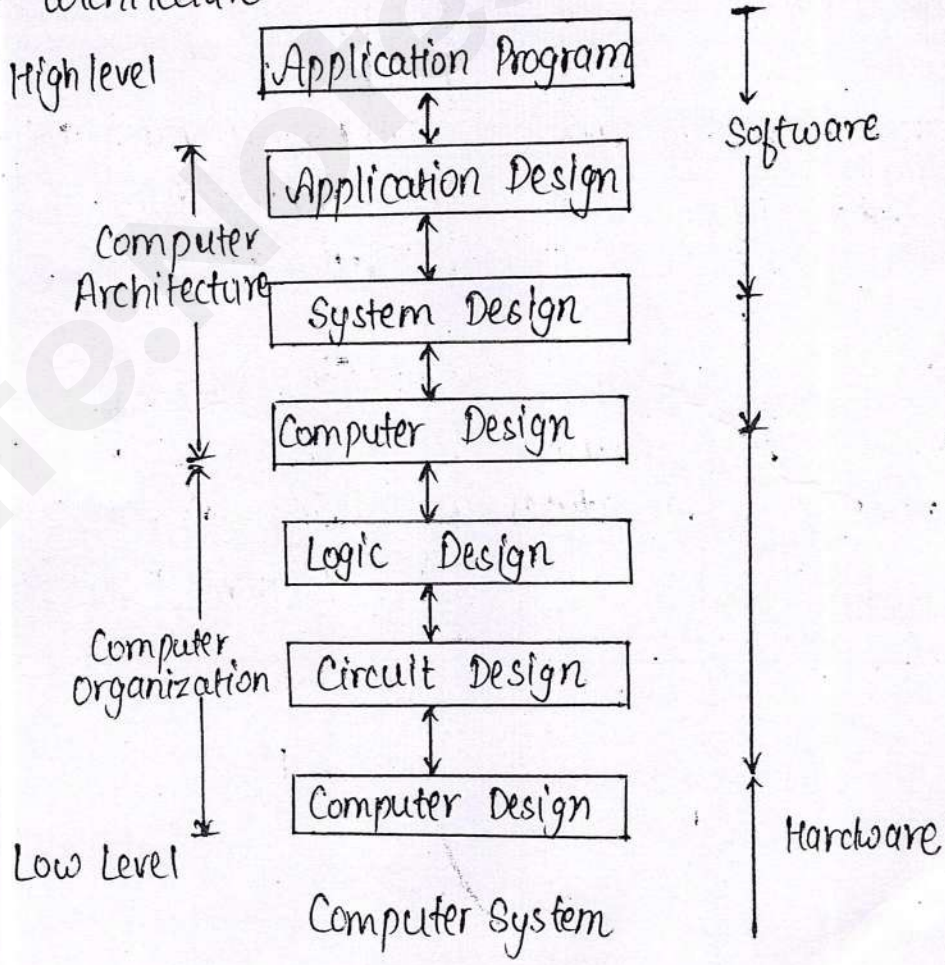
# UNIT:- 01

## Introduction

**Computer Architecture:-** Computer Architecture refers to the design of internal workings of a computer system including the CPU, memory and the other hardware. It involves decisions about the organisation of the hardware such as instructions set architecture, the data path design and the control unit Design.

**Computer Organisation:-** Computer organization refers to the operational units and their interconnections that implement the architecture specification. It deals with how the components of a computer system are arranged and how they interact to perform the required operation. It concerns with the physical implementation of computer architecture.



# Difference b/w Computer Organization and Architecture

| Computer Architecture | | Computer Organisation |
|---|---|---|
| Computer architecture describes what the computer does. | 1. | The organisation describes how it does it. |
| Computer Architecture deals with the functional behaviour of a computer system. | 2. | It deals with structural relationship. |
| Computer Architecture deals with higher level design issues. | 3. | It deals with low level design issues. |
| As a programmer you can view architecture as a series of instruc-tions, addressing modes and registers. | 4. | The implementation of the architecture is called organization. |
| For designing a computer, its architecture fixed first. | 5. | For designing a computer and organization is decided after its architecture. |
| Computer architecture contains logical functions such as instruction set, registers, data types and addressing modes. | 6. | The computer organization consists of physical units like circuit design, peri-pherals and adders. |

# Digital Computer Architecture

## Functional Units of Digital System

The digital computer consists of five functionally units:- Input, Memory, Arithmetic and Logic, output and control unit.

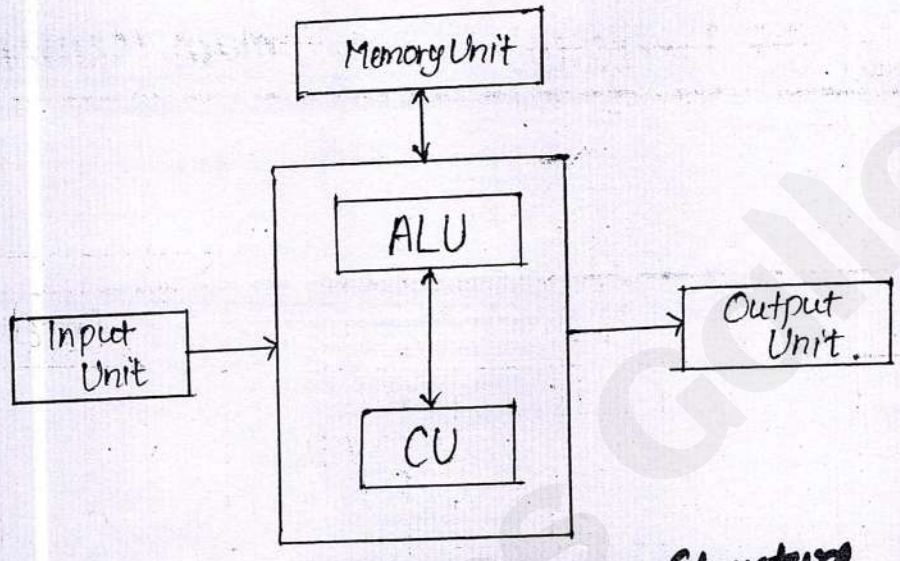



fig. Von- Neumann Structure

**Input Unit:-** The input unit accepts the digital information from user with the help of input devices such as keyboard, mouse, microphones etc.

**Memory Unit:-** The memory unit is used to store programs and take the data. Usually two types of memory devices are used in computer system.

1. Primary storage Memory device
2. Secondary storage Memory device

**ALU:-** Arithmetic and Logic Unit is responsible for performing arithmetical operation such as Add, subtract, Multiplication and Division, logical operations like AND, OR, NOT.

**Control Unit:-** The control unit co-ordinates the control signals or timing signals to determine when a given action is to take place.

**Output Unit:-** Output Unit sends the processed result to the user using output devices such as monitor, printer etc

## Connections blw the Processor and the main Memory

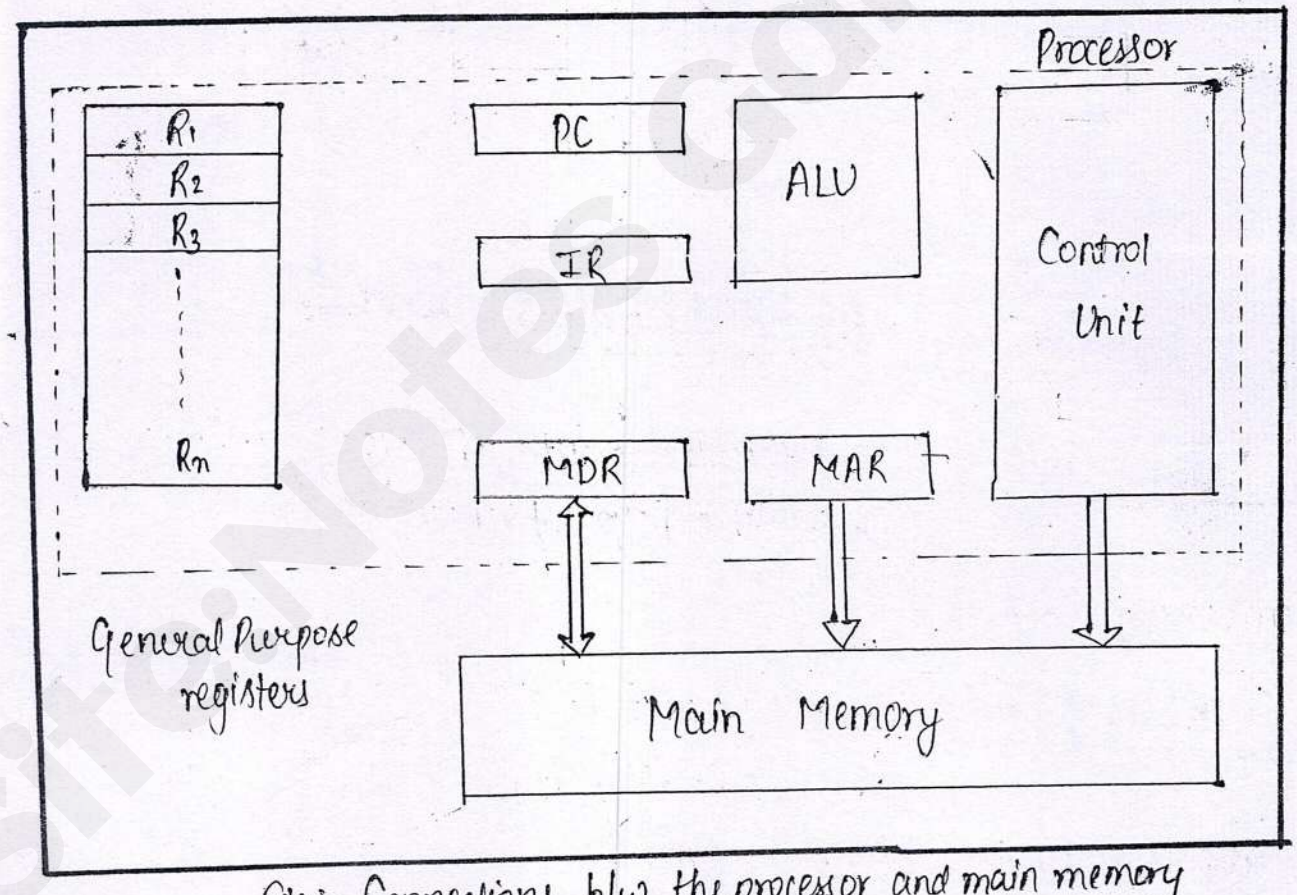



Fig:- Connections blw the processor and main memory

To perform execution of instruction, the processor contains a no. of registers used for temporary storage of data and some special function register as shown in figure.

The special function register include program counter, instruction register, memory address register, and memory data register.

1. **Program Counter:-** A program is a series of instructions stored in the memory. These instructions tell the CPU exactly how to get the desired result.

* The sequence of instructions execution is monitored by the program counter (PC).
* It keeps track of which instruction is being executed and what the next instruction will be.

2. **Instruction Register (IR):-** It is used to hold the instructions that is currently being executed.

3. **Memory Address Register & Memory Data Register:-**

* These registers are used to handle the data transfer between the main memory and the processor.
* The MAR holds the address of the main memory to or from which data is to be transferred.
* The MDR is also known as MBR (Memory Buffer Register) contains the data to be written into or read from the addressed word of the main memory.

4. **General Purpose Register:-** These are used to hold the operands for arithmetical and logical operation and/or used to store the result of the operation.

**Buses:-** A group of wires, called bus is used to provide necessary signals for communication between modules.

* Bus is a shared transmission medium.-
  1. Must only be used by one device at a time.
  2. When used to connect major components of computer system [CPU, memory, input output] is called a system bus.

* The system bus is separated into three functional groups:-

1. Data Bus
2. Address Bus
3. Control Bus

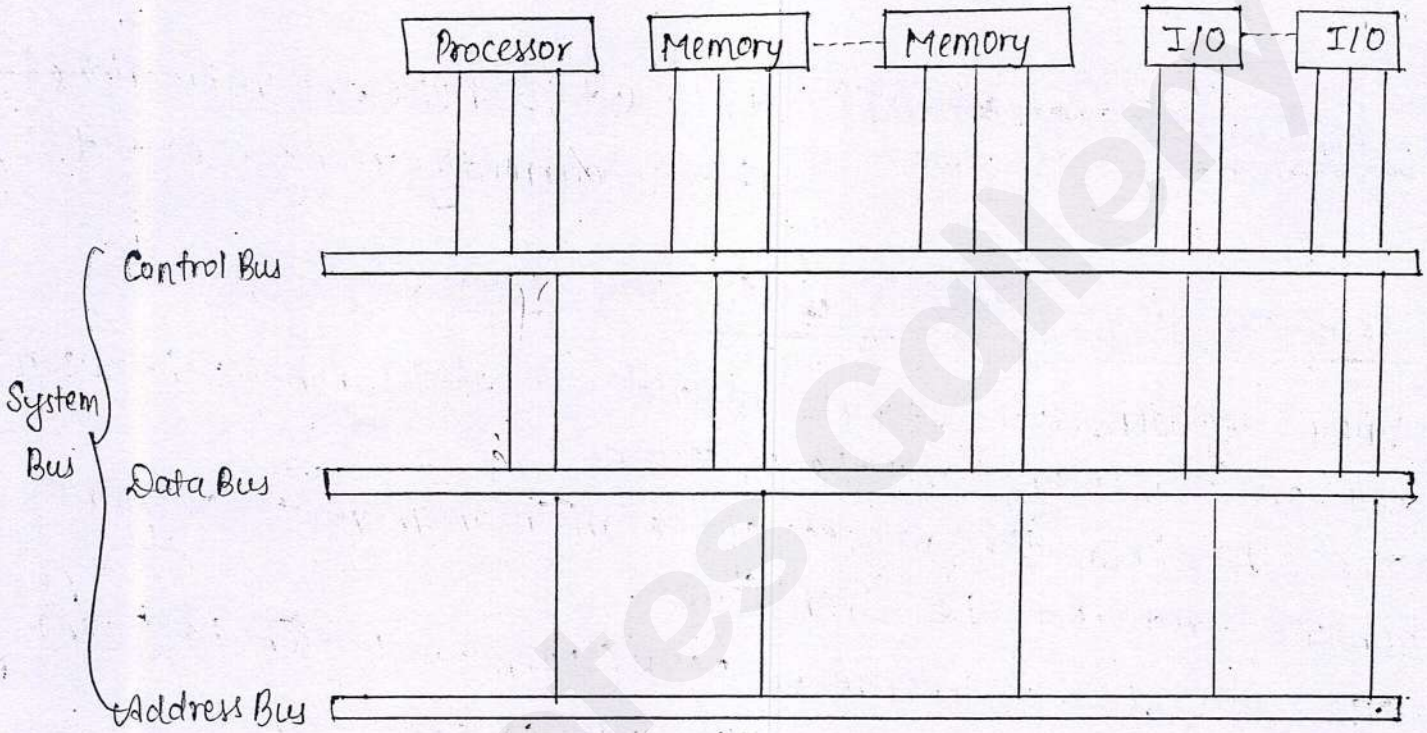



Fig:- Bus Interconnection System

**Data Buses:-** Data bus lines are bidirectional. CPU can read data on these lines from memory or from port as well as send data out on these lines to a memory location or to a port.

**Address Buses:-** It is a unidirectional Bus. width determines the maximum possible memory capacity of the system.

**Control Buses:-** Control access to ~~an~~ use of the data and address lines.

Control lines include:-

* Memory read and memory write
* Input/Output read and input/output write
* Transfer acknowledgement.
* Bus Request
* Bus Grant

## Types of Bus Structure

1. **Single Bus Structure :-** In single bus structure, address bus, data bus and control bus are shown by a single bus called system bus.

In single bus structure, all units are connected to common bus system called system bus. However, with the single bus only two units can communicate with each other at a time.

The bus control lines are used to multiple request for use of the bus.

The main advantage of single bus structure is its low cost and its flexibility for attaching peripheral devices.

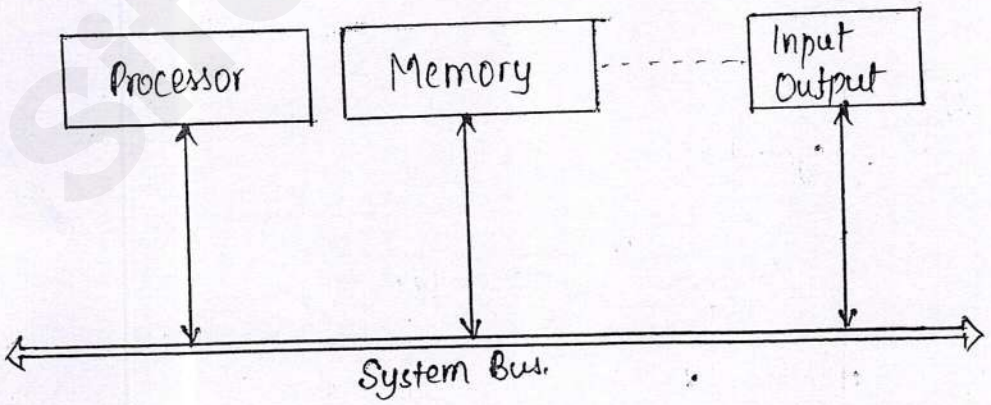



Fig. Single Bus Structure

**Multiple Bus Structure:** Large number of devices on a single bus will cause performance suffer due to propogation delay and the bus may become a bottle neck.

Nowadays, the data transfer rates for video controllers and network interfaces are growing rapidly. The need of high speed shared bus is impractical to satisfy with a single bus. Thus, most computer system uses multiple buses. These buses have hierarical structure as shown in figure:-

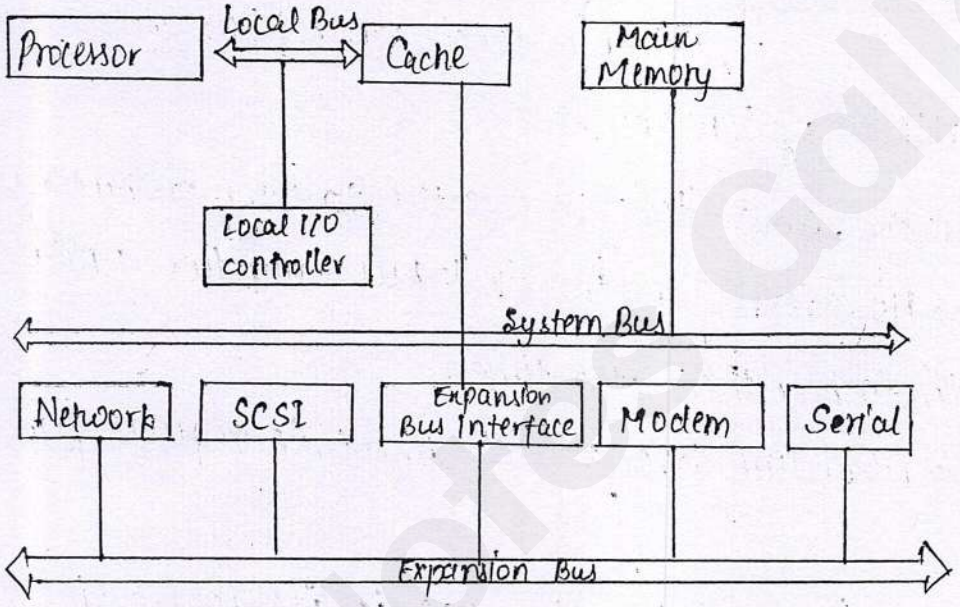



**Fig. 1 Traditional Bus Configuration**

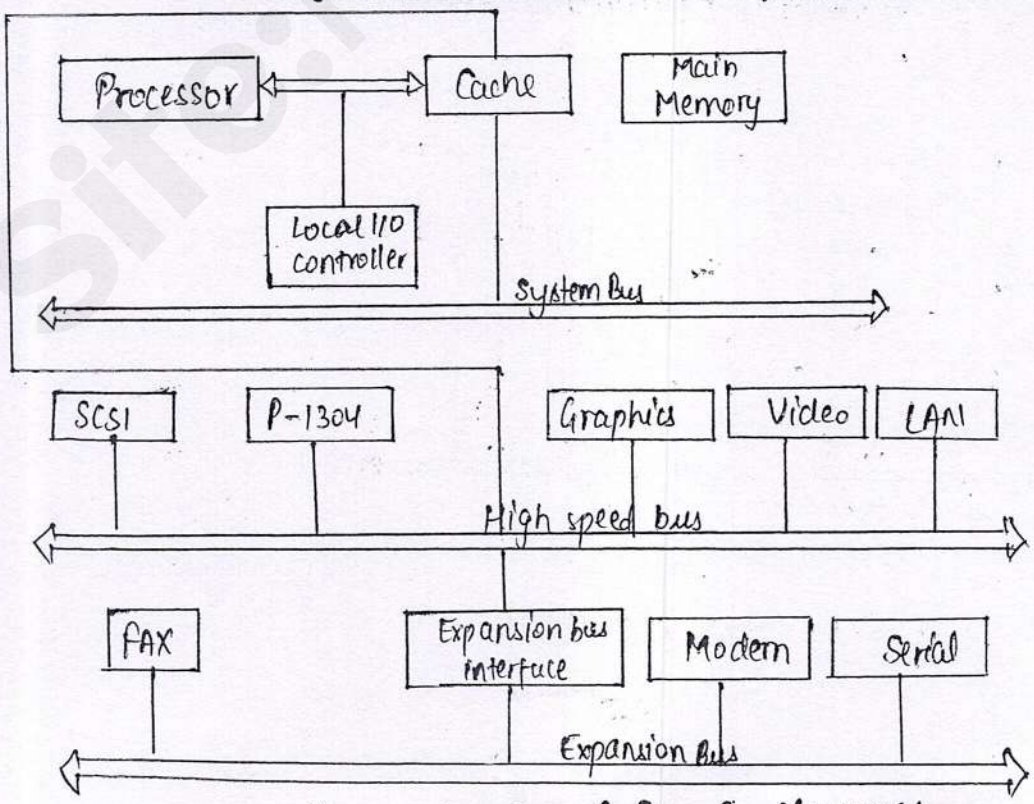
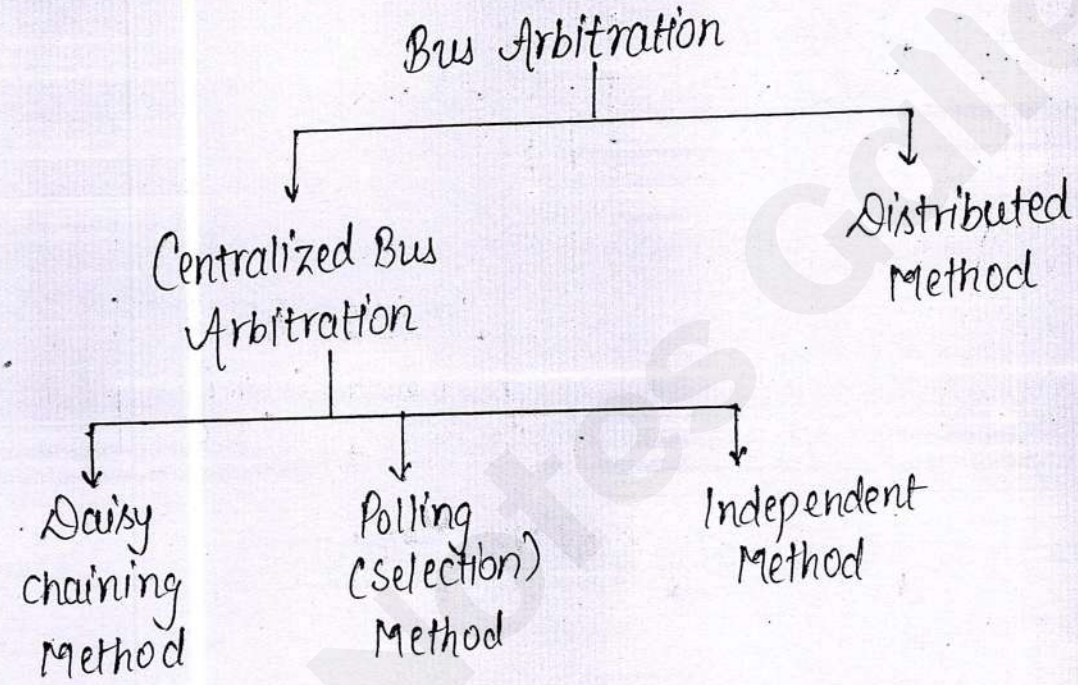



**Fig. 2 High Speed Bus Configuration**

**Bus Arbitration:** The device that is allowed to initiate data transfer on the bus at any given time is called the bus master.

Bus arbitration is the process by which the next device to become the bus master is selected and bus mastership is transferred to it. The selection of bus master is usually done on the priority bases.

## Approaches to Bus Arbitration




**Centralized Bus Arbitration:** In centralized bus arbitration, a single bus arbitor performs the required arbitration. The bus arbitor may be the processor or a separate controller connected to the bus.

Centralized bus approach is implemented by three methods:

**Daisy Chaining Method:** It is a simple and cheaper method. All masters make use of the same line for bus request.

In response to a bus request, the controller sends a bus grant if the bus is free. If the

* The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus.

* This master blocks the propogation of the bus grant signal, activate the busy line and gains the control of the bus. Therefore, any other requesting module will not receive the grant signal.

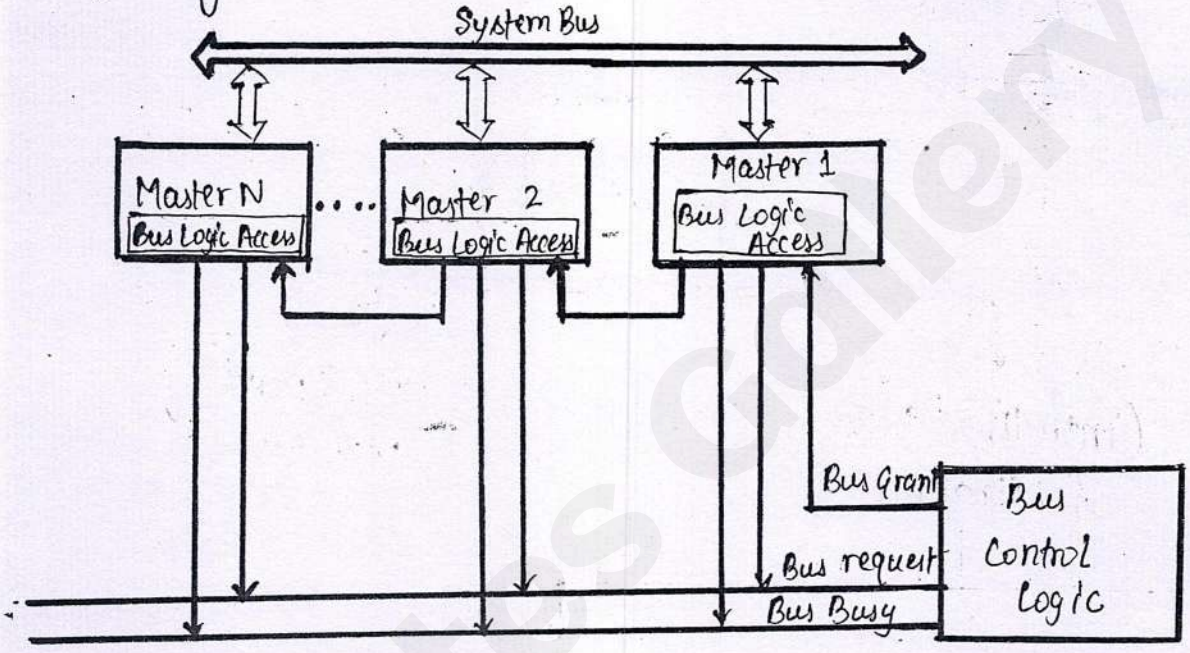



Fig. Daisy Chaining Method

**Advantages:** * It is a simple and cheaper method.
* It requires the least no. of lines and this no. is independent of the no. of masters in the system.

**Disadvantages:** * Failure of anyone master causes the whole system to fail.
* The priority of master is fixed by its physical location.
* The propagation delay of bus grant signal is proportional to the no. of masters in the system. This makes arbitration time slow. and hence limits the no. of masters in the system.

**Polling Method:** *In this, the controller is used to generate the addresses for the masters.

* No. of address lines required depends on the no. of masters connected in the system. For eg: there are eight (8) masters connected in the system, at least 3 address lines are required.

* In response to a bus request, controller generates a sequence no. of master's address. When a requesting master recognizes its address, it activates the busy line and begins to use the bus.

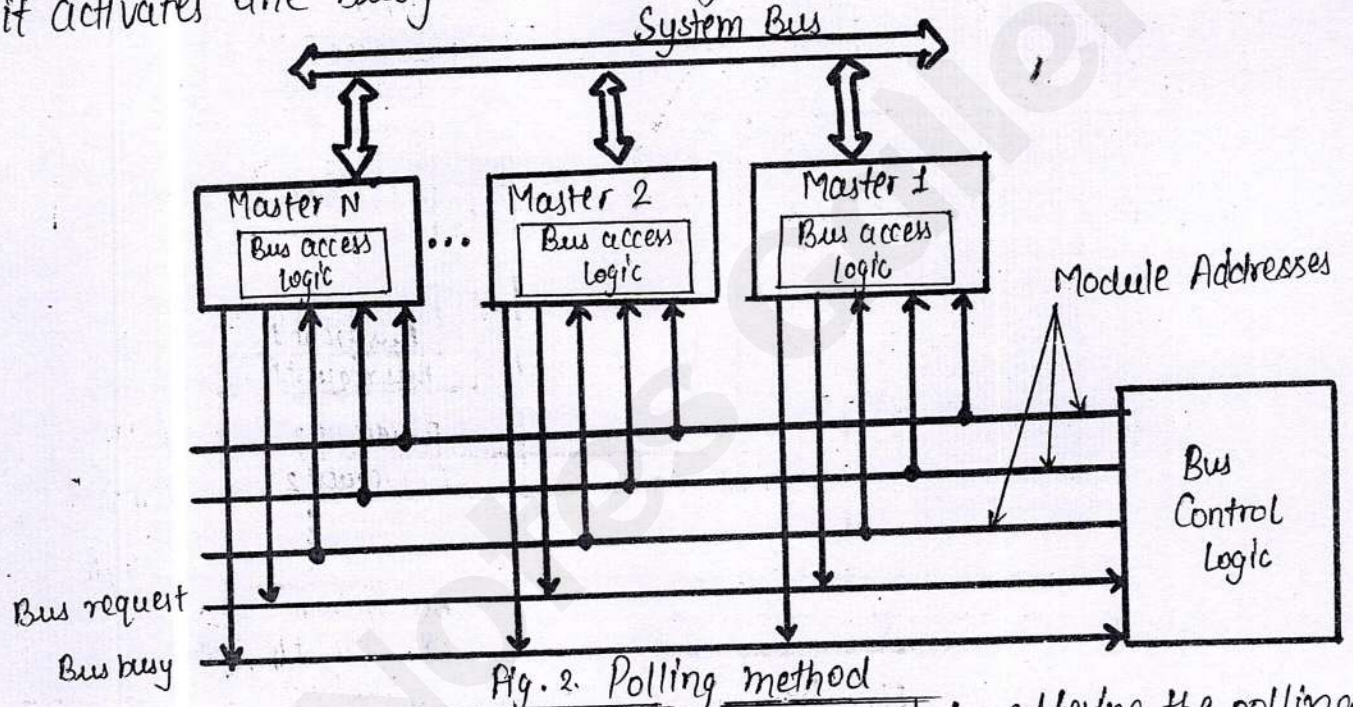



Fig. 2. Polling method

**Advantages:** * The priority can be changed by altering the polling sequence stored in the controller.

* If the one module fails, entire system does not fail.

**Disadvantages:** * No. of address lines are variable. This may affect cost. This may require more time for configuration.

3. **Independent Method:-** * In this scheme, each master has a separate pair of bus request and bus grant lines and each pair has a priority assigned to it.

* The controller selects higher priority request first and activates signal for it

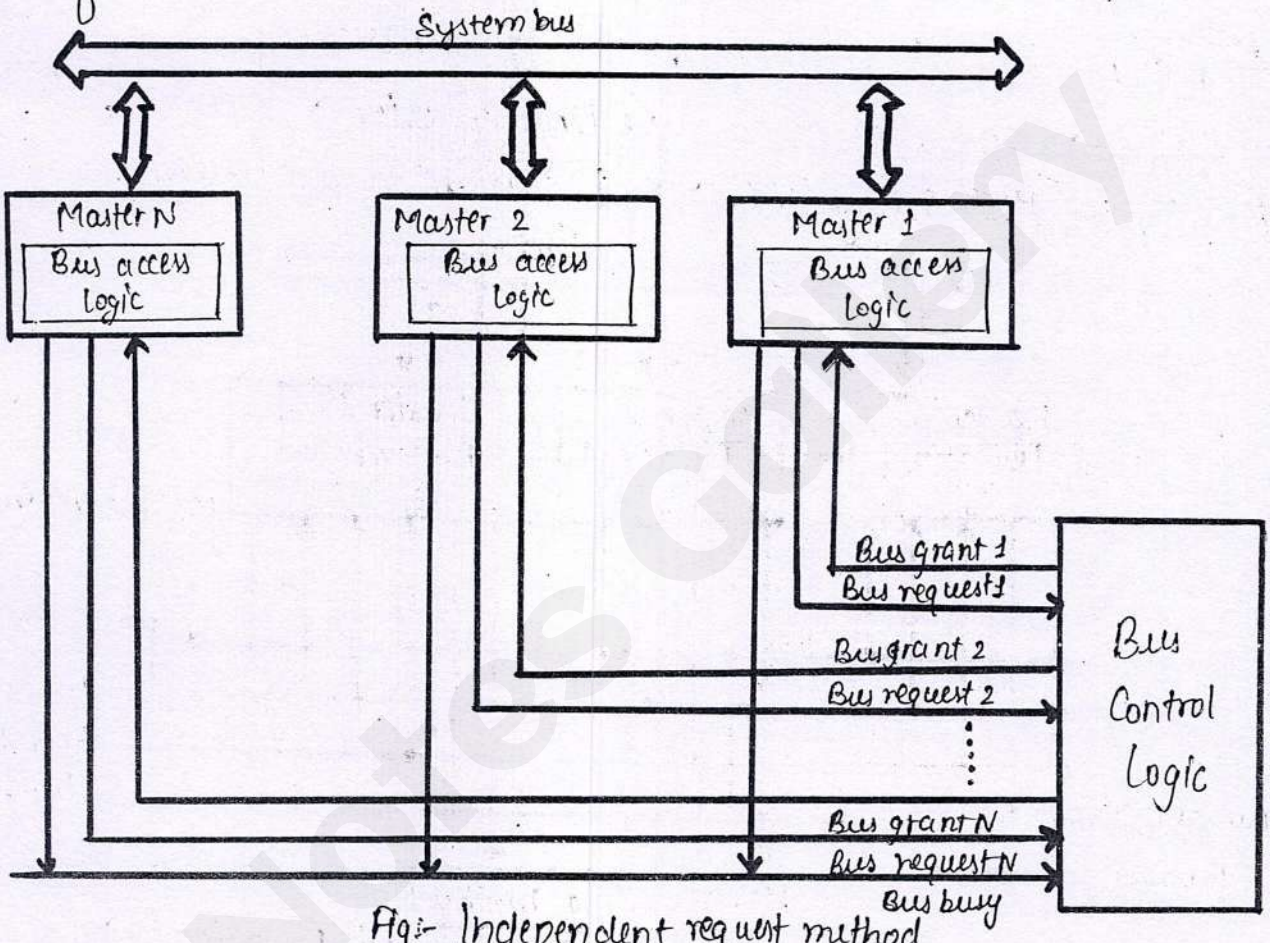



Fig:- Independent request method

**Advantages:-** Due to separate pairs of the bus request and bus grant signals, arbitration is fast and independent of the no. of masters in the system.

**Disadvantages:-** It requires more bus request and grant signals (=2×N lines for N modules).

# Distributed Bus Arbitration

In distributed bus arbitration, all devices participate in the selection of the next busmaster. In this scheme each device on the bus is assigned a 4-bit identification number. The no. of bits used for identification depends on the no. of devices connected on the bus. When one or more devices request for the control of bus, they initiate ~~when or~~ (start) arbitration signal and place their 4-bit identification number. In this scheme, the device having highest no. has highest priority.

# Register Transfer Language

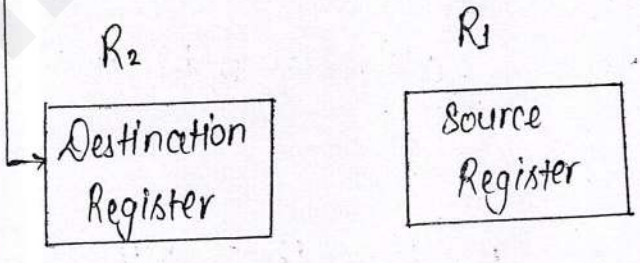
The microoperation is an elementary operation performed in a computer system. When the data transfer of any microoperation occur in between registers is known as register transfer.

The symbolic notations is used to describe or represent any micro-operation belongs to register transfer forms a statement or syntax is known as register Transfer Language.

**Example:**

(i) $R_2 \leftarrow R_1$

(ii) $R_1 \leftarrow R_2 \oplus R_3$



## :- Features of RTL :-

1. The presentation of digital function in register Transfer Logic is very user friendly.
2. Usage of register is easier instead of flipflops and gates.
3. It describes the information flow and processing task among the data stored in the registers. in a concise and precise manner

## :- Basic Components of RTL :-

# The Register Transfer Logic Method uses four basic components to describe digital system. These are as follows:-

1. **Registers & their functions:** Registers are the electronic circuit where information can be stored. The register includes all its counter parts such as shift register, counters and many more.
2. **Information:** The information is stored in the registers. The information may be binary no.s, alphanumeric characteres, control information or any other binary coded information.
3. **Operations:** The operations perform on the information stored in the registers. The operation performed on the data are called microoperations. The operation may be arithmetic, or logical operations.
4. **Control Functions:** The control functions that activate operations and control the sequence of operation. The control function is basically, a binary variable, when it is logic one, then it initiates the operation otherwise it deactivates the operatio

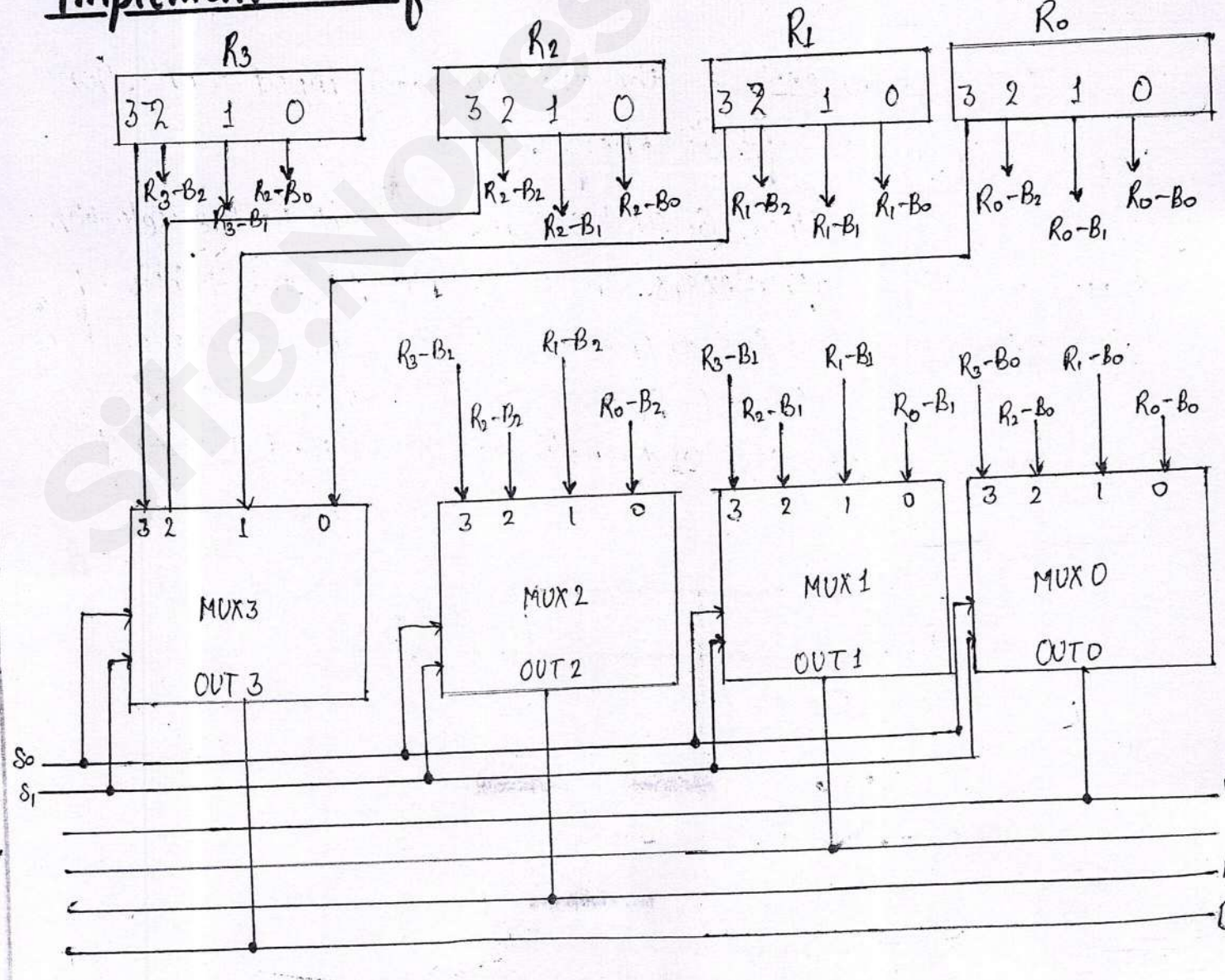
**Bus Transfer** :- A digital computer has many registers and it is necessary to provide data path between them to transfer information from one register to another. If separate lines are used between each register, there will be excess no. of wires and controlling of those wires make circuit complex.

A common bus consist of a set of common lines one for each bit of a register through which binary information is transferred one at a time.

The common bus scheme implemented into two ways -

1. Using multiplexer
2. Using Tri-State Buffer

## Implementation of Common Bus Using Multiplexer



The figure shows the implementation of common bus system for 4 registers using multiplexers. Each register has four bits numbered 0 through 3 and they are routed through multiplexer to the common bus. Here, four multiplexers are used to select four bits of the source register. Each multiplexer has four input lines, two select lines, one output line. The four input lines of multiplexer zero are connected to the bit zero, outputs of four registers such that bit zero of register is connected to input zero, bit zero of register one is connected to input one, bit zero of register two is connected to input two and bit zero of register three is connected to input three. Similarly inputs for MUX 1 are connected to bit 1 outputs, inputs for MUX 2 are connected to bit 2 and inputs for MUX 3 are connected to bit 3, outputs of register 0 through 3.

To avoid the complexity of the diagram only input connections for MUX 3 are physically shown.

The two selection lines $S_1$ and $S_0$ are connected to the selection inputs of all 4 Multiplexers. These lines choose the 4-bits of one register and transfer them into the 4-line, common bus through outlines. When $S_1 S_0 = 00$ the input zero of all four multiplexers are selected and applied to the output to transfer them on the common bus.

| $S_1$ | $S_0$ | Selected Register |
|---|---|---|
| 0 | 0 | Register 0 |
| 0 | 1 | Register 1 |
| 1 | 0 | Register 2 |
| 1 | 1 | Register 3 |

Table 7.2 Selection Table

# Memory Transfer

A memory is a collection of storage cell. Each cell stores one bit of information, the memory stores binary information in groups of bits called word. To access information from a particular word from the main memory to access, each word has different address.

The transfer of information from a memory word to the outside environment is called a read operation. The transfer of new information to be stored into the memory is called a write operation.

The data transfer between memory and processor takes place through the use of two processor registers usually called MAR (Memory Address Register) and Memory Data Register (MDR).

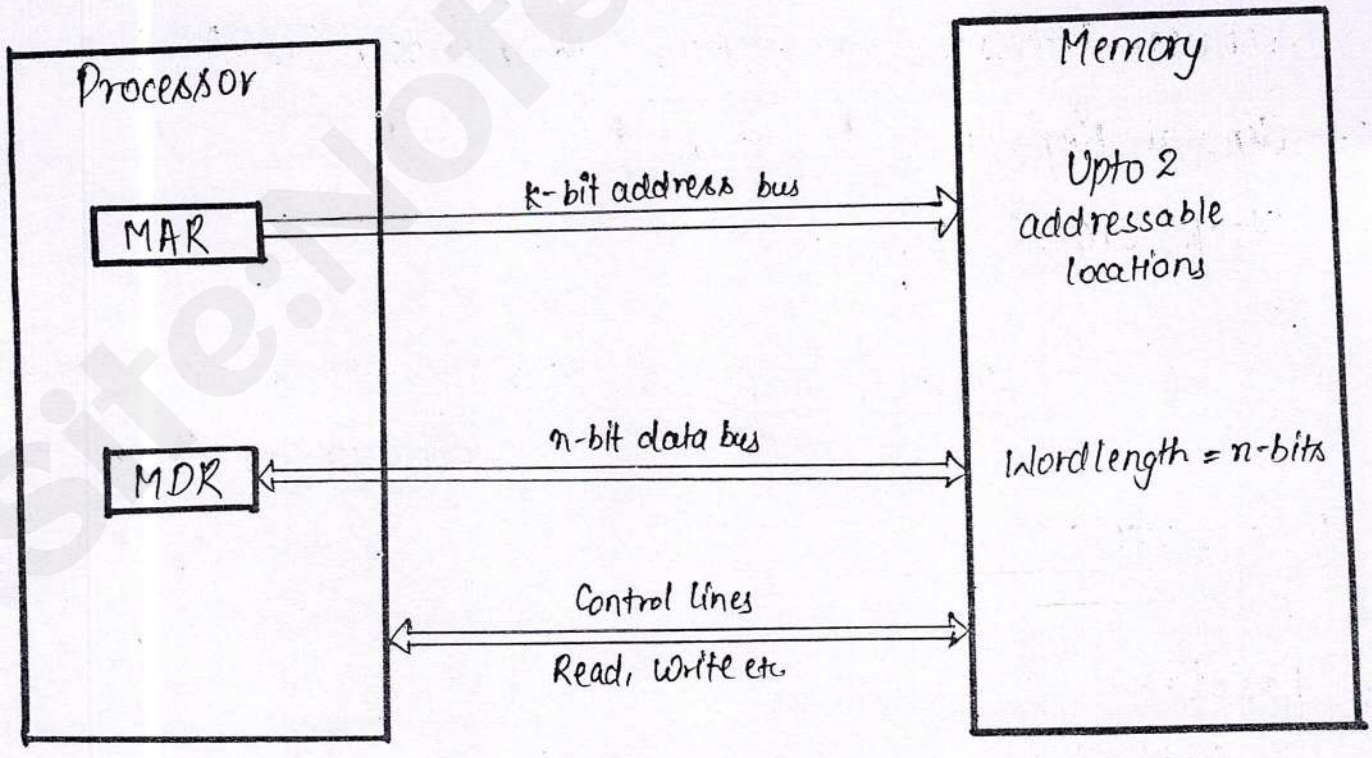



Fig. 1.7.16 Connection between memory and processor

In read operation, the address of the Memory word is specified by address register, AR and the data word read from the memory is loaded into data register, DR, ~~discrete op~~ This read operation is represented as:-

| Read : DR ⟵ M[AR] |
|---|

→ memory से data fetch किया जाता है।

The right operation transfers the contents of a data register to a memory word selected by the address in the address register. The right operation is represented as:

| Write : M[AR] ⟵ DR |
|---|

→ Data pass किया जा रहा है जो memory में store हो जाता है।

# Processor Organization

There are three types of processor organization:-

1. General Processor Organization (Accumulator based)
2. General Register Organization
3. Stack Organization

1. **General Processor Organization:** The general processor organization includes three major logic devices:-
   1. ALU
   2. Registers
   3. Control Unit

   The internal data bus is used to transmit data between these logic devices.

1. **ALU:** It is major logic device contains the processors data processing logic.

The internal data bus of the processor is connected to the two input of ALU through the temporary register and the accumulator. The output is connected to the internal data buses.

The ALU works on either one or two data words depending on operation. It performs logical and arithmetical operation.

The output is stored in accumulator (A).

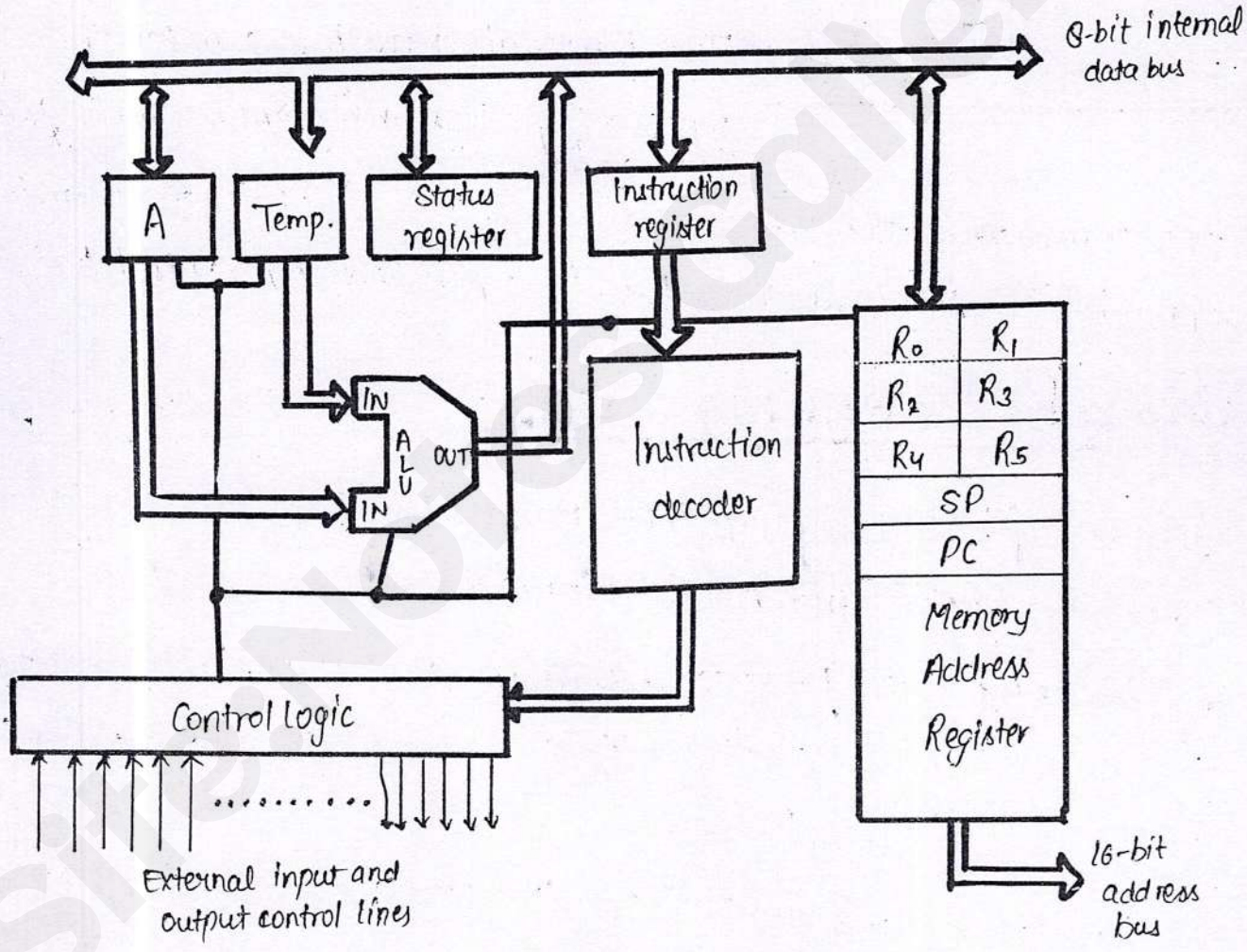



Fig. General Processor organization

2. **Registers:** The basic registers found in the most of the processors are the accumulator, program counter, stack pointer, the status register, the general purpose registors, the memory address register, the instruction register and the temporary data register.

3 **Control Logic:** The control logic is responsible for working of all other parts of the processor together. It maintains the synchronization in operation of different parts in the processor.
The control logic receives the signal from instruction decoder which decodes the instructions stored in the instruction register
The control logic generates the control signal necessary to carry out this instruction.

-;- **Internal Data Bus:** The internal data bus connects the different parts of processor together and it enables the communication between these parts. The data transfer through the internal data bus is controlled by control logic.

## 2. General Register Organization

It shows that how registers are selected and how data flow between register and ALU takes place.
A decoder is used to select a particular register. The output of each register is connected to two multiplexers ~~two~~ to from the two buses A and B.
The selection line in each multiplexer select the input data for a particular bus.
The A and B buses form the two inputs of an ALU.
The operation select line decides the micro-operation to be performed.
The result of the microoperation is available at the output bus.
The output bus is connected to the inputs of all registers, thus

the selecting a destination register, it is possible to store the result in it.

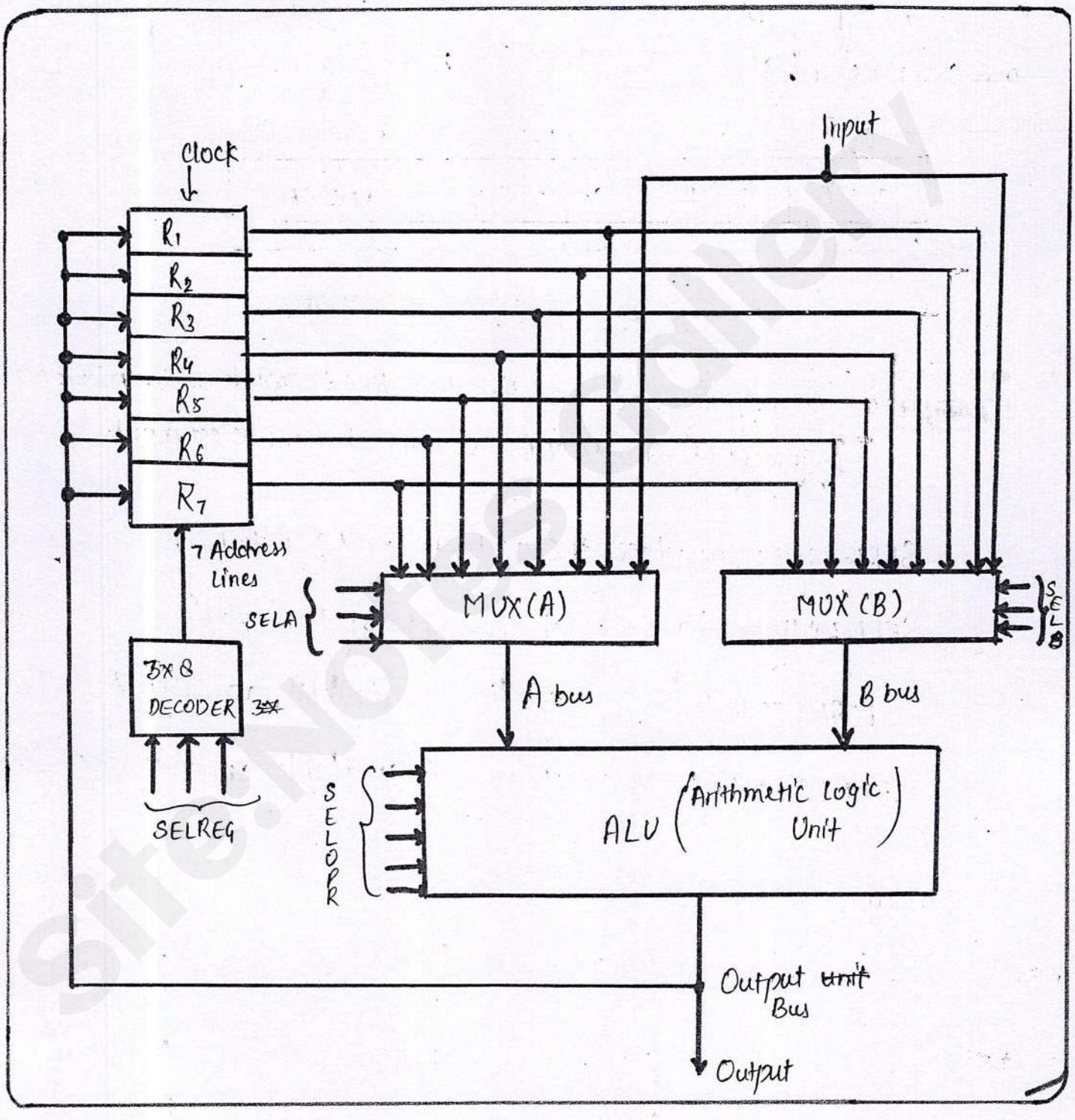
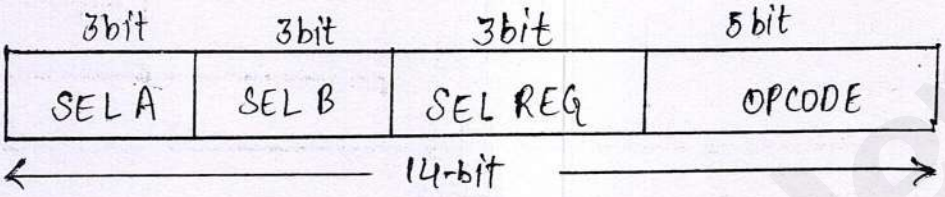



Fig. 2.2.1 General bus organization for CPU registers

**Control Bus: Word:-** The binary conversion of any micro-operation is known as control word. The combined value of a binary selection is specified by control word.

Collection of 14-binary bits which specify any micro operation known as control word. It is represented as :-




General format for Control word

**Example:-** To perform the operation $R_3 \leftarrow R_1 + R_2$ we have to provide following binary selection variables to the select inputs

SELA : 001 - To place the contents of R1 into bus A

SELB : 010 - To place the contents of $R_2$ into bus B.

SELOPR : 10010 - To perform the arithmetic addition A+B

SELREG : 011 - To place the result available on output bus in $R_3$

| Binary Code | SELA | SELB | SELREG |
|---|---|---|---|
| 0 0 0 | Input | Input | — |
| 0 0 1 | $R_1$ | $R_1$ | $R_1$ |
| 0 1 0 | $R_2$ | $R_2$ | $R_2$ |
| 0 1 1 | $R_3$ | $R_3$ | $R_3$ |
| 1 0 0 | $R_4$ | $R_4$ | $R_4$ |
| 1 0 1 | $R_5$ | $R_5$ | $R_5$ |
| 1 1 0 | $R_6$ | $R_6$ | $R_6$ |
| 1 1 1 | $R_7$ | $R_7$ | $R_7$ |

| Operation selection code | Operation |
|---|---|
| 00000 | Transfer A |
| 00001 | Increment A |
| 00010 | A+B |
| 00101 | A-B |
| 00110 | Decrement A |
| 01000 | A AND B |
| 01010 | A OR B |
| 01100 | A XOR B |
| 01110 | Complement A |
| 10000 | shift right A |
| 11000 | Shift left A |

## 3. Stack Organization

Stack is a list of data elements usually words or bytes with the accessing restrictions that the elements can be added or removed at one end of the list only. This end is called the top of the stack and another end is known as bottom of the stack. Stack structure is also known as Last In First Out (LIFO) list. The term PUSH and POP are used to describe placing a new data element on the stack and removing the top of the data element from the stack respectively.

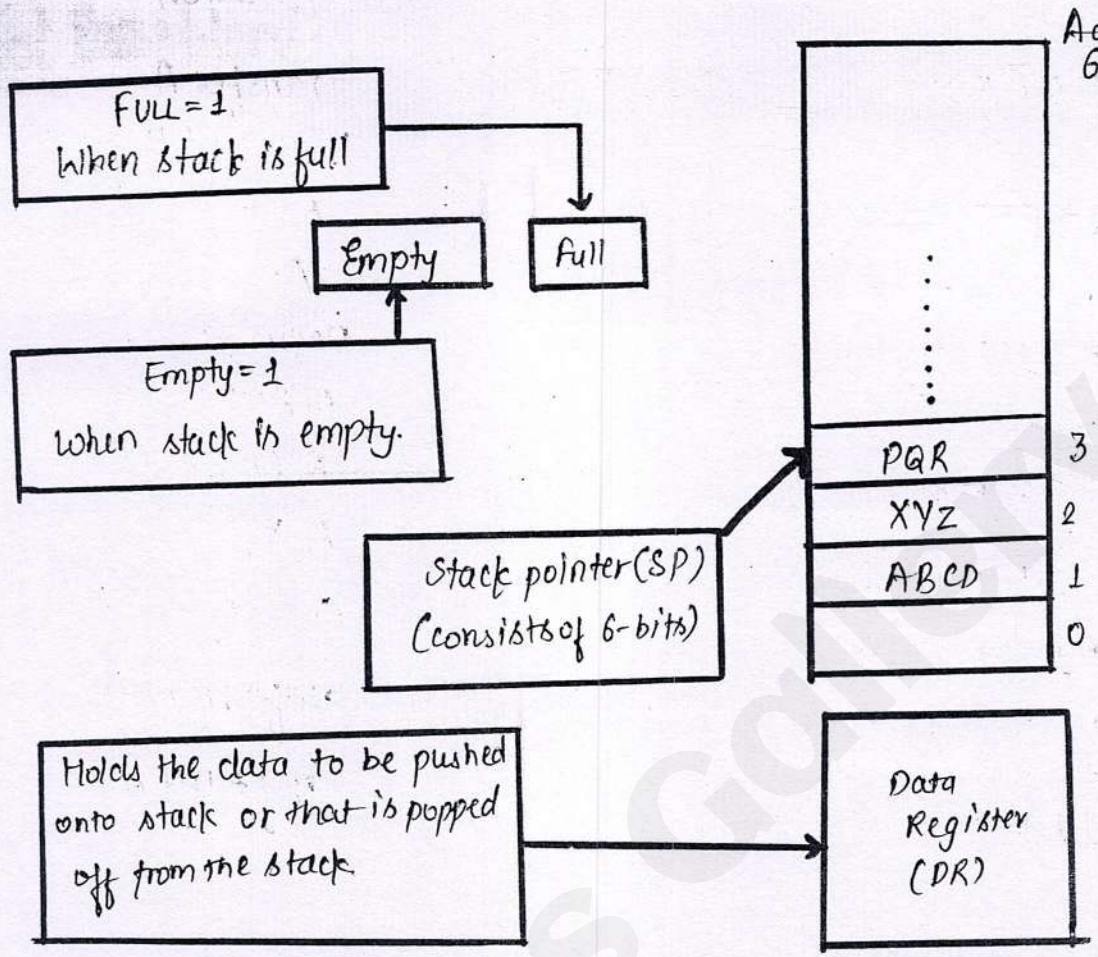



Fig:- Block-diagram of 64-word stack

## Working of POP and PUSH:

POP (Performed if stack is not empty i.e. if EMPTY=0)

DR ← M[SP] — Read item from the top of stack.

SP ← SP-1 — Decrement stack pointer.

If (SP=0) then (EMPTY ← 1) — Check if stack is empty.

FULL ← 0 — Mark the stack not full.

PUSH (Performed if stack is not full i.e. if FULL=0)

MP ← SP+1 — Increment stack pointer.

M[SP] ← DR — write item on the top of stack.

If (SP=0) then (FULL ← 1) — Check if stack is full.

EMPTY ← 0 — Mark the stack not empty.

## Register Stack

A stack can be placed in a portion of a memory unit or it can be organized as a collection of finite no. of CPU registers. In a figure, 32 bit register stack is represented. The stack pointer holds the address of the register that is currently the top of the stack..

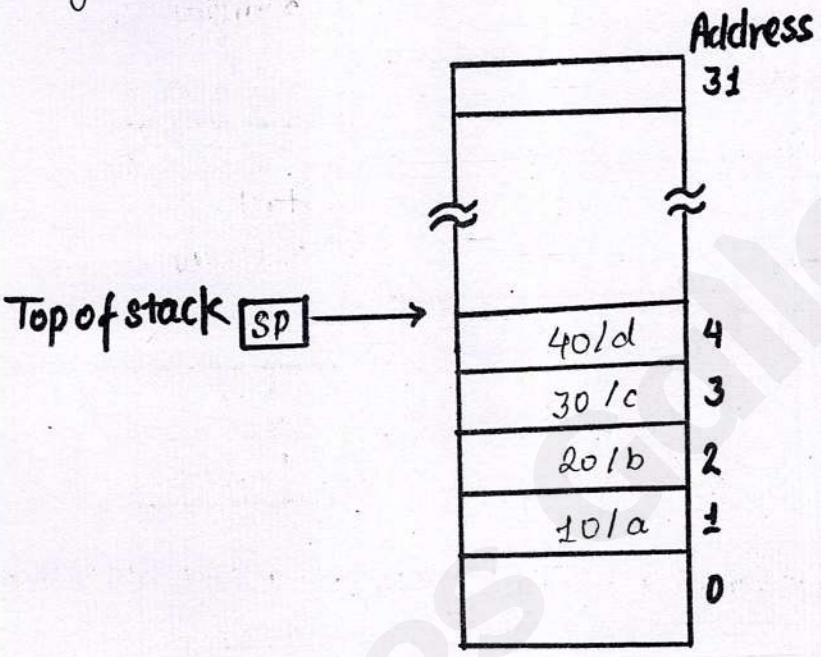
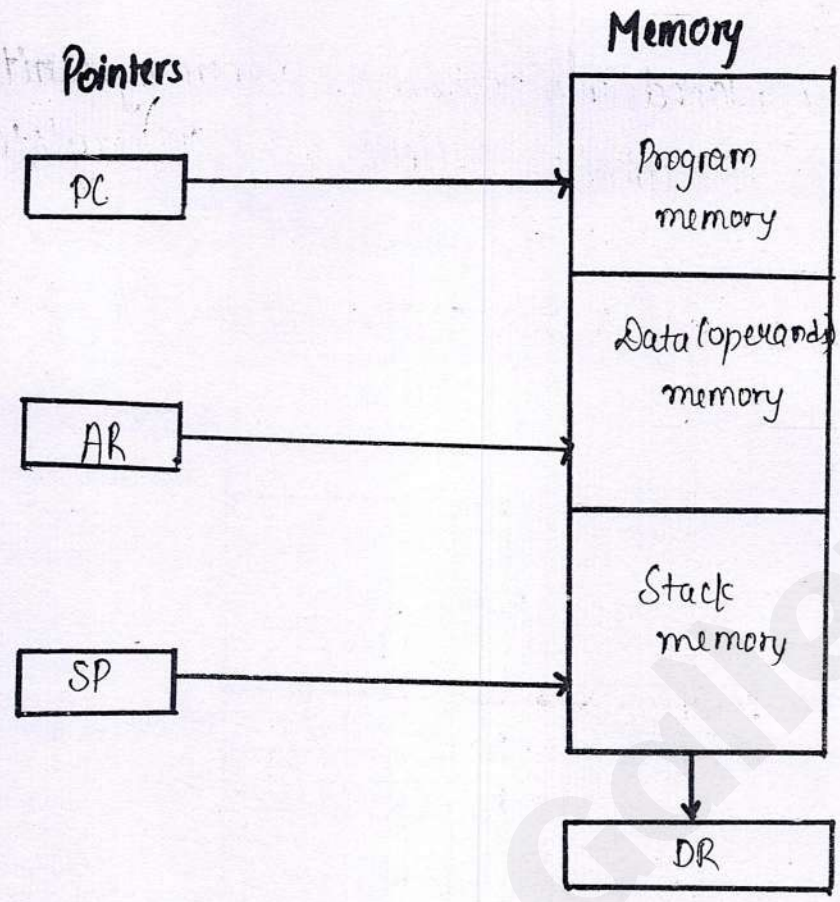



Fig 2.31: Block Diagram of a 32-word register stack

## Memory Stack

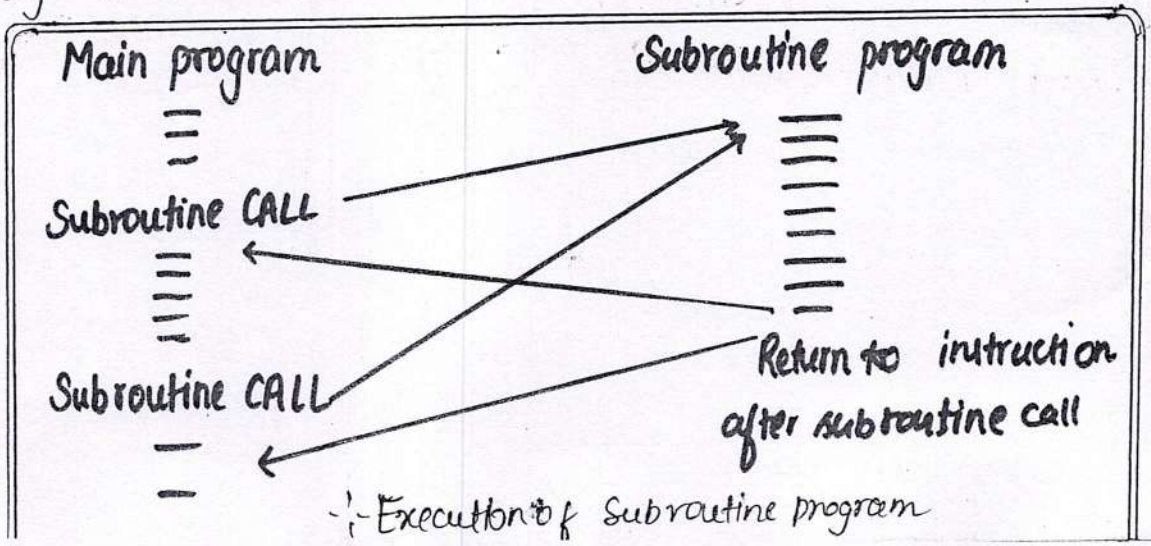
The operation of memory stack is exactly similar to the register stack. However, it is implemented using computer memory instead of CPU register array.

The memory stack has an advantage of large size. but the operation on it is slower than that of register stack. This is because register stack is internal to the CPU and does not need any memory access.




Sharing of computer memory by program data and stack data.

**Sub-routine:-** In some situations, it is better to execute part of a program that is not in sequence with the main program. For eg. there may be a part of a program that must be repeated many times during the execution for the entire program. Rather than writting repeated program again and again the programmer can write that part only once. This part is written separately. The part of the program which is written separately is called subroutine



# Addressing Modes

The address of the main memory can be specified directly within the instruction.

The rules for interpreting the address fields in the instruction is known as addressing modes.

There are following types of addressing modes:-

| S.No | Addressing mode | Example |
|---|---|---|
| 1 | Register mode | MOV $R_1, R_2$ |
| 2. | Absolute mode | MOV $A_1$ 2000 |
| 3. | Immediate mode | MOV A, #20 |
| 4. | Register ~~mode~~ indirect | MOV A, $(R_0)$ |
| 5. | Auto-Increment | MOV $(R_2)$, $+R_0$ |
| 6. | Index mode | MOV 20 $[R_1]$, $R_0$ |
| 7. | Relative mode | JNZ BACK |
| 8. | Auto-Decrement | MOV $(R_1)$, $-R_0$ |
| 9. | Implied mode | CMA |

**Register mode:-** The operand is the content of processor register. The name of register is specified in the instruction.

Eg MOV $R_1, R_2$ ⇒ This instruction copies the content of register $R_2$ to $R_1$.

**Absolute or Direct Mode:-** The address of the location of the operand is given explicitly. (in a clear and detailed manner) in the instruction.

Eg:- MOV A, 2000. ⇒ This instruction copies the contents of memory location 2000 into the register A.

**Immediate Mode:-** In this addressing mode, the operand is given explicitly in the instruction.

Eg:- MOV A, #20 ⇒ This instruction copies the operand 20 in the register A.

**Register Indirect Mode:-** The effective address of the operand is the content of a register on the main memory location whose address is given explicitly in the instruction.

Eg:- $EA = R$

Eg:- MOV A, $(R_0)$ ⇒ This instruction copies the contents of memory address by the contents of register $R_0$ into the register A.

**Index Mode:-** The effective address of the operand is generated by adding a constant value (specified in the instruction) called offset to the content of register.

$$EA = R + offset \rightarrow (X_R)$$

↳ address part given in instruction

Eg:- MOV 20, $[R_1]$, $R_0$ ⇒ This instruction loads the content of $R_0$ into the memory location whose address is calculated by the addition of the content of Register $R_1$ and constant value (offset or displacement) 20.

**Relative Mode:-** The effective address is determined by the index mode using program counter in place of the general purpose register.

$$EA = PC + \text{Address part of Instruction}$$

Eg:- $JN_Z$ BACK ⇒ This instruction causes program execution to go to the branch target location identified by the name BACK, if branch condition is satisfied.

**Auto Increment Mode:-** The effective address of the operand is the contents of the register specified in the instruction after accessing the operand the contents of register are ~~impl~~ incremented to address the next location.

Eg: MOV $(R_2)+, R_0$ ⇒ This instruction copies the content of register $R_0$ into the memory location whose address is specified by content of register $R_2$. After copy operation, the content of register $R_0$ are automatically incremented by 1.

**Auto Decrement Mode:-** The contents of register specified in the instruction are decremented and then they are used as an effective address to access the memory location.

Eg MOV $(R_1), -R_0$ ⇒ This instruction initially decrements the content of register $R_0$ and then decremented contents of register $R_0$ are used to address the memory location

**Implied Mode:-** In this addressing mode, opcode is specified in detailed and clear manner. This operation is performed in accumulator.

Eg:- **CMA** ⇒ The above instruction complements the content of accumulator.

**Example 2.4.2⇒** At memory address 200, two word instruction, load to AC is stored with a mode bit as a most significant bit. A location 201 the address stored is 500. At location 202 next instruction is stored. The following are stored at different memory locations as shown ahead.

| Memory location (Address) | Memory contents |
|---|---|
| 399 | 450 |
| 400 | 700 |
| 500 | 800 |
| 600 | 900 |
| 702 | 325 |
| 800 | 300 |

If the content of PC is 200, while the contents of register R1 is 400. XR register is 100. If all the numbers and addresses are in decimal no. find out the contents of AC and effective address for the following addressing modes.

i) Direct address ii) Indirect address iii) Relative address
iv) Indexed address v) Register Indirect addressing mode.

Soln:

| | | |
|---|---|---|
| 200 | LOAD TO AC | MODE Bit |
| 201 | 500 | |
| 202 | Next Add. Instruction | |
| | ⋮ | |
| 399 | 450 | |
| 400 | 700 | |
| 500 | 800 | |
| 600 | 900 | |
| 702 | 325 | |
| 800 | 300 | |

opcode

operands

Given: PC = 200
$R_1$ = 400
(Index) $X_R$ = 100
↓
offset

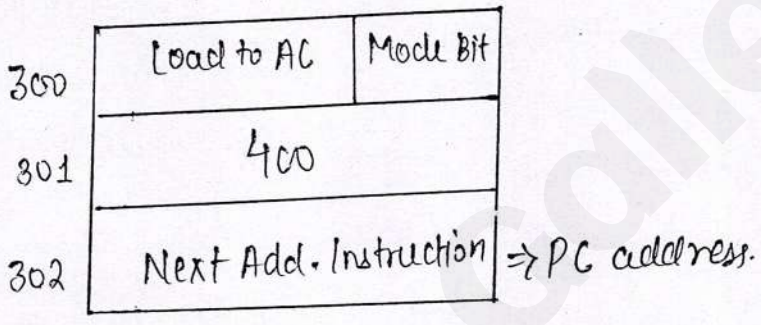
| | EA | AC |
|---|---|---|
| Direct Mode | 500 | 800 |
| Indirect Add. | [500] = 800 | 300 |
| Relative Add. Mode. | PC + Add Part Given to Instruction<br>Add<br>202 + 500 = 702 | 325 |
| Index | offset + Add. Part in Instruc<br>100 + 500 = 600 | 900 |
| Register Indirect | $R_1$ = 400 | 700 |
| Register Direct | — | 400 |

Ques. An instruction is stored at location 300 with its address field at location 301. The address field has the value 400. A processor register R1 contains the number 200. Evaluate the effective address if the addressing mode of the instruction is:

(i) Immediate (ii) Direct (iii) Register Indirect (iv) Relative

(v) Index with R1 as the index register. $\Rightarrow \{X_R = R_1\}$

Soln $R_1 = 200$




(i) EA = 301

(ii) EA = 400

(iii) EA = $R_1$ = 200

(iv) EA = PC Add. + Add. part given in instruction

EA = 302 + 400

EA = 702

(v) EA = $X_R$ + Add. part given in Ins. $\{X_R = R_1\}$

= 200 + 400

EA = 600