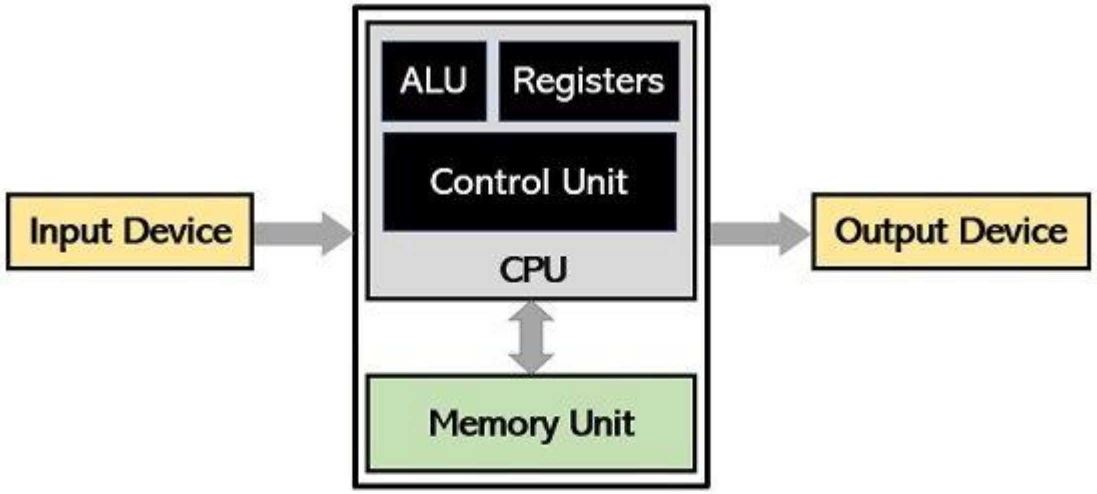
Unit -2

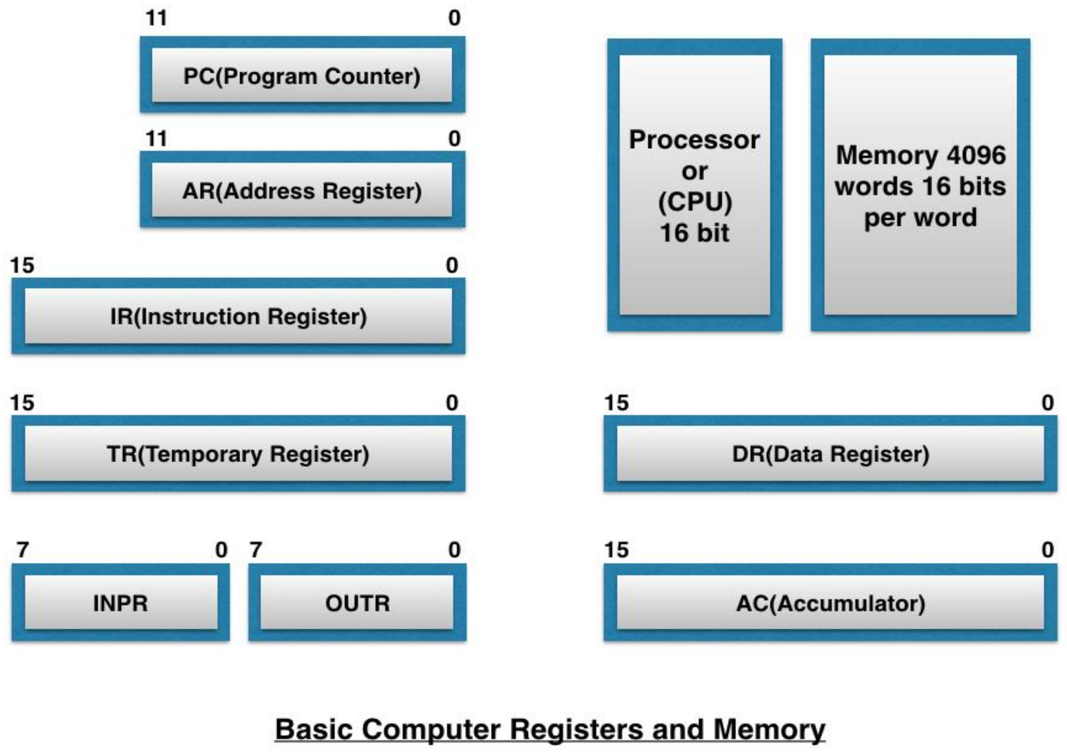
ALU Design

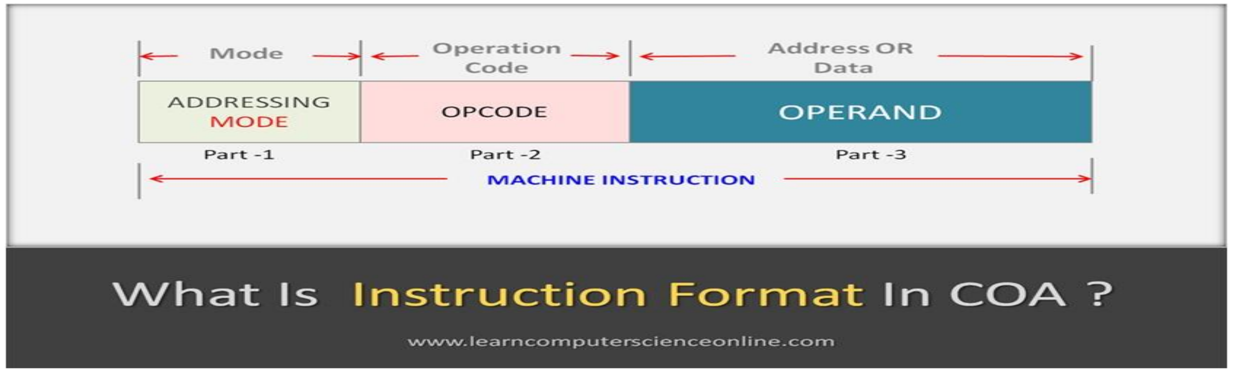
- Memory Unit –RAM
- RAM basically stores data(int a =10) and set of instructions (c=a+b).
- Input- keyboard, mouse,etc.
- Output- Monitor, Printer, etc.
- CPU- converts input to output.
- ALU- arithmetic and logical unit (consists of different types of circuits for performing operation such as ADD, SUB, MUL, DIV, , OR, Shift, etc.)
- Register- fastest memory available to store temporary data.
- Requirements of register is to pace up with CPU speed up. (Embedded on cpu)
- Register- Program counter, Accumulator, Input and output register, memory register
- Control unit – Timing and control signal
- Timing defines the order in which instructions are executed(for performing actions at particular time).
- Control signals enables read and write operation inside registers(how and which register to enables)
- Connection between them is using bus



Von Neumann Architecture

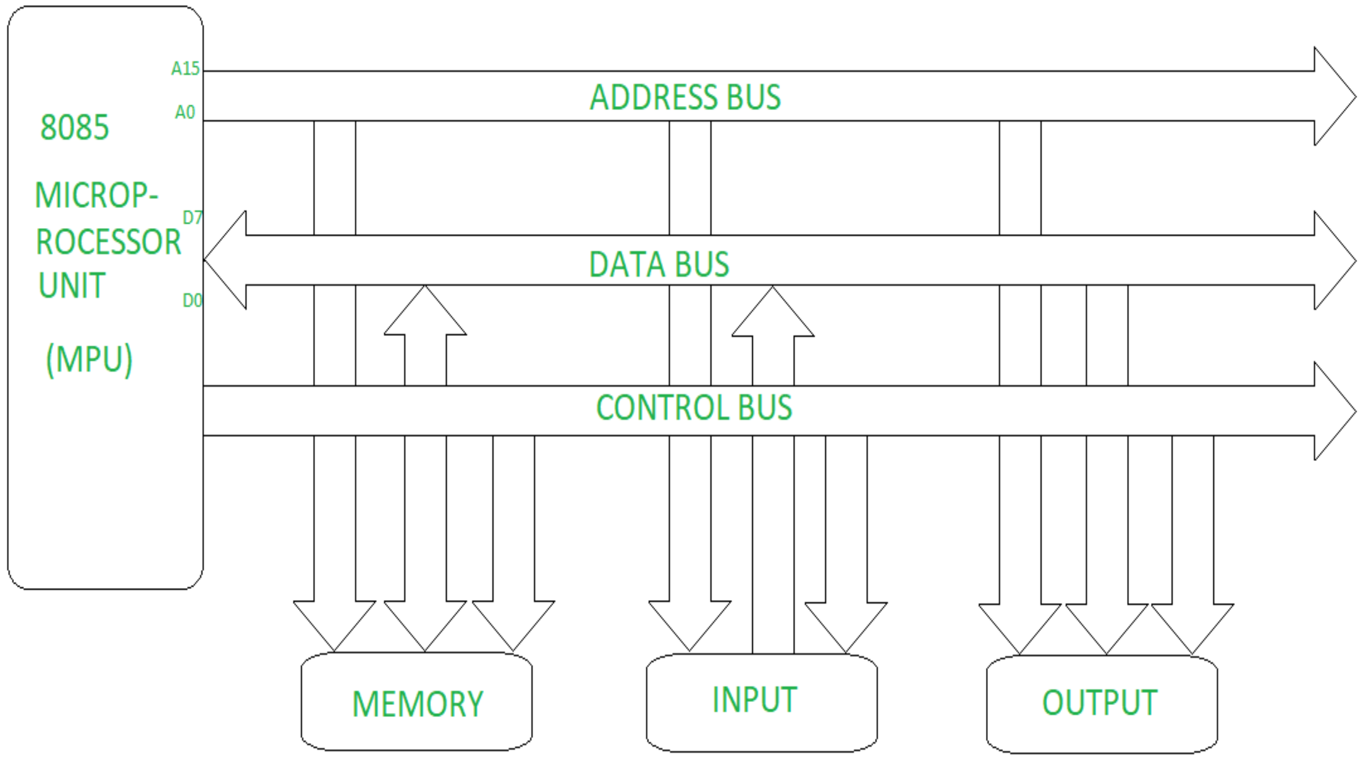
Different types of Register





- Generally, register is 16 bit. How to decide size of registers
- As in memory 4096 = number of words(number of slots)
- 16 = size of each word
- Word is memory representable unit.
- Memory – byte addressable (8bit)
- - word addressable (8, 16 , 32,64)
- With help of address register we fetch location of instructions/data in memory
- Size of address register =log(No. of words)=12 bit
- Data register is to store data
- Size of data register= size of each word = 16 bit
- AC is used for storing intermediate data (ALU)
- Size of AC= 16 bit
- PC is used to store address of next instruction
- Size of PC = 12 bit
- IR stores the instruction going to execute. (size=16)
- TR stores temporary data. (size=16)
- INPR to fetch data from input device.
- OUTR to display data at output devices
- Size of INPR and OUTR doesn’t depend on memory size

Types of Buses

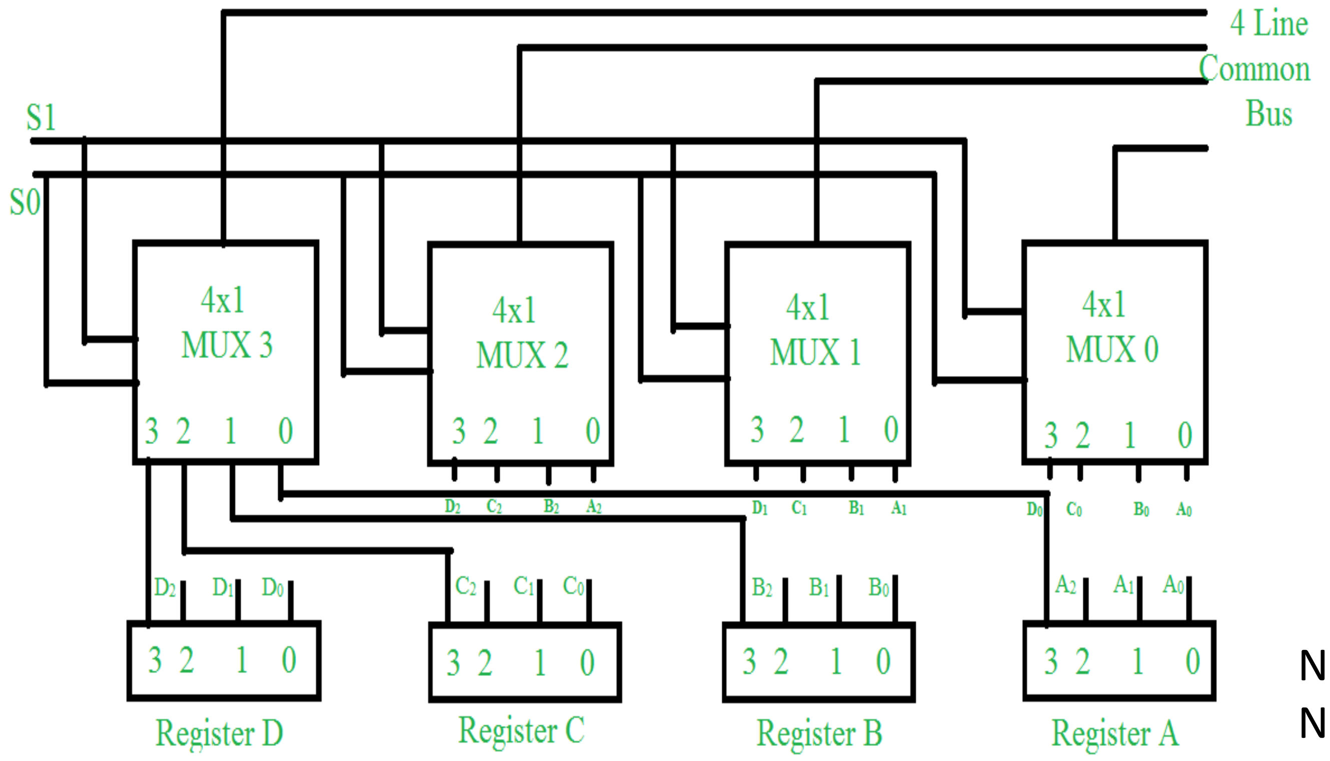


Bus organization system of 8085 Microprocessor

- Address bus carry the address of memory or I/O to from which data is fetched/stored. Here it is 16-bit address. Unidirectional and generated by CPU.
- Data bus is to carry data. It is bidirectional . The data is stored or transferred to/between register. It depends on word size
- Control bus are used to carry control and timing signal. Thus, it determine the sequence of instructions for desired output. It is performed by control unit. Generally, perform read and write operations
- E.g., load signal, increment, clear (LD, INC, CLR)

Common bus system using multiplexer

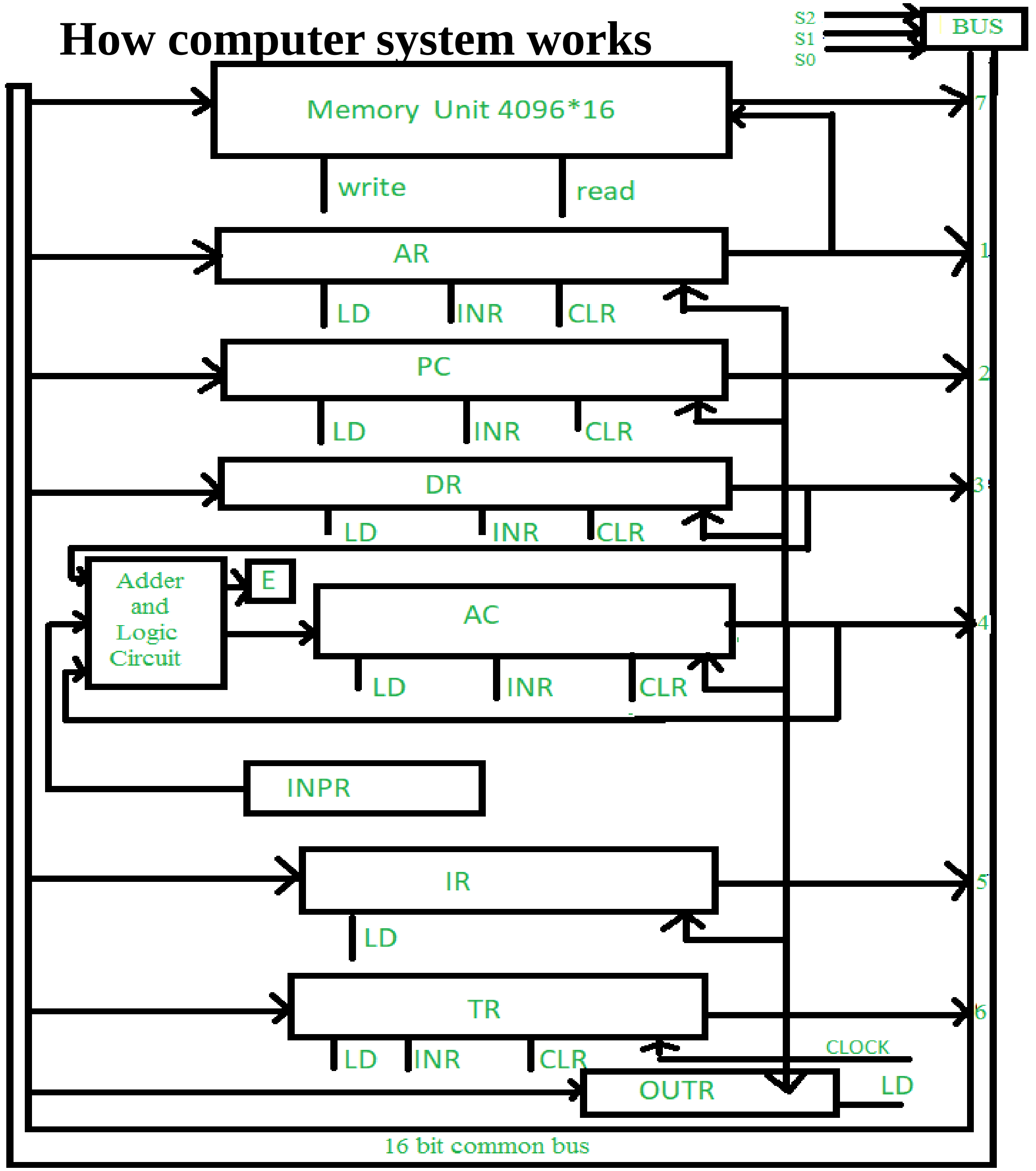
- How different register uses same bus for communication
- Consider 4-bit register



Select Lines combination S1 S0	Register Selected
00	Register A
01	Register B
10	Register C
11	Register D

Number of multiplexer= No. of bits in register
Number of i/p in multiplexer= No. of register

How computer system works



- Bus is implemented using multiplexer
- Number of select lines = log(N)
- 16-bit bus= word size
- Once the data is loaded in bus , the register to whom we want them input , we enable there LD=1.
- Start with PC, so S2S1S0=010. Now loaded address is passed to AR using LD=1.
- Now address is passed from AR to Memory using address bus.
- Now make S2S1S0=1 for passing data to common bus.
- Now LD=1 of DR enable data at DR, DR-> ALU and store result in AC.
- Now S2S1S0=100 for result on common bus and with LD=1 of OUTR, data is displayed at output devices

E.g. X= (A+B)

LOAD A	// AC <– M[A]
ADD B	// AC <– AC + M[B]

Types of Instructions

- Instruction = Binary sequence + operation
- In General computer it is mainly of 3 types
  - Data transfer instructions
  - Data manipulation instructions
  - Program control instructions
- Data transfer instructions are used to transfer or load instruction/data between register , register to memory.
  - E.g. LOAD, STORE, MOVE, PUSH, POP
- Data Manipulation instructions are used to perform operations on data.
  - E.g.- ADD, SUB, MUL, AND , OR, SHIFT
- Program control instructions are used to transfer control based on decision or flow of instruction.
  - E.g. IF, FOR, WHILE

Data Transfer Instructions

- For transferring data from/to register(R) , memory(M) and i/o unit (IO).
    - MOV R1, R2- data transfer from source(R2) to destination(R1)(copy)
    - MOV R1, 500 – Immediate addressing mode
    - The MOV instruction can be written in different ways depending upon the addressing mode.
- LD(LOAD) – load data from register (AC) to memory and vice-versa
- Store (STA) – generally data from register is passed to memory.
- Exchange(XCHG) R1,R2- for swapping data between R1 and R2.
- Input(In)- data from input device to memory.
- Output(OUT)- from memory to output device.
- Push and Pop- used when memory is used as stack. Push-data insert, Pop- data output.

Arithmetic Instructions

- Add –addition operation
- Sub- subtraction operation
- MUL- Multiplication operation
- DIV- Division operation
- INC- increment by constant
- DEC- decrement by constant
- Add with carry – add and store the carry for future references
- Sub with borrow- subtract with carry operation
- Negate- to represent number in negative number

<u>Instruction vs Microoperation</u>

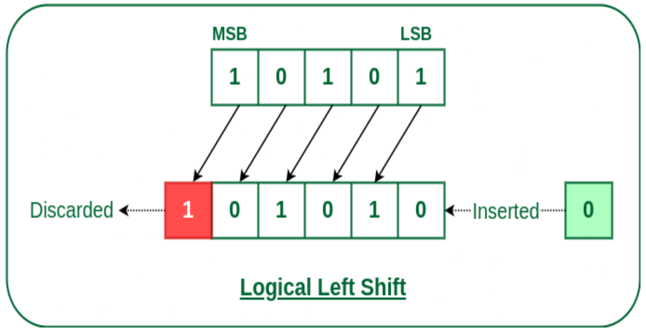
- Generally, execution of instruction consists of multiple microoperations
- For e.g. c= a+b add-instruction, while loading content and then storing along with addition constitute microoperation

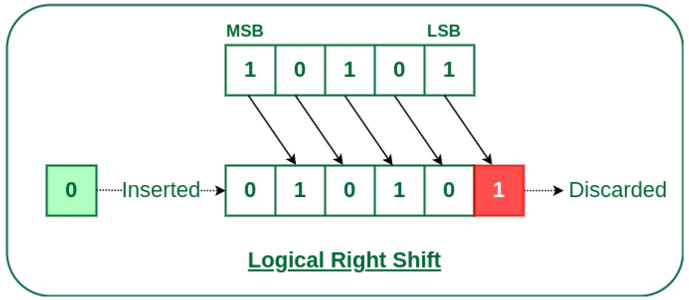
Logical Instructions

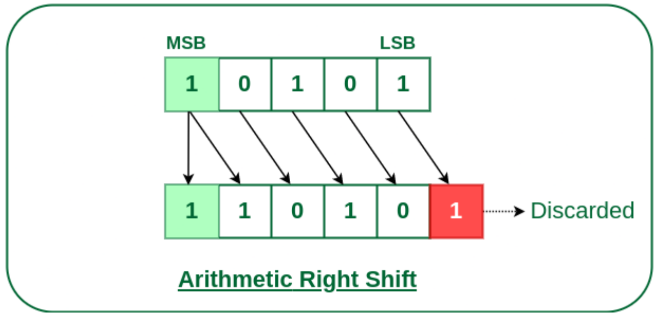
- Complement (COM or NOT) – 1 -> 0 and 0 -> 1
- Clear(CLR)- set all bits to 0. (With clock pulse)
- AND- perform logical and operations (.)
- OR – perform logical or operations (+)
- Ex-OR- perform Ex-OR operations on data
- Clear carry (CLRC)- to clear carry
- Set carry(STC)- carry bit set to 1.(used in 2’s complement calculation)
- CMC- complement carry
- EI- Enable interrupt or interrupt is occur.
- DI- Disable interrupt

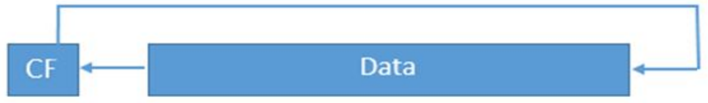
Shift Instructions

- Logical shift left- shift the bit left and replace LSB bit with 0. (multiplication operation)
- Logical shift right- shift the bit right and replace MSB bit with 0. (Division operation)
- Arithmetic shift right- applicable to signed number, replace the MSB bit with signed bit.
- Arithmetic shift left- shift the bit left and replace LSB bit with 0.
- Rotate right- same as logical shift right with MSB is replaced with LSB.
- Rotate left- same as logical shift left with LSB is replaced with MSB bit.
- Rotate right through carry- perform shift right by replacing carry with LSB bit
- Rotate left through carry- perform shift left by replacing LSB bit with carry



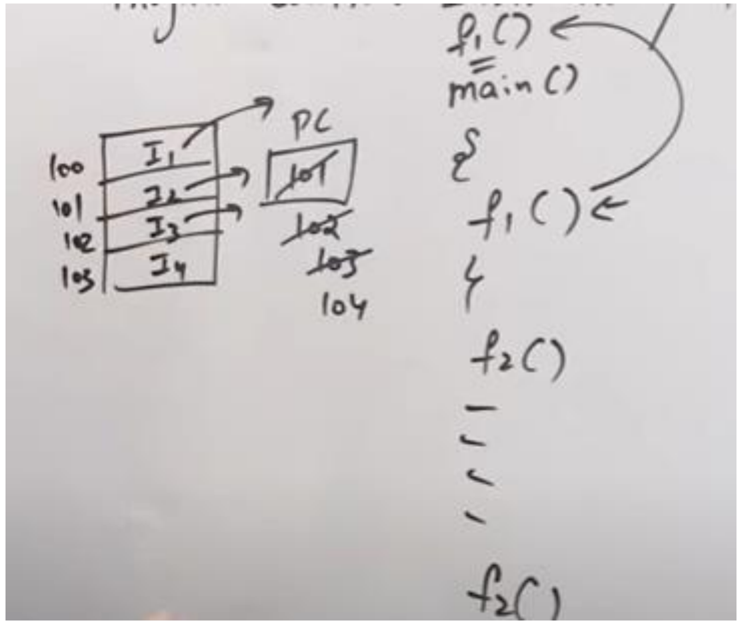






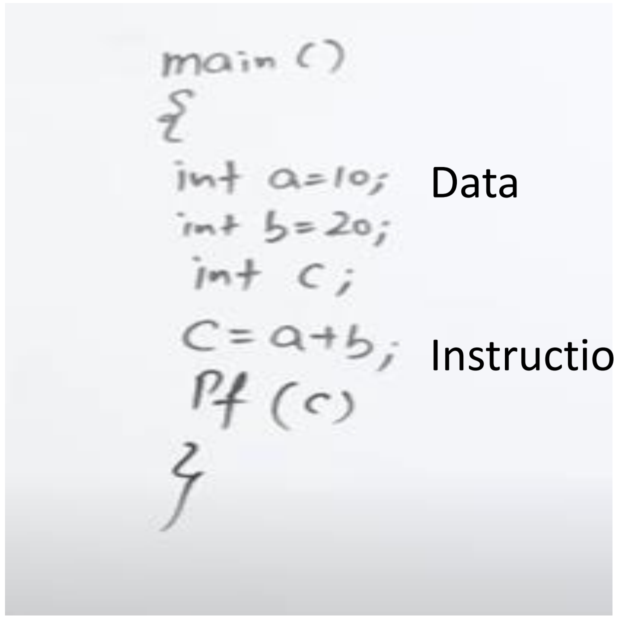
Program Control Instructions/ Transfer of control

- Generally, instructions are executed in sequential order.
- Branch instruction comes under this category
  - Unconditional – simply transfer PC to refer address
    - JMP 2000 , PC-> 2000
    - B 3000
    - SKP
    - CALL 3000
    - RETURN
  - Conditional- transfer control only when condition is true , if false proceed with next sequential address instruction
    - BE r1, r2 2000
    - BNZ r1 2000



Instruction format

- First program -> compiler -> machine language (set of instructions).
- Next step is to fetch instruction.
- Mode –addressing mode (operand in memory , register and data directly what means/treat)
- Opcode-types of operations
- Operand- address of data/ data directly/ address of address of data
- Memory -> IR (decode operation)
- Based on data and opcode, ALU perform operation and store/transfer data to OUTR.
- Size of instruction depends on the type of CPU organisation.
- Types of organisation- zero address, single accumulator, 2 address and 3 address instruction

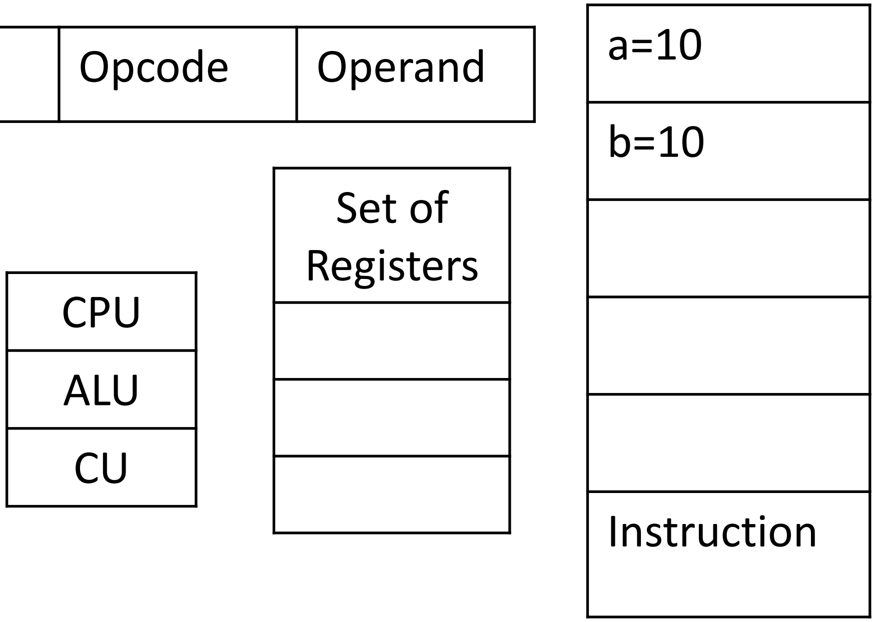


```
main ()
{
 int a=10;
 int b=20;
 int c;
 c=a+b;
 Pf (c)
}
```

Data

Instructions

Mode	Opcode	Operand



Types of CPU organisation

- It defines the size of instruction format.
- It is of three types- single accumulator, general register, stack organisation.
- Mode- from where operand is fetched(memory, register)
- Single accumulator is used in simple computer. Only few register (AC, PC, IR). Cheapest system as only one register. ALU -> AC (generally 16 bit).
- PC-> Memory -> IR -> AC -> ALU -> AC/OUTR.
- It supports one address instruction i.e., address of single operand.
- For e.g. X= A+B

  LD A(AC <- M[A])

  Add B (AC <- AC+M[B])

Mode	Opcode	Operand

General register organisation

- More register as compared to single accumulator, so cost is high.
- Registers- AC, PC, IR, set of registers.
- Supports 2 and 3 address instruction format.

2 address instruction			
Mode	Opcode	Destination	Source

3 address instruction				
Mode	Opcode	Destination	Source1	Source2

```
X= (A+B)* (C+D)
MOV R1, A
ADD R1, B
MOV R2, C
ADD R2, D
MUL R1, R2
STR X, R1
```

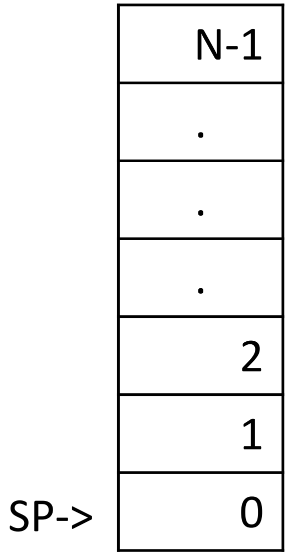
```
X= (A+B)* (C+D)
ADD R1, A, B
ADD R2, C,D
MUL X, R1, R2
```

Register Stack Organisation

- Here we set of registers (Beside AC, PC, IR,) and we organise them in form of stack.
- Stack pointer (address) keep of track of location of register where we perform the data insertion and deletion.
- Stack –PUSH and POP
- Supports zero address instructions

  (a+b)*(c+d)
- Fastest but costlier

Push a
Push b
Add
Push c
Push d
Add
MUL
P



Push
SP=SP+1
M[SP] <- DR

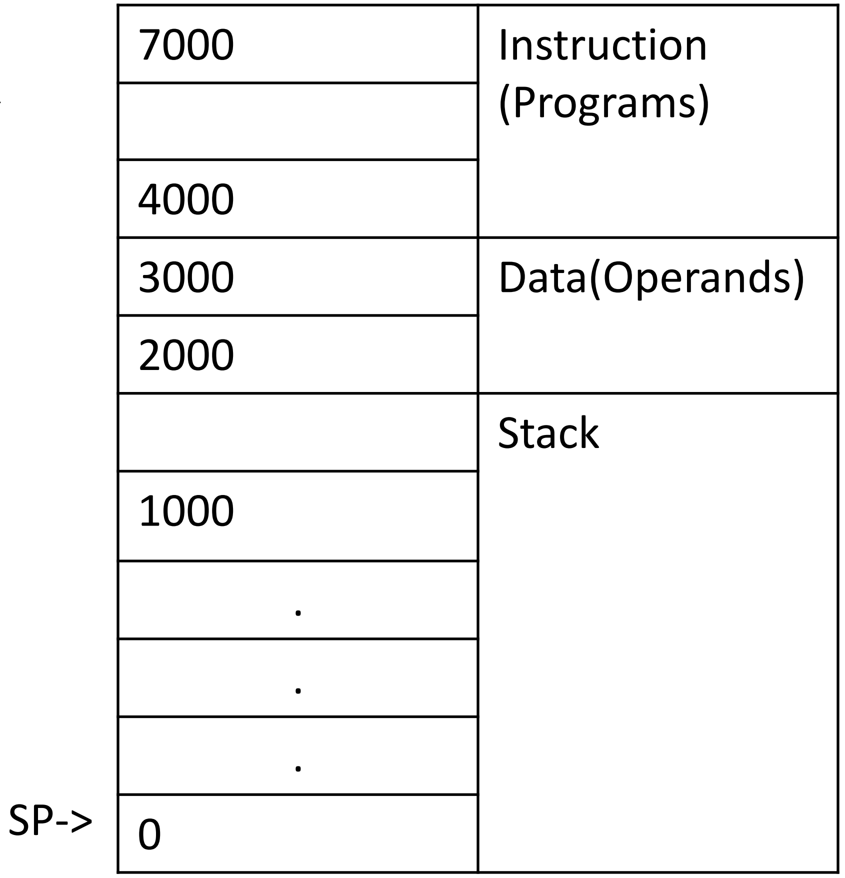
Pop
DR <- M[SP]
SP=SP-1

Full	Empty
0	1
1	0

Memory Stack Organisation

- Here we implement remaining section of memory as stack
- Also support zero address instruction
- Only difference is that it incur less cost
- Instruction set contains Push and Pop

Push a
Push b
Add



Addressing mode

Opcode	**Operand**

- Addressing mode is used for dealing with operands bits(how to treat them.)
- Operand – data, Opcode- types of operation. Data-register, memory
- Operand- data, address of data, address of address of data.
- Data- constant, variable. Data –directly, but for variables-we use address, register name.
- Addressing mode helps in decode where the variable is stored.
- Addressing mode help in reducing the instruction size.(Instead of storing data in memory, store data in register- Benefit).
- RISC have 5-6 addressing mode while CISC have more addressing mode.

- Implied addressing mode
- Immediate addressing mode
- Register addressing mode
- Register indirect addressing mode
- Auto-increment/decrement addressing mode
- Direct addressing mode
- Indirect addressing mode
- Relative addressing mode
- Indexed addressing mode
- Base register addressing mode

Implied Addressing mode

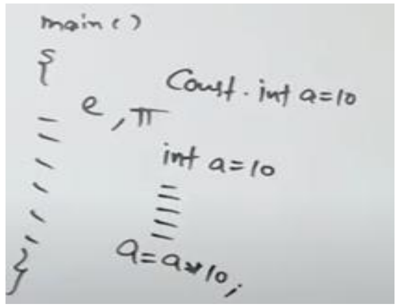
- Operand is specified implicitly in the definition of instruction.
- Use zero address and one address instructions.
- For e.g., INC A, push ,pop, CRC, CLA
- Source and destination in the instruction.
- Instruction size is very less

Register Direct mode

- Operand is present in the register.
- Register number is written in operand.
- For e.g., LD R1, AC<- R1
- EA= R1
- Instruction size depends on the number of register.

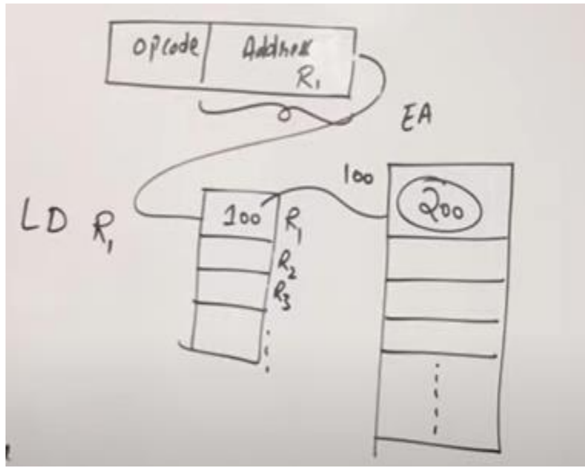
Immediate Addressing mode

- Operand is directly provided as constant
- No calculation required for EA(Effective address)
- For e.g., Add R1, #3.
- Constant size is restricted by address size



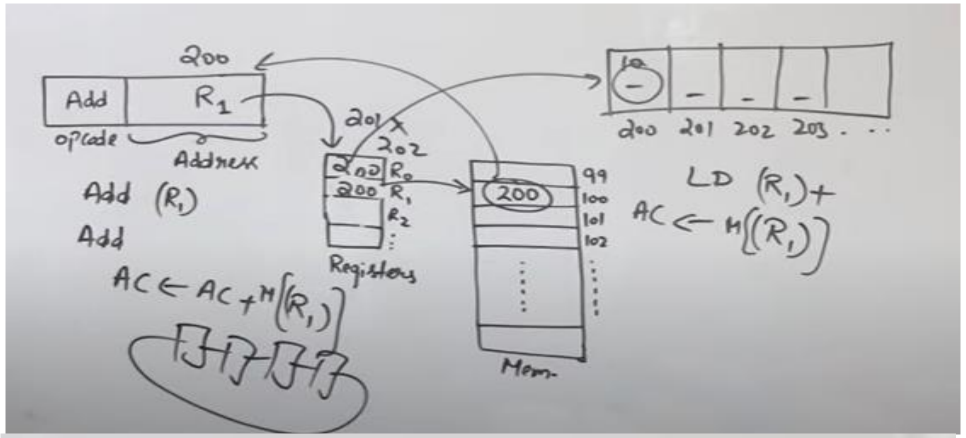
Register Indirect mode

- Register contain the address of memory containing operand.
- (Register number) is written in operand.
- For e.g., LD (R1), AC<- M[R1]
- EA= M[R1]
- Instruction size same as register direct mode.



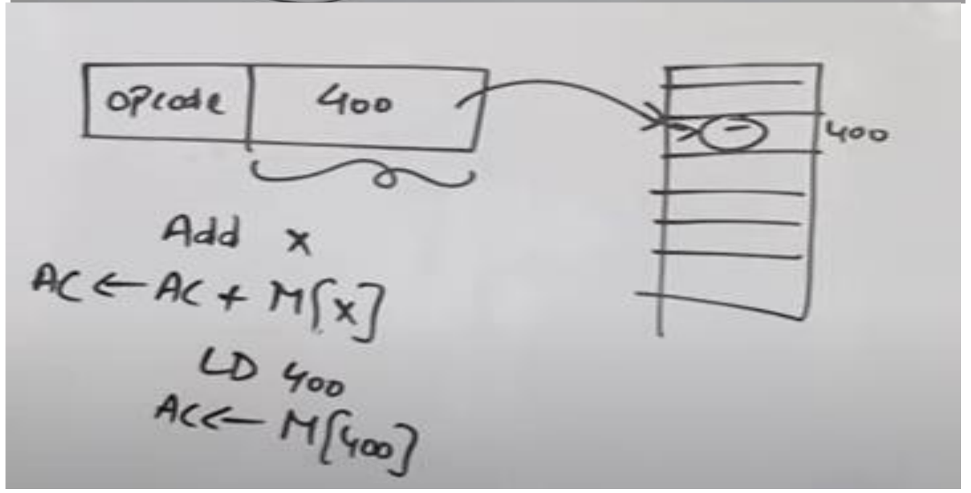
Autoincrement or Auto-Decrement mode

- Special case of register indirect addressing mode
- LD $(R1)_+$, LD $(R1)_-$
- Series of data(Select data from table).



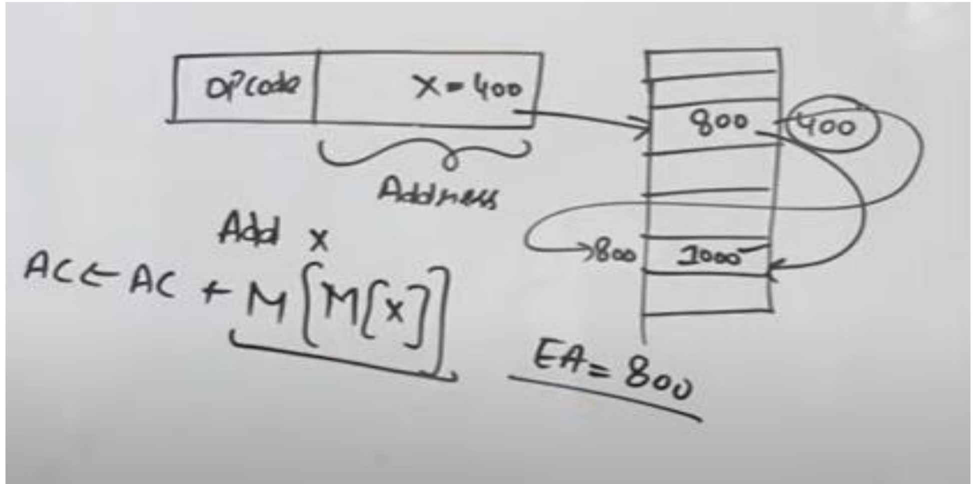
Direct /Absolute Addressing mode

- Actual address(memory) is given in operand
- Use to access variables
- EA= M[X]
- Instruction size depends on number of words



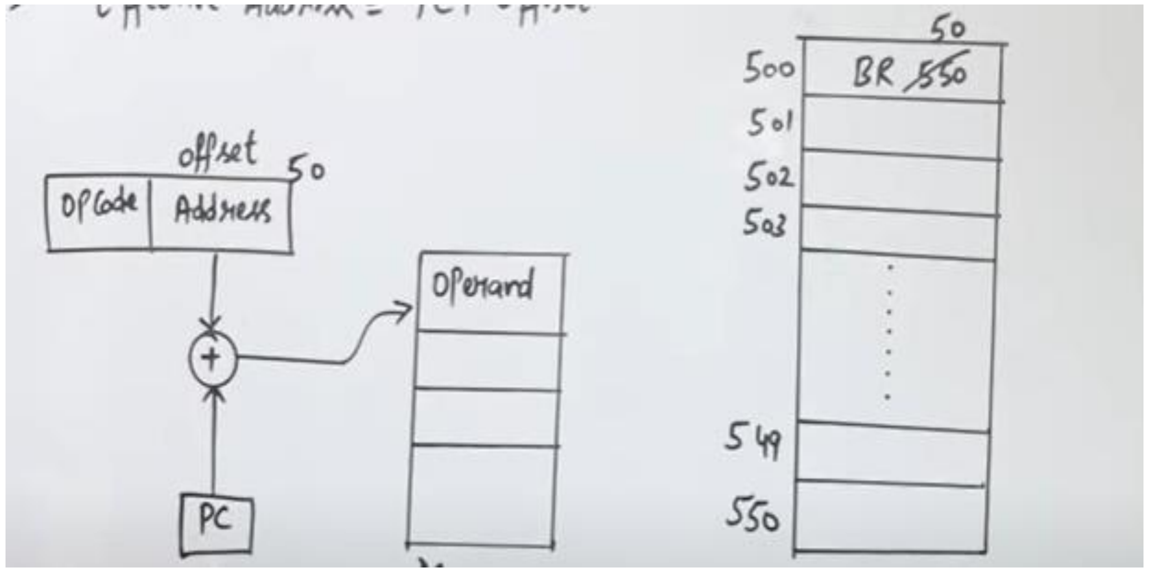
Indirect Addressing mode

- 2 memory access for getting to operand
- Use to implement pointers and passing parameters
- EA= M[M[X]]
- E.g. Add (X)
- Instruction size depends on number of words
- Computation increases



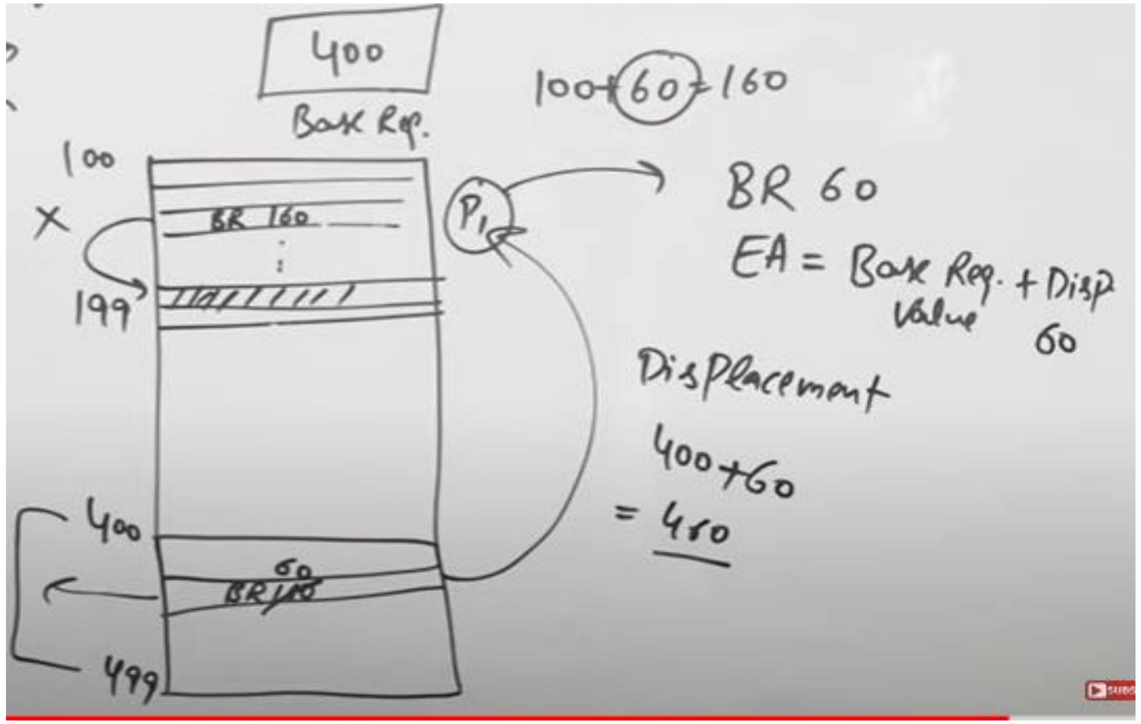
Relative addressing mode

- Program control instruction (JMP, B)
- EA = PC + offset(reducing size of operand)
- E.g. BR 50
- Offset is always one less.



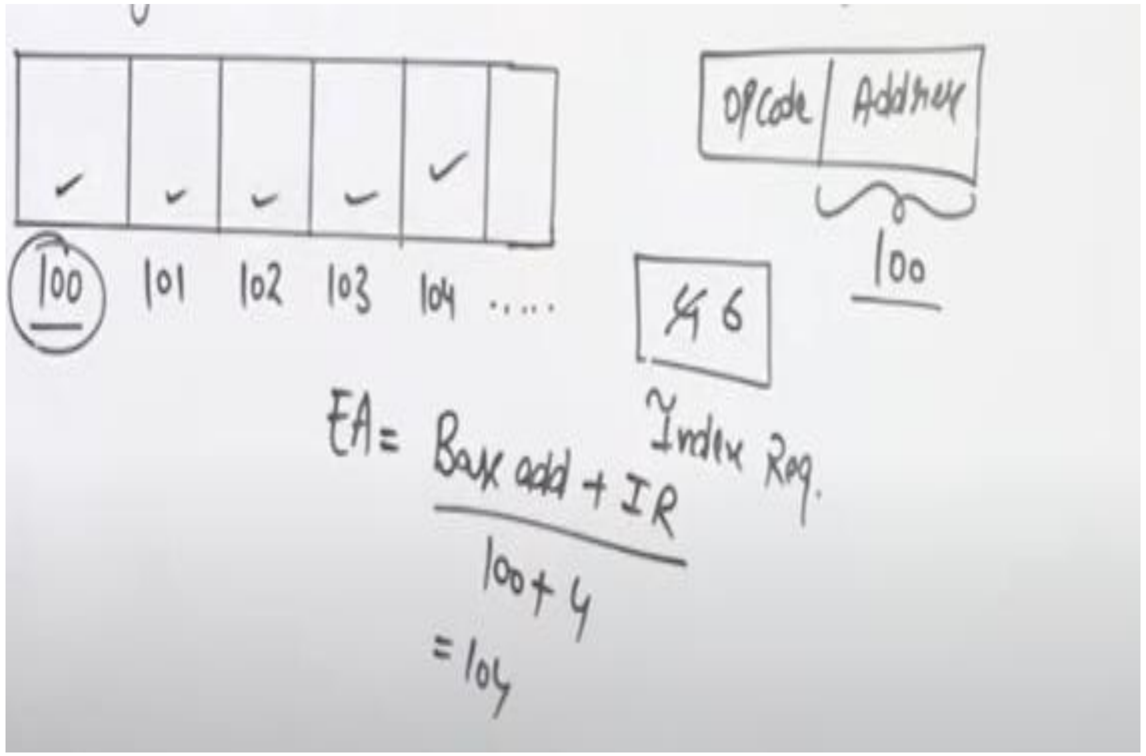
Base register addressing mode

- Used in program relocation
- RAM is limited so swap in –out program. So, program relocate to new address
- Base register is used to store relocated program address



Indexed Addressing mode

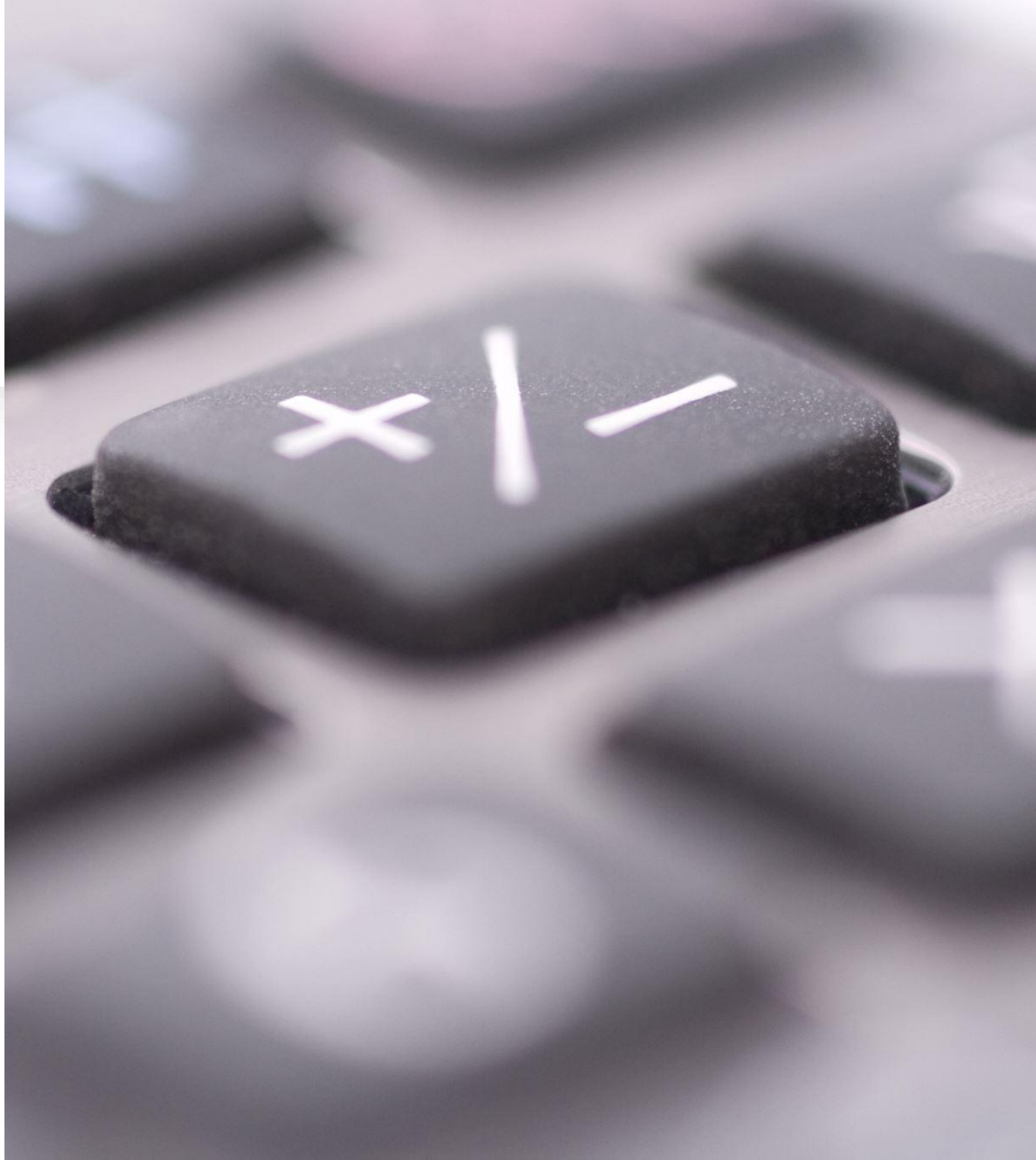
- Used to access or implement array efficiently.
- Multiple registers required to implement.
- Any element can be access without changing any instructions
- Base address is provided in the operand
- One index register is taken for accessing particular item
- EA= Base address + Index register



Instruction Cycle

Instruction Cycle

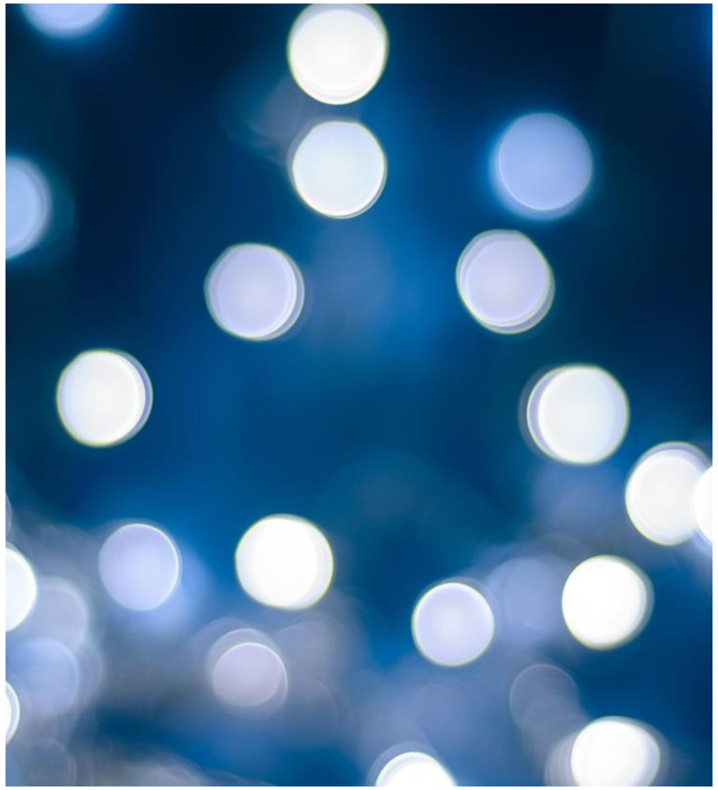
- A program residing in the memory unit of the computer consists of a sequence of instructions.
- .The program is executed in the computer by going through a cycle for each instruction
- In the basic computer each instruction cycle consists of the following phases:
- 1. Fetch an instruction from memory.
- 2. Decode the instruction.
- 3. Read the effective address from memory if the instruction has an indirect address.
- 4. Execute the instruction.



Fetch and Decode

Initially, the program counter PC is loaded with the address of the first instruction in the program.

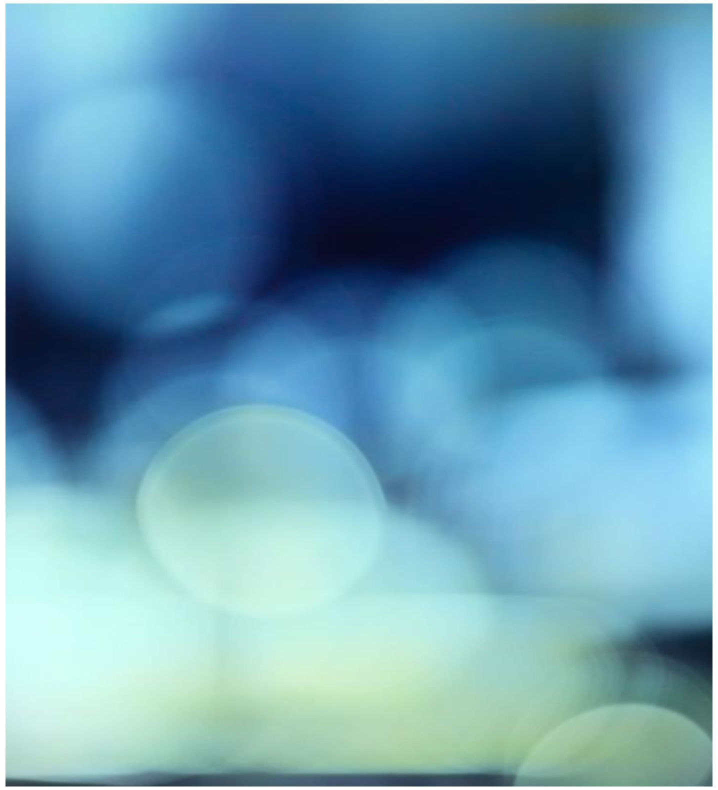
The sequence counter SC is cleared to 0, providing a decoded timing signal To. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2, and so on. The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

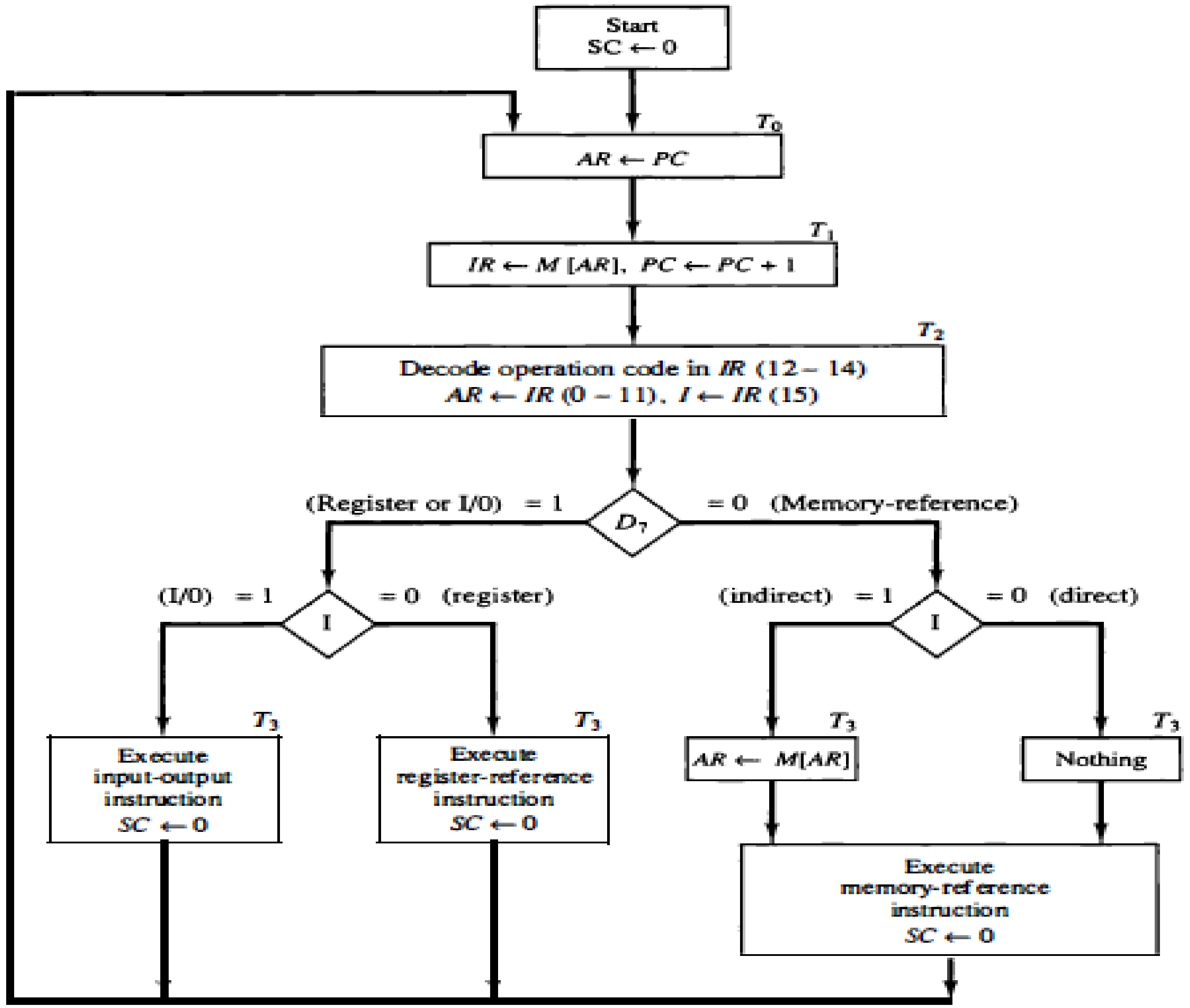


T0: AR <- PC

T,: IR <-M[AR], PC <- PC + 1

T2: D0, • • • , D7 <- Decode IR(12-14), AR <--- IR(0-11), 1 <--- IR(l5)




Start
SC ← 0
T0
AR ← PC
T1
IR ← M [AR], PC ← PC + 1
T2
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)
(Register or I/0) = 1
D7
= 0 (Memory-reference)
(I/0) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T3
Execute input-output instruction SC ← 0
T3
Execute register-reference instruction SC ← 0
T3
AR ← M[AR]
T3
Nothing
Execute memory-reference instruction SC ← 0

Timing and Control, Instruction Cycle

The timing for all registers in the basic computer is controlled by a master clock generator. The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit. The clock pulses do not change the state of a register unless the register is enabled by a control signal.

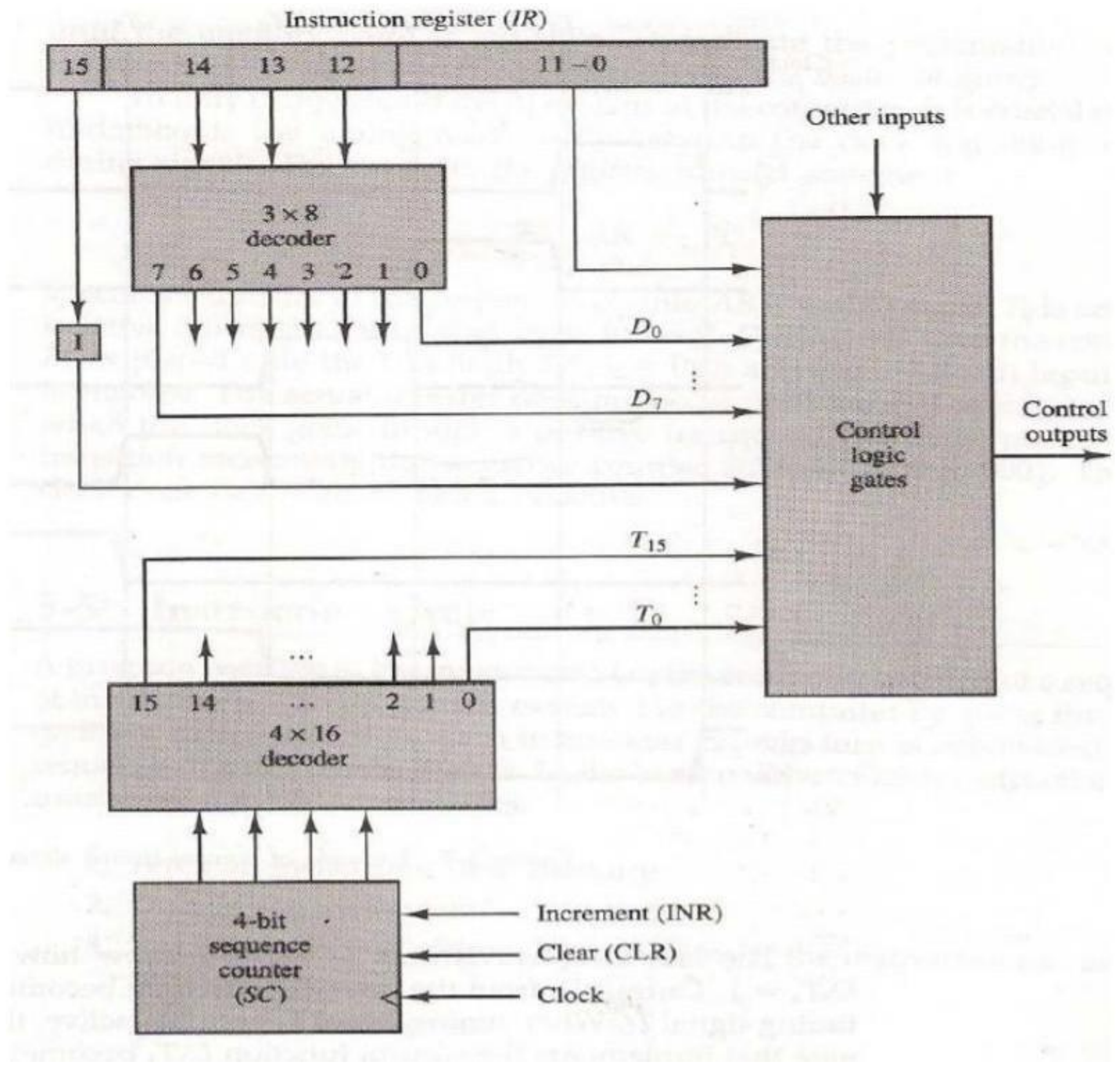
The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers and micro-operations for the accumulator.

There are two major types of control organization:

- **Hardwired Control**
- **Micro-programmed Control**

Hardwired Control	**Micro-programmed Control**
The control logic is implemented with gates, flip-flops, decoders and other digital circuits.	The control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro operations.
The advantage is that it can be optimized to produce a fast mode of operation.	Compared with the hardwired control operation is slow.
Requires changes in the wiring among the various components if the design has to be modified or changed.	Required changes or modifications can be doneby updating the microprogram in control memory.

The block diagram of the hardwired control unit is shown below:



The control unit of basic computers consists of two decoders, a sequence counter and a number of control logic gates.

- An instruction read from memory is placed in the instruction register (IR). It is divided into three parts:
  a.The I bit
  b.The operation code and
  c.Bits 0 through 11.
- The operation code in bits 12 through 14 are decoded with a 3*8 decoder. The eight outputs of the decoder are designated by the symbols D0 through D7.

- Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I.
- Bits 0 through 11 are applied to the control logic gates.

- The 4-bit sequence counter can count in binary from 0 through 15.
- The outputs of the counter are decoded into 16 timing signals T0 through T15.
- The sequence counter SC can be incremented or cleared synchronously.
- The counter is incremented to provide the sequence of timing signals out of the 4*16 decoder.
- consider the case where SC is incremented to provide timing signals T0, Tv T2, T3, and T4 in sequence. At time T4, SC is cleared to 0 if decoder output D3 is active. This is expressed symbolically by the statement
- D3T4: SC <- 0

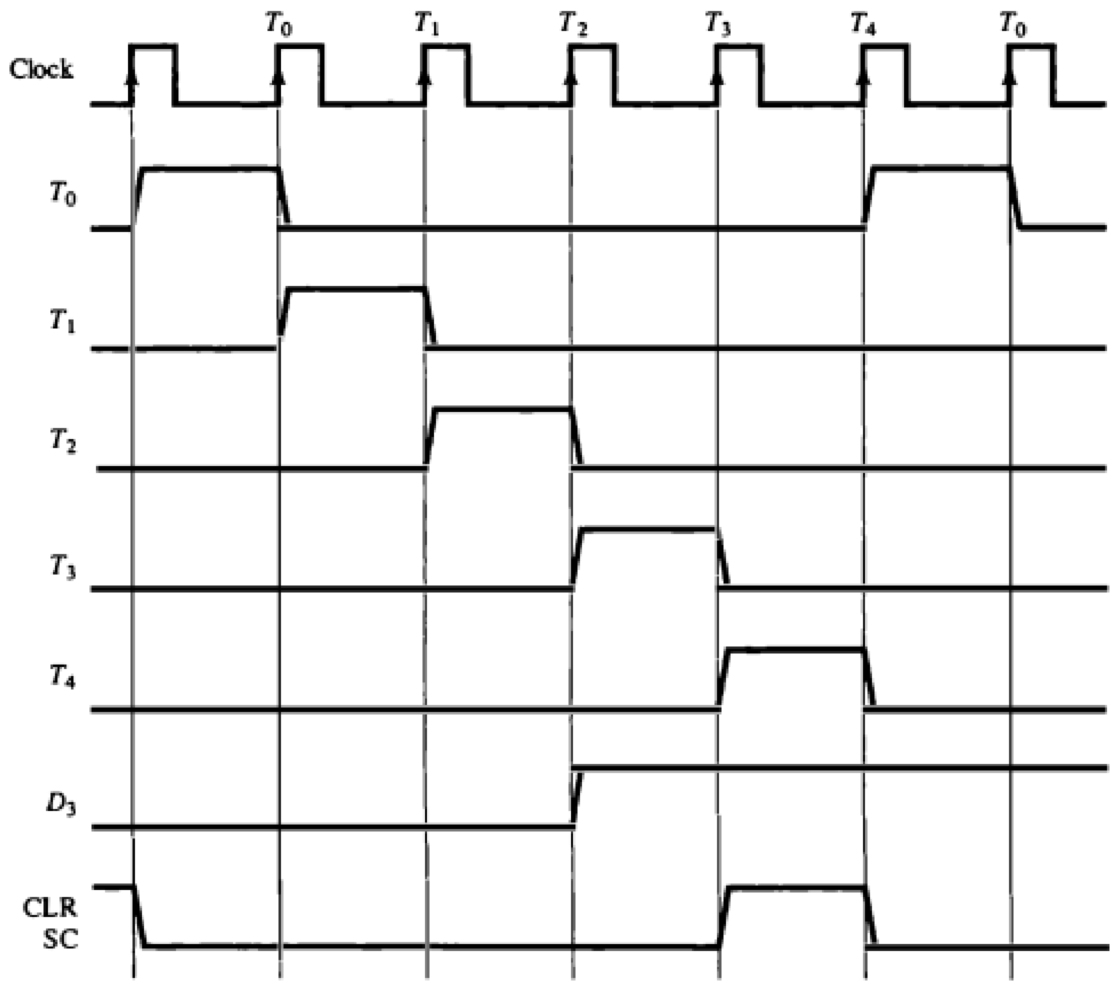


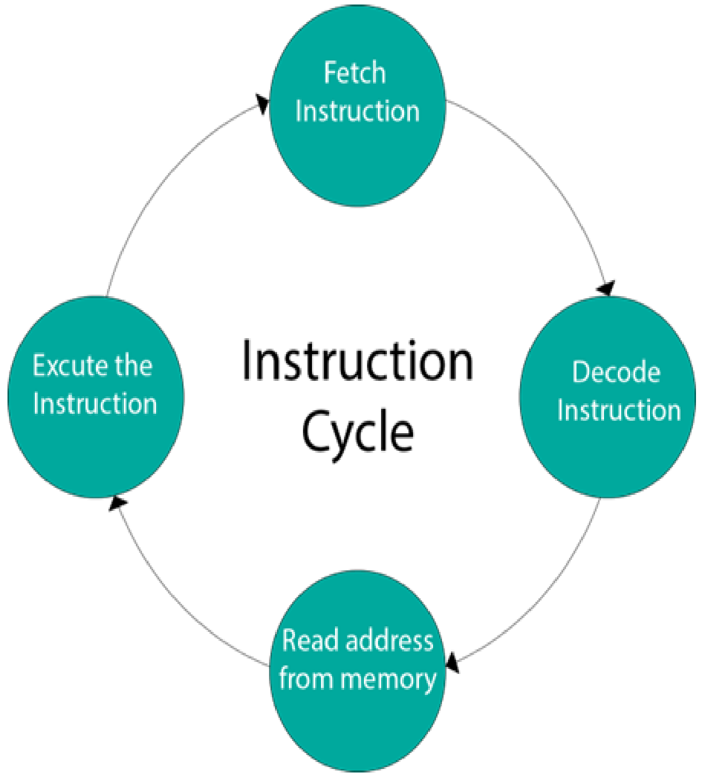
Figure 5-7 Example of control timing signals.

Instruction Cycle:

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle of each instruction. Each instruction cycle in turn is subdivided into a sequence of sub cycles or phases.

In the basic computer each instruction cycle consists of the 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.



Interrupt and its Types

- Interrupt is an event that disrupt the normal flow of operation.

<u>Types of Interrupt:</u>

1)Hardware Interrupt

- This interrupt is present at the hardware pins of processor.
- It can be further two types- internal and external interrupt.
- External interrupt caused by external devices such as (I/o devices, power supply , etc.)
- Internal interrupt- internal components of processor(timer, invalid opcode, , stack overflow, etc.)

2) Software Interrupt:

- Here interrupt is defined in the instruction set of the processor (E.g. Supervisor call)

3) Maskable Interrupt:

- This interrupt deals with low priority interrupt and can be enable or disabled.

4) Non-Maskable interrupt:

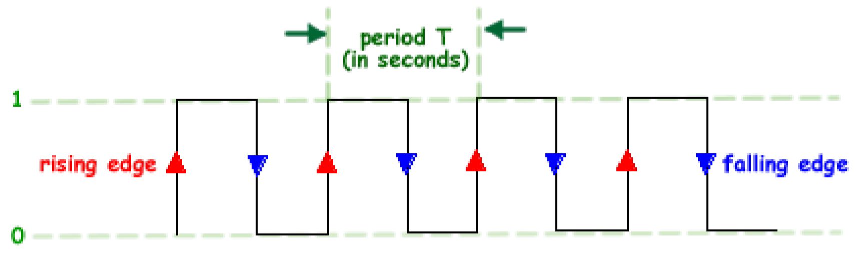
- Interrupt is always present in the enable state and deals with high priority components.(Power failure)

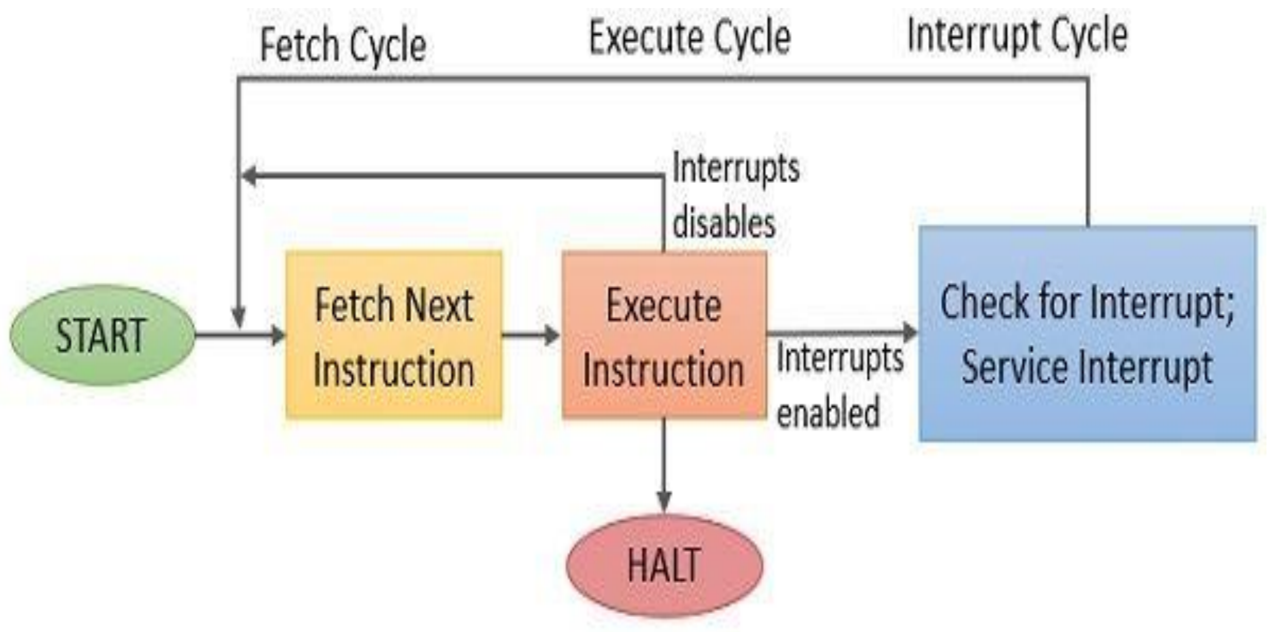
Level triggered Interrupt

- This interrupt are enable based on the level of clock pulse .

Edge triggered interrupt

- This Interrupt is enabled based on the raising edge transition or falling edge transition of clock signal.





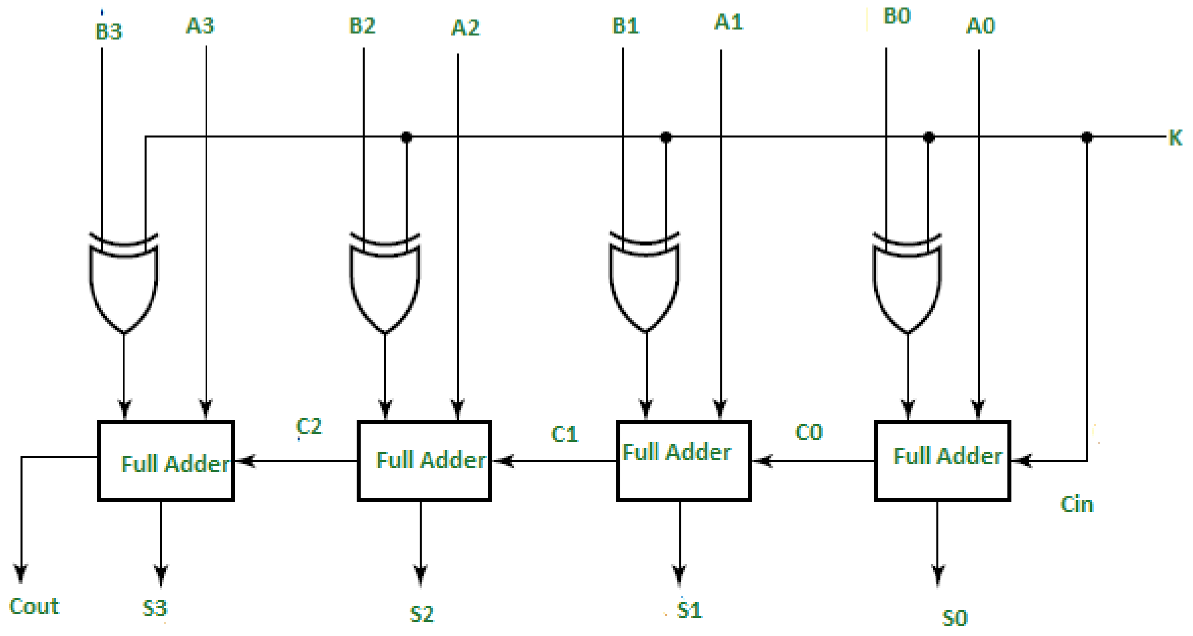
Instruction Cycle with Interrupts

Binary Adder and Subtractor

- A **Binary Adder-Subtractor** is capable of both the addition and subtraction of binary numbers in one circuit itself.
- The operation is performed depending on the binary value the control signal holds. It is one of the components of the ALU (Arithmetic Logic Unit).
- This Circuit Requires prerequisite knowledge of Exor Gate, Binary Addition and Subtraction, and Full Adder.
- Let's consider two 4-bit binary numbers A and B as inputs to the Digital Circuit for the operation with digits

  A0 A1 A2 A3 for A

  B0 B1 B2 B3 for B
- The circuit consists of 4 full adders since we are performing operations on 4-bit numbers. There is a control line K that holds a binary value of either 0 or 1 which determines that the operation is carried out is addition or subtraction.



- A0- direct input, B0 exor K- second input, two output- S, C
- When K =1, B0 exor 1= B0' (Complement Operation)
- When k =0, B0 exor 0 = B0 (Simple input)
- Thus K =1 , subtractor-A0+B0'+Cin
- For k =0 , adder- A0+B0+Cin
- For n bit number we require n full adder.

E.g.-*Assume that we have two 3-bit numbers, i.e., X=100 and Y=011, and feed them in Full-Adder as an input.*

X0 = 0 X1 = 0 X2 = 1

Y0 = 1 Y1 = 1 & Y2 = 0

For K=0:

- Y0⊕K=Y0 and Cin=K=0
- So, from first Full-Adder
  - S0 = X0+Y0+Cin , S0= 0+1+0
    S0=1,C0=0
  - S1 = X1+Y1+C0, S1 = 0+1+0
    S1=1 and C1=0
  - S2 = X2+Y2+C1, S2 = 1+0+0
    S2=1 and C2=0,
- X= 100 =4,Y = 011 = 3
- Sum = 0111 = 7

For K=1

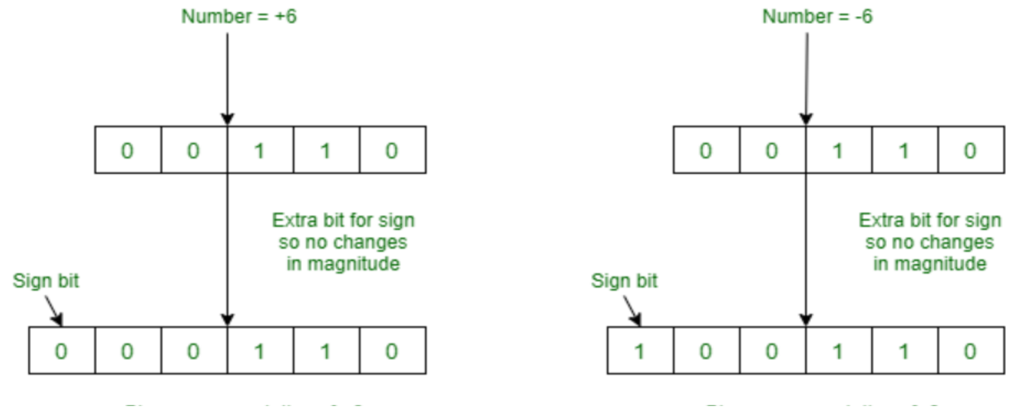
- Y0⊕K=Y0' and Cin=k=1

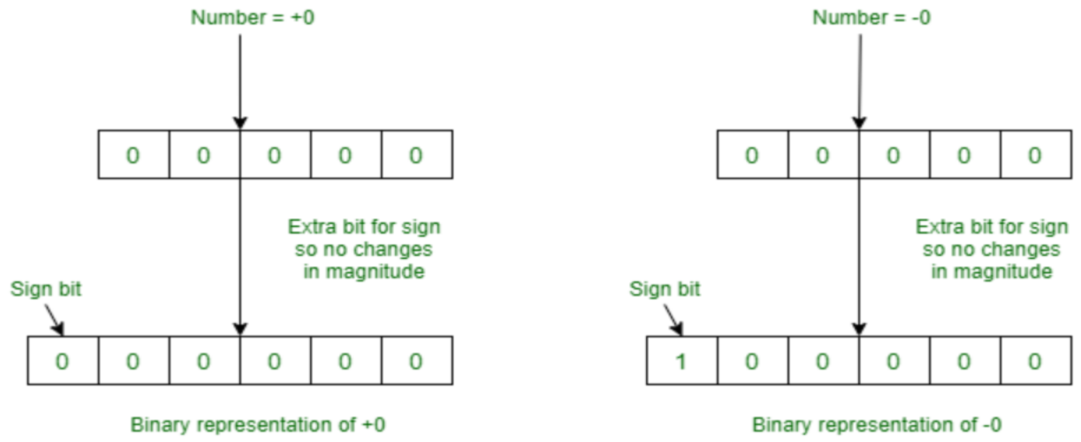
So,

  - S0 = X0+Y0'+Cin, S0 = 0+0+1
    S0=1 and C0=0
  - S1 = X1+Y1’+C0, S1 = 0+0+0
    S1=0 and C1=0
  - S2 = X2+Y2’+C1, S2 = 1+1+0
    S2=0 and C2=0
- Thus, X = 010 = 4, Y = 011 = 3
- Difference = 001 = 1

Representation of Negative Binary Numbers

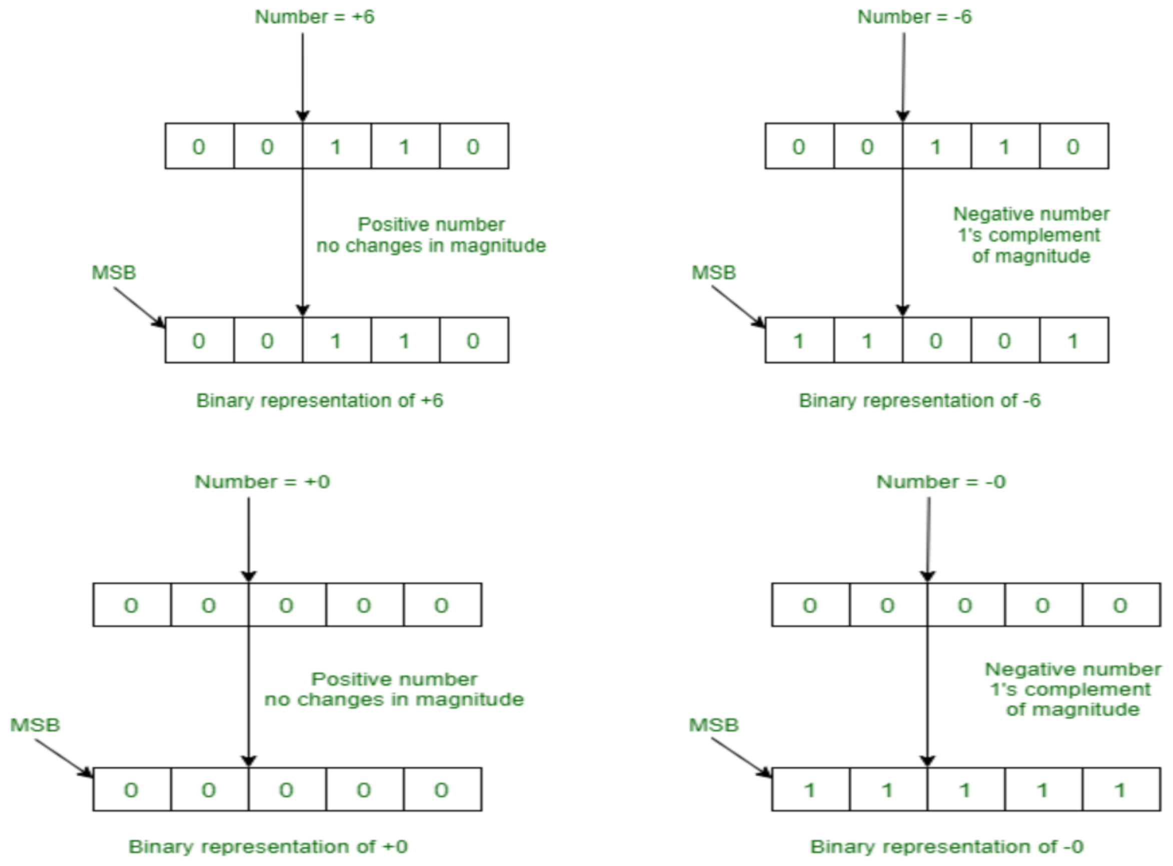
- We represent negative binary numbers using a minus symbol in front of them.
- an extra bit or flag called sign bit or sign flag in the Binary number representation system for signed numbers.
- 0- positive 1- negative.
- **Ways to represent magnitudes :**
  These are: Sign-Magnitude method, 1's Complement method, and 2's complement method.
- **Signed Magnitude Method :**
  We only add an extra sign bit to recognize negative(1) and positive numbers(0).


Number = +6
0 0 1 1 0
Extra bit for sign so no changes in magnitude
Sign bit
0 0 0 1 1 0
Number = -6
0 0 1 1 0
Extra bit for sign so no changes in magnitude
Sign bit
1 0 0 1 1 0


Number = +0
0 0 0 0 0
Extra bit for sign so no changes in magnitude
Sign bit
0 0 0 0 0 0
Binary representation of +0
Number = -0
0 0 0 0 0
Extra bit for sign so no changes in magnitude
Sign bit
1 0 0 0 0 0
Binary representation of -0

- **<u>1’s Complement</u> Method :**

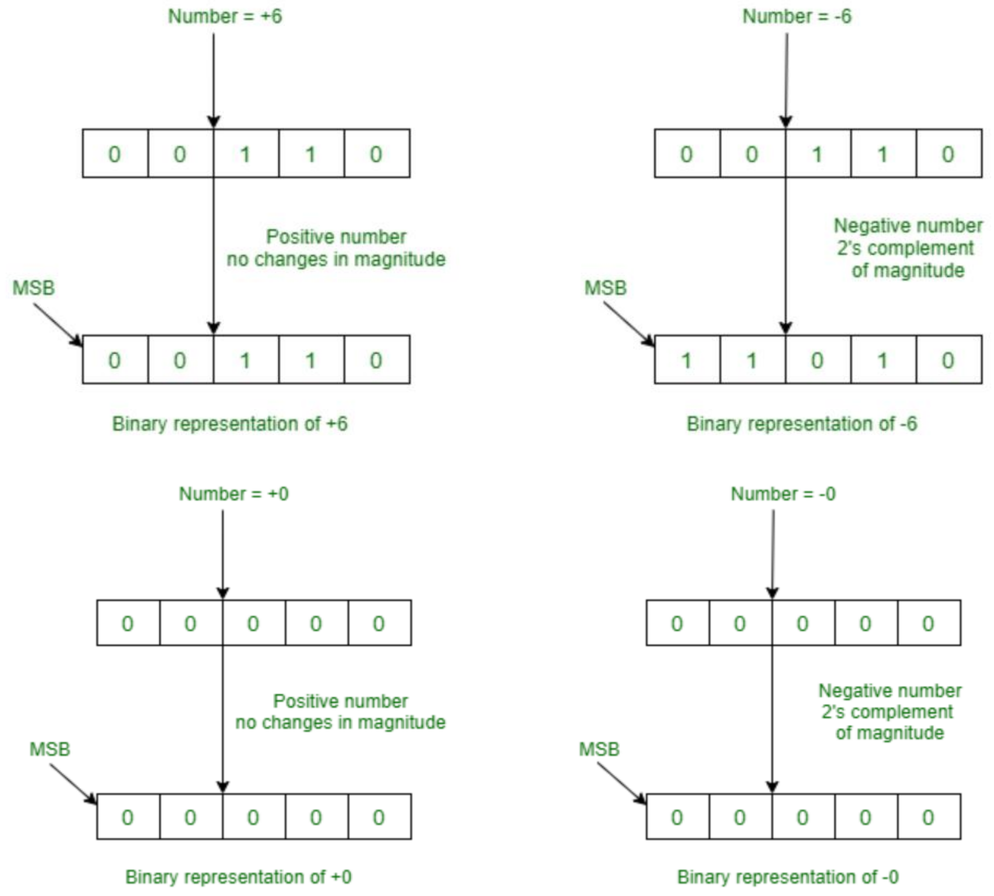
Please note that MSB is always Sign bit, if it 0, then there are no changes. MSB 1 in negative numbers. We only take 1’s complement of negative numbers to represent in the computer.



Number = +6
0 0 1 1 0
Positive number no changes in magnitude
MSB
0 0 1 1 0
Binary representation of +6
Number = -6
0 0 1 1 0
Negative number 1's complement of magnitude
MSB
1 1 0 0 1
Binary representation of -6
Number = +0
0 0 0 0 0
Positive number no changes in magnitude
MSB
0 0 0 0 0
Binary representation of +0
Number = -0
0 0 0 0 0
Negative number 1's complement of magnitude
MSB
1 1 1 1 1
Binary representation of -0

<u>2’s Complement</u> Method :

We only take 2’s complement of negative numbers to represent in the computer



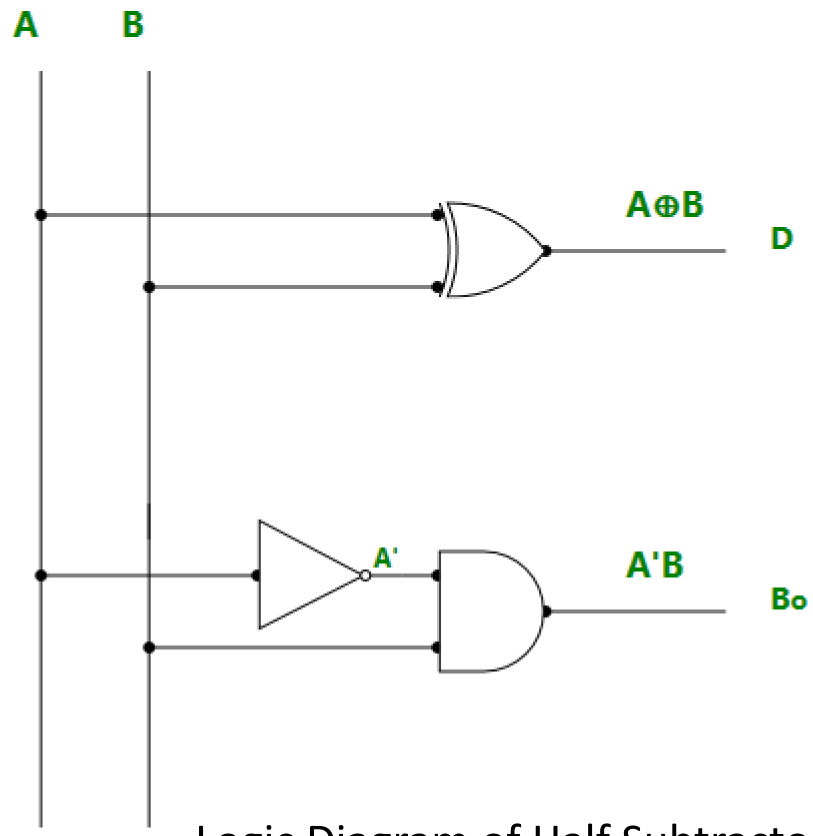
Number = +6
0 0 1 1 0
Positive number no changes in magnitude
MSB
0 0 1 1 0
Binary representation of +6
Number = -6
0 0 1 1 0
Negative number 2's complement of magnitude
MSB
1 1 0 1 0
Binary representation of -6
Number = +0
0 0 0 0 0
Positive number no changes in magnitude
MSB
0 0 0 0 0
Binary representation of +0
Number = -0
0 0 0 0 0
Negative number 2's complement of magnitude
MSB
0 0 0 0 0
Binary representation of -0

Inputs		Outputs	
A	B	D	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

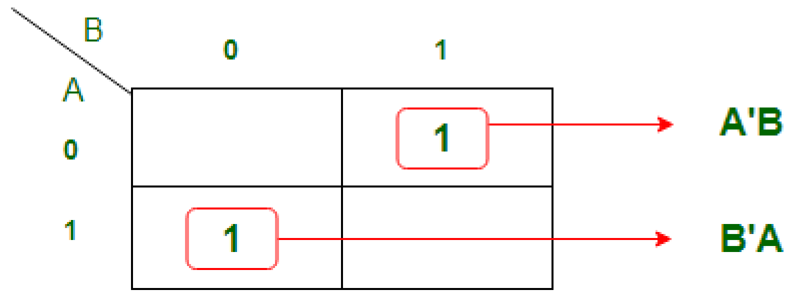
D = A'B + AB'

D= A XOR B

B_o = A'B



Logic Diagram of Half Subtractor



A'B+B'A=A xor B