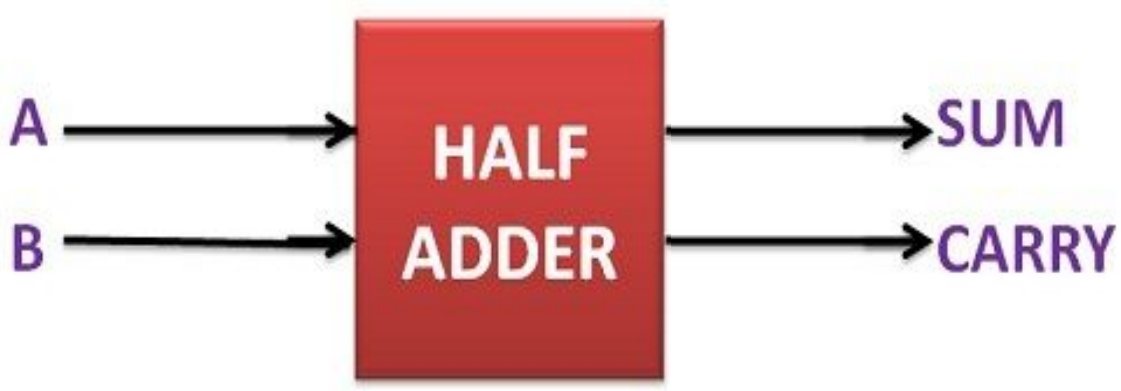
Module 3
Processor Organization & Architecture

Prepared by:

Prof. Datta Deshmukh

Half adder: Definition and Block diagram

- Performs the addition of two single bit numbers (A and B) and generates two outputs (Sum and Carry)
- From left side, the inputs 'A' & 'B' are applied, while to the right side, the outputs 'SUM' and 'CARRY' are obtained as shown below:



Half adder: Truth table, Equations and Logical diagram

- For two inputs 'A' & 'B', the 'Sum' & 'Carry' outputs are shown below in the Truth table:

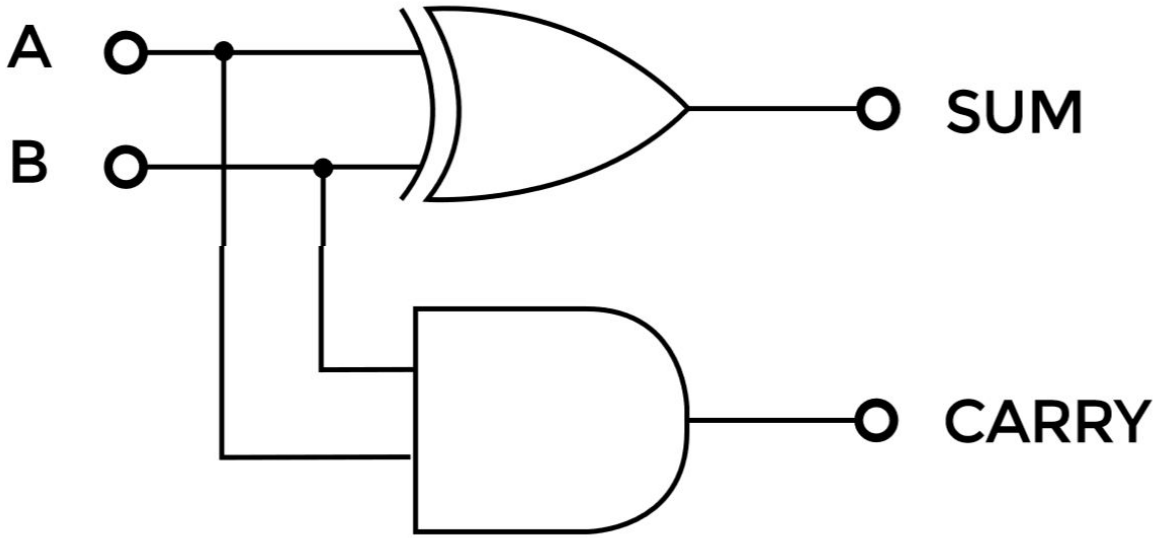
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The output equations are:

SUM = A~.B + A.B~ = (A) **EXOR** (B)

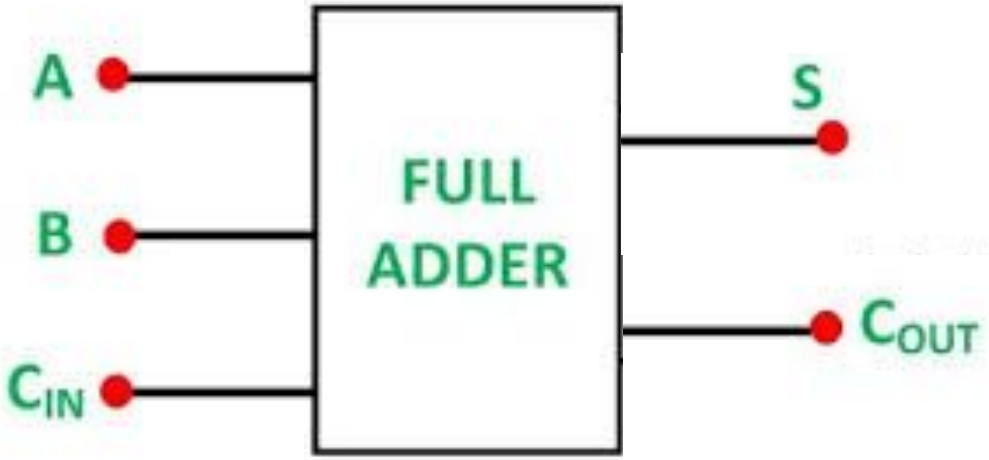
CARRY = A.B

Logical diagram of Half adder is:



Full adder: Definition and Block diagram

- Performs the addition of three single bit numbers (A, B and C_{in}) and generates two outputs (Sum and Carry Output)
- From left side, the regular inputs 'A' & 'B' alongwith input carry 'C_{IN}'are applied, while to the right side, the outputs sum (S) and carry output (C_{OUT}) are obtained as shown below:



Full adder: Truth table & Output equations

INPUTS			OUTPUTS	
A	B	C_{in}	SUM	CARRY OUT
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

$$\text{Sum} = \overline{A}\,\overline{B}\,C_{in} + \overline{A}\,B\,\overline{C}_{in} + A\,\overline{B}\,\overline{C}_{in} + A\,B\,C_{in}$$

$$= (A) \text{ EXOR } (B) \text{ EXOR } (C_{in})$$

$$\text{Carry} = \overline{A}\,B\,C_{in} + A\,\overline{B}\,C_{in} + A\,B\,\overline{C}_{in} + A\,B\,C_{in}$$

$$= A\,B + A\,C_{in} + B\,C_{in}$$

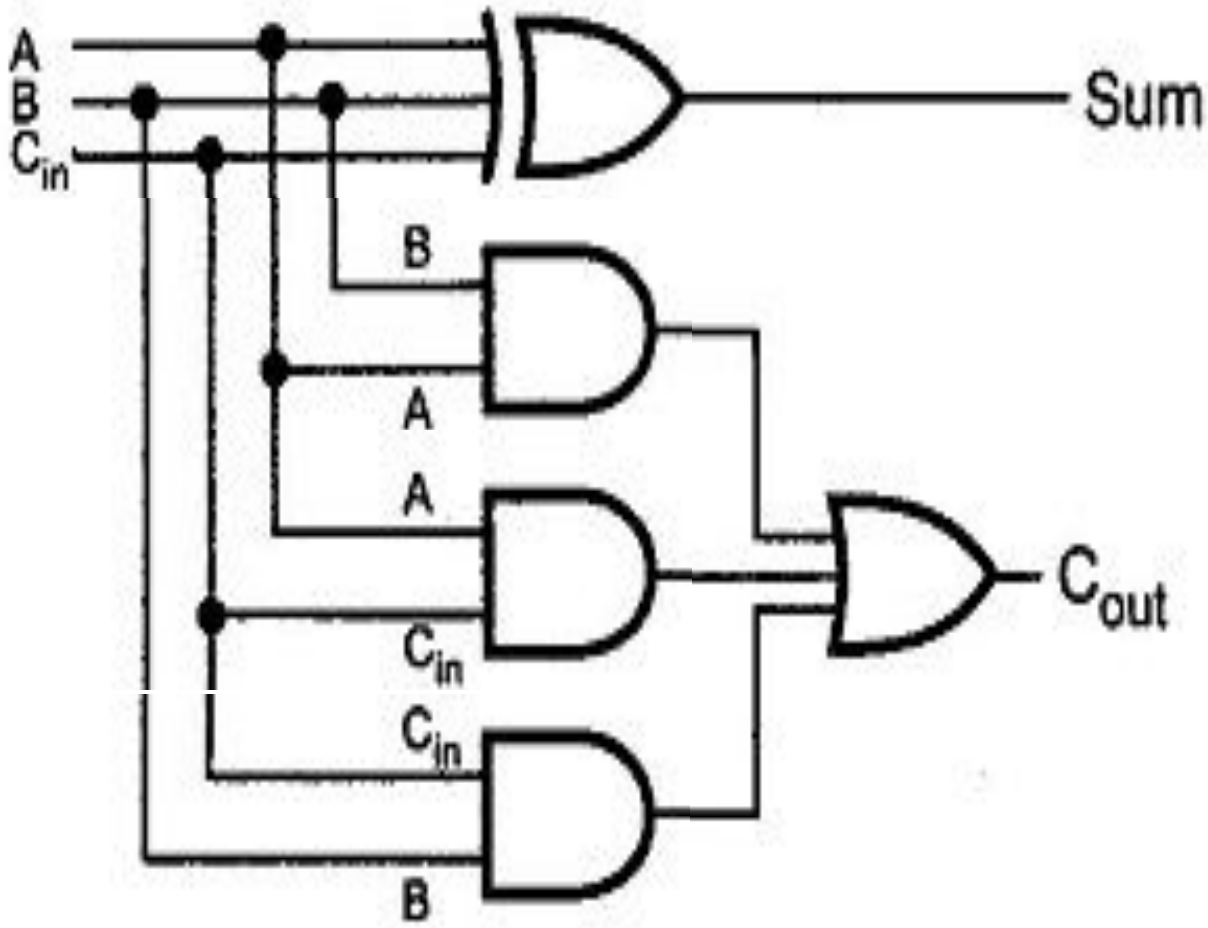
Full adder: Equations and Logical diagram

$$\text{Sum} = \overline{A}\,\overline{B}\,C_{in} + \overline{A}\,B\,\overline{C}_{in} + A\,\overline{B}\,\overline{C}_{in} + A\,B\,C_{in}$$

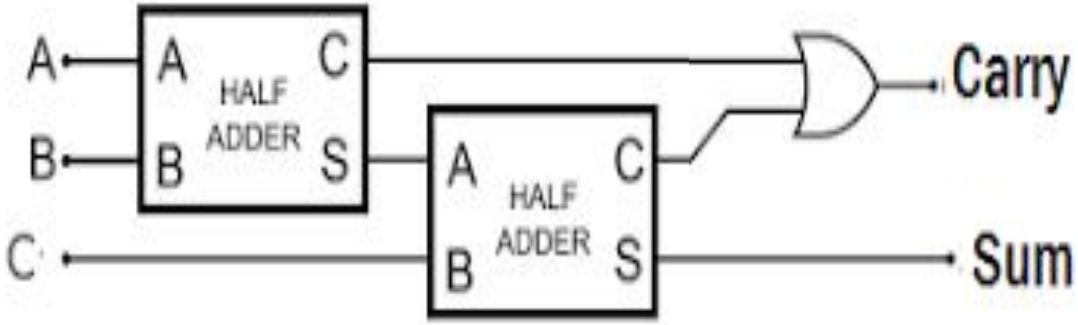
$$= (A) \text{ EXOR } (B) \text{ EXOR } (C_{in})$$

$$\text{Carry} = \overline{A}\,B\,C_{in} + A\,\overline{B}\,C_{in} + A\,B\,\overline{C}_{in} + A\,B\,C_{in}$$

$$= A\,B + A\,C_{in} + B\,C_{in}$$



Full adder design by cascading two half adders



- First Half Adder (left side) performs addition of two single bit numbers: 'A' & 'B'
- Second Half Adder (middle) performs the addition of sum bit of the first half adder with the carry input 'C'
- Second Half Adder generates the final 'Sum' output of the Full adder
- The carries generated by both the adders are merged together by 'OR' gate to generate the final output 'Carry'
- The block diagrams can be replaced by the logical diagrams of the half adders

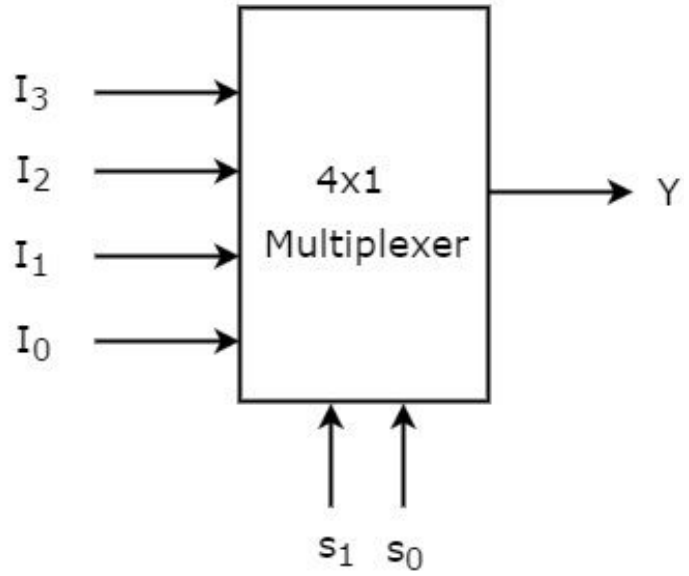
Multiplexer: The general concepts

- Multiplex means 'Many to one'
- Hence, Multiplexer (MUX) has many inputs and one output line
- MUX has 'n' select lines, used to select one out of '2^n' inputs and the logic on the selected input line is steered on the single output line

Common MUXs include:

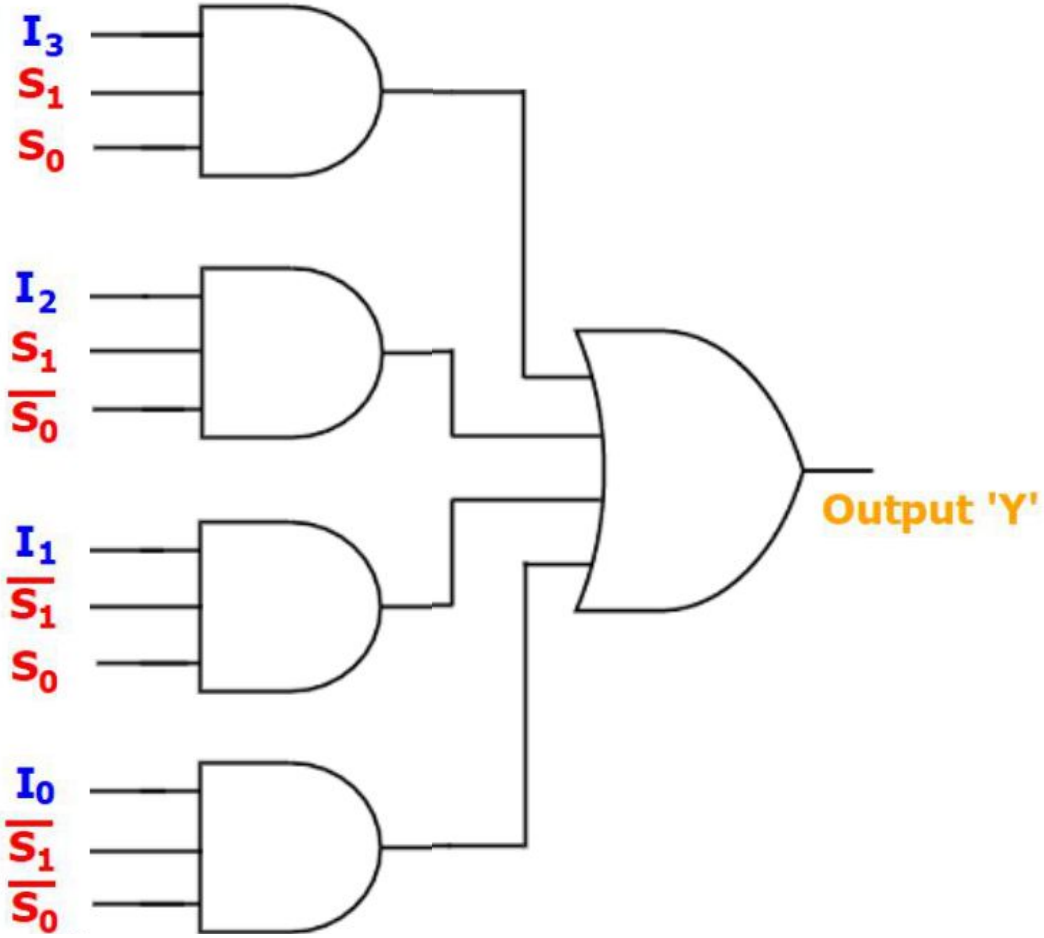
1. 4:1 MUX (4 inputs, 1 output and 2 select lines)
2. 8:1 MUX (8 inputs, 1 output and 3 select lines)
3. 16:1 MUX (16 inputs, 1 output and 4 select lines)

4:1 MUX : Block diagram, Truth table & Logical diagram



Truth table of 4:1 MUX

Select lines		Output
S1	S0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



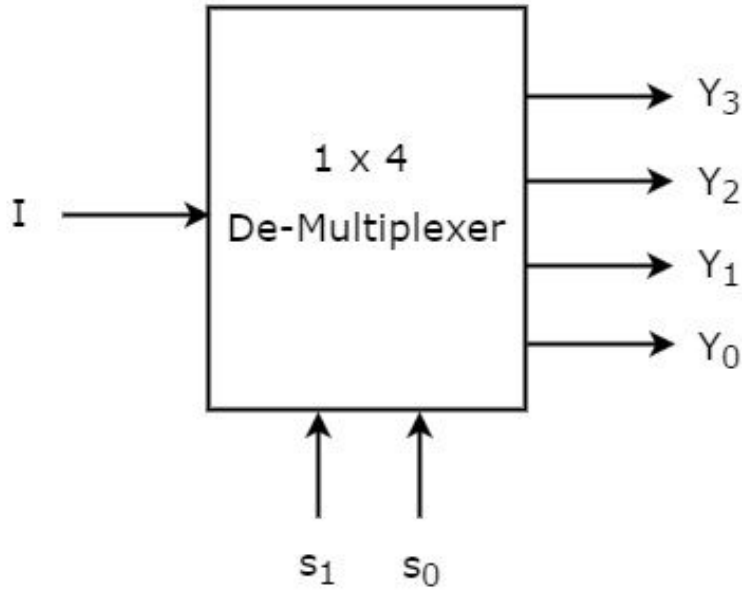
Demultiplexer: The general concepts

- Demultiplex means ' One to Many'
- Hence, Demultiplexer (DEMUX) has one input and many output lines
- DEMUX also has 'n' select lines, used to select one out of '2^n' output lines and the logic on single input is steered to the selected output line

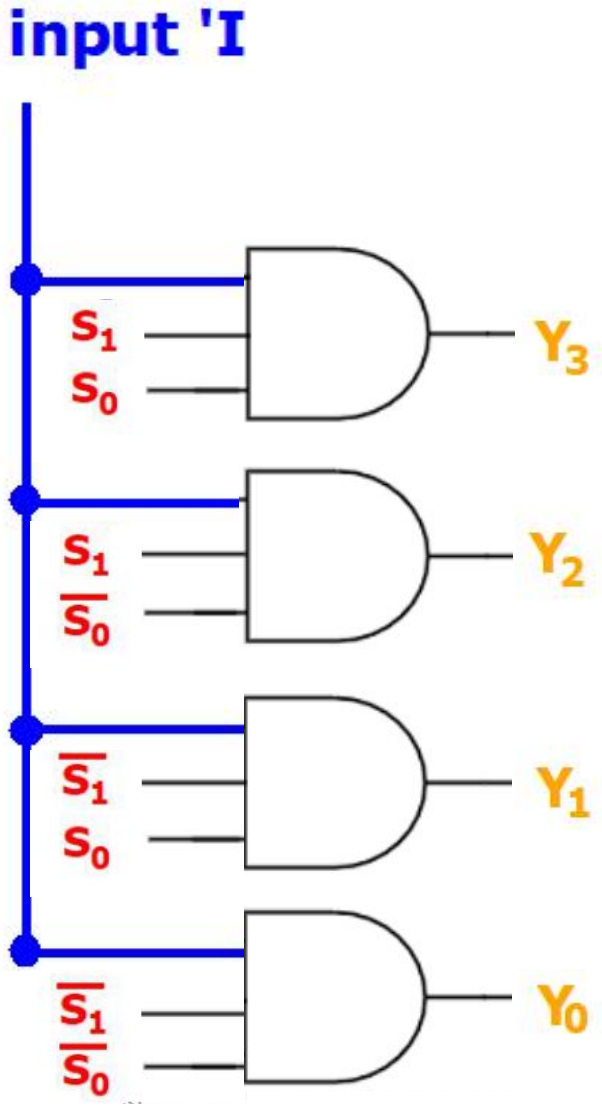
Common DEMUXs include:

1. 1:4 DEMUX (1 input, 4 outputs and 2 select lines)
2. 1:8 DEMUX (1 input, 8 outputs and 3 select lines)
3. 1:16 DEMUX (1 input, 16 outputs and 4 select lines)

1:4 DEMUX : Block diagram, Truth table & Logical diagram



Select lines		Outputs			
S1	S0	Y3	Y2	Y1	Y0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0



Encoders: Theory and types

- Encodes the inputs into the output coded data
- When any one of the input is activated, the output code is generated

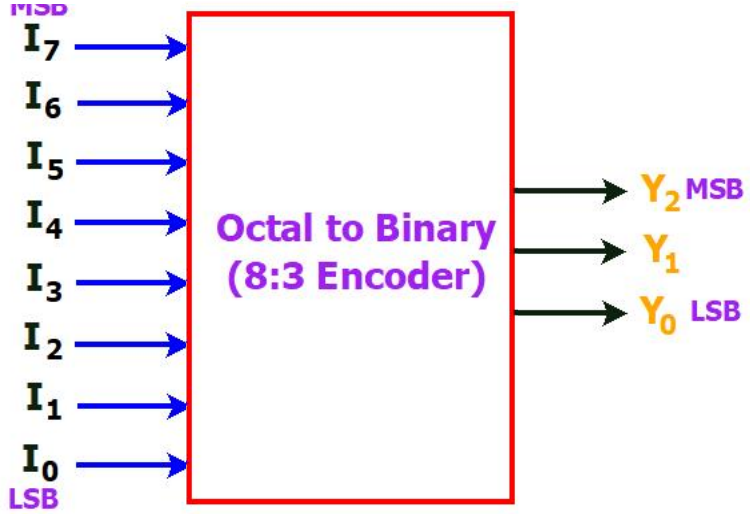
Octal to Binary encoder (8:3 encoder)

When one of the input (out of 8 inputs) is activated, then 3 bit binary sequence number of the activated input is generated

Decimal to BCD encoder (10:4 encoder)

When one of the input pertaining to decimal values '0' to '9' is activated, then the output is 4 bit BCD equivalent of that activated input

8:3 (Octal to Binary) encoder: Block diagram to Design



Truth Table

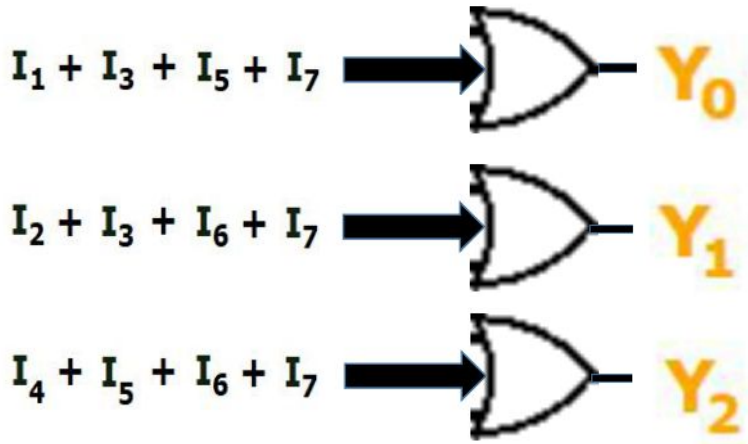
Inputs								Outputs		
MSB I_7	I_6	I_5	I_4	I_3	I_2	I_1	LSB I_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Output Equations:

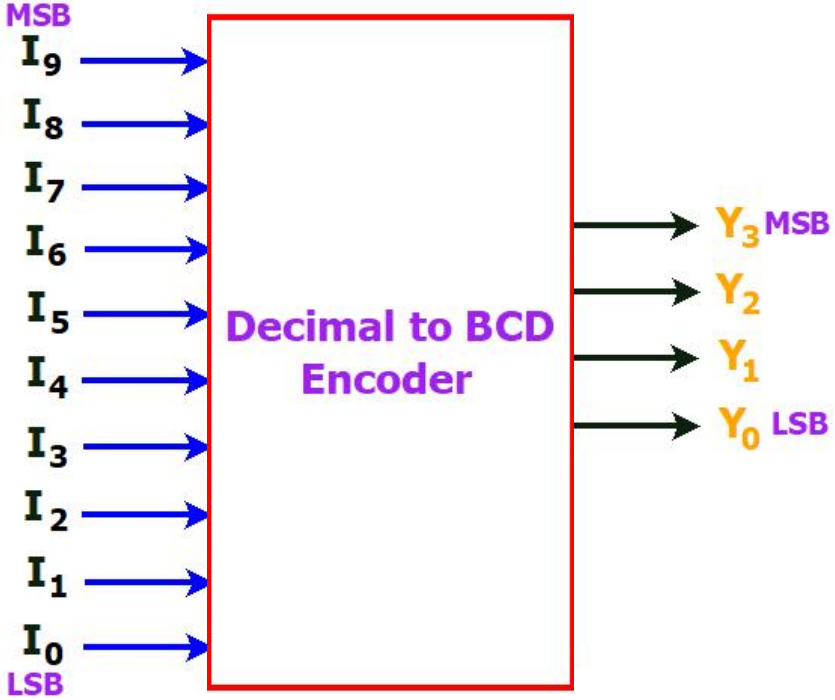
$$Y_0 = I_1 + I_3 + I_5 + I_7$$

$$Y_1 = I_2 + I_3 + I_6 + I_7$$

$$Y_2 = I_4 + I_5 + I_6 + I_7$$



Decimal to BCD encoder: Block, Truth table, Equations



Truth Table

Inputs										Outputs			
MSB I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	LSB I_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

Output Equations:

$$Y_0 = I_1 + I_3 + I_5 + I_7 + I_9$$

$$Y_1 = I_2 + I_3 + I_6 + I_7$$

$$Y_2 = I_4 + I_5 + I_6 + I_7$$

$$Y_3 = I_8 + I_9$$

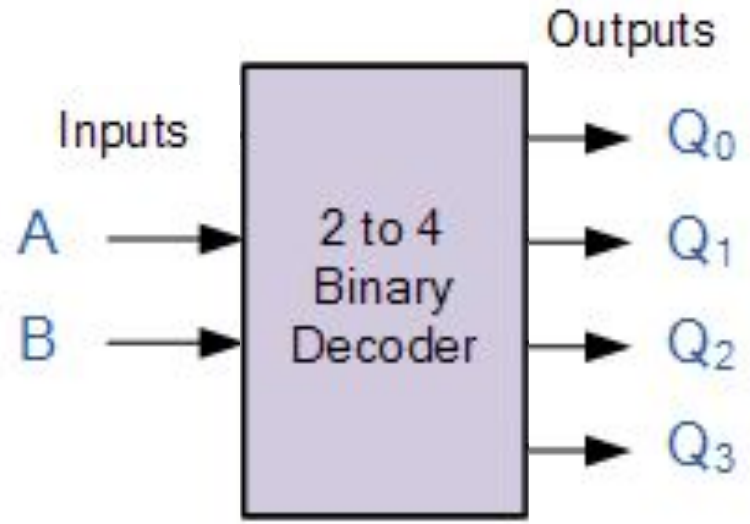
Decoder: Theory and types

- Accepts the coded information as the input and activates one particular output line
- Normally, the input coded information is in Binary

Decoders are of the following types:

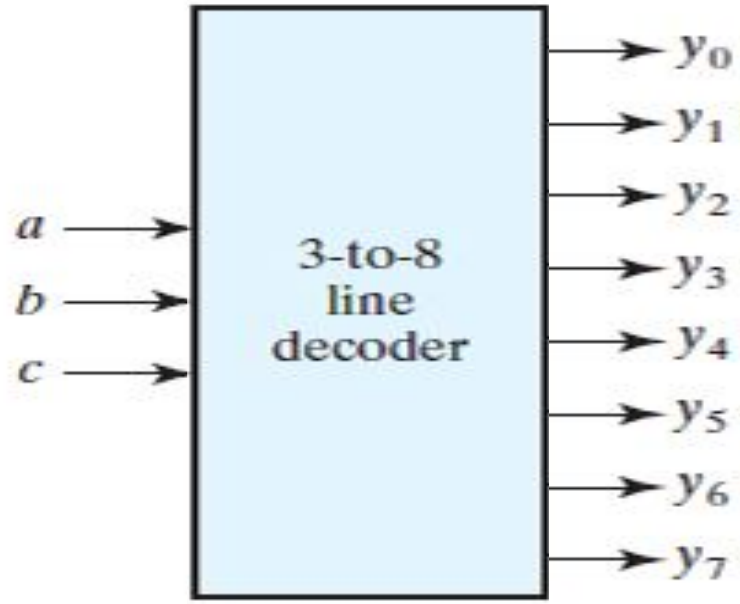
1. 2:4 Decoder (2 inputs and 2^2 = 4 outputs)
2. 3:8 Decoder (3 inputs and 2^3 = 8 outputs)
3. 4:16 Decoder (4 inputs and 2^4 = 16 outputs)
4. BCD to Seven Segment Decoder (4 bit BCD input & 7 segment outputs)

2:4 and 3:8 Binary Decoders with active high outout (Block diagram & Truth table)



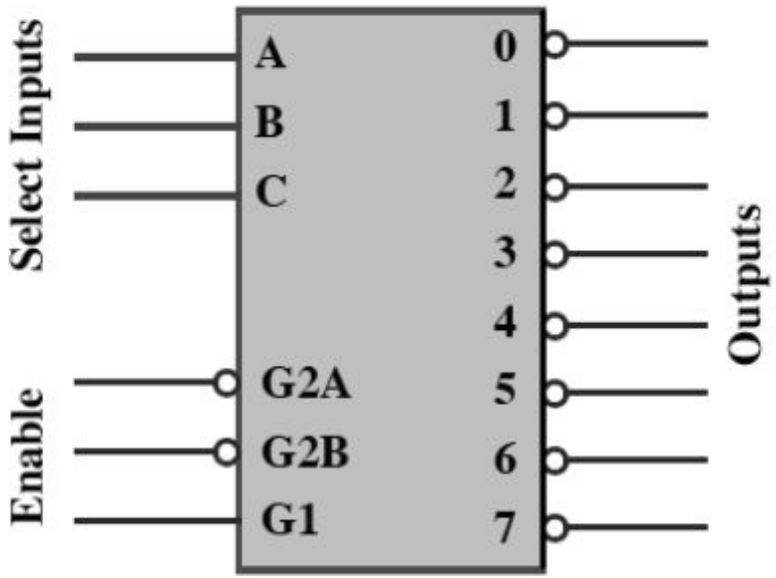
Truth Table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



a	b	c	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Practical 3:8 Decoder with active low output (IC 74 LS 138)

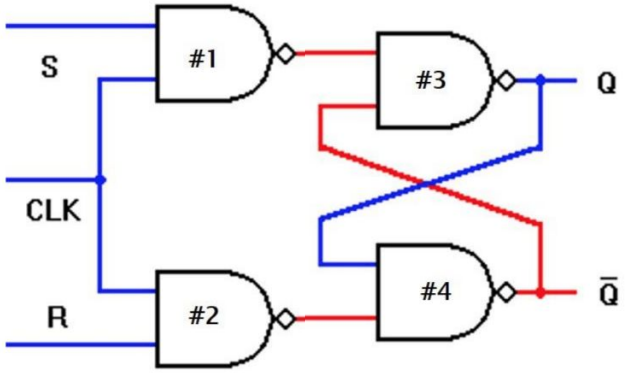


Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$
0	0	1	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0

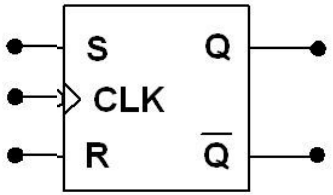
Clocked SR flip-flop: Design, Symbol, Truth table & Operation

Designed using SR latch (NAND gate #3, #4) and Steering logic (NAND #1, #2)

Logical design & symbol:



Holds 1 bit info.



When CLK = 0:

O/Ps of NAND#1 & #2 = 1. The latch (NAND#3, #4) maintains **previous state**

When CLK = 1:

Depending upon S & R inputs, **proper output obtained as per the truth table**

Truth table (with CLK present, CLK = 1)

S	R	Q	State
0	0	Previous State	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Forbidden

When S = R = 0: #1 = #2 = 1. Hence, #3 & #4 maintains previous state

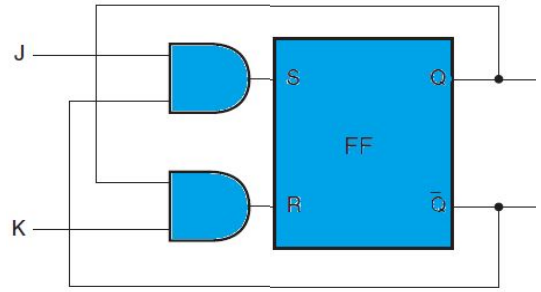
When S = 0, R = 1: #1 = 1, #2 = 0. It makes, Q~ = 1. Hence, Q = 0. Output is Reset.

When S = 1, R = 0: #1 = 0, #2 = 1. It makes, Q = 1. Means, Output is Set.

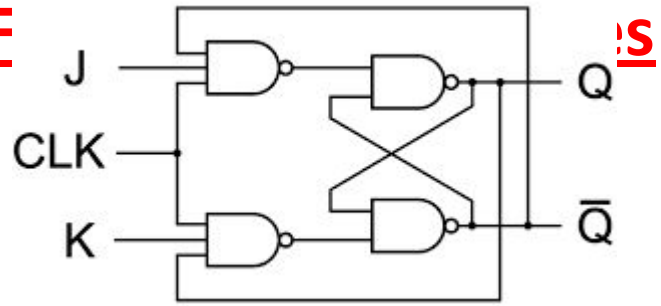
When S = R = 1: Makes #1 = #2 = 0. Hence, #3 = #4 = 1 simulatneously (impossible)

JK flip-flop: Design, Symbol, Operation & Truth table

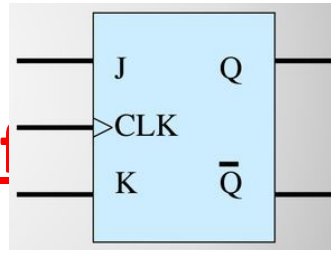
Using SR flipflop & feedback



JK F...s



Symbol of...



Operation of JK flip-flop:

When J = K = 0: Makes S = R = 0. Hence, previous state is maintained.

When J = 0, K = 1: Makes S = 0 & hence, Q = 0

When J = 1, K = 0: Makes R = 0, Q~ = 0. Hence, Q =1

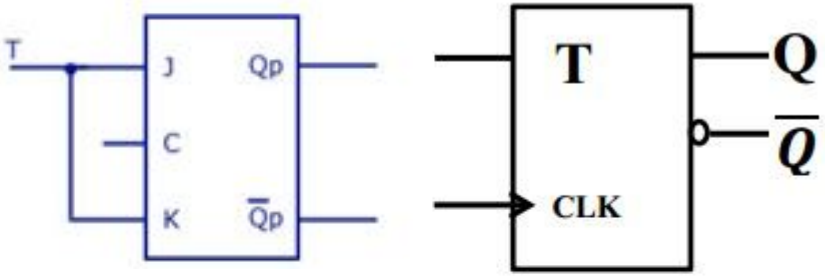
When J = K = 1: Output toggles (changes state)

Truth table of JK flip-flop:

J	K	CLK	Q
0	0	↑	Q_0 (no change)
1	0	↑	1
0	1	↑	0
1	1	↑	$\bar{Q}_0$ (toggles)

Toggle (T), Delay (D) using JK flip flop: Design, Truth table

T flip flop using JK flip flop & symbol



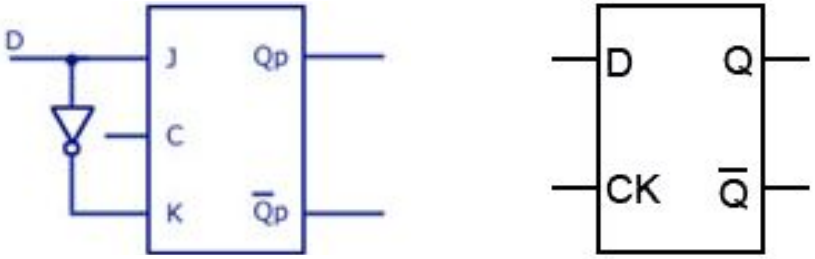
Truth table of T flip flop

Input	Outputs	
	Present State	Next State
T	Q_n	Q_{n+1}
1	0	1
1	1	0

When T = 0, maintains previous state

T flip flops are used in counter designs

D flip flop using JK flip flop & symbol



Truth table of D flip flop

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

D	Q	$\overline{Q}$
0	0	1
1	1	0

Delay flip flop transfers the logic of D input to the output after 1 CK pulse (generates a delay of 1 CK)

Used in the design of digital delay lines

CPU (Processor) Register Organization

- 2 major types of registers:
  - User visible registers
  - Control & status registers
- User visible registers:
  - These are General Purpose Registers (GPRs) which hold data at the time of processing.
  - Some of the registers are also used to address memory.
- Control & status registers:
  - Used to control the operation of the processor.
  - Most of these registers are not visible to the user

User visible (available) registers of the Processor

- User visible address registers include segment registers, Index registers, Stack Pointer
- Segment register:
  - Holds the base address of the segment.
  - Multiple segment registers such as data segment register, code segment register, stock segment register etc.
- Index registers: Used for indexed addressing
- Stack pointer: Points the "Top of the stack" in execution of
  - PUSH, POP instructions
  - Procedure calls
  - Interrupt processing

Control & Status registers of the Processor

- **Condition codes or flags:** Individual bits that are set/reset by the CPU as result of operation
- **Program counter (PC):** Points the address of the next instruction to be fetched
- **Instruction Register (IR):** Contains the Op-code of the instruction to be executed
- **Memory address register (MAR):** Points memory address while fetching data/operand
- **Memory buffer Register (MBR):** Holds the data to be read from or written into the memory

Instruction format

- Instruction format depends upon type of the instruction

Addressing modes of the generic Processor

Defines the method to refer data/operand

Common addressing modes:

1) Immediate addressing mode
2) Direct addressing mode
3) Indirect addressing mode
4) Register addressing mode
5) Register Indirect addressing mode
6) Displacement addressing mode
7) Stack addressing

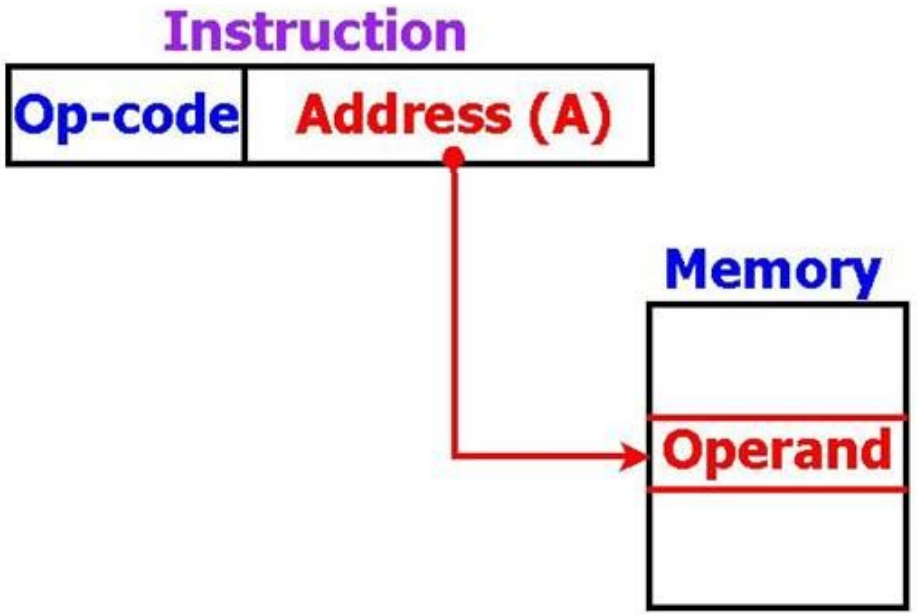
Immediate Addressing mode

- Data defined as immediate number within instruction
- Example: Intel 8085 instruction, ADI 20

  Adds data (20)H to accumulator

Direct Addressing mode

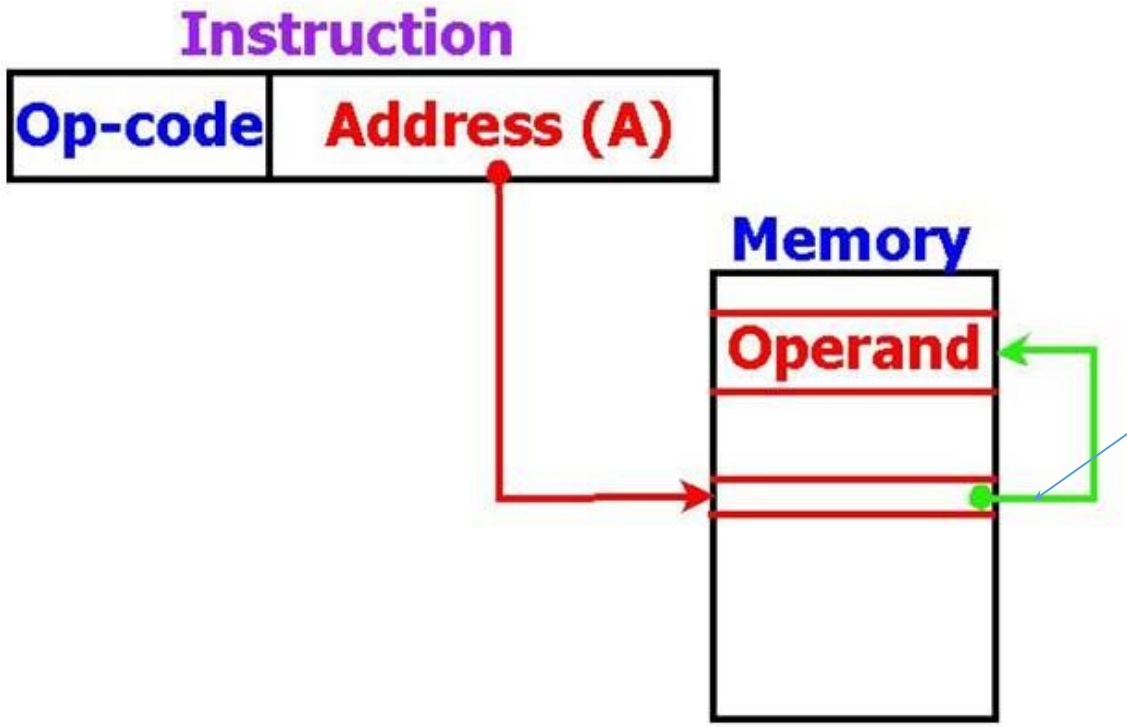
- Address of operand (data) is defined directly within instruction



- Example: Intel 8085 instruction : LDA 1000
  Loads 8 bit data from address (1000)H to accumulator

Indirect Addressing mode

Bracket indicates indirect addressing



Mov A, [3000h]

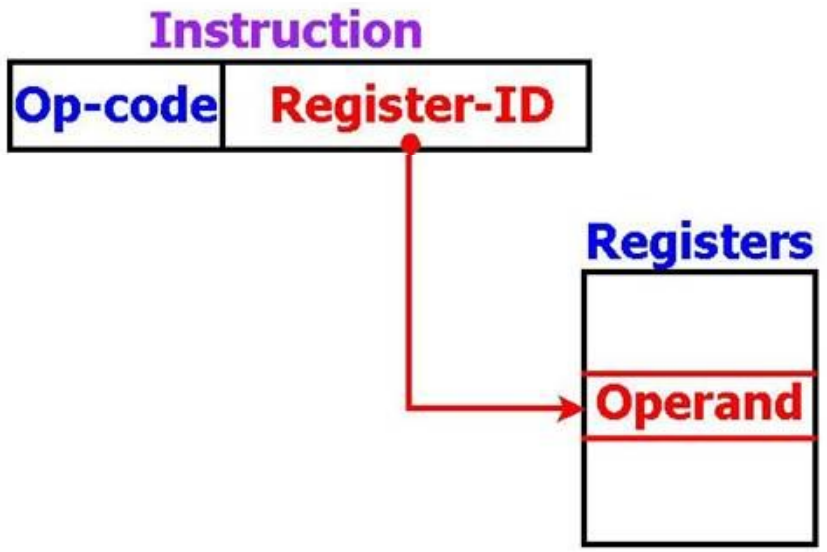
H = 20h L = 00h
2000h

- Address “A” points memory location
- This memory location points another memory location that has the operand

Register Addressing mode

- Operand is defined in register
- Register-ID is defined within the instruction

MOV A, L



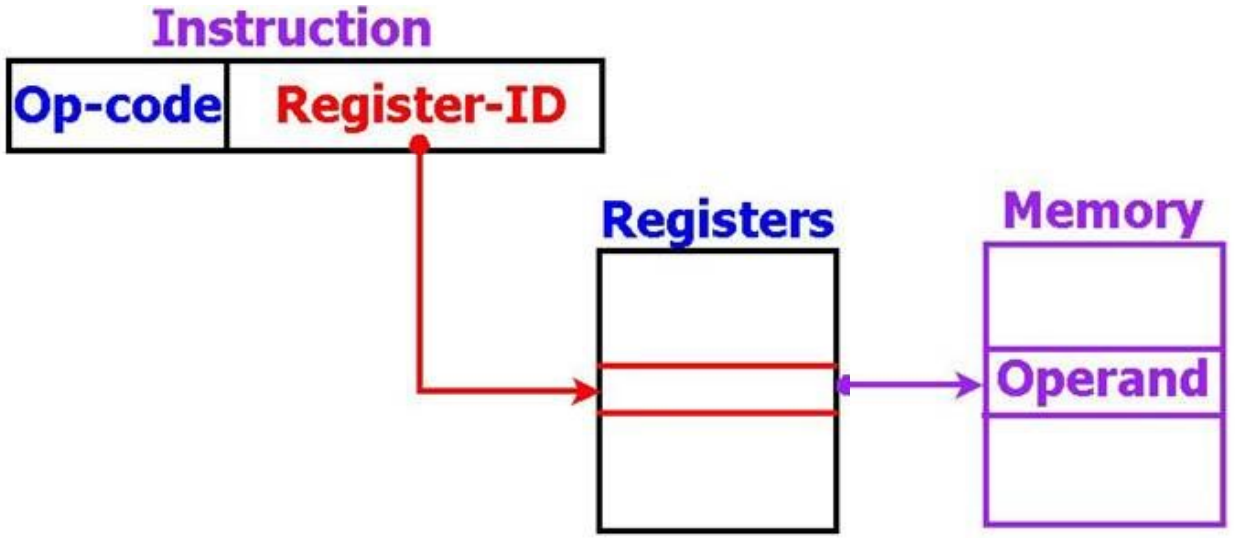
- Example: Intel 8085 instruction, ADD B

  Adds register B contents to accumulator

Register Indirect Addressing mode

- Register or register pair defines the address of the operand

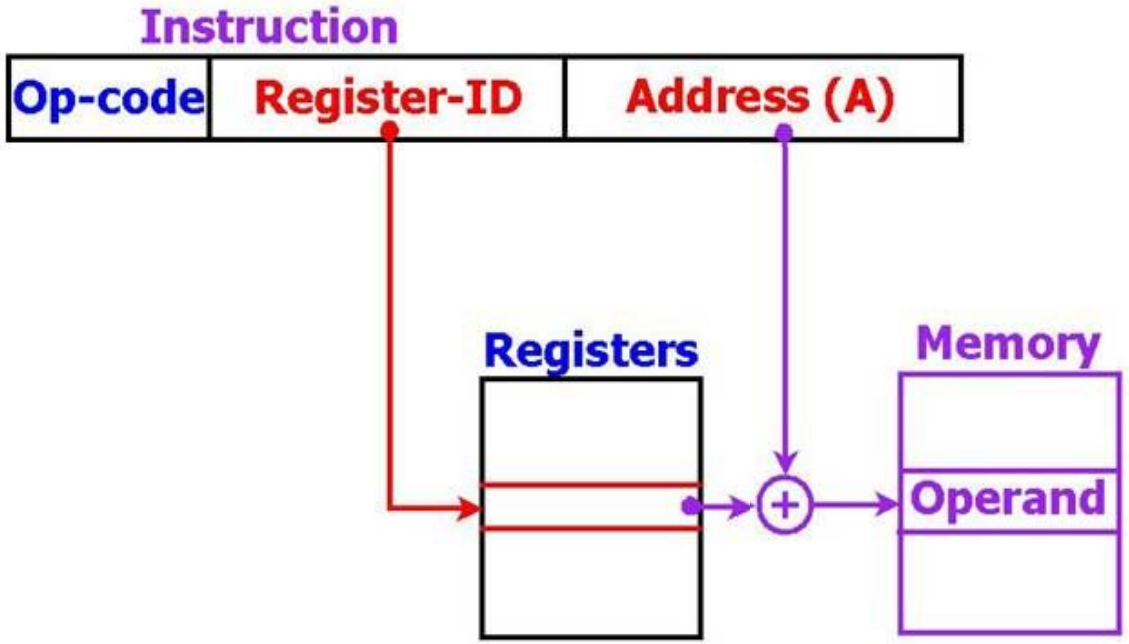
Mov a, [HL]



- Example: Intel 8085 instruction, ADD M
  Adds contents of memory location
  (pointed by HL register pair)to accumulator

Displacement Addressing mode

- Address of the operand = Contents of register + Offset address (A)
- Both Register-ID & Offset address (A) are defined within the instruction



Stack Addressing mode

- Data stored on stack memory
- Stack memory is pointed by stack pointer
- Data accessed by using 2 instructions:
  - PUSH: Data copied onto stack
  - POP: Data retrieved out of stack

Instruction formats of Intel 8085 Processor

Single Byte format

- It contains only one byte i. e. Op-code byte.
- Op-code may specify operation alongwith the register-ID
- Register addressing mode or indirect addressing uses single byte format in 8085 Processor

Two Byte format

- It contains two bytes: First byte is op-code & second byte contains immediate 8 bit data or I/O port address
- Immediate addressing mode uses this format

Three Byte format

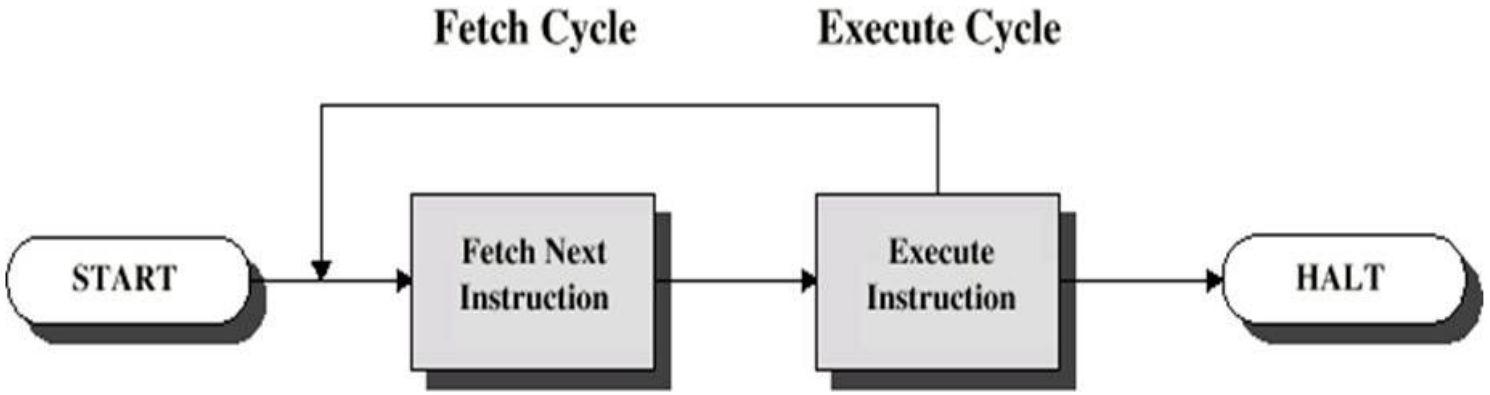
- It contains three bytes: First byte is op-code, Second byte is lower byte of 16 bit data or address & third byte is upper byte of 16 bit data or address
- Direct addressing or Immediate addressing (for 16 bit data) uses this format

Instruction cycle of the Processor

It is the time required to execute one complete instruction

- **Instruction Cycle = Fetch cycle + Execute cycle**
- **Fetch: Reading instruction from memory**
- **Execute: Performing the operation**

I C = fetch + decode +execute



Fetch Cycle

- Program Counter (PC) holds the address of next instruction to be fetched
- Processor fetches instruction op-code from memory address pointed by PC
- Instruction op-code gets loaded into Instruction Register (IR)
- PC is automatically incremented to point the address of the next instruction

Execute Cycle

- Processor decodes/interprets instruction
- Decoding of the instruction is done by using internal instruction decoder
- Decoded instruction is then executed inside ALU (ALU performs the intended operation)
- Execution includes data processing operation (either arithmetic or logical operation on data)
- For every instruction, such fetch & execute cycles are repeated

Instruction Interpretation and Sequencing

Instruction Interpretation:

- Instructions are stored in the memory in the form an Op-code (Operation Code)
- Op-code is unique hexadecimal code for each assembly language instruction
- Op-code must be decoded by the Processor internally using Instruction Decoder
- Instruction decoding decides the action to be performed by the Processor
- Hence, Instruction Decoding = Instruction Interpretation
- After decoding the instruction, the internal control unit generates the control signals for execution of the instruction in the ALU section

Instruction sequencing:

- Performed by loading proper address in the Program Counter (PC) or similar register of the Processor
- Once loaded with address, the PC always points address of the next instruction to be executed and is auto-incremented after completion of current instruction