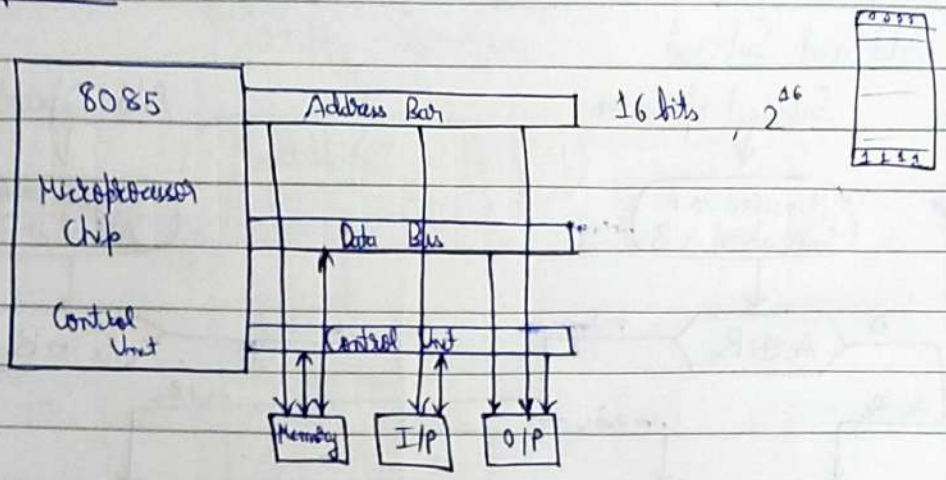
Registers + Microoperations + Control Functions = Digital Computer

- Von Neumann Architecture



__/__/____

- **Types of Buses**



System bus is a communication channel which is used to provide communication between major components of computers.

- **Types of Instructions**

(i) Data Transfer Instructions (DTI) (ii) Data Manipulation Instructions (DMI) (iii) Program Control Instructions (PCI)

DTI: mov, load, Store, Exchange, input, output, push, pop

DMI: Add, sub, Div, mul, INC, DEC, Add with Carry, Sub with Borrow, Negate,

Logic instructions: complement, logical AND, OR, clear, XOR, clear carry, set carry, complement carry, enable interrupt, disable interrupt

- **Excitation Table for All 4 Flip Flops**

SR flip flop

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

D flip flop

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

JK flip flop

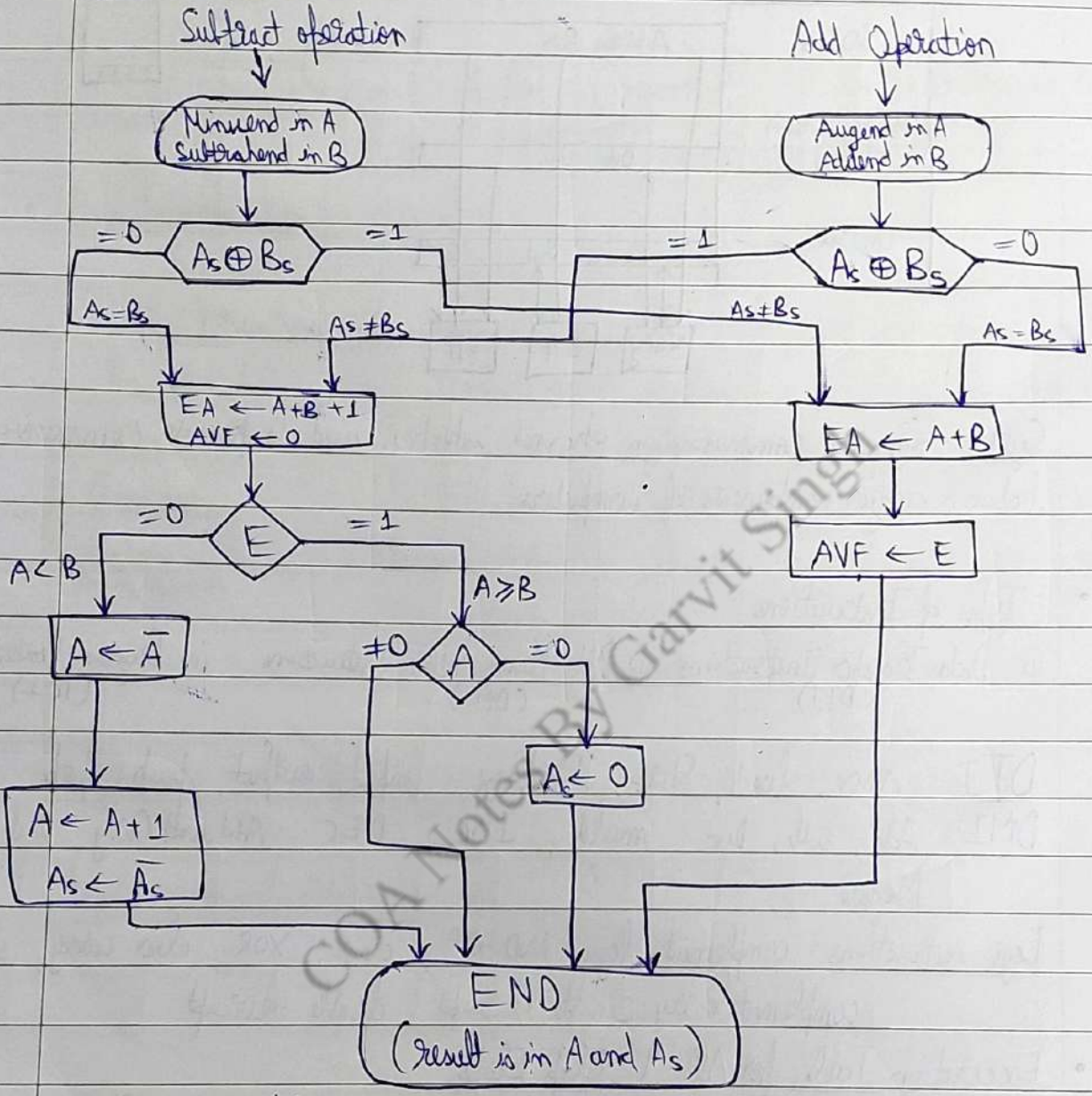
Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

T flip flop

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

COA Notes By Garvit Singh

- **Computer Arithmetic : Booth's Algorithm for + , − , ÷ , ×**
- **Add and Subtract**



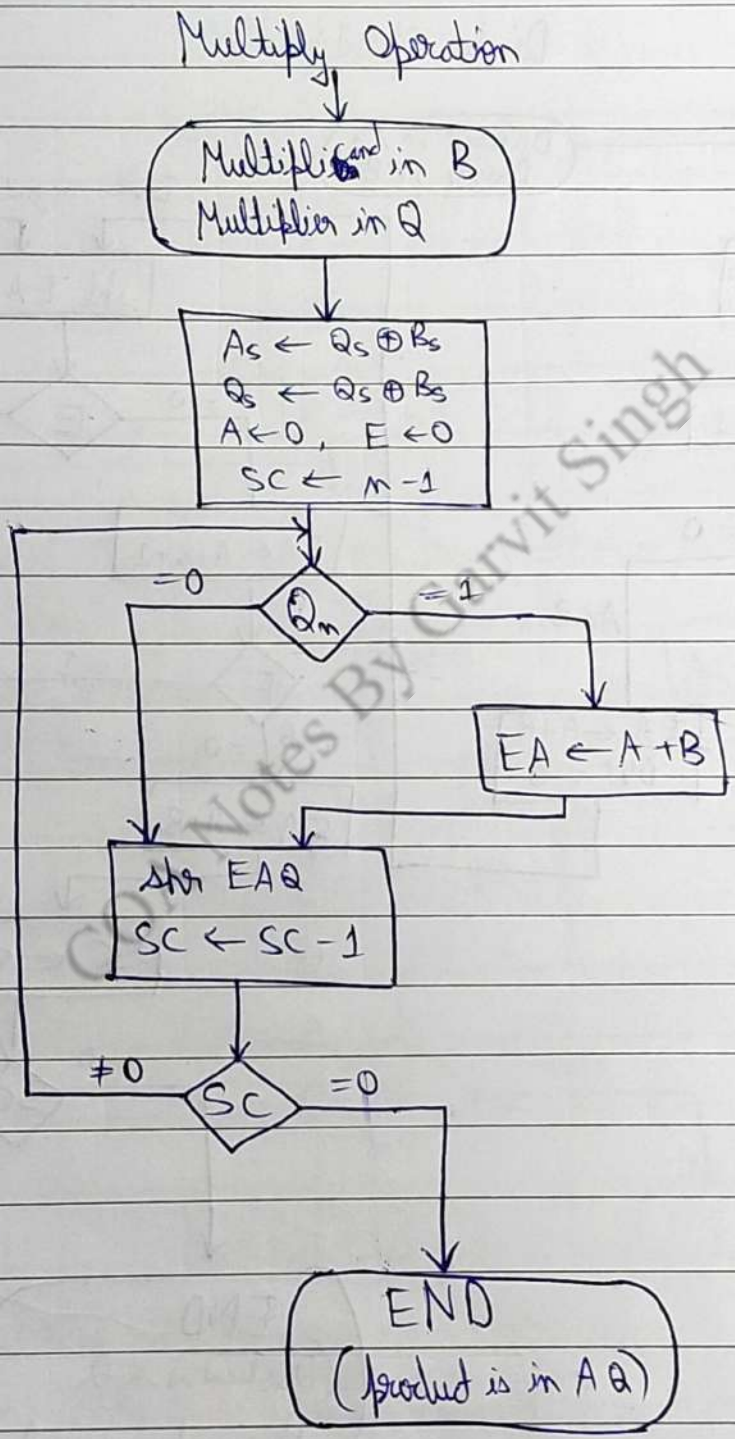
- **Hardware Implementation**

(i) Require two registers for storing value of A and B.
(ii) Two flip flop hold corresponding signs.
(iii) One more register for holding the result
(iv) Parallel adder to perform A+B.
(v) A comparator circuit is required A>B, A<B, A=B
(vi) Two parallel subtractor circuit are needed to perform A−B and B−A.

COA Notes By Garvit Singh

__/__/____

- **Multiply**

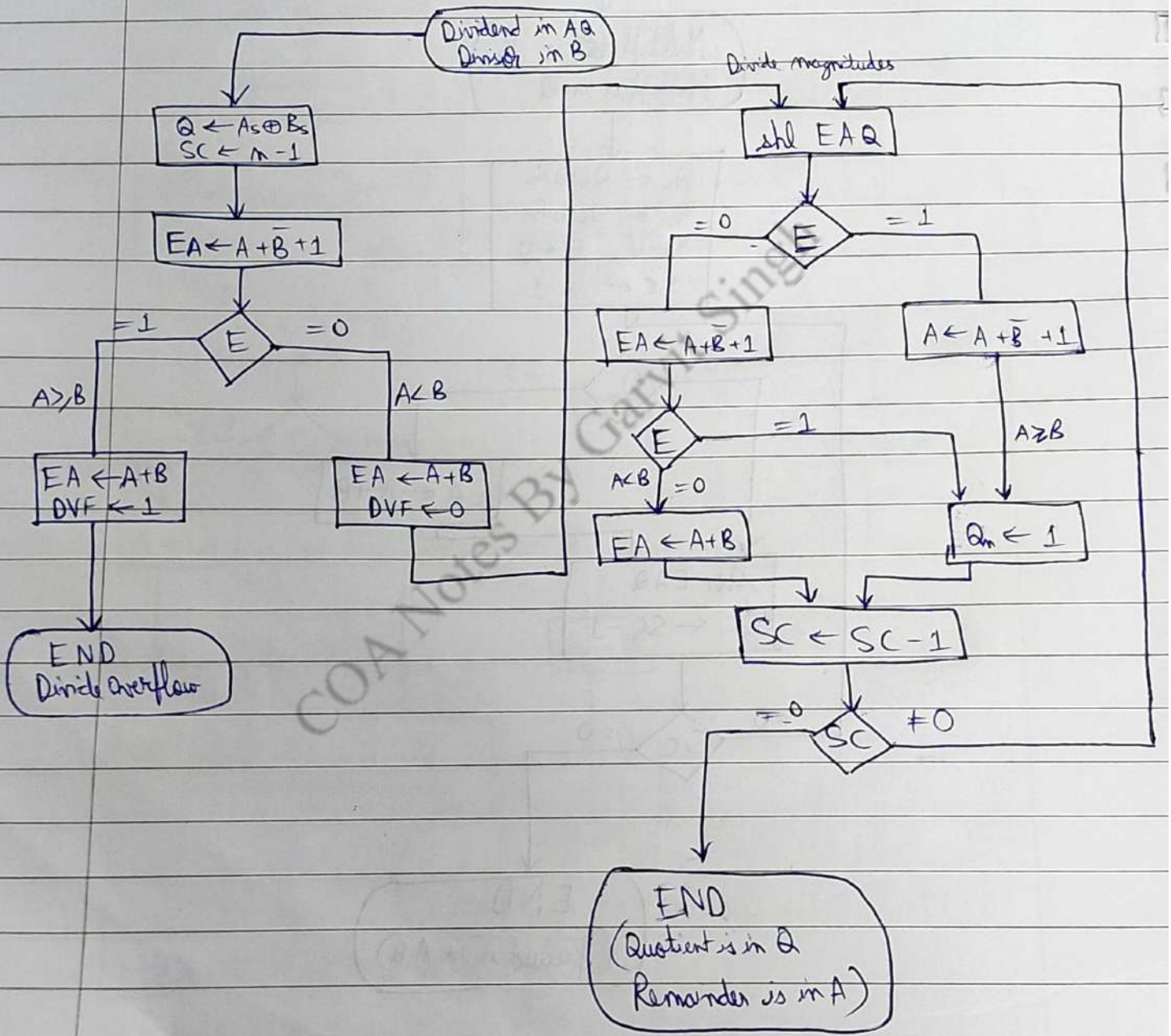


Class Notes By Garvit Singh

__/__/____

- **Divide**

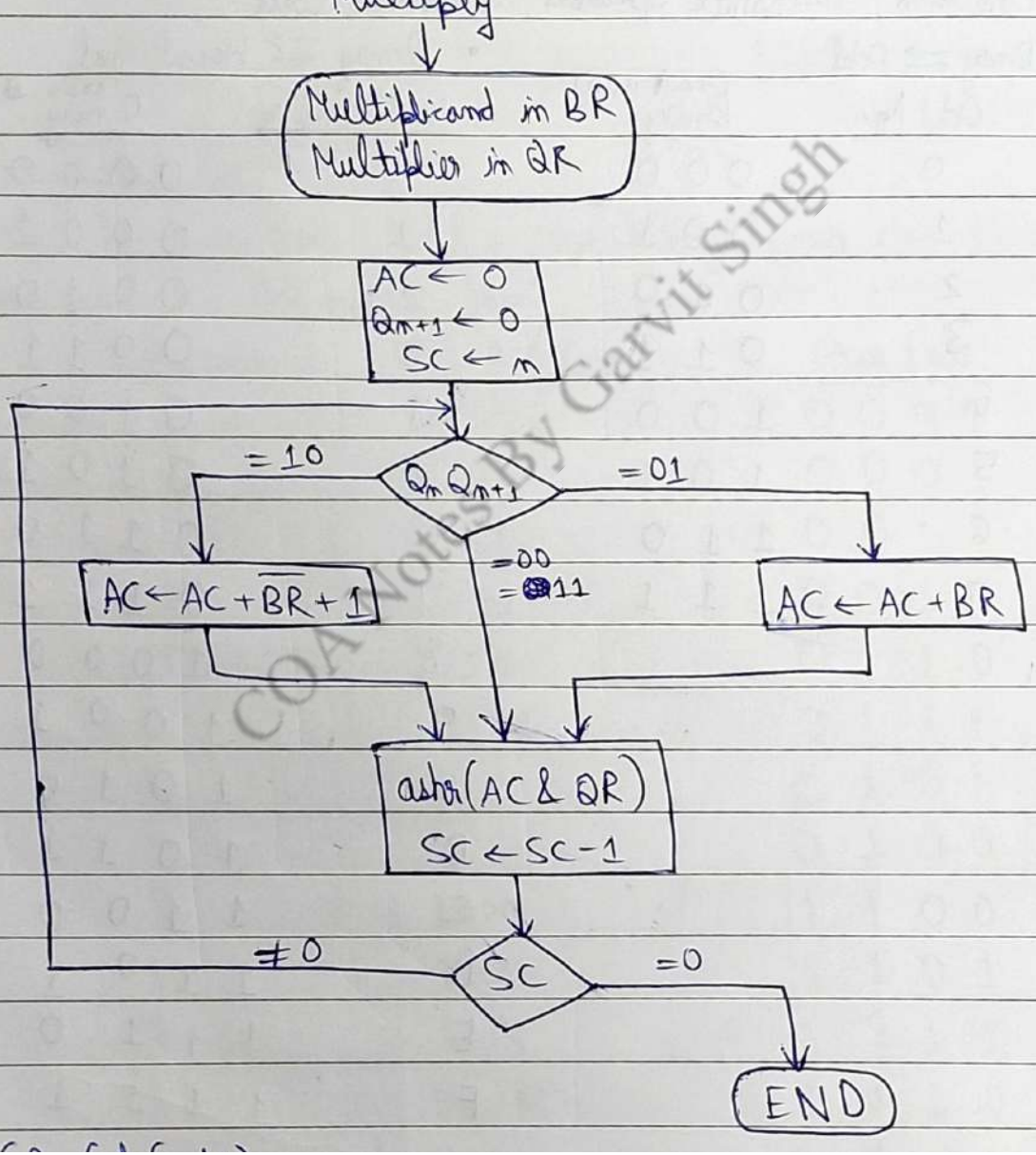
Divide Operation



COA Notes By Garry Singh

__/__/____

- **Multiply for signed-2's complement numbers**



COA Notes By Garvit Singh

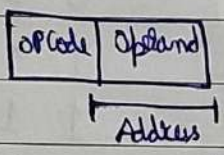
- **Addressing Modes**

1) **Implied Mode**

Operand is specified implicitly in the definition of instruction. Used for zero address and one-address instructions. Ex→ INC A → increment accumulator, CRC → clear carry flag, CLA → complement accumulator.

2) **Immediate Mode**

Operand is directly provided as constant. No computation required to calculate effective address.



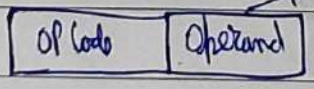
Add R_1, #3

$R_1 \leftarrow R_1 + 3$

Limitation→ The range of constant number supported depends on size of address field.

3) **Register Mode**

Operand is present in the register. Register no. is written in the instruction.



LD R_1

$AC \leftarrow R_1$

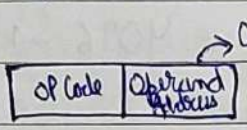
Effective address = (R_i) where i is register no.

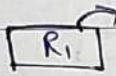
Advantage → Instruction size will be small since we're working with registers. Registers are also faster than main memory. Speed is higher.

COA Notes By Gurjit Singh

4) Register Indirect Mode

Register contains address of operand rather than operand itself





Contains address of operand instead of operand itself.

LD (R1)

AC ← M [(R1)]

Add R1, (R2)

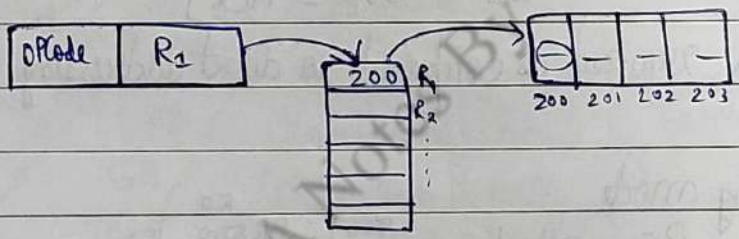
R1 ← R1 + M[(R2)]

Effective address = M[(R1)]

Advantage → Registers are smaller and faster as compared to main memory. Faster computation. Instruction size will be small. Takes up less space.

5) Auto Increment or Auto Decrement Addressing Mode

Special case of register indirect addressing mode



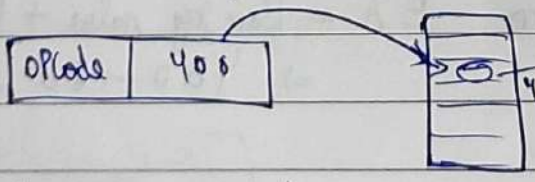
LD (R1) +

AC ← M [(R1)]

Register is auto-incremented or decremented, without having to update the content of the register again and again. Useful if operands are stored in a continuous array in the main memory. Register keeps incrementing and updates to the next memory address of main memory.

6) Direct Addressing Mode (Absolute addressing mode)

Actual address is given in instruction. Used to access variables.



Add X

AC ← AC + M[(X)]

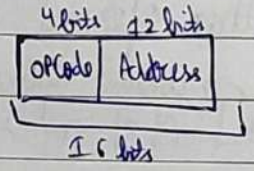
No computation needed.

LD 400

AC ← M[400]

__/__/____

Limitation: Address cannot be larger than the size of instruction.

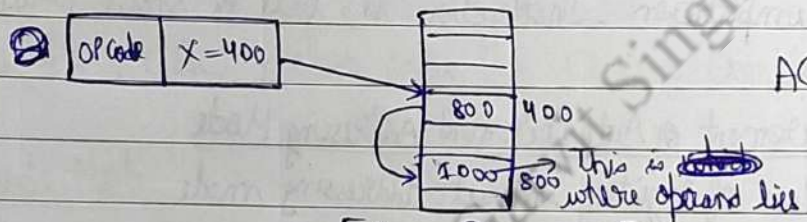


∴ Only $2^{12} - 1 = 4096 - 1 = 4095$ addresses can be accessed using this mode (if instruction format is in 16 bits)

7) Indirect Addressing Mode

i) Used to implement pointers and passing parameters.

ii) 2 memory access required.



Add X

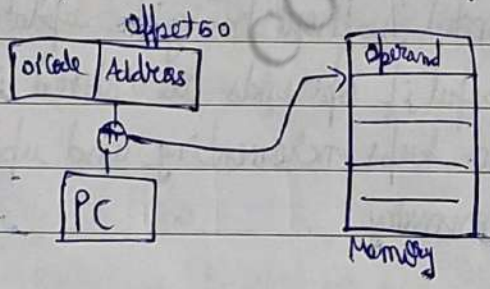
$AC \leftarrow AC + M[M[X]]$

$EA = 800 = M[X]$

More computation required as compared to direct addressing.

8) Relative addressing mode

Effective address = PC + Offset



PC + offset
500 + 50
⇒ 550
Jump to 550

Give the displacement instead of the entire address.

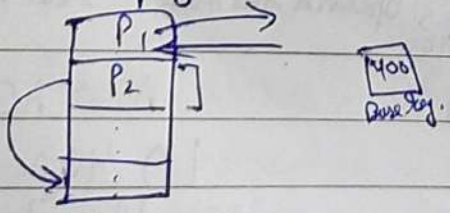
PC gets incremented to 501 after execution

9) Base register addressing mode

Used in program relocation

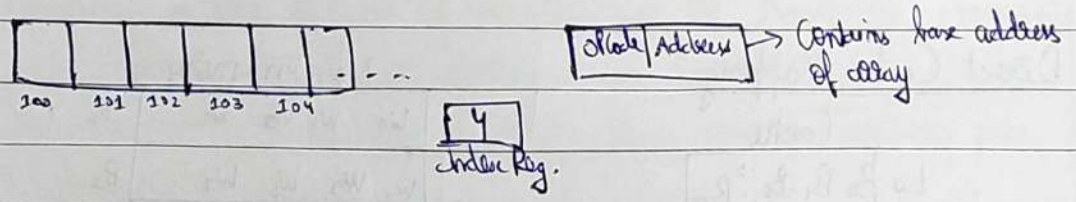
EA = Base reg. value + Disp

⇒ 400 + 60 = 460



10) Indexed Addressing Mode

(i) Used to access or implement array efficiently.
(ii) Multiple registers required to implement.
(iii) Any element can be accessed without changing instruction.



EA = Base address + IR = 100 + 4 = 104

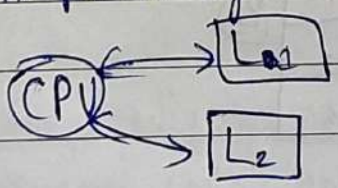
Advantage → Instruction remains constant. No need to update it. Only IR values gets changed

- Memory Hierarchy

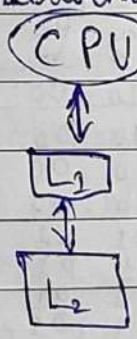
Speed, Capacity, Cost, Stability, Scalability

- 2-level memory organisation

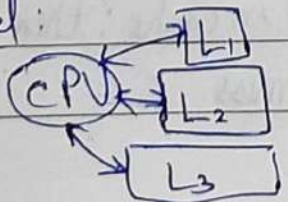
Independent Organisation

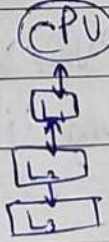


Heirarchial



- 3-level:





© Aakash Singh Garvit Singh

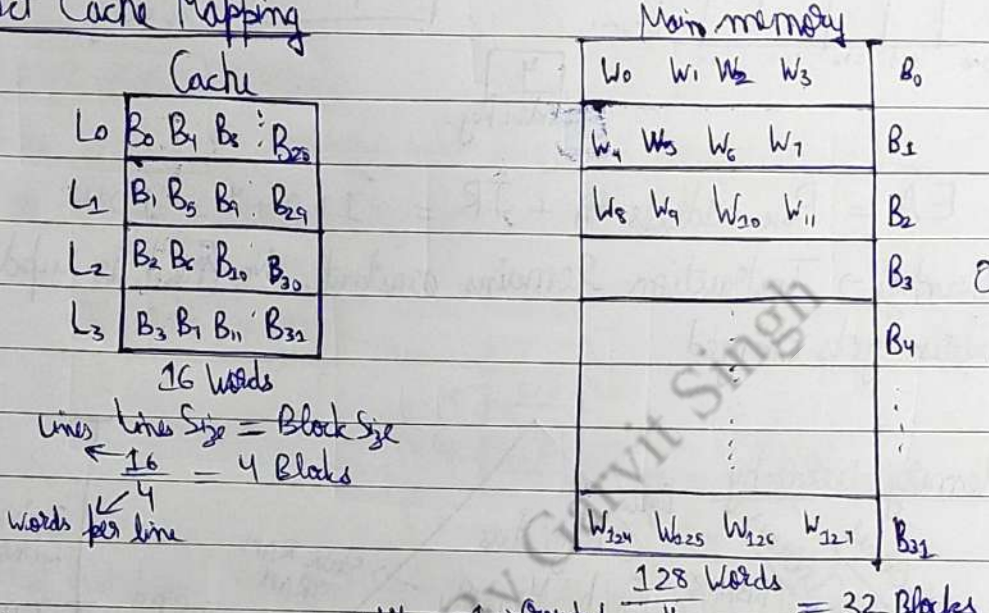
__/__/____

- **Cache Mapping & It's Types**

  Direct Mapping Fully associative K-way set associative

- ~~Pipelining Basics~~

1) Direct Cache Mapping



K mod n ⇒ 1 mod 4 = 1 → Goes to L_1
2 mod 4 = 2 → Goes to L_2

K → Block no. n → No. of lines

Physical Address (PA) (7)

5 bits	2 bits
Block No.	Block offset (size)

00010 | 10
↳ B_2 ↳ 2nd word taken from zero

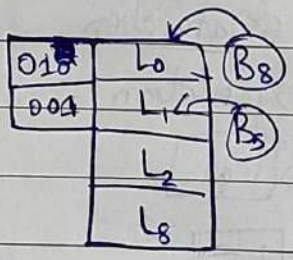
3 bits	2 bits	2 bits
Tag	Line No.	Block offset

Ex-1

010	00	00	W_{32}
010	00	01	W_{33}
010	00	10	W_{34}
010	00	11	W_{35}

Ex-2

001	01	00	→ 20
001	01	01	→ 21
001	01	10	→ 22
001	01	11	→ 23



If the word CPU is searching for is found in cache, then it is called cache hit. If not found, it is called cache miss.

Advantage: Simple method. Searching is easy

Disadvantage: Conflict miss

- Pipelining

(i) Pipelining is the process of arrangement of hardware elements of CPU such that its overall performance is increased.

(ii) Simultaneous execution of more than one instruction takes place in a pipelined processor

(iii) In pipelining, multiple instructions are overlapped in execution.

- Interrupt

Hardware, Software, Internal, External

- CISC vs RISC

CISC	RISC
1) Complex Instruction Set Computer	1) Reduced Instruction Set Computer
2) Large no. of instructions.	2) Less No. of instructions
3) Variable length instruction format.	3) Fixed length instruction format.
4) Large no. of addressing modes	4) Few no. of AM
5) Cost is high	5) Less Cost
6) More powerful.	6) Less powerful
7) Several Cycle instructions.	7) Single Cycle instructions
8) Manipulation directly in memory.	8) Only in registers
9) Microprogrammed control unit.	9) Hardwired Control Unit
10) Mainframes, Motorola 68 00, Intel 8080,	10) MIPS, ARM, SPARC, Fugaku
MULT 2:2, 3:3	LOAD A, 2:2
	LOAD B, 3:3
	PROD A, B
	STORE 2:2, A

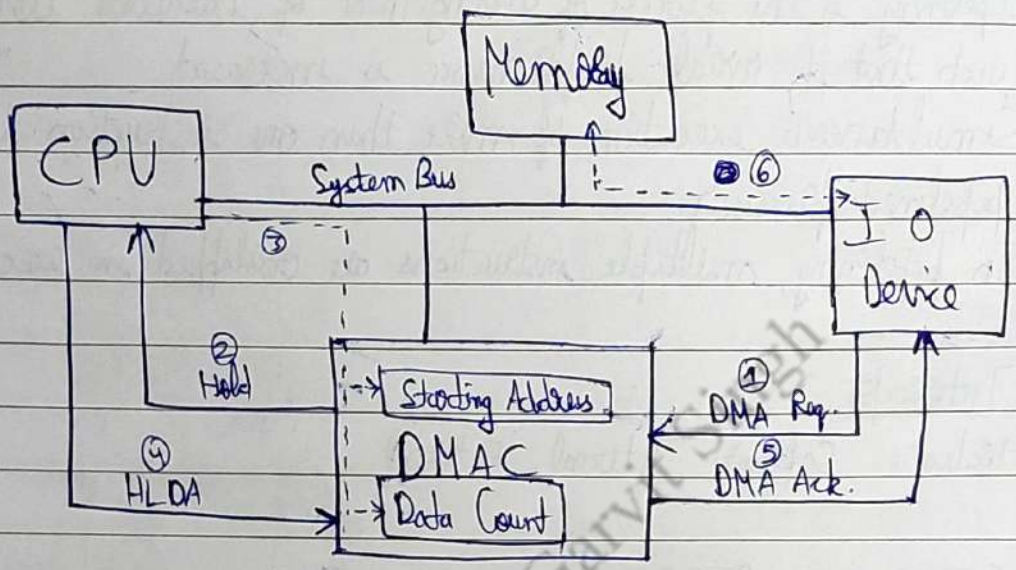
COA Notes By Garvit Singh

__/__/____

- **Direct Memory Access (DMA)**

→ DMA enables data transfer between I/O and memory without CPU intervention.

→ Need a hardware: DMAC (DMA Controller)



Starting address: Memory address, starting from where data transfer should be performed.

Data Count: No. of bytes or words to be transferred.

	Initially	After 1 Byte	After 2 byte	After 3 byte
SA	1000	1001	1002	1003
DC	500	499	498	497

When, DC (data count) reaches zero, transfer is stopped.

→ During DMA transfer CPU can perform only those operations which do not require system bus, which means CPU will be blocked mostly.

→ Control to DMAC is not given for a long period of time.

Modes of DMA transfer:

1) Burst Mode
2) Cycle Stealing
3) Interleaving DMA

1) Burst Mode
→ Burst of data is transferred before CPU takes the control of the buses back.
→ Burst of data could be → Entire data
→ Burst of blocks

2) Cycle Stealing
→ Slow IO devices takes some time to prepare the word
→ During this time, the CPU keeps control of the buses.
→ Once word is ready, CPU gives the control of the buses to DMAC for 1 cycle in which prepared word is transferred to memory.

3) Interleaving DMA
→ Whenever CPU does not require system bus (doing internal work), then only control of the buses will be given to DMAC.
→ CPU will not be blocked due to DMA.
→ Maximum time required for data transfer as compared to burst & cycle stealing mode.
→ Time required for data transfer: Interleaving > Cycle stealing > Burst Mode
→ Speed of data transfer: Burst Mode > Cycle Stealing > Interleaving
→ DMAC is a special purpose processor, which controls data transfer b/w memory and I/O (because it generates add. & control signals for memory).
→ DMA could work even when instruction is executing.

COA Notes By Gurjit Singh