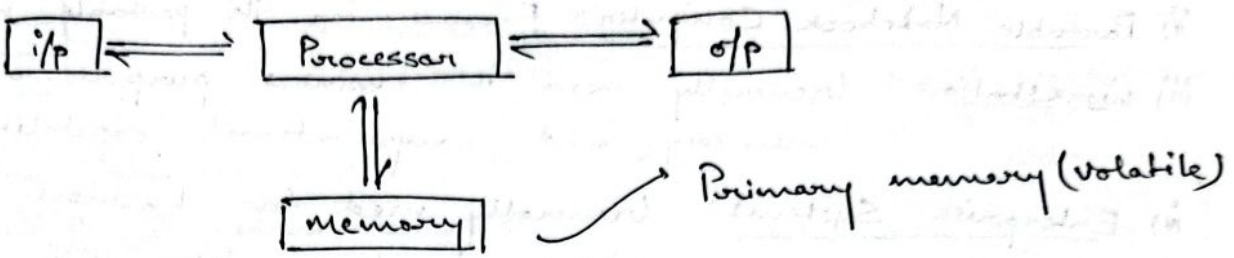
Computer: A computer is an electronic device that accepts input from the outside world and process them according to some predefined instructions and produces output for the outside world

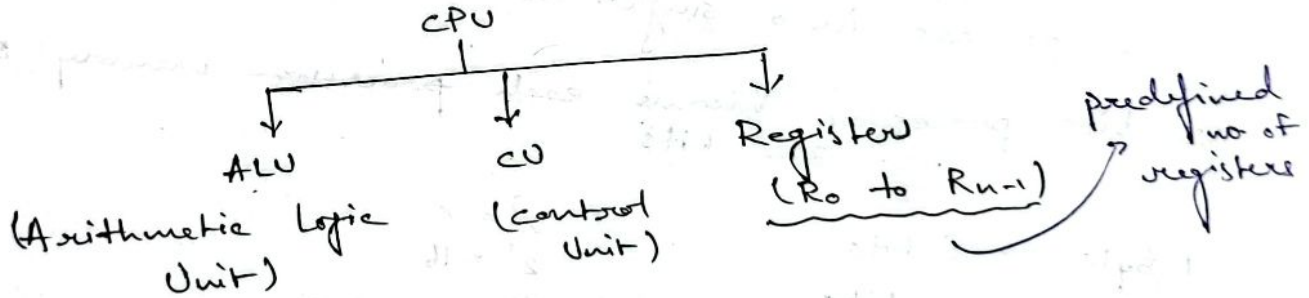


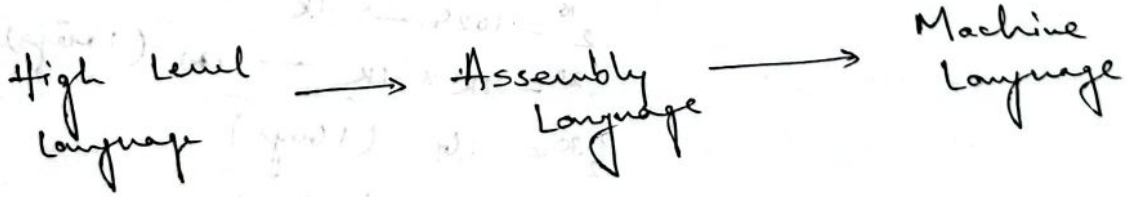
Organisation

It is concerned with the way the hardware components operate and the way they are connected together to form the computer.

Architecture

It is concerned with the structure and the behaviour of the computer as seen by the user. It includes instruction formats, instruction set and techniques for addressing memory.





Types of computers :

Computers are differ in size, construction, computational power and intendent use.

i) Personal computer or desktop : Generally used for personal use

ii) Portable Notebook Computer : Famous for its portable nature

iii) Workstation : Generally used in business purpose with high memory and computational capabilities.

iv) Enterprise Systems : Generally used for business purpose to process medium to large data and it has much more computing power.

v) Server : It contains sizeable database storage units and also capable of handling large volumes of data.

vi) Super computers : Super computers are used for large scale numerical calculations / computations and required in applications such as weather forecasting and aircraft design and simulation.

* The maximum number of bits a processor can fetch or process in a single fetch that is called 'Word'.

32 bits processor means each ~~processor~~ memory stores 32 bits.

1 byte → 8 bits

1 Nibble → 4 bits

$2^3 = 8$

$2^4 = 16$

⋮

$2^{10} = 1024 \rightarrow 1K$

$2^{20} = 1K \times 1K \rightarrow 1M$ (1 Mega)

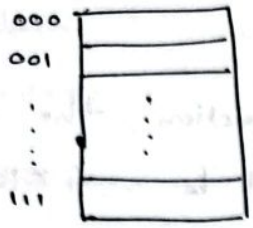
$2^{30} = 1G$ (1 Giga)

$2^{40} = 1T$ (1 Tera)

⋮

Q A processor is connected to a 128G X 32 memory module. What is the width of address and data bus ?

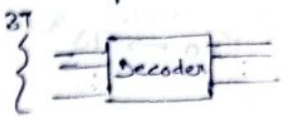
Ans:



Suppose,
No. of bits → 3
Total locations → $2^3 = 8$

3 → Address bus
8 → Data bus

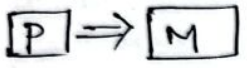
37 bits are required to represent a address



each location holds 32 bits of data

Here, total locations = 128G = $2^7 \times 2^{30} = 2^{37}$
Width of address bus = 37
Width of data bus = 32

So total data stored in the processor = $(2^{37} \times 32)$ bits



processor is giving address to memory (unidirectional)

Processor & memory exchanging data (bidirectional)

Data bus cannot be decoded

Width of address bus = No. of wires coming from the processor to provide address to memory

Processor

Major components of processor are:

i) CPU
ii) ALU
iii) Register set (R_0 to R_{n-1})
iv) Control Unit

One instruction contains so many microoperations and each of the microoperation or microexecution needs a clock pulse. The instruction should be fetched once from the primary memory and then it should be stored into cache memory

Registers have small memory capacity but it is very fast so its the fastest and smallest memory. If the data is inside the registers instead of memory, CPU is very fast.

Control Units job is to control and coordinate all the activities inside the system.

Kilo → 10^3 Mega → 10^6 Giga → 10^9

4 GHz processor means to complete one instruction, the no. of clock pulses generated in every second will be = 4×10^9

Q One instruction requires 7 clock cycles to complete its execution. How much time is required for the instruction if the processor speed is 5 GHz?

5×10^9 clock pulses are generated in 1 second

1 " " " " " $\frac{1}{5 \times 10^9}$

∴ 7 " " " " " $\frac{7}{5 \times 10^9}$

$= 1.4 \times 10^{-9}$ s

$= 1.4$ ns

(Programmer visible register that can be used as operands in instruction)

Register

- Dedicated Register (dedicated for some specific task)
- General Purpose Register (R0 to Rn-1)

1. Program Counter (PC)
2. Memory address register (MAR)
3. Memory data register (MDR)
4. Instruction register (IR)
5. Stack pointer (SP)

} Each register has a particular work and these are the pillars of processor.

Always left → right

ADD R0 R1 → R0+R1 and the result will be stored in R1

ADD R1 R0 →

PC will give the ~~instruction~~ address of the instruction and it will be passed to MAR. After each passing the address will be incremented. PC can store the address of instruction and data. PC means program counter. It stores the address of the instruction of the primary memory.

MAR contains the address of memory locations from where any read or write operation is going to take place.

After reading all the operations all the datas and instructions should be stored into the MDR but if its an instruction it will be moved to IR. If its a data, then after storing into MDR it will be directly moved to the control unit (CU)

Before storing anything into MDR, the memory will read and after the completion the MFC (memory function control) confirms the CU and then it stores into MDR.

Opcode	Operand

The instruction stored in the IR will be then decoded by a decoder circuit and separate the opcode and operand. Then it will check if all the operands are stored in the general registers (R_0 to R_{n-1}) or not. If its stored in an outside accumulator or memory then the particular address should be fetched which will be time taking.

Whatever number of bits are required to represent a memory address will be same for the content stored inside PC because we already know that PC holds the address of the next instruction to be executed.

- **MDR**

  It contains the data that is being read from the memory or data that is being written into the memory during write operation.

PC

(i) PC holds the address of the next instruction.

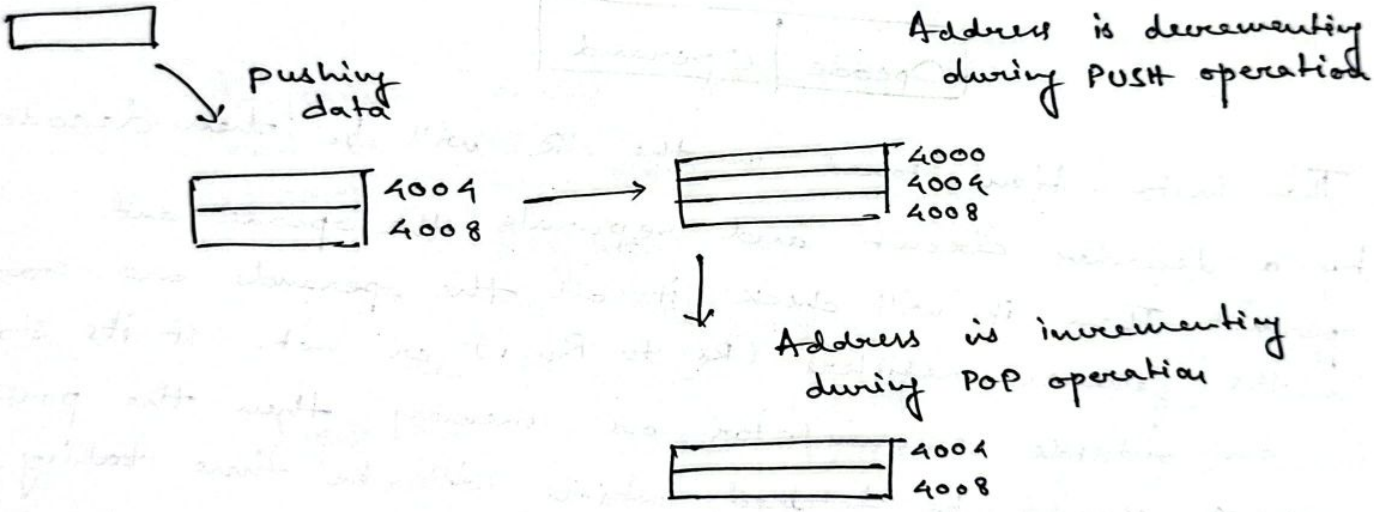
MAR

(i) MAR holds address of the instruction or data into the memory

Whenever read or write operation is going on (memory is busy) the PC doesn't rest, it sends the address of the current instruction to the ALU and increments the address which is the next instruction address.

Stack Pointer (SP):

It points to the top of the stack. Stack goes downward in the memory for 'push' operation
SP is decremented and the 'pop' operation SP is incremented after the operation.



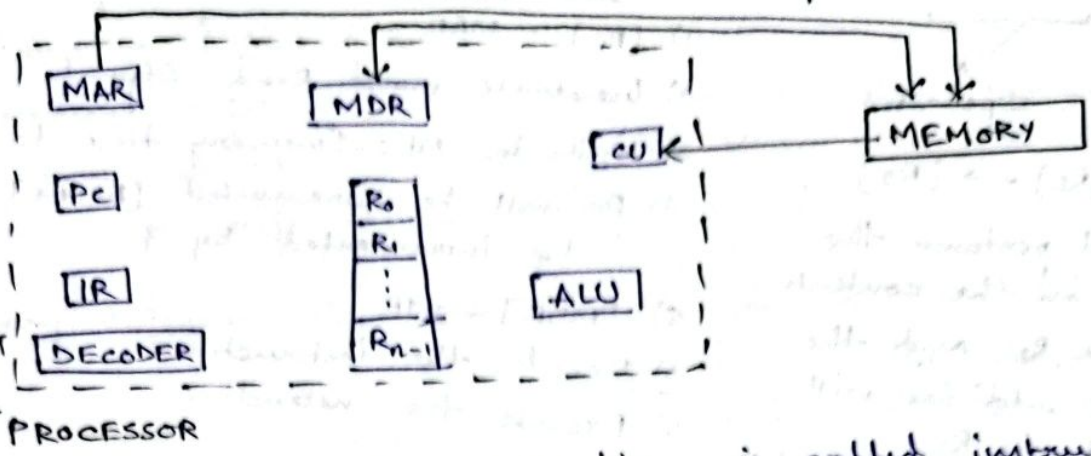
Control Unit (CU):

Control Unit will generate various signal of appropriate time so that one complete activity can be performed in a well behaved manner.

General purpose registers (R0 to Rn-1):

These are programmer visible registers or general purposes registers. Visible registers that can be used as operand in an instruction.

Q How instruction is executed into the processor :



One complete instruction execution is called instruction cycle. Instruction cycle consists of :

i) Fetch the instruction (from memory) pointed by PC into IR
ii) Decode the instruction.
iii) Fetch the operands from memory if required (No need for general registers).
iv) Execute the instruction.
v) Store the results into the memory if required. (Incase it is connected to any output devices like printer then its not required)

---

① FETCH THE INSTRUCTION POINTED BY PC INTO IR

a) Content of Pc [PC] is going to MAR
b) Control Unit will generate Read signal
c) Wait for MFC (memory function complete) to complete
d) content of MDR [MDR] will be transferred to IR

} Instruction cycle

② DECODE THE INSTRUCTIONS

Decoder is connected to IR hence the instruction is decoded immediately and what is the operation to be performed is found out (opcode). Decoder is connected to the control unit (CU) and IR is connected to the decoder.

① ADD R_1, R_2

Opcode — Operands

$[R_1] + [R_2] \longrightarrow [R_2]$

ALU will perform the addition on the contents of R_1 and R_2 and the result of addition will be stored in R_2

Steps

1) [PC] = MAR
2) Generate Read Signal
3) Wait for MFC (memory time to complete)
4) PC will be incremented [PC++] will be incremented by 1
5) [MDR] → IR
6) Decode the instruction
7) Execute the instruction

② ADD A, R_1 where A is the memory location and it is numeric

Steps

1) [PC] = MAR
2) Generate Read Signal
3) Wait for MFC
4) PC will be incremented [PC++]
5) [MDR] → IR
6) Decode the instruction
7) Fetch the operand stored at memory location A.

Sub steps

i) Address part of IR goes to MAR [IR → MAR]
ii) Generate the read signal
iii) Wait for the MFC
iv) [MDR] → Input of ALU

(Data bus is connected to MDR so the data will come to MDR at first)

Memory ⟷ (Data bus) MDR

8) Execute the instruction ALU will perform the addition on the [Memory location A] and R_1 and the result is stored into R_1

Q How many times the memory is referred to execute the instruction ? ADD A, R_1.

One memory reference means once you give an address in MAR then till the data comes in MDR then the whole sequence is referred as memory reference.

Memory reference is 2 for the instruction ADD A, R_1
- → to fetch the instruction
- → to fetch the operand from memory location A.

Q ADD R_i, A

1) [PC] = MAR
2) Generate Read Signal
3) Wait for MFC
4) PC will be incremented [PC++]
5) [MDR] → IR
6) Decode the instruction

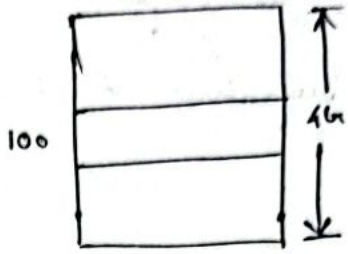
} Upto this steps are always same.

7) Fetch the operand stored at memory location A
   - i) Address part of IR goes to MAR
   - ii) Generate the Read signal
   - iii) Wait for MFC
   - iv) [MDR] → input of ALU
8) Execute the instruction
9) ALU will perform the addition on the contents of memory location A and R_i
10) Store the result into memory location A
    - i) Address part of IR → MAR
    - ii) Result of ALU → MDR
    - iii) CU generates 'write' signal
    - iv) Wait for MFC

Value of memory reference is 3
- → to fetch the instruction
- → to fetch the operand from memory location A
- → to store the result into memory location A

Q A processor is connected to a 4G x 32 bit memory [illegible]. A program is kept in 100th location (100th address of memory) and maximum length of each instruction of the memory is of 32 bits. Then find out size of MAR, MDR, IR and also the content of PC ?

4G x 32 Bit 1 byte = 8 bit
= 4G x 4 bytes
= 16GB (primary memory size)



$4G = 4 \times 2^{30} = 2^2 \times 2^{30} = 2^{32}$

Each location is of 32 bits and each of them can store 4 bytes [32 bits are required to represent one address] to total no. of memory location

[PC] = 100

Address bus → |MAR| = 32

Data bus → |MDR| = 32

|IR| = 32 (As MDR is connected to IR)

[ ] → content
| | → length

Q A processor has 48 bit instructions composted of 2 fields. The first 2 bytes contain the opcode and the remainder contain memory operand address. How many bits are needed for the PC and the IR ?

Instruction → opcode → 16 bits
Instruction → Operand → 32 bits
} both stored in IR

So the no. of locations = 2^{32} with each location of 4 bits.
No. of bits required for IR = (16+32) = 48 bits
No. of bits required for PC = 32 bits → Since IR contains entire instruction

PC contains the address of instruction and the address of instruction = 32 bit.

Performance

The most important measure of the performance of a computer is how quickly it can execute programs.

Each program contains so many instructions. Each of the instructions means are assembly language

Performing,

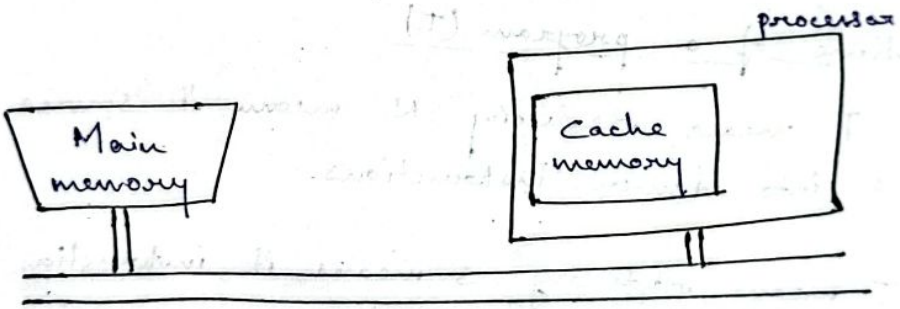
3rd step of 1st instruction
2nd step of 2nd "
1st " " 3rd "

} Parallel Operation or pipelining to boost up the performance using multi bass (harward architecture)

Architecture is always constant, you are just improving the technology by improving the performance.

Three factors improve the performance:

i) Hardware design
ii) Instruction set
iii) Compiler



Single bus architecture (Von Neumann)

Clock:

i) Processor circuits are controlled by a ~~thito~~ timing signal called a clock

ii) The clock defines regular time interval called clock cycle.

iii) To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps such that each step can be completed in one cycle.

• Clock rate

clock rate (R) = $\frac{1}{p}$ Hz

p → length of 1 clock cycle.

= No. of clock cycles per second

Basic Performance Equation

$$T = \frac{NS}{R}$$

T = Processor time required to execute a program that has been prepared in a high level language.

N = No. of actual instructions needed to complete the execution of a program. Some instructions may be executed more than once.

R = clock rate

S = Average number of basic steps needed to execute one machine instruction. Each steps completes in 1 clock cycle.

Execution time of a program (T)

To improve T means reducing N means if source program is compiled into fewer instructions.

To improve T means reducing S means if instruction have smaller no. of basic steps to perform.

To improve T means increasing R means using a high frequency clock.

Q System has 8 MHz processor. One instruction takes 7 clock cycles to be executed. What is the time required to complete one instruction?

8 MHz processor means 8 mega cycles are generated per second

8 MHz = 8×10^6 clock cycles are generated in 1 sec

1 " " " " $\frac{1}{8 \times 10^6}$ sec

7 " " " " $\frac{7}{8 \times 10^6}$ sec

$= 0.75 \times 10^{-6}$ sec

$= 0.75 \mu s$

RISC

- Processor with simple instructions are called reduced instruction set computers.
- Reduced instructions takes one clock cycle.
- Only LOAD and STORE instructions can memory reference
- Supports pipeline
- It requires more of GPR
- In case of simple program program size is large.
- It emphasizes on software technology

CISC

- Processor with complex instructions are called complex instruction set computers.
- Complex instructions takes multiple clock cycle.
- Many instructions can memory reference
- Doesn't support pipeline
- It requires few numbers of GPR
- Code size is small.
- It emphasizes on hardware technology.

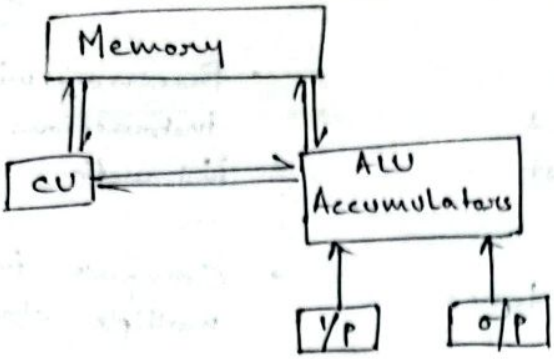
Von Newmann Architecture

John Von-Newmann has invented a machine in institute of advance studies in 1945 - 1952 which is named as stored ~~pro~~ program digital computer or Von-Newmann architecture.

Stored program digital computer is one that keeps its program and data in RAM. Before to this machine there were program controlled computers were present.

There are 4 subsystems in Von Newmann architecture

(I) Memory
(II) Input / Output
(III) ALU
(IV) CU

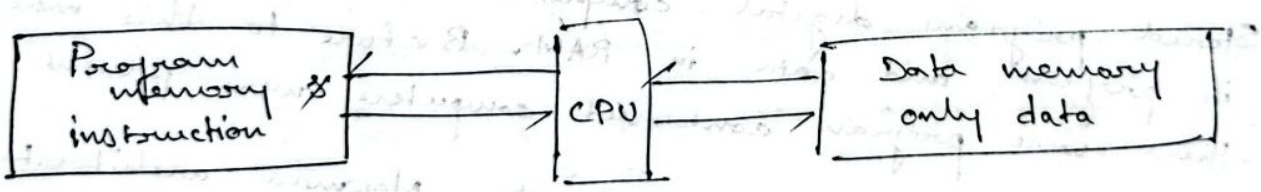


Here single bus is used to connect different components
The earlier computing machine had fixed programs or disc calculations. The idea of store programs replaced the fixed program concept by creating instruction set architecture.

- Disadvantages

- In stored program computers, the manufacturing program can damage its own program, others program and also OS.
- The separation b/w CPU and memory by using a single bus will limit the communication path between processor and memory.

Harvard Architecture



It is a computer architecture with physically separate storage and signal path for interconnection and data.

- Instructions are stored in punch card and data is stored in electromagnetic counters.
- Size of instruction memory is more than the data memory.
- In this architecture, program and data read simultaneously from the program memory and data memory. thus it is faster.

Processor Clock

Processor circuits are controlled by a timing signal called clock. If p is the clock cycle then clock rate.

$$R = 1/p$$

* A processor operates on a frequency. If we know the frequency then we will find the clock cycle time or time period or clock cycle using $T = 1/f$

or $f = 1/T$.

$1\ KHz = 10^3\ Hz =$ 1 milli second
$1\ MHz = 10^6\ Hz =$ 1 micro second
$1\ GHz = 10^9\ Hz =$ 1 nano second

If the frequency of processor is 1 KHz, then clock cycle is equal to $1/10^3$ or 10^{-3} sec = 1 ms

Q A system has 100 MHz processor. A program contains 50 instructions. Each instruction takes 10 clock cycles. Find out the performance of the system.?

$$T = \frac{N \times S}{R} = \frac{50 \times 10}{100 \times 10^6} = 5 \times 10^{-6} \text{ sec} = 5 \mu s$$

$$\text{Performance} = \frac{1}{T} = \frac{1}{5 \times 10^{-6}} = \frac{1}{\text{CPU execution time}} = 0.2 \text{ MHz}$$

Q There are two systems. Both are executing a program. One system has 100 MHz processor. Program contains 100 instructions. Each instruction takes 10 clock cycles (cc). Other system has 150 MHz processor. Program contains 50 instructions and each instruction takes 25 cc. Which system is better in performance ?

$$\text{Performance of computer 1} \left(\frac{1}{T_1}\right) = \frac{R}{N \times S} = \frac{100 \times 10^6}{100 \times 10} = 10^5 \text{ Hz}$$

$$\text{Performance of computer 2} \left(\frac{1}{T_2}\right) = \frac{150 \times 10^6}{50 \times 25} = 1.2 \times 10^5 \text{ Hz}$$

Total no. of clock pulses required by a program = N X CPI

N → No. of instruction in a single program

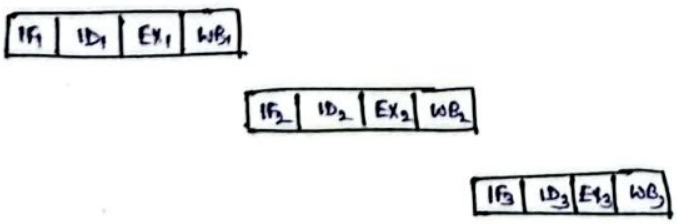
CPI → clock cycles per instruction

Pipelining

Execution of each program parallely done to speed up the system

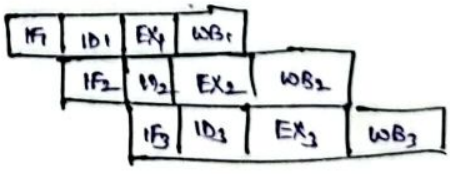
Execution of each instruction is passed through different stages sequentially that is -

i) Fetching (IF)
ii) Decoding (ID)
iii) Executing (EX)
iv) Write Back (WB)



Von Neumann Architecture

(Total clock pulse → 12)



Harvard Architecture

(Total clock pulse → 6)

A substantial improvement in performance can be achieved by overlapping the execution of successive instructions using a technique called pipelining.

Difference b/w RISC and CISC

RISC	CISC
(i) Port size is large	(i) Port size is small
(ii) No. of basic steps of an instruction is less	(ii) No. of basic steps of an instruction is more
(iii) It supports pipelining	(iii) It doesn't support pipelining
(iv) It takes one clock cycle	(iv) It takes multiple clock cycle
(v) LOAD and STORE commands are memory references	(v) Many commands are memory references
(vi) Requires more no. of general purpose registers	(vii) Requires less no. of general purpose registers

Instruction Set:

(i) Reduced Instruction Set Computers (RISC)

(ii) Complex Instruction Set Computers (CISC)

CPU Organisation:

i) Single Accumulator Organisation (One Address Instruction)

ii) General register Organisation (Two & Three Address Instruction)

iii) Stack Organisation (Zero Address Instruction)

X = (A + B) * (C + D)

- 3 Address Instruction

```
ADD  R1, A, B     // R1 ← Mem[A] + Mem[B]
ADD  R2, C, D     // R2 ← Mem[C] + Mem[D]
MUL  X, R1, R2    // X ← Mem[R1] + Mem[R2]
```

The order of operands vary from architecture to architecture

- 2 Address Instruction

```
MOV  R1, A        // R1 ← Mem[A]
ADD  R1, B        // R1 ← [R1] + Mem[B]
MOV  R2, C        // R2 ← Mem[C]
ADD  R2, D        // R2 ← [R2] + Mem[D]
MUL  R1 * R2      // R1 ← [R1] * [R2]
MOV  X, R1        // X ← [R1]
```

Super Scaler Operations:

* Multiple functional units are used so the multiple instructions can be executed parallely
* A higher degree of concurrency is achieved. Multiple instructions pipelines are implemented

One Address Instruction:

This is accumulator based. Accumulator is also one of the general purpose registers that is R0 to Rn-1. This uses accumulator register for all data manipulation. Here accumulator is assumed to be one of the operands for all instructions.

LOAD: We are using LOAD to fetch the data from memory to accumulator.

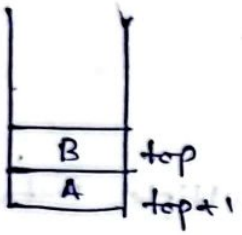
STORE: From accumulator to memory

X = (A+B) * (C+D)

```
LOAD   A      // Acc. ← Mem [A]
ADD    B      // Acc ← [Acc] + Mem [B]
STORE  T      // Mem [T] ← [Acc]
LOAD   C      // Acc. ← Mem [C]
ADD    D      // Acc ← [Acc] + Mem [D]
MUL    T      // Acc ← [Acc] * Mem [T]
STORE  X      // Mem [X] ← [Acc]
```

Zero Address Instruction

pop operation → Address will increase downwards
push operation → Address will decrease upwards



Mem [top+1] + Mem [top] → Mem [top+1] (A+B)
Mem [top+1] - Mem [top] → Mem [top+1] (A-B)
Mem [top] - Mem [top+1] → Mem [top+1] (B-A)

A stack organised computer doesn't use the address field for the instructions like ADD, SUB, XOR, OR. Operand is specified only for PUSH, POP operations. The top 2 contains from the stack is popped out and the operation is performed and their result is pushed back onto the stack.

```
PUSH   A      // Stack [top] ← Mem [A]
PUSH   B      // Stack [top] ← Mem [B]
ADD           // Stack [top+1] ← stack [top+1] + Stack [top]
PUSH   C      // Stack [top] ← Mem [C]
PUSH   D      // Stack [top] ← Mem [D]
ADD           // Stack [top+1] ← stack [top+1] + Stack [top]
MUL           // Stack [top+1] ← Stack [top+1] + Stack [top]
POP X         // Mem [X] ← Stack [top+1]
```

Q $X = (P + Q) * R / S + (T - U)/V$

3 Address Instruction

```
ADD  R1, P, Q      // R1 ← Mem[P] + Mem[Q]
MUL  R1, R1, R     // R1 ← [R1] * Mem[R]
DIV  R1, R1, S     // R1 ← [R1] / Mem[S]
SUB  R2, T, U      // R2 ← Mem[T] - Mem[U]
DIV  R2, R2, V     // R2 ← [R2] / Mem[V]
ADD  X, R1, R2     // X ← [R1] + [R2]
```

2 Address Instruction

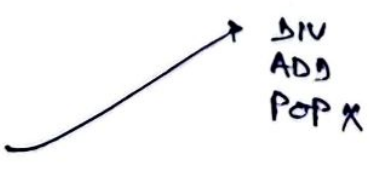
```
MOV  R1, P         // R1 ← Mem[A]
ADD  R1, Q         // R1 ← [R1] + Mem[Q]
MUL  R1, R         // R1 ← [R1] * Mem[R]
DIV  R1, S         // R1 ← [R1] / Mem[S]
MOV  R2, T         // R2 ← Mem[T]
SUB  R2, U         // R2 ← [R2] - Mem[U]
DIV  R2, V         // R2 ← [R2] / Mem[V]
ADD  R1, R2        // R1 ← [R1] + [R2]
MOV  X, R1         // X ← [R1]
```

1 Address instruction

```
LOAD  P            // Acc. ← Mem[P]
ADD   Q            // Acc ← [Acc] + Mem[Q]
MUL   R            // Acc ← [Acc] * Mem[R]
DIV   S            // Acc ← [Acc] / Mem[S]
STORE M            // Mem[M] ← [Acc]
LOAD  T            // Acc. ← Mem[T]
SUB   U            // Acc ← [Acc] - Mem[U]
DIV   V            // Acc. ← [Acc] / Mem[V]
ADD   M            // Acc. ← [Acc] + Mem[M]
STORE X            // Mem[X] ← Acc
```

Zero Address Instruction

```
PUSH P
PUSH Q
ADD
PUSH R
MUL
PUSH S
DIV
PUSH T
PUSH U
SUB
PUSH V
```



RISC (Reduced Instruction Set Computer)

(i) All data manipulation operations take place using 3 address instructions and all three registers are general purpose registers.

(ii) To transfer data from memory to general purpose register, we use load Ri, M where Ri is a general purpose register (GPR) and M is a memory location.

(iii) To transfer data from GPR to memory location, we store M, Ri.

Q $X = (A+B) * (C+D)$

```
LOAD  A, R1      // R1 ← mem[A]
LOAD  B, R2      // R2 ← mem[B]
LOAD  C, R3      // R3 ← mem[C]
LOAD  D, R4      // R4 ← mem[D]
ADD   R1, R2, R5 // R5 ← R1+R2
ADD   R3, R4, R6 // R6 ← R3+R4
MUL   R5, R6, R7 // R7 ← R5+R6
STORE R7, X      // Mem[X] ← R7
```

Q $X = (P+Q) * R/S + (T-U)/V$

```
LOAD  P, R1
LOAD  Q, R2
ADD   R1, R2, R1
LOAD  R, R2
MUL   R1, R2, R1
LOAD  T, R2
LOAD  U, R3
SUB   R2, R3, R2
LOAD  V, R3
DIV   R2, R3, R2
ADD   R1, R2, R1
STORE R1, X
```

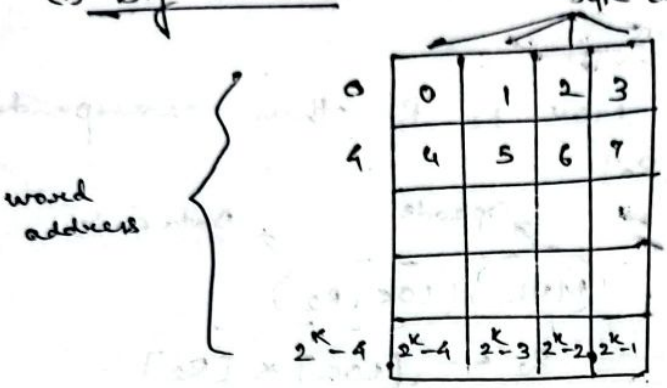
Byte Addressing

There are two types of byte addressing

i) Big endian
ii) Little endian

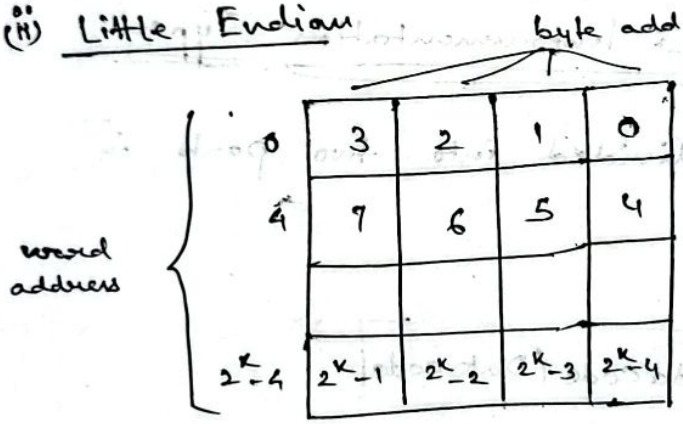
Two ways of addressing byte addresses across words

(i) Big Endian



When lower byte address are assigned to the MSB part of the word then it is called big eidian assignment

(ii) Little Endian



When lower byte addresses are assigned to the LSB part of the word then it is called little endian assignment

Instruction and instruction sequencing

A computer must have instructions capable of performing 4 types of operations

(i) Data transfer b/w processor register
(ii) ALU operations on data
(iii) Programming sequencing and control
(iv) I/O transfer

- Register transfer notation (RTN)

  Transfer of information from one location in the computer to other locations is called RTN

There are different types of location:

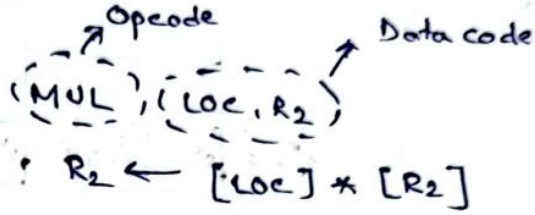
i) Memory location :
Eg: A, B, LOC, DATA

ii) Processor registers
Eg: R_0, R_1, ... R_{n-1}

iii) Registers in the i/o devices
Eg: DATAIN (input device)
DATAOUT (output)
INSTATUS, OUTSTATUS

→ If it is one instruction MOV R_0 R_1 then corresponding RTN notation is $R_1 \leftarrow [R_0]$

→ ADD A, R_0
$R_0 \leftarrow [A] + [R_0]$



■ CPU Organisation / Basic Instruction Types

Every instruction is divided into two parts in assembly language

Op code	Add code / Data code

There are different types of instructions :

i) Three address inst
ii) Two address "
} General register organisation

iii) One address "
iv) Zero address "
} Simple accumulator organisation
Stack organisation

Addressing Modes

The way of representation of an operand in an instruction are called addressing modes

- There are different types of addressing modes:

(i) Implied Addressing Mode
(ii) Immediate Addressing Mode
(iii) Register " "
(iv) Direct " " / Absolute Addressing Mode
(v) Indirect
    - Register Indirect
    - Memory Indirect
(vi) Index
(vii) Relative
(viii) Base
(ix) Auto increment addressing mode
(x) " decrement " "

Implied Addressing Mode

In this mode the operands are specified implicitly in instruction

Ex: COM → Complement Accumulator

All register reference instruction that uses accumulator and all zero address instruction are implied addressing mode.

Immediate Addressing Mode

This addressing mode is used to deal with the constants.

In this mode operands are directly specified in instruction

Ex: MOV #200, R1 (# → constant type)

OR

MVI 200, R1 (200 → constant)

Ex: A = B + 10

```
MOV  B, R1
ADD  #10, R1
STORE R1, A
```

Direct / Absolute Addressing Modes

In this mode, the operand is a memory location or memory address given explicitly in the instruction

Ex: ADD A, B or ADD 1000, B

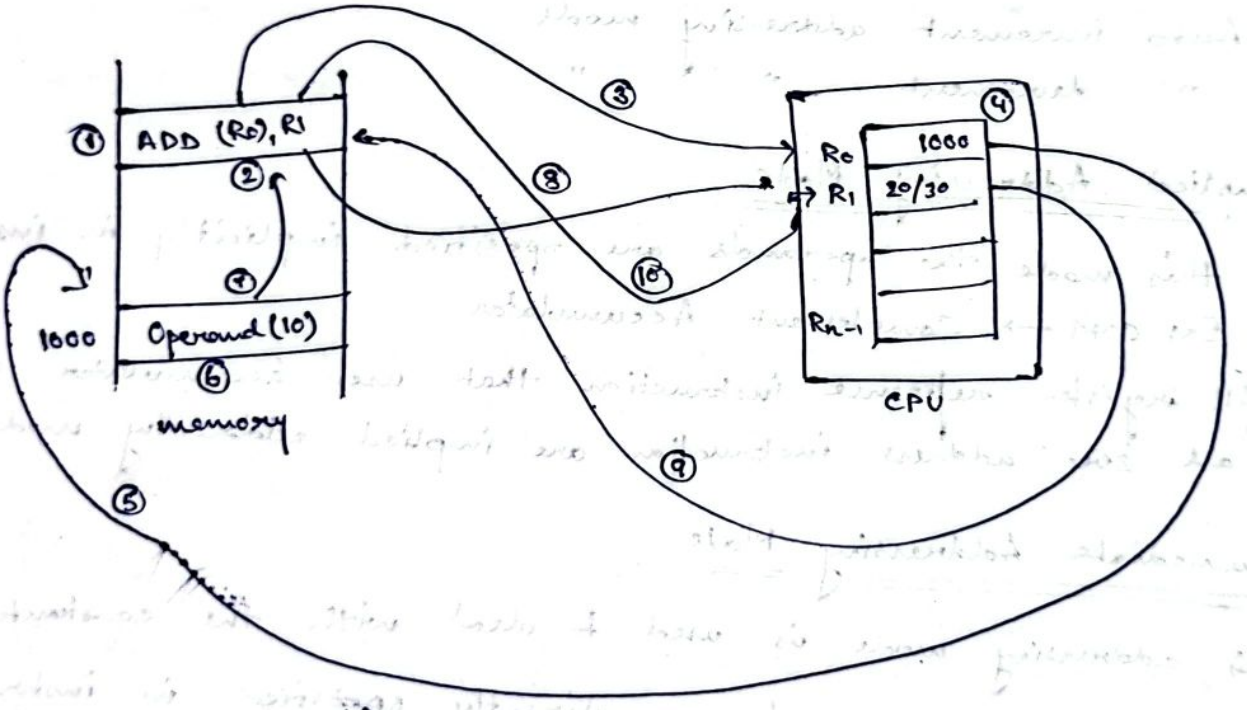
Here the effective address is equal to the address part of the instruction.

Effective address provides the information from which the operand is determined.

Ex: ADD 1000, R1 , MOV A, R1

Indirect Addressing Mode

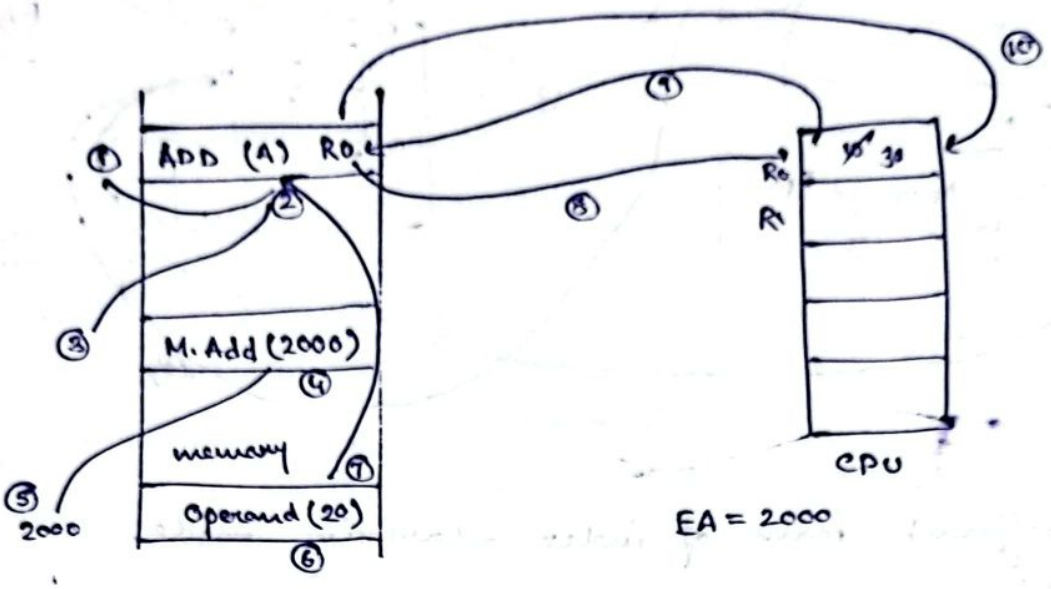
- Register indirect addressing mode: The effective address of the operand is the content of the processor register whose address appears in the instructions.



EA = 1000 (operand)

- Memory indirect addressing modes

The EA of an operand is the content of a memory instruction whose address appears in the instruction.



Q Write ~~an second~~ assembly code to add n numbers

Address	Contents	
	MOV	N ; Ri
	MOV	#NUM1 , R2
	Clear	R0
Loop →	ADD	(R2) , R0
	ADD	#4 , R2
	DEC	R1
	Branch >0	Loop
	MOV	R0 , SOM

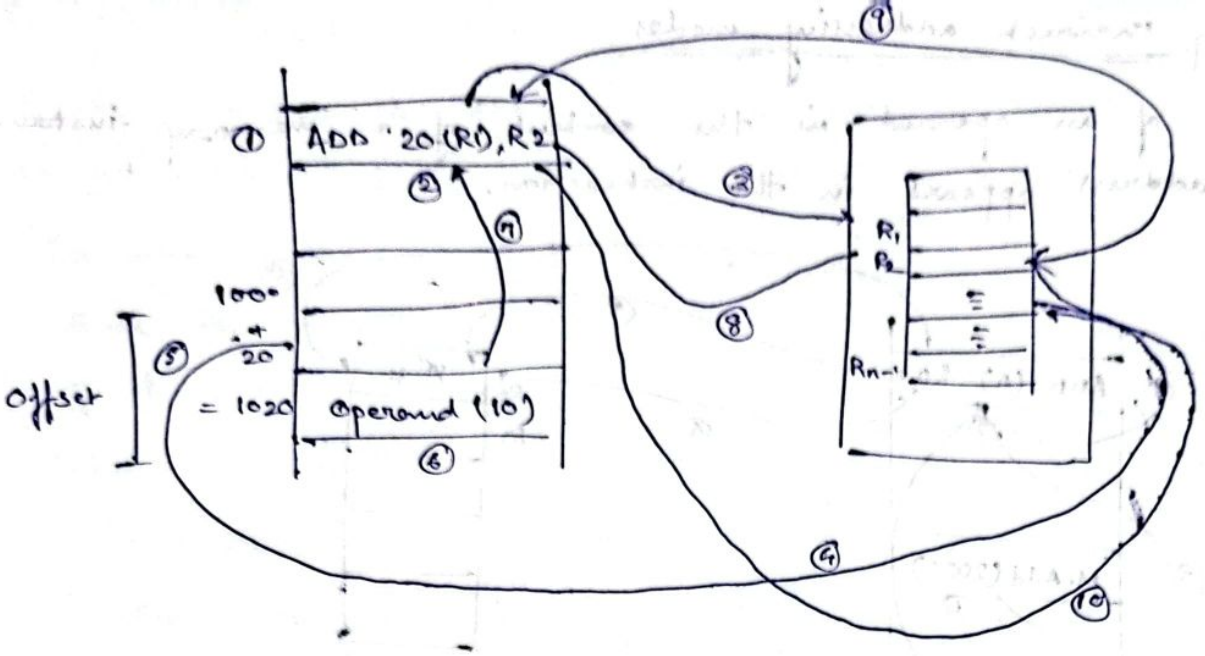
- Index Register

X(Ri) ⟶ IR

where X is a constant value or a address part of the instruction

and Ri is the index register

EA = X + [Ri]



There are different form of index addressing mode.

(i) (R_i, R_j)

$EA = [R_i] + [R_j]$

It is used to represent double dimensional array

(ii) $\overbrace{X}^{\text{offset}}(R_i, R_j)$

$EA = X + [R_i] + [R_j]$

It is used to represent three dimensional array

This R_i is a index register or general purpose register.

- Relative Addressing Mode

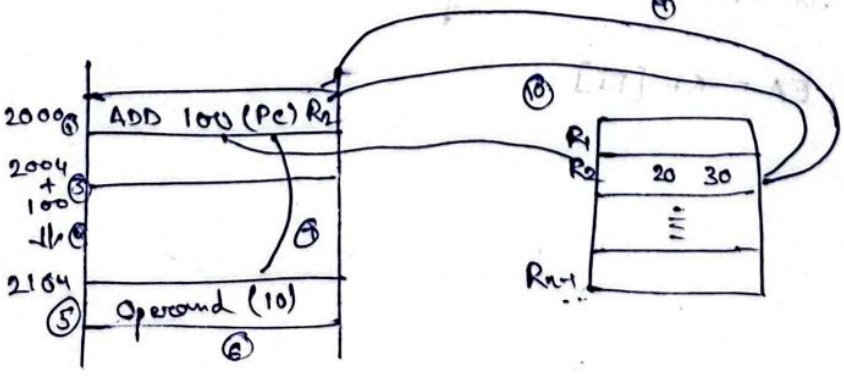
Denoted by X(PC)

where X ⟶ Offset / constant value / address

PC ⟶ program counter

Effective Address (EA) = X + [PC]

It is called relative addressing mode because the address location identified is relative to the PC.



- **Base Addressing Mode**

In this mode the content of the base register it added to the address part of the instruction to obtain the effective address.

ADD 1000 (BR), R2

EA = 1000 + [BR]

- **Auto increment addressing mode**

Denoted by :- (Ri)+

It has two operations :

(i) First access the operands

(ii) Increment the register that contains the operand address

first fetch the operand then increment the content of register depending upon the memory location.

- **Auto decrement addressing mode**

Denoted by :- -(Ri)

It has two operations :

(i) Decrement the register that contains operand address

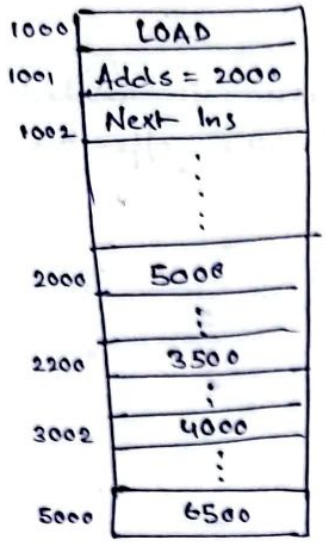
(ii) Access the operand from new decremented address

Q A 2 word instruction LOAD is stored at location 1000 and its address field at location 1001. The address field has the value 2000 and the value stored at 2000 is 5000 and at 5000 it is 6500. The word stores at 2200, 3002, 2000, 3500, 4000 respectively, Index register has the value 2000. Evaluate the effective address (EA) and operand if the addressing mode of instruction as follows :

i) Memory indirect addressing mode

ii) Relative addressing mode

iii) Index addressing mode



i) Memory indirect → Address of the address of the operand

Middle address → EA

EA = 5000

[EA] = 6500 ↳ operand.

(ii) Relative addressing X (PC)

EA = X + [PC]

= 2000 + 1002

= 3002

[EA] = 4000

(iii) Index addressing X (Ri)

EA = X + [Ri]

= 2000 + 200

= 2200

[EA] = 3500

Q An instruction is kept in memory at an address 300 and the memory address 301 occupies the address field of the instruction which is shown below. The opcode is used to add the content of accumulator with an operand. The content of accumulator is 100 and the content register R_5 is 400. Find out the content of accumulator and effective address of operand if the addressing mode is (i) immediate (ii) direct (iii) register direct (iv) indirect (v) register indirect

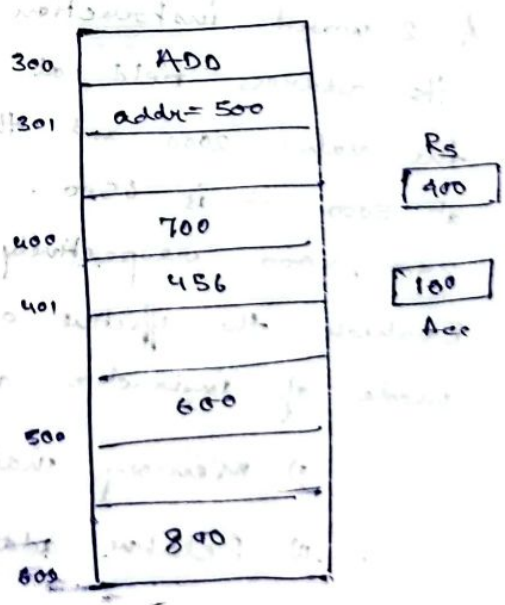
(i) Immediate addressing mode

EA = NULL

[Acc] = [Acc] + Imm. Operand (Value of the add. field)

= 100 + 500

= 600



(II) Direct Addressing Mode :

EA = Value of address field = 500

Acc = [Acc] + mem [500]

= 100 + 600

= 700

(III) Register direct addressing mode :

EA = Ri = Name of the register = R5

Acc = [Acc] + [R5]

= 100 + 400

= 500

(IV) Indirect Addressing mode :

EA = [Value of the address field] = mem [500] = 600

[Acc] = [Acc] + mem [EA]

= 100 + 800

= 900

(V) Register Indirect addressing mode :

EA = [Ri] = [R5] = 400

[Acc] = [Acc] + mem [EA]

= 100 + mem [400]

= 100 + 700

= 800

Q If PC = 2004 after executing the instruction pointed by PC. what is the updated value of PC ? (Initial value of R1 = 10, R2 = 100)

memory location	instruction
2000	ADD R1 R2
2004	BNZ 1000
2008	Next Instr.

R1 + R2 = 110 ≠ 0

BNZ is a conditional branch instruction, it will jump to the instruction specified in its label field if the result of the previous instruction is not equal to zero

By the time the instruction at 2004 is executed, the Pc will incremented to a point to the next instruction in sequence.

Hence Pc = 2008 + 1000

= 3008

Q If Pc = 2004, after executing the instruction pointed by Pc which will be the updated value of Pc ? (Initial value of $R_1 = 10$, $R_2 = 100$)

memory location	Instruction
2000	ADD R1, R2
2004	BZ 1000
2008	Next Instn.

Result of addition of R_1 and R_2 is 110 $\neq 0$

BZ is a conditional branch condition instruction, it will jump to the instruction specified in its Label field, if the result of the previous instruction is EQUAL to ZERO.

By the time the instruction at 2004 is executed, the Pc will be incremented to point to the next instruction in sequence

Hence PC = 2008 as the branch condition is not true.

Q Initial value of $R_1 = 10$, $R_2 = -100$

memory location	Instruction
2000	ADD R1, R2
2004	Branch >0 1000
2008	Next instruction

$R_1 + R_2 = -90$ is not greater than 0

Branch > 0 is a conditional branch instruction, it will jump to the instruction specified in its labelled field, if the result of the previous instruction is greater than zero

By the time the instruction at 2004 is executed, the PC will be incremented to point to the next instruction in sequence (2008)

Hence PC = 2008 as the branch condition is not true.

Q If PC = 2048 and at 2048 memory location "JUMP 100" instruction is there. After executing this instruction, what will be the value of PC ?

memory Location	Instruction
2048	JUMP 100
2052	Next Instruction

JUMP is an unconditional branch instruction irrespective of the result of the ~~process~~ previous instruction it will jump to the instruction specified in the label field [No condition checking]

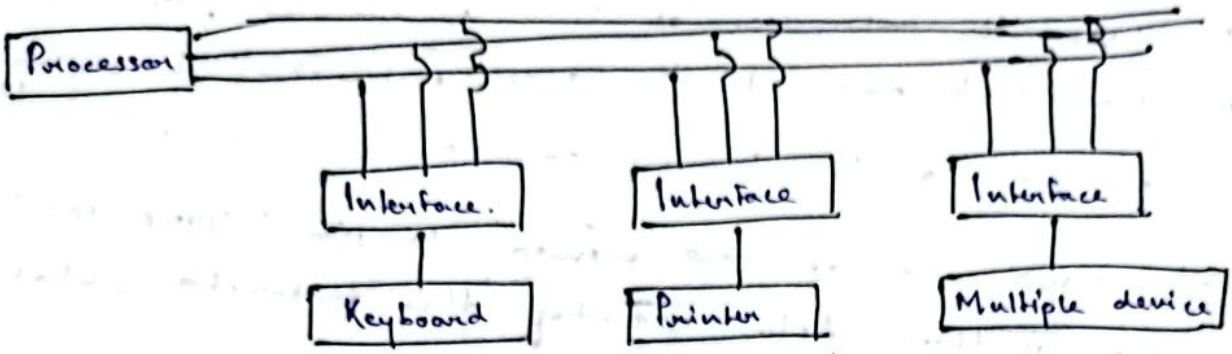
By the time, the instruction at 2048 is executed, the PC will be incremented to point to the next instruction in sequence (2052).

Hence PC = 2052 + 1000

= 3052

Input Output Organisation

Input - output devices aren't directly connected to the processor It is connected via memory.

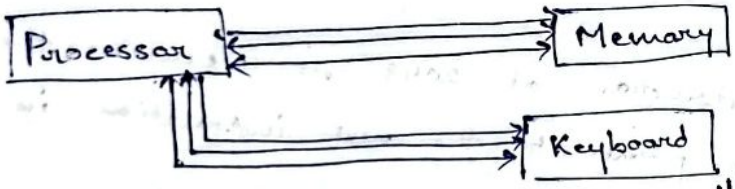


There are 4 types of I/O commands

(i) Control command
(ii) Status "
(iii) Data o/p "
(iv) Data i/p "

There are 3 ways memory & I/O devices are connected with the preparation.

Case I



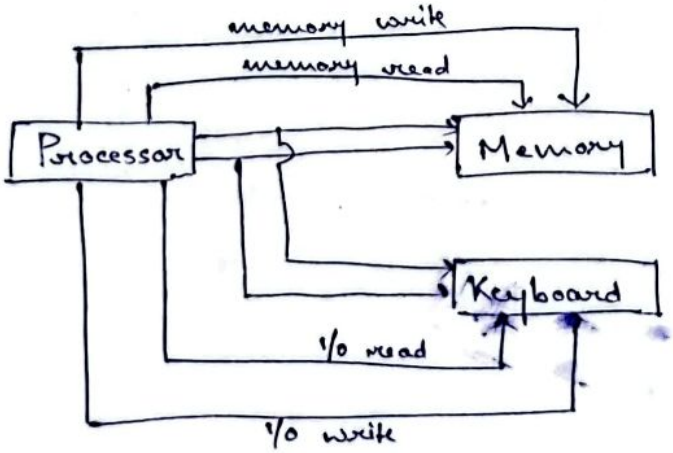
Address bus → Unidirectional
Data bus → Birectional
Control bus → "

Disadvantages:
High cost in manufacturing

Use two separate buses one for memory and one for I/O

memory write

Case II



Use one common bus for both but separate control lines for each

This is called I/O mapped I/O or isolated I/O

Case III Here address bus, data bus and control bus are common, it is called memory mapped I/O because some of the addresses are there for storing the address of I/O devices.

memory mapped I/O	I/O Mapped I/O
① Some of the addresses will be allocated to memory location and others for I/O devices	① Memory and I/O devices will be addressed by the same addresses
② Total no. of memory and I/O devices will be less	② Total no of memory and I/O devices will be more as they uses all the address
③ Some instruction like LOAD and STORE can be used for I/O operations	③ Separate instructions are used for memory and I/O devices.
④ The address will itself identify whether it is a memory or I/O operation.	④ I/O device can have some address so extra hardware is required to distinguish.
⑤ Length of instruction is more since addressing is not fixed.	⑤ Length of instruction is less

■ Mode of transfer:

There are 3 different methods of data transfer

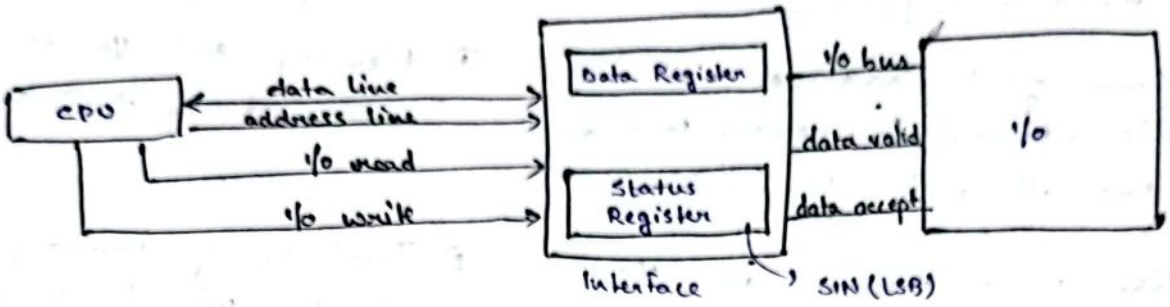
(i) Program control data transfer
(ii) Interrupt initiated I/O
} Controlled by the processor

(iii) Direct memory access (DMA) ——→ Not controlled by the processor

Data and status register all are of 1 byte to store an ASCII char.

① Program control data transfer :

LSB of status register ⟶ Sin = 1 ⟶ read by memory
Sin = 0 ⟶ after reading



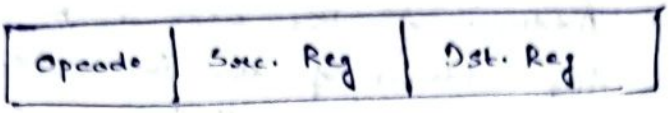
Sin = 1 (When a valid character is entered) ⇒ Data is transferred from Keyboard to memory

Sin = 0 (invalid character)

Disadvantage

Here the CPU wastes its time by checking the flag register. instead of doing any useful computational work.

Q A general purpose register organisation has a 16 bit instruction consisting of opcode, source register and a destination register. It supports 7 no. of arithmetic operations and 6 no. of logical operations. Find the total no. of maximum registers present in the system.



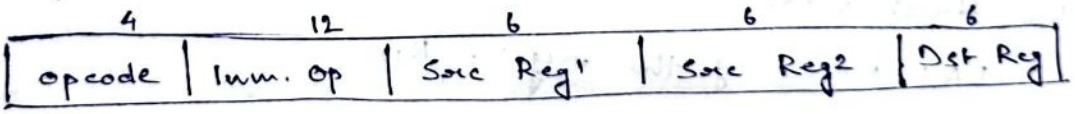
To find the total no. of opcodes
= total no. of operations possible
= 7 no. of arithmetic opcode + 6 no. of logical opcode
= 13 operations.

No. of bits required for opcode = 4 bits ($2^4 = 16 > 13$ ✓, $2^3 = 8 < 13$ ×)

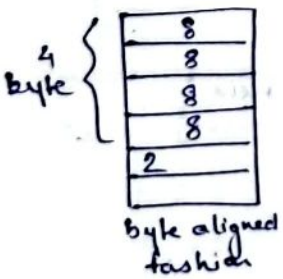
No. of bits required for registers = 16 − 4
= 12 bits (for 2 reg)
= 12/2 = 6 bits (for each reg)

Max. no. of registers = 2^6 = 64 (R_0 to R_{63})

Q Consider a processor with 64 registers that supports 12 instructions. Each instruction has 5 distinct fields namely opcode, 2 source register fields, 1 destination register field and a 12 bit immediate value. Each instruction must be stored in memory in a byte aligned fashion. If a program has 100 instructions. What is the amount of memory (in bytes) consumed by a program



Length of instruction = (4+12+6+6+6) = 34 bits



34 bits = (8×4) + 2 = 4 bytes + 2 bits

So total bytes required for a single instruction = (4+1) = 5 bytes

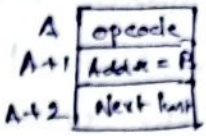
We have to allocate an entire byte space

∴ Memory required for 100 instructions = (100×5) = 500 bytes

Q A two word instruction is stored in a location A. The operand part of instruction holds B. If the addressing mode is relative, the operand is available in which location ?

In relative mode,

EA = Updated value of PC + offset
= A + 2 + B



Updated value of PC = Address of the next instruction i.e. the address of the instruction following the branch instruction.

Q A relative mode branch type instruction is stored in memory at an address 750. The branch is made to an address 500. What should be the value of the relative address field of the instruction ?

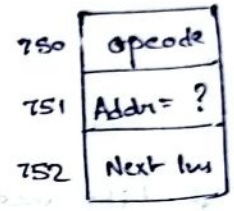
In relative mode,

EA = Updated value of PC + Offset

500 = 752 + Offset

Offset = -252

Address field value = -252



Q An instruction is stored at location 200 with its address field having the value 10. A processor register R10 contains the value 210 which is also used as index register. Evaluate the effective address of the operand if the addressing mode of the instruction is (i) direct (ii) register direct (iii) register indirect (iv) indexed

Direct Addressing mode,

EA = value of address field = 10

Register direct mode

EA = Name of the reg = R_{10}

Register indirect mode

EA = [R10] = 210

Index addressing mode,

EA = [Index reg] + Offset (Value of the address field)

= 210 + 10

= 220

① A two word instruction is stored in memory at an address designated by the symbol P. The address field of the instruction (Stored at P+1) is designated by the symbol Q. The operand used during the execution of the instruction is stored at an address symbolized by EA. An index register contains the value X. State how EA is calculated from the other addresses if the addressing mode of the instruction is direct, indirect, relative and indexed.

Direct addressing mode

EA = Value of address field
= Q

Memory indirect addressing mode

EA = [Value of address field] = [Q]

Relative addressing mode

EA = Updated [PC] + offset (value of address field)
= P + 2 + Q

Index Addressing Mode

EA = [Index reg] + Offset (Value of the address field)
= X + Q

② A two word instruction LOAD is stored at location 1000 with its address field at location 1001. The address field has the value 2000 and the value stored at 2000 is 5000 and at 5000 is 6500. The words stored at 2200, 3002 are 3500, 4000 respectively. An index register has value 200. Evaluate the effective address and operand if addressing mode of the instruction is as follows:

(i) memory indirect (ii) relative (iii) index.

memory indirect addressing mode

EA = [value of address field]

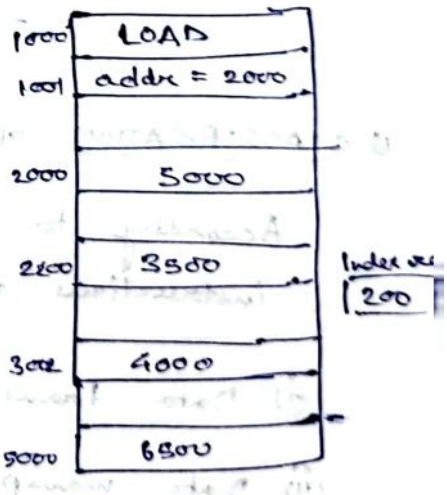
[2000] = 5000 Operand = [5000] = 6500

Relative addressing mode

EA = Updated [PC] + Offset (Value of address field)

= 1002 + 2000

= 3002 Operand = [3002] = 4000



Index addressing mode

EA = [Index reg] + Offset (Value of address field)

= 200 + 2000

= 2200

Operand = [2200] = 3500

■ Memory Reference : The number of times we refer to memory is called memory reference.

(i) To fetch an instruction from memory for execution, requires one fetch cycle / memory reference.

(ii) If any instruction is having one memory operand then 2 memory reference is required.

(iii) If the result need to be stored in a memory location in the above case then 3 memory reference is required.

Ex: ADD R1, R2
Fetch = 1
TMR = 1

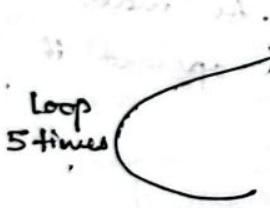
ADD A, R1
Instr fetch = 1
Operand = 1
TMR = 2

INC R1
TMR = 1

INC A
TMR = 3

ADD A, B
Fetch Instr = 1
1st operand = 1
2nd operand = 1
Result = 1
4

Q Calculate the total no. of memory references for execution of the following program.



MOV #5, R1 ⇒ 1
ADD (R0), R2 ⇒ 2
DEC R2 ⇒ 1
Branch > 0 Loop ⇒ 2
R1

TMR = (1+2+1) 5 + 1 = 26

■ CLASSIFICATION OF INSTRUCTION BASED ON COMPUTATION

According to operation carried out by the ~~operati~~ computer, instructions are classified into three types:

(i) Data transferred instruction

(ii) Data manipulation instruction → ~~Instruc~~ Arithmetic, Logical, Shift

(iii) Program control instruction

- **Data transferred instruction** : The instruction move data from one location to another location without changing data content

Name	Mnemonic	Meaning
LOAD	LD	memory to register
STORE	ST	Register to memory
MOVE	MOV	Transfer b/w registers
Exchange	XCH	Swaps the content of data.
Input	IN	
Output	OUT	

- **Data manipulation instruction** :
  - **Arithmetic instruction** :

Name	Mnemonic
Increment	INC
Decrement	DEC
Addition	ADD
Subtraction	SUB
Multiplication	MUL
Division	DIV
Add with carry	ADDC
Sub with borrow	SUBB
Neglate	NEG (2's complement)

  - **Logical instruction**

Name	Mnemonic
CLEAR	CLR
Complement	COM
AND	ANA
OR	oR
Exclusive OR	XOR
Clear Carry	CLRC
Set Carry	SETC
Enable interrupt	EI
Disable interrupt	DI

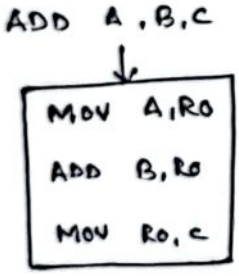
- Shift Instruction

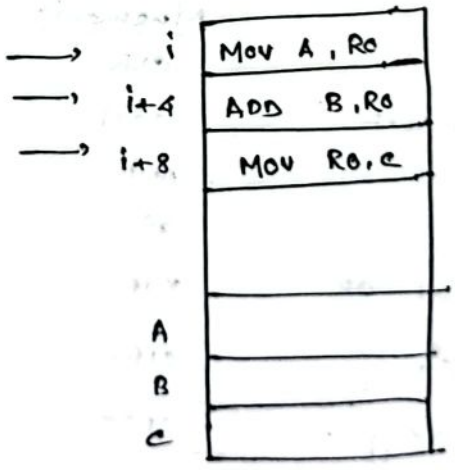
Name	Mnemonic
Logical Shift Right	SHR / LShiftR
Logical Shift Left	SHL / LShiftL
Arithmetic Shift Right	ASHR / AShiftR
Arithmetic Shift Left	ASHL / AShiftL
Rotate Right	ROR / RotateR
Rotate Left	ROL / RotateL
Rotate right through carry	RORC
Rotate left through carry	ROLC

- **Program Control Instructions** : To control the programs

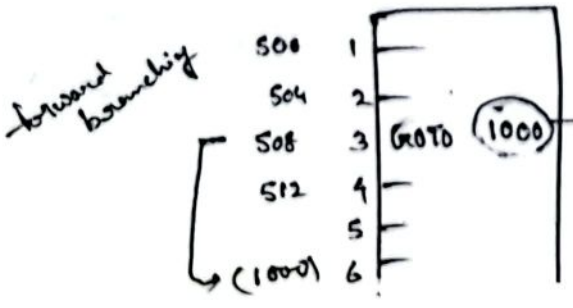
Name	Mnemonic
Branch	BR ⇒ Conditional Jump
Jump	JMP ⇒ Unconditional Jump
Subroutine Call	CALL
Subroutine routine	RET
Compare	CMP
Test	TST

- **Instruction Execution and Straight Line Sequencing**





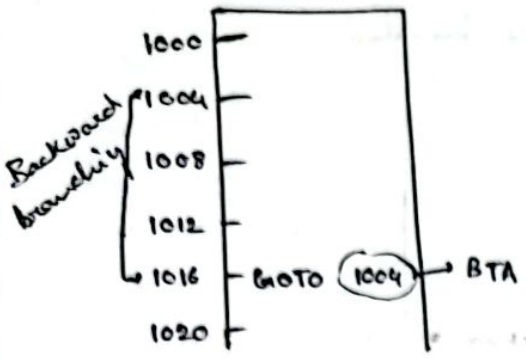
- **Branching**
  - Forward
  - Backward



PC = 512

Branch target address
(while executing this address, we will come to know which is my BTA)

{1000 − 512 = 488 (offset)}



Content of PC is 1020

[offset = BTA - PC]

" = 1004 - 1020

= -16

Condition Code :

- The processor keeps track results of various operation performed by the conditional branch operation.
- This information is recorded in individual but called individual code flags.
- These flags are grouped together in a special processor register called condition code register or status register.

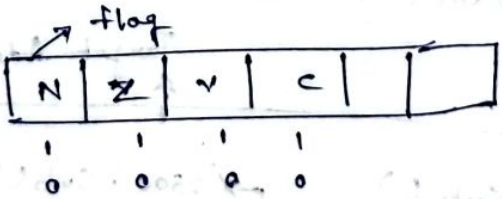
There are 4 types of flag used :

① N (Negative) : It is set to 1 if the result is -ve otherwise cleared to 0.

② Z (zero) : It is set to 1 if the result is 0 otherwise cleared to 0.

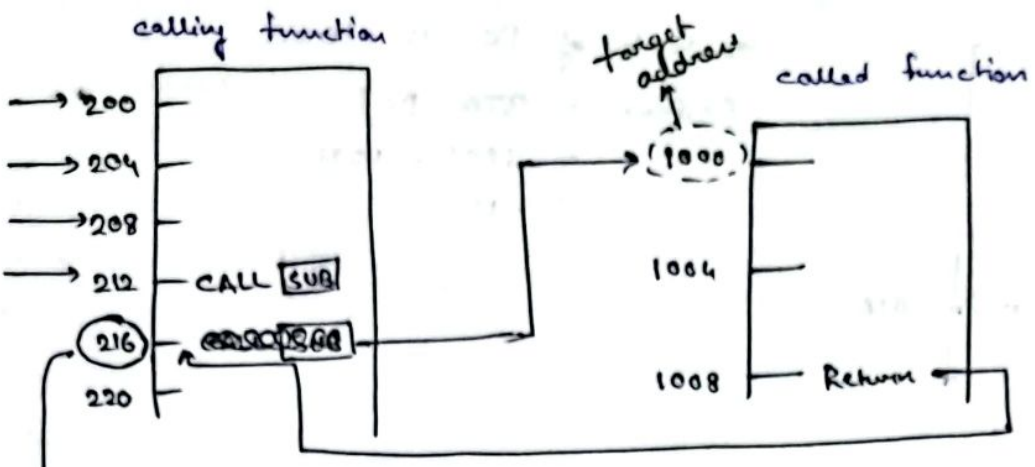
③ V (Overflow) : It is set to 1 if arithmetic overflow occurs otherwise cleared to 0.

④ C (Carry) : It is set to 1 if a carry out results from the operation otherwise cleared to 0.

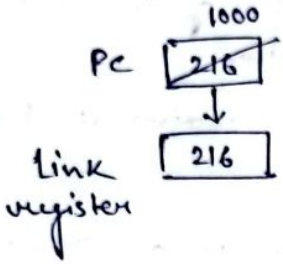


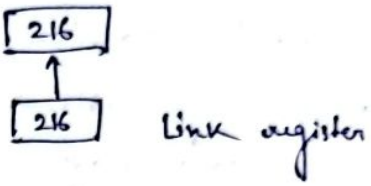
Subroutine

- When we perform a particular subtask many times on different data value then, subtask is known as subroutine.
- It is used to save the memory space



Return address of subroutine

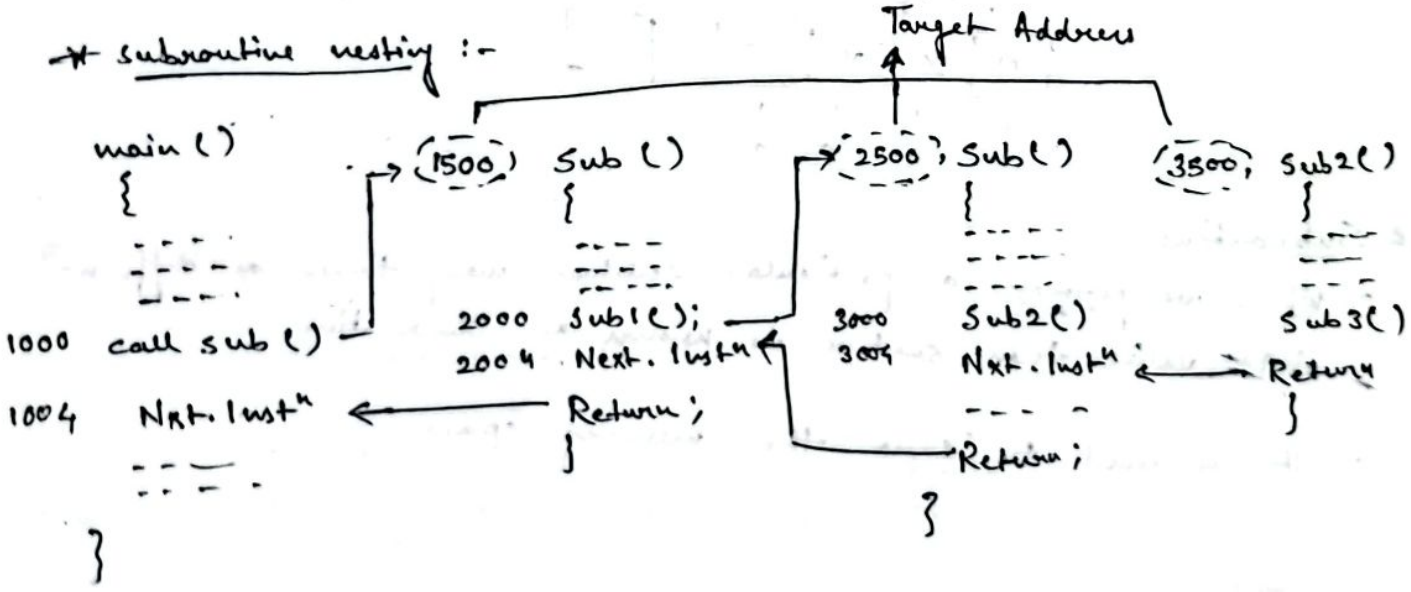




- The way of execution of call and return instruction of subroutine is called subroutine linkage method.
- Here subroutine linkage method saves the return address of subroutine in link register.
- A call instruction is a branch instruction that performs following operations:
  - (i) Stores the content of PC to link register
  - (ii) Branch to target address.

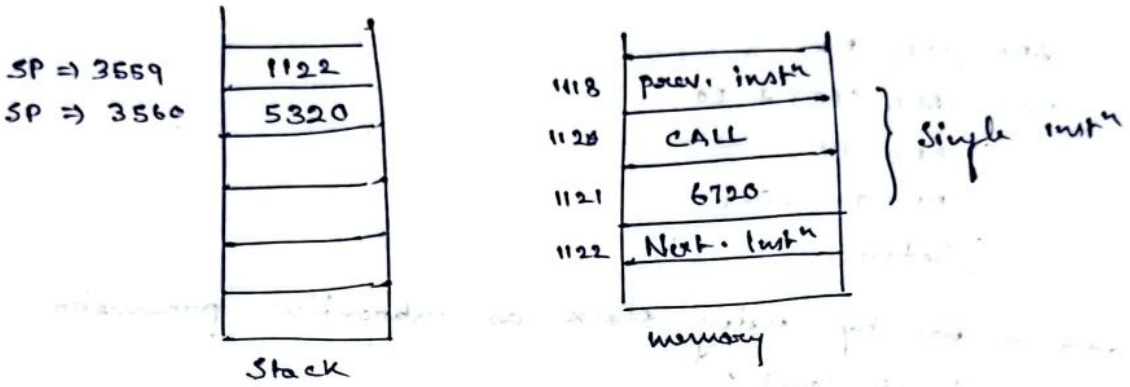
The return instruction is a branch instruction that performs
- (i) Branch to the address contain in the link register

* Subroutine nesting :-



Q The content of the top of the stack is 5320. The content of stack pointer (SP) is 3560. A two word call subroutine inst^n located in memory at address 1120 followed by an address field of 6720 at ~~address~~ location 1121. What is the content of PC, SP and TOS ?

(i) Before call instruction execution.
(ii) After " " "
(iii) After return from the subroutine



Before call inst^n

Value of SP = 3560
" TOS = 5320
" PC = 1120

After call instruction

PC : 6720 (address from the call instruction address field)
SP : 3560 (unchanged)
TOS : 5320 (unchanged)

After return from subroutine

PC : 1122 (address immediately after call instruction)
SP : 3560
TOS : 5320

Parameter passing: The parameter of the subroutine may be processor registers or memory locations.

Q WAP to add n no. by using subroutine calling program

```
MOV   N,R1 ;
MOV   #NUM1 , R2
CALL  ADD
MOV   R0, SUM
Subroutine

ADD   CLR  R0
LOOP  ADD  (R2) + R0
      DEC  R1
      R1 > 0   LOOP
      Return.
```

Q WAP to add n no. by using stack as subroutine parameter
Assuming TOS is at level 1

calling program

```
MOV   #NUM1 , - (SP)
MOV   N , - (SP)
CALL  ADD
MOV   4(SP), SUM
ADD   #8, SP
```

Subroutine

```
ADD   Movemultiple   R0-R2 , - (SP)
MOV   16(SP) , R1
MOV   20 (SP) , R2
CLR   R0
LOOP  ADD   (R2)+ , R0
      DEC   R1
      R1>0  LOOP
      MOV   R0 , 20(SP)
Movemultiple  (SP)+ ,  R0 - R2
Return
```

BASIC PROCESSING UNIT

Executing an instruction

- Fetch the content of the memory location points to be the Pc. The contents of this location are loaded into IR (fetch phase)

$$IR \leftarrow [PC]$$

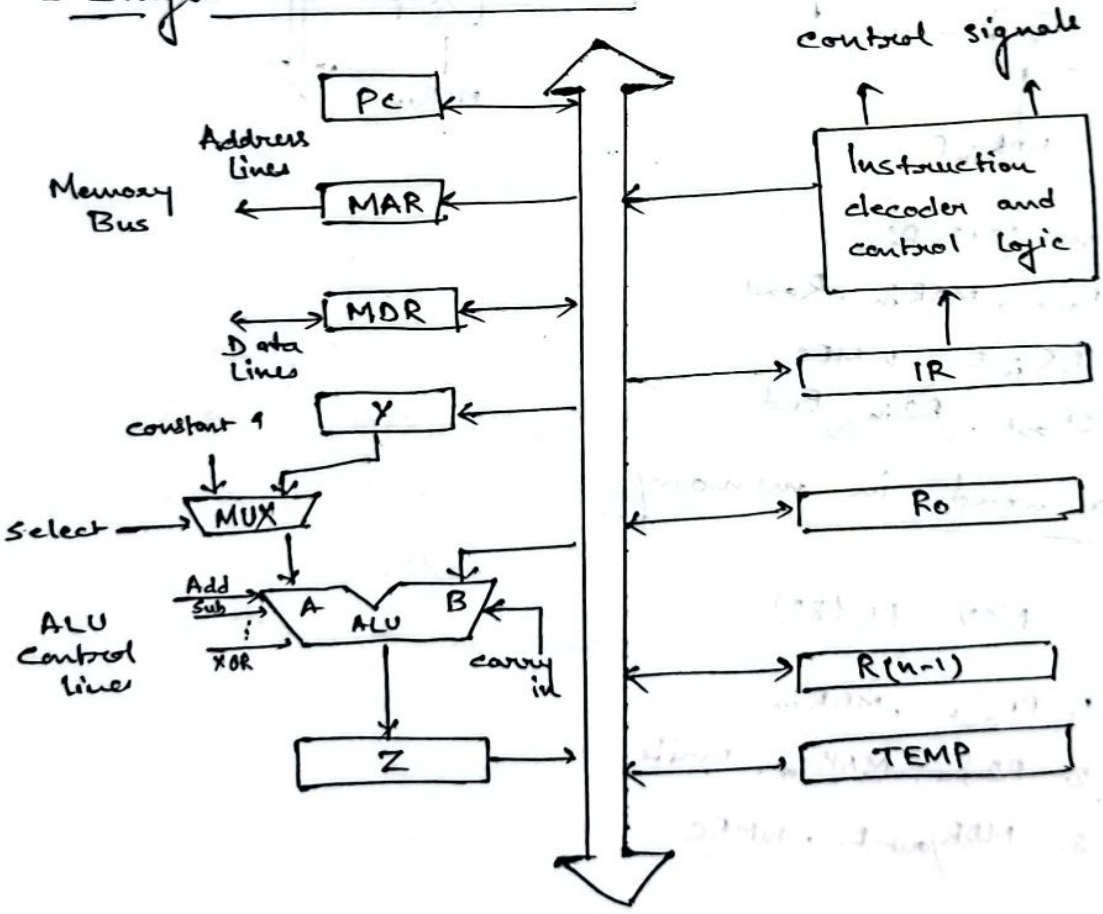
- Assuming that the memory is byte addressable, increament the content of the Pc by 4 (fetch phase)

$$Pc \leftarrow [PC] + 4$$

- Carry out the actions specified by the instruction in the IR (execution phase).

In case where an instruction occupies more than one word, steps 1 and 2 must be repeated as many times as neccessary to fetch the complete instruction.

Single bus architecture



- MDR has two inputs and two outputs
- 3 temp registers used - Y, Z, temp
- WMFC — Wait for memory function completed

Performing an arithmetic operation

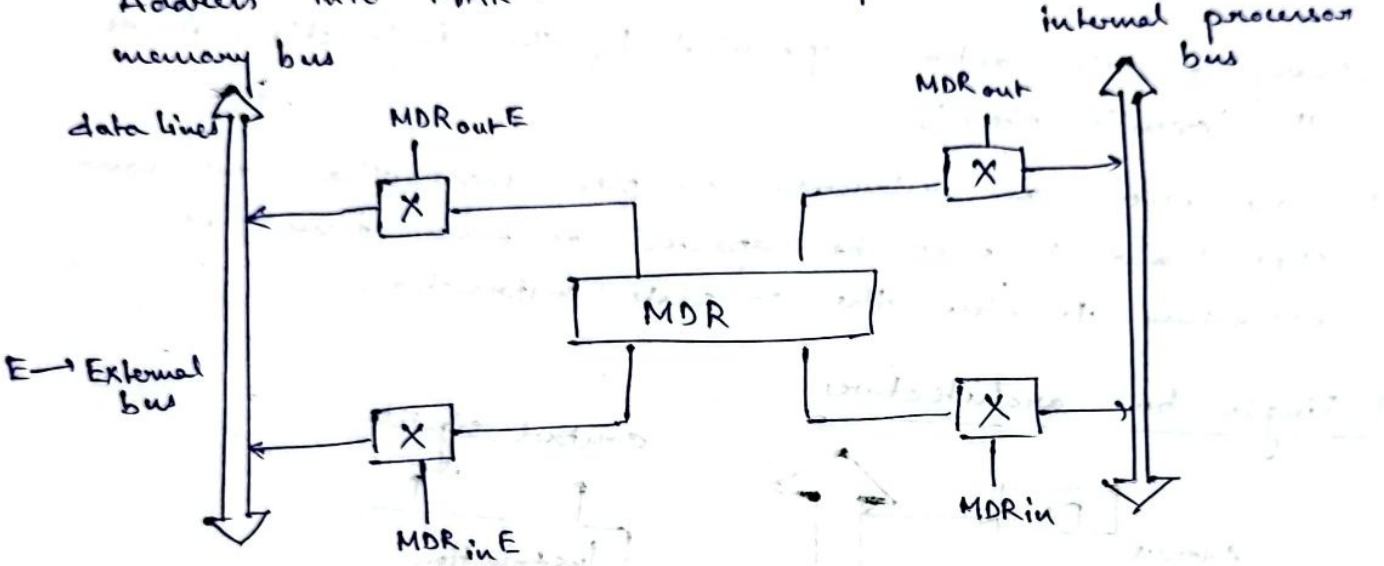
ADD R1, R0 || R2 ⟵ [R1] + [R2]

The corresponding micro instruction is

1. R1out , Yin
2. R2out , Select Y , ADD , Zin
3. Zout , R2in , End

Fetch / Read a word from the memory

Address into MAR ; issue Read operation ; Data into MDR



MOV (R1), R2

1. R1out , MARin , Read
2. MDRinE , WMFC
3. MDRout , R2in , End

Storing a word in memory

MOV R1, (R2)

1. R1out , MDRin
2. R2out , MARin , Write
3. MDRoutE , WMFC

Execution of complete instruction

Q Write the micro instruction for the following : ADD R1, R2, R3

Fetching input from memory:
1. PC_{out}, MARin, Read, Select 4, ADD, Zin
2. Z_{out}, PCin, Yin, WMFC
3. MDRout, IRin

4. R1out, Yin
5. R2out, Select Y, ADD, Zin
6. Z_{out}, R3in, End

Q Write the micro routine for the given instruction: ADD (R3), R1

1. PC_{out}, MARin, Read, Select 4, ADD, Zin
2. Z_{out}, PCin, Yin, WMFC
3. MDRout, IRin
4. Offset_field_of_IRout, Yin
5. ~~R1out~~ R3out, Select Y, ADD, Zin
6. Z_{out}, MARin, Read
7. R1out, Yin, WMFC
8. MDRout, Select Y, ADD, Zin
9. Z_{out}, Rin, End

Q ADD (R3)-, R1

1. PC_{out}, MARin, Read, Select 4, ADD, Zin
2. Z_{out}, PCin, Yin, WMFC
3. MDRout, IRin
4. R3out, MARin, Read, Select 4, SUB, Zin
5. Z_{out}, R3in
6. R1out, Yin, WMFC
7. MDRout, Select Y, ADD, Zin
8. Z_{out}, Rin, End

Example-6:

- Write the sequence of control steps required for the single bus structure for the following instruction

Jump L1

Which unconditionally transfers the controls to location L1 for further execution of instructions.

Solution

Step	Action	
1.	**PC_{out}, MAR_{in}, Read, Select 4, ADD, Z_{in}**	Fetch the instruction
2.	**Z_{out}, PC_{in}, Y_{in}, WMFC**	
3.	**MDR_{out}, IR_{in}**	
4.	Offset-field-of-IRout, Select Y,ADD, Zin	Execute the instruction
5.	Zout, PCin, End	

Example-6:

- Write the sequence of control steps required for the single bus structure for the following instruction

Branch <0 Label1

Which transfers the controls to location Label1 for further execution of instructions if the condition satisfied by Branch<0.

Solution

Step	Action	
1.	PC_{out}, MAR_{in}, Read, Select 4, ADD, Z_{in}	Fetch the instruction
2.	Z_{out}, PC_{in}, Y_{in}, WMFC	
3.	MDR_{out}, IR_{in}	
4.	Offset-field-of-IRout, Select Y,ADD, Zin if N=0, then End	Execute the instruction
5.	Zout, PCin, End	