COMPUTER ARCHITECTURE & ORGANISATION [CAO]

Text Book :-

1. Computer Organization, Carl Hamacher, Zvonko Vranesic, Safwat Zaky, 5th Edition, Mc. Grawhill.

Syllabus :- UNIT-1 BASIC STRUCTURE OF COMPUTERS

→ Functional Unit
→ Basic Operational Concepts
→ Bus Structures
→ System Software
→ Performance
→ The history Of computer development
→ Machine Instructions & programs:
→ Instruction and Instruction sequencing
→ Register Transfer Notation
→ Assembly Language Notation
→ Basic Instruction Types.

Computer and computer Types :-

(a) Computer :- It is a fast Calculating machine that accepts digitized input information, processes it according to a list of internally stored instructions and produces the resulting Output information.

→ List of instructions is called a computer program.
→ Internal Storage is called computer memory

(b) Computer Types :- There are different types are available

(1) Desktop Computers (Personal computer)
(2) Notebook computers
(3) Workstations
(4) Enterprise Systems
(5) Servers
(6) Super Computers

(1) Desktop computers :- It has processing and storage units Visual display and audio output units, and a keyboard that can all be located easily on a home (or) office desk. The storage media includes - hard disks, CD-Rom's etc. The most common form of Desktop computers is personal computer which has found wide use in home, schools and business offices.

(2) Notebook computers :- These are compact Version of Personal computer with all of these components package into a single unit the size of a thin briefcase. It is portable.

(3) Workstations :- These have more computational power than Personal computers, used in Engineering applications, especially for interactive design work.

(4) Enterprise Systems (or) Mainframes :- These are used for business data procesing in Medium to large corporations that require much more computing power & storage capacity than workstations can provide

(5) Servers :- They are capable of handling large Volumes of requests to access the data, & it contain sizable database storage units.

(6) Super Computers :- These are used for large scale numerical calculations required in applications such as weather forecasting, aircraft design & simulations.

* Functional Units :-

The Below figure shows the basic functional units of a Computer :-

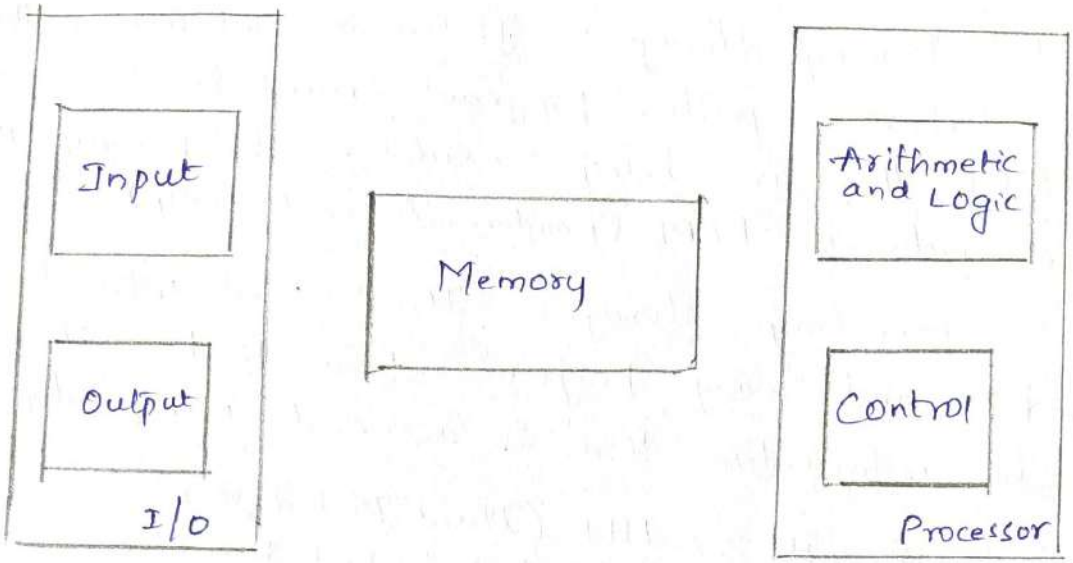


Fig (1) : Basic functional units of a computer.

A Computer Consists of five functionally independent main Parts are :-

(1) Input
(2) Memory
(3) Arthimetic & logic Unit
(4) Output
and (5) Control Unit.

(1) Input unit :-

Computers accept coded information through input units, which read the data. The most well known input device is the keyboard. The Other Input devices are :-

Ex :- → Mouse
→ Joystick
→ Scanner
→ Touch screen
→ light pen.. Etc.

(2) Memory Unit :- The function of Memory unit is to Store programs and data. There are two types of Storages they are : (i) Primary Storage
(ii) secondary Storage.

(i) Primary Storage :- It is a Fast memory that operates at Electronic Speeds. programs must be stored in the memory while they are being Executed. The primary memory of a Computer is RAM (Random Accen memory)

(ii) Secondary storage :- It is used when large amounts of data and many programs have to be stored, particularly for information that is accessed infrequently.

Example :- HDD (Hard Disk drive)
FDD (Floppy Disk drive)
Magnetic disks & tapes
Optical disks (CD-ROM) (compact disk Read only memory)

(3) Arithmetic and Logic Unit :-

The Computer operations are Executed in the Arithmetic and Logic unit (ALU) of the processor.

Ex:- Suppose two numbers located in memory are to be added
→ They are brought into the processor, and the actual addition is ~~Out~~ Carried out by the ALU
→ Sum may then be Stored in memory (or) retained in the procend for immediate use.

(4) Output Unit :-

It is the Counterpart of the input unit The function of output unit is to Send processed results to the outside world.

Examples of Output units are

- Printer
- Monitor
- Plotter ... Etc.,

(5) Control unit :-

The Control Unit is Effectively the nerve center that Sends Control Signals to other units and senses their States.

The task of Control Unit is to coordinate all units like memory, ALU, I/O Units properly by sending control Signals.

All activities inside the machine are directed and Controlled by control unit

Thus the Operation of a computer can be Summarized as follows:

(i) computer accepts information in form of programs & data through input unit and Store in the memory unit

(ii) Information Stored in memory is fetched, into an ALU (Arithmetic Logic Unit) where it is processed

(iii) processed information leaves the Computer through Output Unit

Above all activities inside the machine is directed by

* **Basic Operational Concepts :-**

The basic function of computer is to Execute program, sequence of instructions. These instructions are stored in the Computer memory. Through input unit all instructions are loaded into computer memory, after Processing the data the result is either stored back into the computer memory or sent to the outside could through the output port.

→ Transfers between the memory and the processor are Stored by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals.

The data are then transferred to (or) from the memory.

The below figure Shows Connections between processor and the memory.

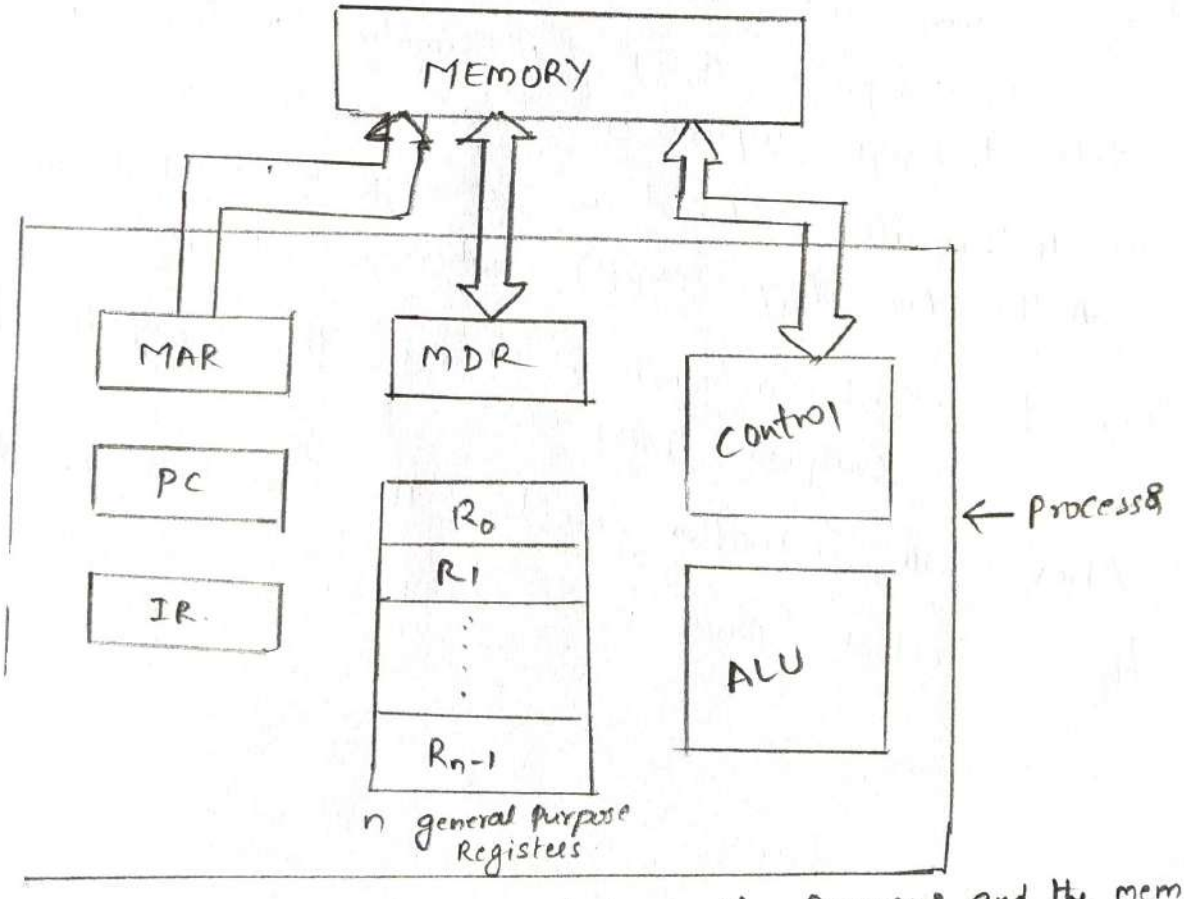


fig:- Connections between the processor and the memory.

To perform Execution of instruction, in addition to the Arithmetic logic unit, Control Unit, the processor Conlains a number of Registers used for temporary Slorage of data, and some Special function registers as shown in figure The Special function registers are : (1) program Counter (PC)
(2) Instruction Register (IR)
(3) Memory Address Register (MAR)
(4) Memory Data Register (MDR)

Program Counter (PC) :- It is one of the most important registers in processor (CPU). It keeps track of the Execution of a program and it Contains the memory address of the next instruction to be fetched and Executed. During Execution of an instruction, the contents of the "PC" are updated to correspond address of next instruction to be Executed.

(2) Instruction Register :- (IR) :- It holds the instruction that is currently being Executed. The output is available to Control circuits, which generates the timing signals that control Various processing Elements involved in Executing the instruction

(3) Memory Address Register (MAR) :- It holds the address of the main memory to be accessed.

(4) Memory Data Register (MDR) :- It Contains the data to be written into (or) read out of the addressed location.

Along with this there are n-general purpose registers in processor Unit as shown in figure. They are R_0 to R_{n-1} registers.

In addition to transferring data between the memory and the processor, the Computer accepts data from Input devices and Sends data to Output devices.

* Bus Structures :-

When a word of data is transferred between units, all its bits are transferred in parallel, i.e., the bits are transferred simultaneously over many wires (or) lines, and one bit per line.

* A group of lines that serves as a connecting path for several devices is called a bus.

→ In addition to the lines that carry data, the bus must have lines for address and control purposes. So Address bus, data bus, control bus together are called as System bus.

The simplest way to interconnect functional units is to use a single bus and it shown in below fig(a). Single bus structure is low cost and its flexibility for attaching peripheral devices.

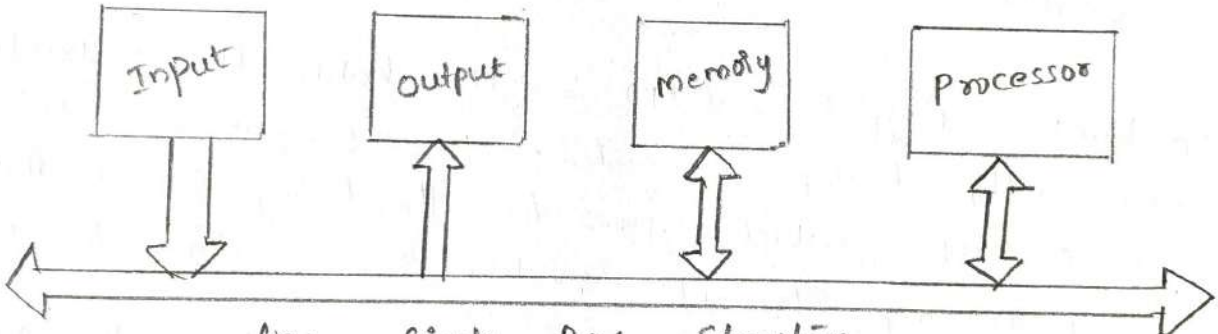


fig:- Single Bus Structure

The bus can be used for only one transfer at a time, only two units can actively use the bus at any given time. Bus control lines are used to arbitrate multiple requests for use of the bus.

→ Multiple bus structure is also available, Systems which contain multiple buses achieve more concurrency in operations by allowing two (or) more transfers to be carried out at the same time. This leads to better performance but at an increased cost.

* System Software :-

For a user to Enter and run an application program, the computer must already contain Some system software in its memory.

→ System Software is a collection of programs that are Executed as needed to perform function such as :

(i) Receiving and interpreting user Commands.

(ii) Entering and editing application programs and storing them as files in secondary storage devices.

(iii) File Management :- storage and Retrieval of files in secondary storage devices [Ex: hard disk (or) floppy disks etc]

(iv) Running Standard application programs such as word processors, Spreadsheets etc.

(v) Controlling I/O units to receive input information and produce output results.

(vi) Translating programs from source form prepared by user into Object form consisting of machine instructions.

(vii) Linking and Running user written application programs.

(viii) Debugging the user written application programs.

The System Software is thus responsible for the Coordination of all activities in a Computing System.

→ Application programs are usually written in a high-level Programming language, such as C, C++, Java or Fortran etc in which programmer specifies mathematical (or) text processing operations.

Compiler translates the high level language Program into Suitable machine language program containing instructions such as Add, Load etc in a System Software.

→ Editor is a System Program that all programmers used for Entering and Editing application programs.

Operating system (OS) :- It is a large program (or) actually a collection of routines, that is used to control the sharing of and interaction among various computer units as they Execute application programs.

The OS routines Perform the tasks required to assign computer resources to individual application Programs.

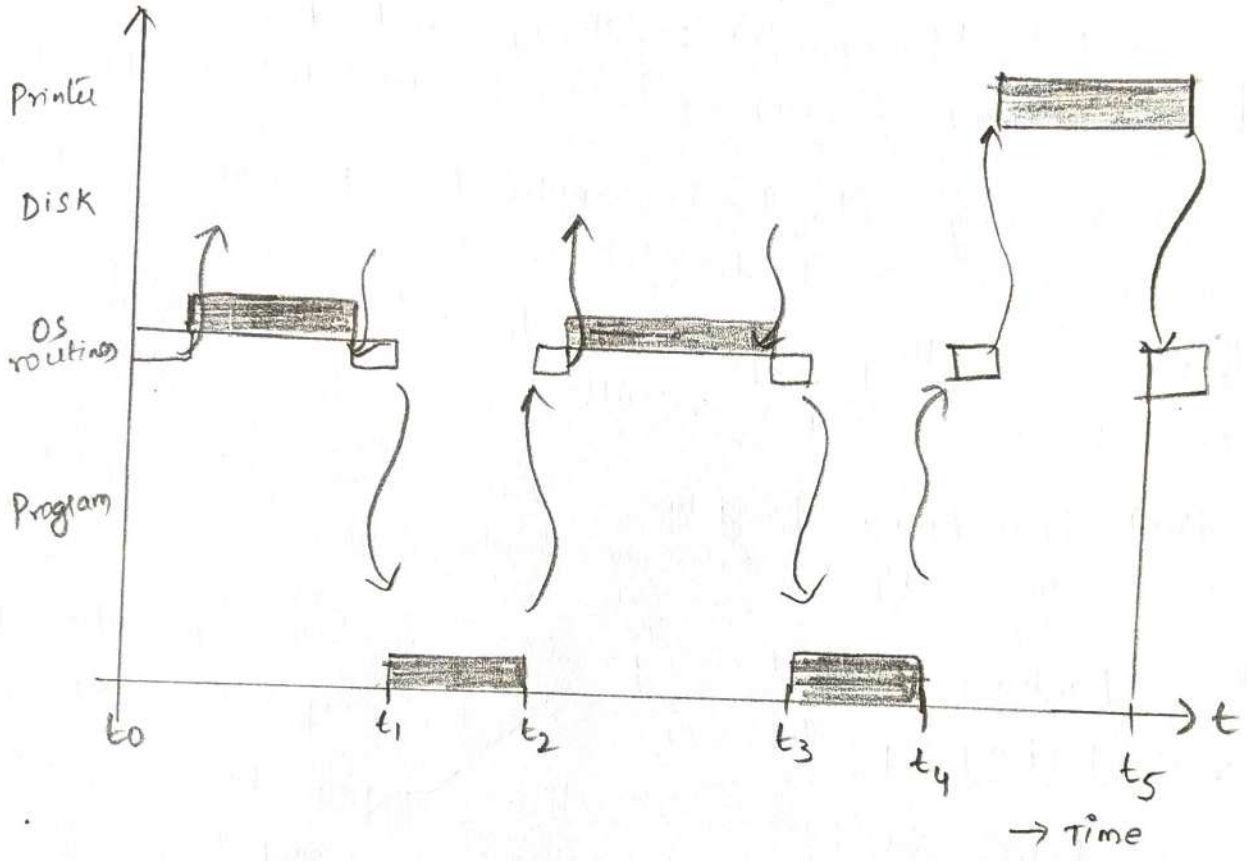


fig:- User program and OS routine sharing of the processor

The Operating System (OS) manages the Concurrent Execution of several application programs called as multiprogramming (or) Multitasking.

* PERFORMANCE :-

The most important measure of the performance of a computer is how quickly it can execute programs. The computer user is always interested in reducing the time between the start and completion of program, i.e., reducing the execution time (or) response time. Thus as response time is reduces, "throughput" increases (Amount of work done in a given time).

→ The speed with which a computer executes programs is affected by the design of its hardware and its machine language instructions.

→ The performance of com. processor is consider during the period of processor is active.

→ The processor cache is depicted as :-

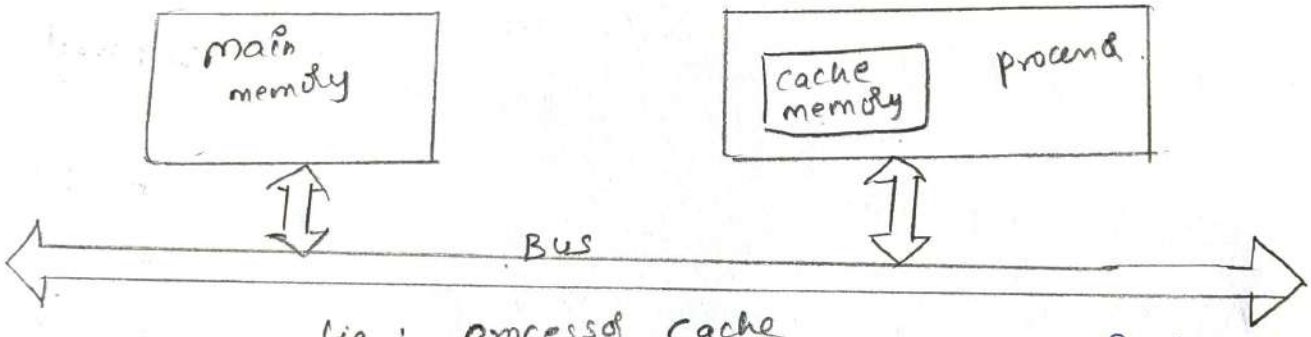


fig :- processor cache

→ At the start of execution, all program instructions and the required data are stored in the main memory.

→ As execution proceeds, instructions are fetched one by one over the bus into the processor, and a copy is placed in the cache.

→ when the execution of an instruction calls for data located in the main memory, data are fetched and a copy is placed in the cache.

→ Later, if the same instruction (or) data is needed a then it reads directly from the cache.

System performance is measured by :-

(i) Processor clock :-

The processor circuits are controlled by a timing signal called a clock.

→ A clock defines regular time intervals, called clock cycles.

→ To Execute a machine instruction, the processor divides the actions to be performed into a sequence of basic steps, such that Each Step can be completed in one clock cycle

→ The length 'P' of one clock cycle is an important parameters that effects processor performance.

Its inverse is the clock Rate, $R = \frac{1}{P}$ which is measured in cycles per second.

In standard Electrical Engineering terminology, the term cycles Per Second is called Hertz.

(ii) Basic performance EQuation :-

Let "T" be the processor time required to Execute a program.

→ Assume that complete Execution of the program requires the Execution of 'N' machine language instructions.

→ Number 'N' is the actual number of instruction Executions and is not necessarily Equal to the number of machine instructions in the object program.

→ Suppose that the average number of basic steps needed to Execute one machine instruction is 'S', where each step is completed in one clock cycle.

→ If the clock rate is R cycles per second, the program Execution time is given by

$$T = \frac{N \times S}{R}$$

This is referred as Basic performance Equation

(iii) Pipelining and Superscalar Operation :-

→ A substantial improvement in Performance can be achieved by overlapping the execution of Successive instructions, using a technique called Pipelining.

Ex:- Add R1, R2, R3

Above instruction adds, the contents of Registers R1, R2 & Places the sum into R3 Register.

→ The processor can read the next instruction from the memory while the addition operation is being performed.

(iv) Clock Rate :-

There are Two possibilites for increasing the Clock Rate, (R).

(1) First :- Improving the integrated circuit (IC) technology makes logic circuits faster, which reduces the time needed to complete a basic step. This allows the clock period (P) to be reduced and Clock Rate (R) to be increased.

(2) Second :- Reducing the amount of processing done in one basic step also makes it possible to reduce the clock period (P).

(V) CISC and RISC :-

Simple instructions require a small number of basic steps to Execute.

Complex instructions involve a large number of steps

A Key consideration in comparing the two choices is the use of pipelining.

CISC & RISC are used for complex instructions.

CISC Complex Instruction set computers (CISC)

RISC : Reduced Instruction Set Computers (RISC)

vi, Compilers :-

→ A compiler translates a high level language program into a sequence of machine instructions. To reduce 'N' (No. of instructions) for execution, we need to have a suitable machine instruction set and a compiler that makes good use of it.

→ An optimizing compiler takes advantage of various features of the target processor to reduce the product N×S, which is the total number of clock cycles needed to execute a program.

viii, Performance Measurement :-

Computer designers use performance estimates to evaluate the effectiveness of new features.

→ Manufacturers use performance indicators in the marketing process. Buyers use such data to choose among many available computer models.

The computer community adopted the idea of measuring computer performance using benchmark programs. To make comparision possible, standardized programs must be used. The performance measure is the time it takes a computer to execute a given benchmark.

A non profit organization called SPEC [system performance evaluation corporation) selects and publishes representative application programs for different application domains

The SPEC rating is computed as,

$$\text{SPEC Rating} = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

The overall SPEC Rating for the computer is given by

$$\text{SPEC Rating} = \left[\prod_{i=1}^{n} \text{SPEC}_i\right]^{1/n}$$

Here 'n' is No. of Program Test Cycle

* History Of Computer Development :-

There are five generations of Computers.

(1) First generation (1946-1959)
(2) Second generation (1959-1965)
(3) Third generation (1965-1971)
(4) Fourth generation (1971-1980)
(5) Fifth generation (1980 - onwards)

(1) First generation :- (1946-1959)

In the first generation, it made use of Vaccum tubes which are the only electronic component available during these days, they can calculate in milliseconds.

→J.P. Eckert & J.W. mauchy invented the first successful electronic Computer called "ENIAC" [Electronic Numeric Integrated & Calculator"].

These Vaccum tubes are big in size, weight was about 30 tones. and they are costly and they require large cooling system.

(2) Second generation :- (1959-1965)

In this 2nd generation Computers were based on transistors instead of Vaccum tubes. The size of Electron Component decreased compared to first generation Computers. Assembly language and punch cards were used for input. It calculate data in micro seconds, low cost compared to Vaccum tubes

Ex:- Honey well 400, IBM 7094 etc.,

(3) THIRD GENERATION :- (1965-1971)

In this 3rd generation computers are based on Integrated circuits, these Computers were Cheaper as compared to second generation. IC was invented by Robert Noyce and Jack Kilby, and IC was a Single Component containing number of transistors. IC not only reduces the size of the Computer but it also improves the performance of the Computer as compared to previous generations Computers.

It has big Storage capacity. mouse & Keyboard are used as Input. Operating System is used for better performance. The computational time is in nanoseconds.

(4) Fourth Generation :- (1971- 1980)

In this 4th generation, computers uses a technology based on microprocessor. Microprocend is used for any logical and arithmetic functions to be performed in any program. Graphics User Interface (GUI) technology is used. The Computational time is good, Size is less. All types of high level languages Can be used in this type of computers

Ex :- IBM 4341, DEC 10, PUP 11 etc.,

→ Concurrency, pipelining, Caches and Virtual memories are evolved, to produce high performance Computing Concepts System.

Fifth Generation :- (1980- till now)

This generation is based on Artifical Intelligence and are capable of learning and self organization. In this generation a device can respond to natural language input This generation is based on ULSI (ultra large scale integration) technology. It is more reliable & faster, available in different Sizes and unique features. It provides user friendly interfaces with multimedia features. Advancement of Super Conductor technology.

Ex :- Desktop, Laptop, Notebook etc.,

* Machine Instructions and programs :-

In this we deal with the following topics

→ Instructions and instruction Sequencing
- Register Transfer Notation
- Assembly Language Notation
- Basic instruction types.

* Instructions and instruction sequencing :-

A Computer must have instructions capable of performing four types of operations.

(1) Data transfers Between the memory and the Processor registers.

(2) Arithmetic and logic operations on data

(3) Program sequencing and control

(4) I/O transfers.

* Register Transfer Notation :-

To transfer any information from one location to another location in the computer, the possible locations that may be involved in such transfers are memory locations, Processor registers, (or) Registers in the I/O subsystem. The memory locations addresses may be :

LOC, PLACE, A, VAR2 ; and register names be R_0, R_5 Etc.

and I/O names may be :- DATAIN, OUTSTATUS Etc.,

→ The contents of a location are denoted by placing square brackets around the name of the location.

Ex :- (1) $R1 \leftarrow [LOC]$

means that the contents of memory location 'LOC' are transferred into processor Register R1.

Ex :- (2) $R3 \leftarrow [R1] + [R2]$ [this is known as Register Transfer Notation [RTN]]

means the operation adds the contents of Register R1 & R2 and then places their sum into register R3.

* Assembly language Notation :-

Assembly language Format is a notation to represent machine instructions and programs.

Ex :- (1) MOVe LOC, R1

means data transfered from memory location 'LOC' to Process Register R1, and the contents of 'LOC' are unchanged by Execution of this instruction, but the old contents of Register R1 are overwritten.

Ex :- (2) Add R1, R2, R3

It means adding two numbers contained in process of register R1 and R2 and placing their Sum in R3, specified by above instruction.

* Basic Instruction Types :- (or) Instruction Formats :-

The computer perform tasks on the basis of instructions provided. An instruction in computer comprises of groups called fields. The common fields are (1) opcode (operation code) and (2) Operand (Data).

On the basis of number of address, instructions are Classified as four types, they are :

(1) Three - address instruction
(2) Two - address instruction
(3) One - address instruction
(4) Zero - address instruction

(1) Three Address instruction :-

An instruction which is having three Operands, they are Three Address instruction.

Syntax :-

opcode	source 1, Source 2, Destination

Ex :- Add A, B, C (∵ similar to C ← [A] + [B])

Here Add is opcode, A, B are Source Operands, C is destination Operand.

(ii) Two address Instructions :-

An Instructions having two operands they are known as Two address instructions.

Syntax :- | Opcode | Source, Destination |

Ex :- (1) Move B, C [means move is opcode, B is source, C is destination]

(2) ADD A, C

↓

Add is opcode, A is source operand, C is destination Operand.

(iii) One-address Instructions :-

An instruction having only one operand is known as one-address Instructions. When a second Operand is needed, then processor register called Accumulator is used.

Syntax :- | Opcode | Operand |

Ex :- (1) LOAD A

It means 'Load' instruction copies the contents of memory location 'A' into the Accumulator.

Ex :- (2) Store A

It means 'store' instruction copies the Contents of the Accumulator into memory location 'A'.

Ex :- (3) Add A

It means contents of memory location 'A' is added to the contents of accumulator register and place the Sum back into the accumulator.

Note :- Accumulator is nothing but the processor Register.

(iv) Zero-Address instruction :- An instruction having zero Operands is called Zero Address instructions. It uses stack Operations PUSH & POP to perform.

Ex :- (1) PUSH A

(2) POP C Etc.,

Example :- Evaluate this Expression in different Instruction types ?

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

Three Address Instruction :

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

Two Address Instruction :-

```
MOV A, R1       [∵ R1 ← [A])
ADD B, R1       ( R1 ← [R1] + [B] )
MOV C, R2       [∵ R2 ← [C])
ADD D, R2       [∵ R2 ← [R2] + [D])
MUL R1, R2      (∵ R2 ← [R1] * [R2])
MOV R2, X       (∵ X ← [R2])
```

One Address Instruction :-

```
LOAD A          (∵ AC ← [A])
ADD B           (∵ AC ← [AC] + [B])
STORE T1        (∵ T1 ← [AC])
LOAD C          (∵ AC ← [C])
ADD D           (∵ AC ← [AC] + [D])
MUL T1          (∵ AC ← [AC] * [T1])
STORE X         (∵ X ← [AC])
```

Zero Address Instruction :-

```
PUSH A          (∵ TOS ← [A])
PUSH B          (∵ TOS ← [B])
ADD             (∵ TOS ← [A] + [B])
PUSH C          (∵ TOS ← [C])
PUSH D          (∵ TOS ← [D])
ADD             (∵ TOS ← [C] + [D])
MUL             (∵ TOS ← ([A] + [B]) * ([C] + [D]))
POP X           (∵ X ← [TOS])
```

UNIT-II Addressing Modes :

→ Basic Input/output Operations
→ Role of stacks and Queues in Computer programming Equation.

→ Component Of Instructions :
- Logic Instructions
- Shift & Rotate Instruction

→ Type Of Instructions
- Arithmetic & Logic Instructions.
- Branch Instructions.

→ Addressing Modes
→ I/O operations.

* Addressing Mode :-

The different ways in which the location of an operand is specified in an instruction are referred to as Addressing Modes.

(or)

The way in which effictive Addren of an operand is Specified in the instruction is called as Addressing modes.

→ Effective Address (EA) (or) Location (LOC) :-

EA is the address of the Exact memory location where the value of the operand is present.

→ For Each instruction we have two fields they are Opcode & Operand. Addressing mode is focuses mainly on operand's addresses in the instruction.

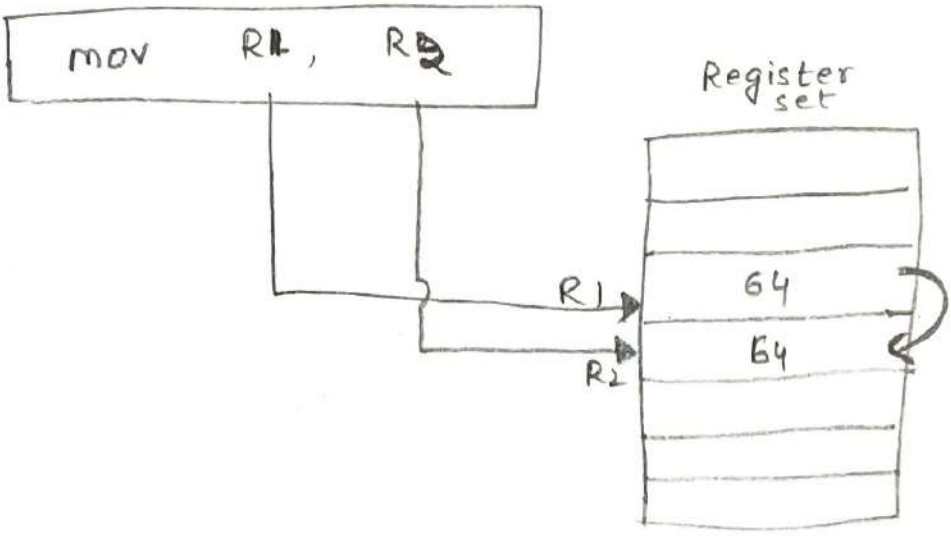
There are different kinds of Addressing modes. They are

(1) Register Addressing Mode
(2) Direct Addressing mode (or) Absolute Mode → Implementation of Variables & constants
(3) Immediate Addressing Mode
(4) Register Indirect Addressing mode → Indirection & pointers
(5) Index Addressing mode
    - Base with Index
    - Base with Index & Offset
    } Indexing & Arrays
(6) Relative Addressing mode → Relative Addressing
(7) Auto Increment mode
(8) Auto decrement mode
} → Additional Modes

(1) Register Addressing Mode :- The instruction which uses Processor registers to represent operands is the instruction in Register Addressing Mode. Here Effective Address is a register where the Value of the Operand is present (EA = R)

Ex :- MOV R1, R2

In the above Example, Move Instruction uses Registers to represent both of its operands. The contents in Register R1 is moved into Register R2.

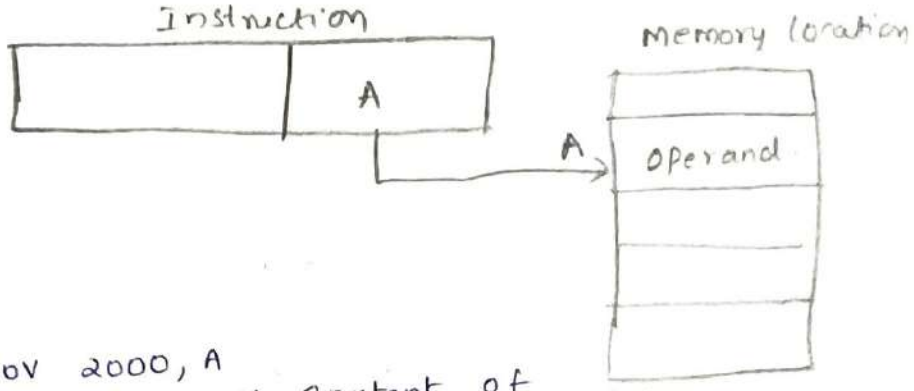


(2) Direct (Absolute) Addressing Mode :-

It is also known as Absolute Addressing mode. In this Addressing mode, the Operand is in a memory location.

Ex:- ADD A, B

In the above Example contents in the location A, B are added and result is Saved in 'B' Location.



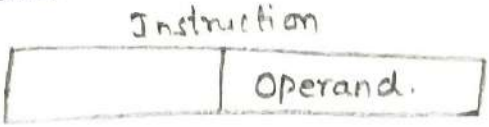
Ex:(2) MOV 2000, A
means Content of memory location 2000 into the Register 'A'.

(3) Immediate Addressing Mode :-

In Immediate Addressing mode, the Value of the Operand is Explicitly mentioned in the instruction itself.

Ex:- MOV #200, R1

In the above Example Value 200 is moved into Register 'R1'.
# indicates the data (or) Value.



(4) Register Indirect Addressing Mode :-

A Processor Register is used to hold the address of a memory location where the Operand is placed, that type of instructions is called Register Indirect Addressing Mode.
It is Referred to as pointers. The Indirect mode is denoted by placing the register inside the parenthesis

Ex:- MOV (R2), R3

In the above Example, The ~~Value (of)~~ Addren in R2 register is first reads, and contents (of) data in memory location R2 is read and then its value is moved into R3 register

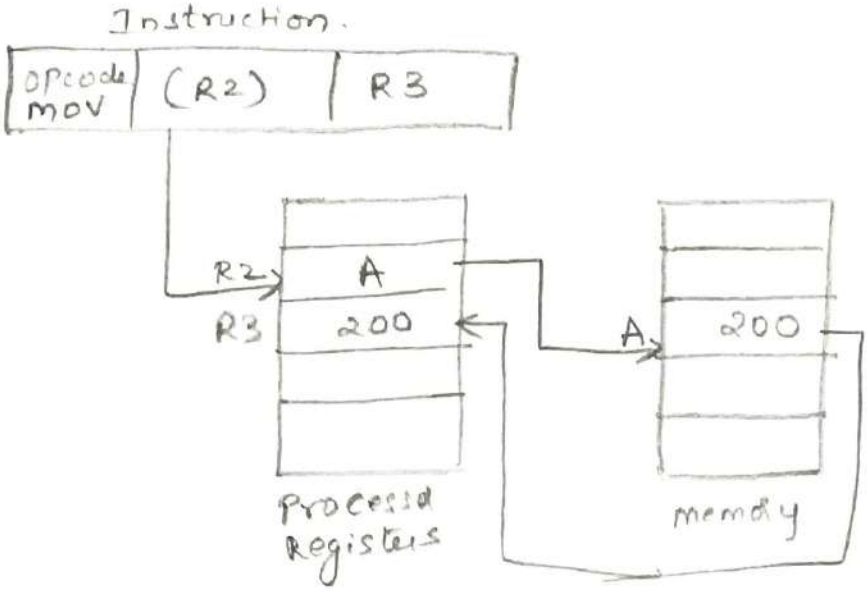


fig:- Register Indirect Addressing mode

(5) Index Addressing Mode :-

This Addressing mode is useful in dealing with lists and Arrays. In this mode the Effective address is generated by adding a constant to the registers Content. The register may be either a special register(or) general purposer register called Index Register.

Index mode Symbolically represenled as: X(Ri)

The Effective Addren of the Operand is given by

$$EA = X + [Ri]$$

Where X is constant, [Ri] → Contents in Ri registers.
X is Offset(or) displacement.

There are two types of Index Addressing mode. They are

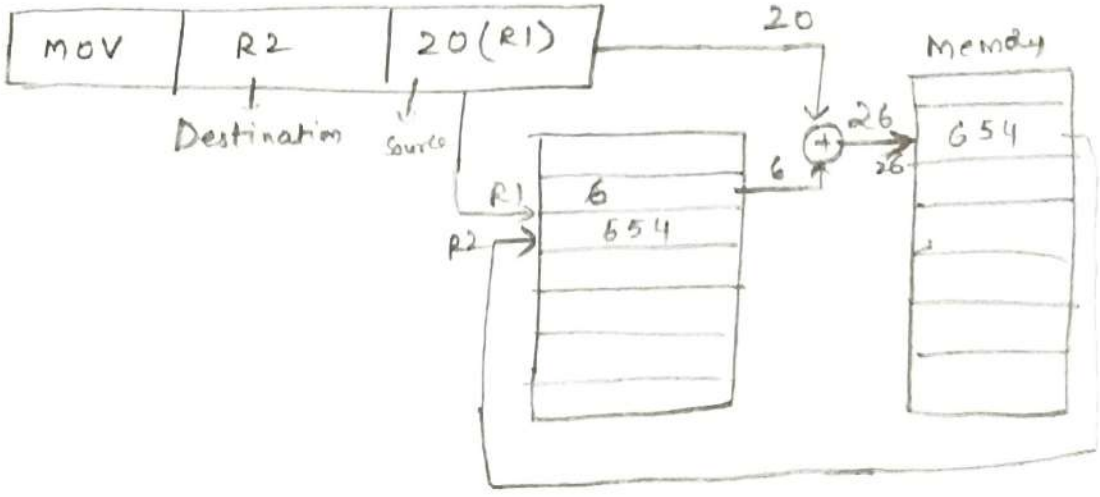
(i) Base with Index Addressing mode.

(ii) Base with Index & Offset Addressing mode.

Index mode

Ex:- MOV R2, 20(R1)

In the above Example, the addition of the contents of Register R1 and Constant Value (offset & displacement) 20 is added, and Effective address is calculated, and in that Address, the Value of data is moved into Register R2.

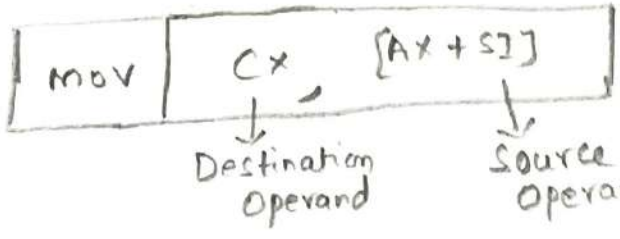


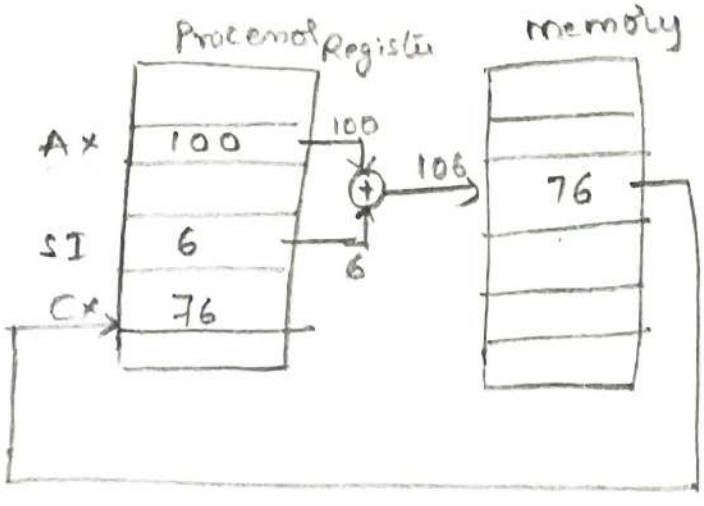
Base Indexed Addressing modes :-

Ex:- MOV CX, [AX + SI]

destination, source

In the above Example AX is Base register, SI is Index register both Values (or) data in these registers are added, and we get a addren called Effective Addren, In that address the corresponding Value (or) data is moved into CX register.

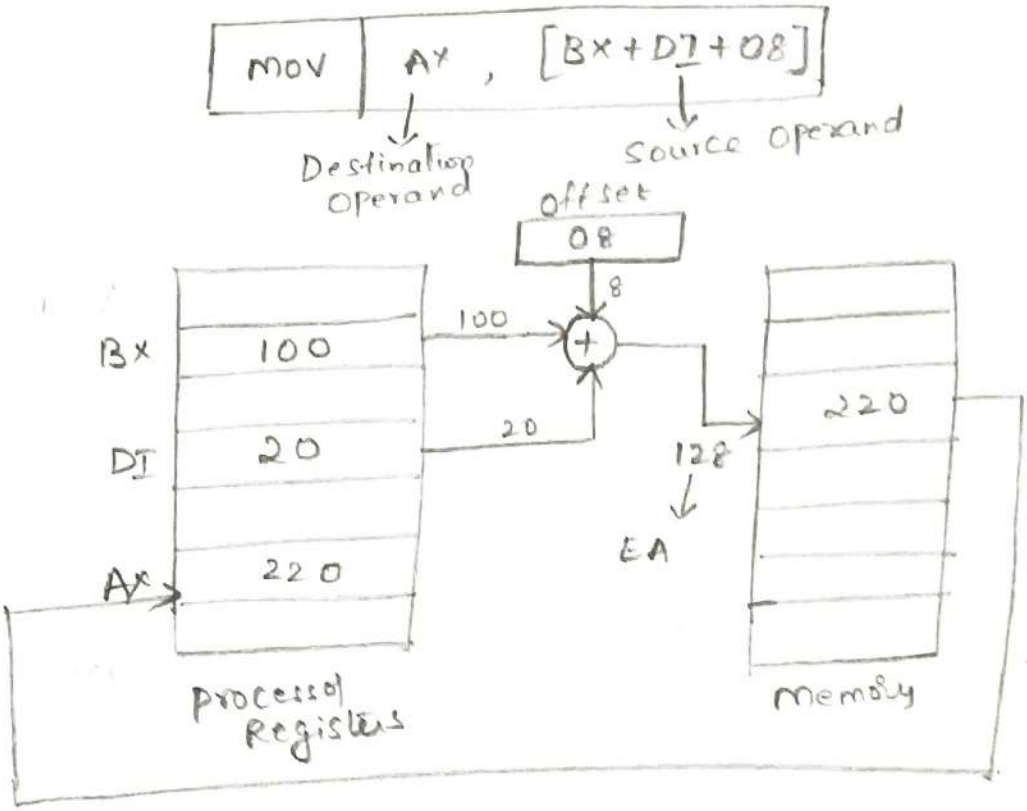




Base Indexed with Offset Addressing mode :-

Ex:- MOV AX, [BX + DI + 08]

In the above Example we have Base register BX, Index Register DI and displacement (or) Offset (08) is added together then we get an Effective address, in that address of memory the data (or) Value is present which is Moved into AX register



(6) Relative Addressing Mode :-

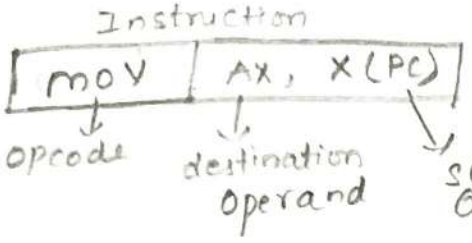
The Index mode uses general purpose Registers, whereas Relative Addrening mode uses Program Counter (PC).

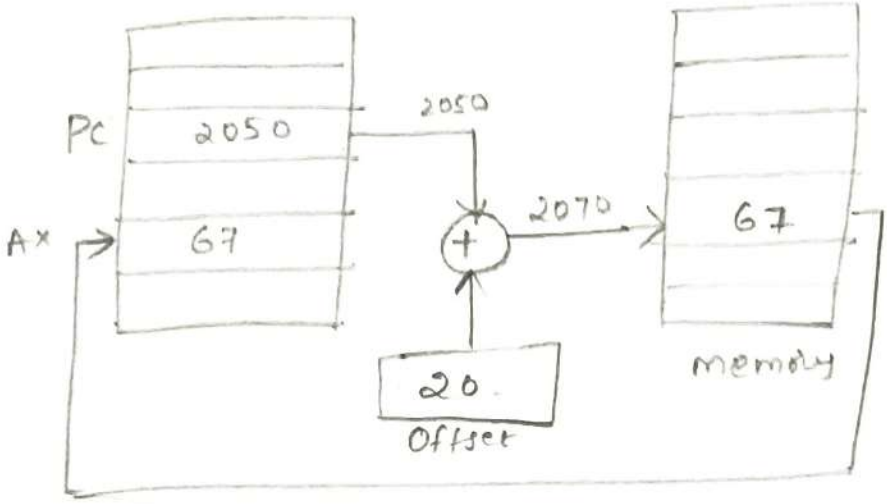
Symbolic Represenlation is X(PC)

EA = X + (PC)

In this Value in PC (Program Counter) is added with Offset (or) displacement then we get an Effective Addren, where the operand is present.

Ex:- MOV AX, X(PC)





(7) Auto Increment Addressing modes :-

The Effective address of the operand is the contents of a register specified in the instruction. After Accessing the operand (or) data, the contents of the register are incremented to address the next location.

Symbolic represented as : (Ri) +

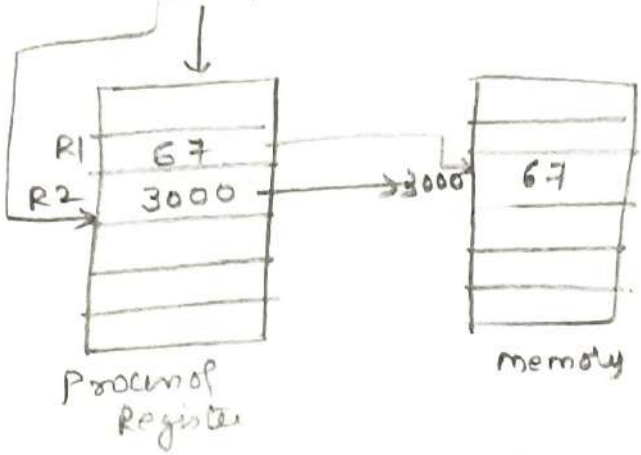
Effective Addren of the operand is

EA = [Ri]
Increment Ri

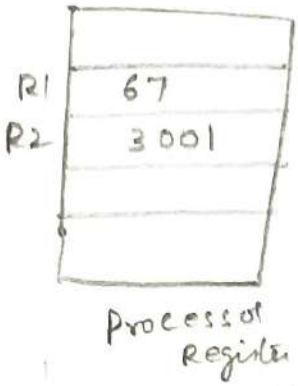
Ex:- MOV (R2), +R1

In the above instruction, the contents of R1 is ~~moved~~ from ~~into~~ memory location whose addren is specified by contents in Register R2. After 'move' operation, the contents of Register R2 is automatically incremented by 1.

Ex:- MOV [R2], +R1



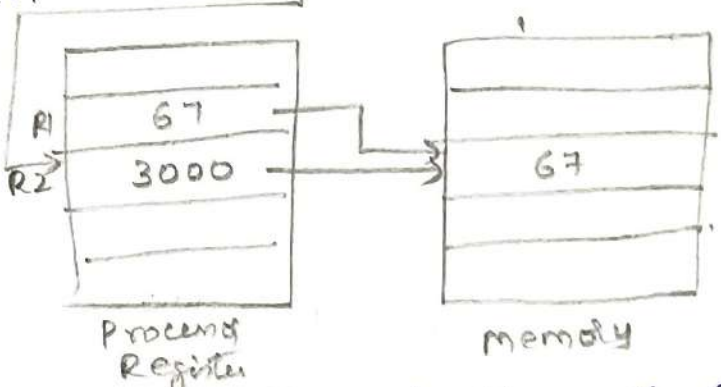
After Execution of Above instruction, Value in R2 is incremented automatically by 1



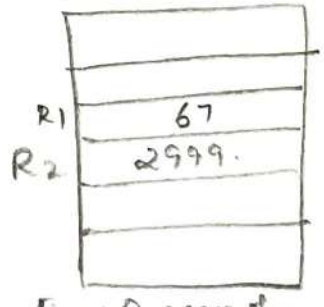
(8) Auto Decrement Addressing Modes :-

Address of the operand is in a Register whose Value is decremented after fetching the operand from the address.

Ex:- MOV [R2], -R1



After Execution of Above instruction, the data in a register is decremented automatically by 1.



* Basic Input / output Operations :-

The transfer of data between Keyboard and processor and display device is called Input / output data transfer.

Suppose, a input from keyboard is read a character and produces character output on a display screen. This simple way of performing such Input / output tasks uses a method known as program-Controlled I/O.

→ The rate of data transfer from the keyboard is limited by typing Speed of the user and the rate of data transfer to the display device is determined by rate at which characters can be transmitted over the link between the Computer & the display device.

→ The rate of output data transfer to display is much higher than the input data rate from the keyboard, however both of these rates are much slower than Speed of a processor that can Execute many millions of instructions per second.

→ Due to the Speed difference between these devices, we have to use Synchronisation mechanism for proper transfer of data between them.

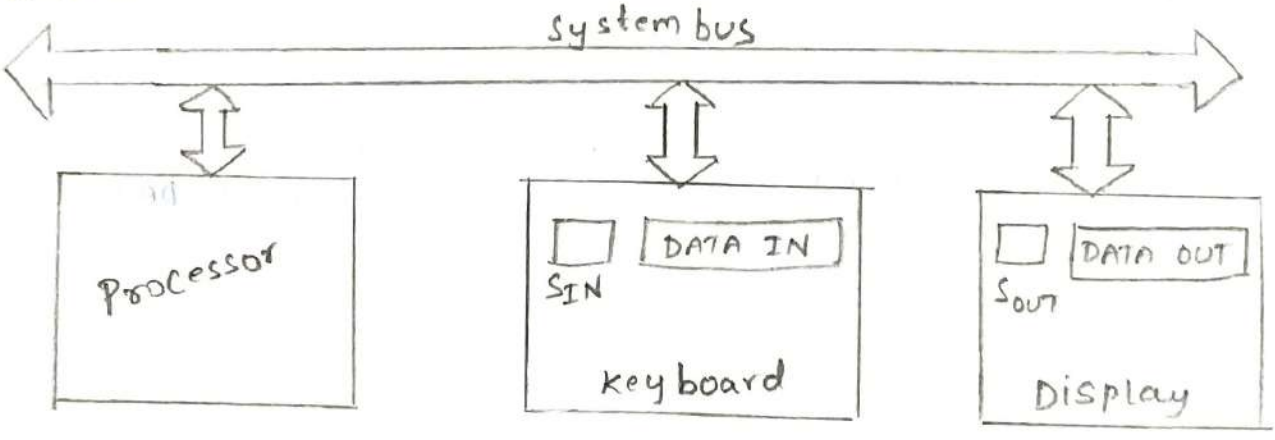


fig :- Bus Connection for processor, Keyboard and display

The above figure shows, S_{IN} and S_{OUT} status bits are used to Synchronize data transfer between Keyboard, Processor and data transfer between display & processor respectively.

→ If a Key is Pressed, the corresponding character code is Stored in the DATA IN register.

→ Now S_{IN} Status bit is set (1) to indicate, valid character code is available in DATA IN register.

→ Now processor Checks status bit S_{IN}, if $S_{IN}=1$, then it reads the Contents of the DATA IN register.

→ After completion of read operation S_{IN} status bit is automatically reset to (0) i.e., $S_{IN}=0$.

→ If another key is pressed, same procedure repeats.

Example:

```
READWAIT    Branch to READWAIT if SIN = 0
            Input from DATAIN to R1
```

Instruction:-

```
Movbyte  DATAIN , R1
```

DATAIN: Source operand; R1: destination operand

→ When characters are transferred from the processor to display, DATA OUT register, and Status bit S_{OUT} is used for transfer.

→ When $S_{OUT}=1$, display is ready to receive character, so processor Sends data to DATA OUT register and clears Status bit $S_{OUT}=0$.

Example:-

```
WRITEWAIT   Branch to WRITEWAIT if SOUT = 0
            Output from R1 to DATAOUT
```

Instruction:-

```
Movbyte  R1, DATAOUT
```

R1: Source operand; DATAOUT: destination operand

The contents of R1 Register can be transferred to DATAOUT.

* The Role of Stacks and Queues in Computer Programming Evaluations

STACK :- Stack is a list of data Elements, where the insertion and deletion takes place at one end, Known as top.

The Stack is also called as LAST IN FIRST OUT (LIFO), That means, the Element which is inserted in the stack at last must be deleted first.

Ex:- A pile of trays in cafeteria

For insertion and deletion of Elements from the stack, we use terms called "PUSH" and "POP" respectively.

A Stack is a data structure where we can store a group of items.

→ A Set of memory locations in memory is nothing but a Stack as shown in below figure.

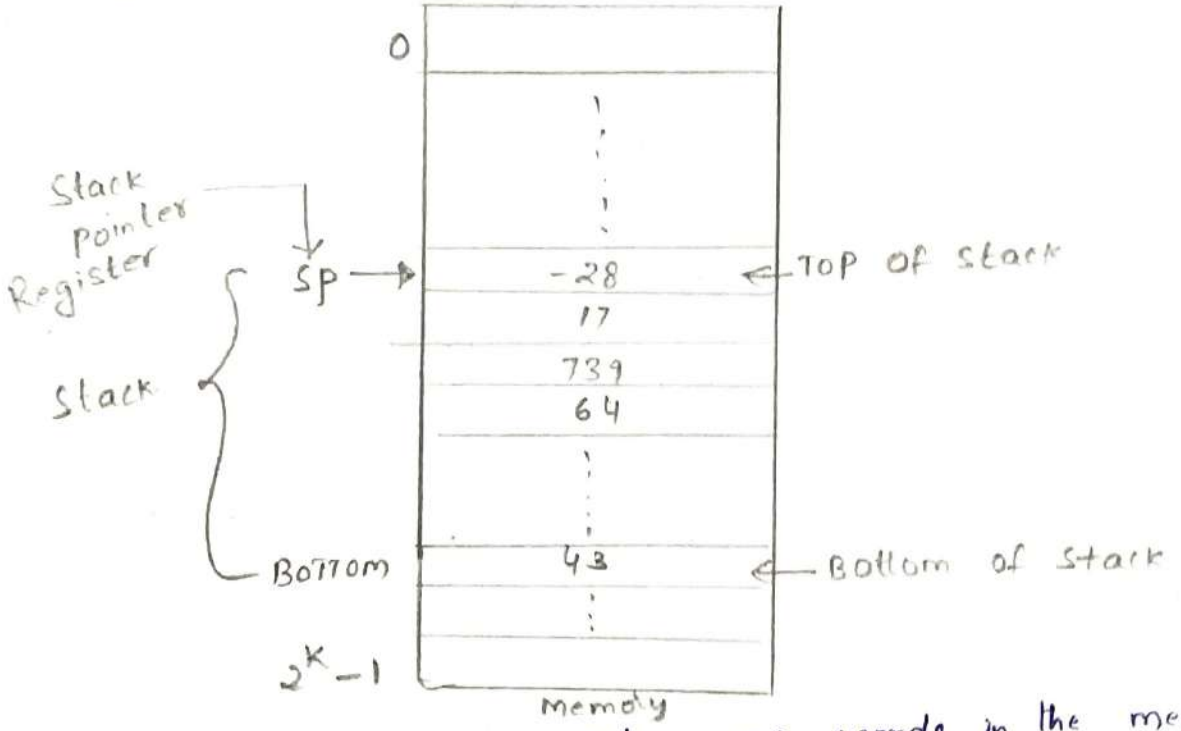


Fig:- A stack of words in the memory

The first Element is placed at location 'BOTTOM', and new Elements are pushed onto the Stack and are placed in Succesively lower Address location.

The address of Top of Stack (TOS) is placed in a register called Stack pointer (SP) Register, which is a Special purpose

register in processor.

Consider a scenario like: the main memory capacity is 2048 MB. In that stack occupies memory locations from 2000 to 1500 location. Initially the stack is empty. Lets have a stack pointer whose name is sp to point the Elements in the stack.

PUSH Operation :-

Adding (or) inserting an Element to a stack is called 'PUSH' operation. when adding an Element to a Stack we need to decrement the Stack pointer (SP). Then after decrementing the stack the Element to be pushed on to the new location points to by Stack pointer (SP).

Ex:- If we want to place a data item 'A' on to stack

```
SP = SP - 1
MOVE A, [SP]
```

In the above Example SP is decremented, then A is moved into address of sp register.

→ Above Example can also be represented, in the following way if we use Auto decrement mode

Ex:-
```
MOVE -A, -[SP]
```

POP OPERATION :-

Removing (or) deleting an Element from the Stack is called "pop" Operation. when deleting an Element from stack, we need to increment the stack pointer (SP). to show next top Element. To Implement pop Operation first we need to copy the item and then just increment the Stack pointer.

Ex:-
```
MOVE [SP], TEMP
Sp = Sp+1
```

In the above Example, the Element in stack points address location is moved (or removed into TEMP Valiable, then Stack pointer is incremented to next top Element.

If Auto Increment Addressing mode is used for above Example, it can be represented as follows : (1)

Ex:- MOVE [SP]+, TEMP

QUEUE :- Queue is a linear list of Elements in which Insertion (or) addition, and deletion can be done.

For data Adding (or) insertion, and for retrieving data, Queues will follow a basis known as FIRST-IN-FIRST-OUT (FIFO). Data is inserted (or) added at high-address End (Rear), for this we use Enqueue Operation.
For Data Retreving (or) removing at lower Addren end (front) for this we use dequeue operation.

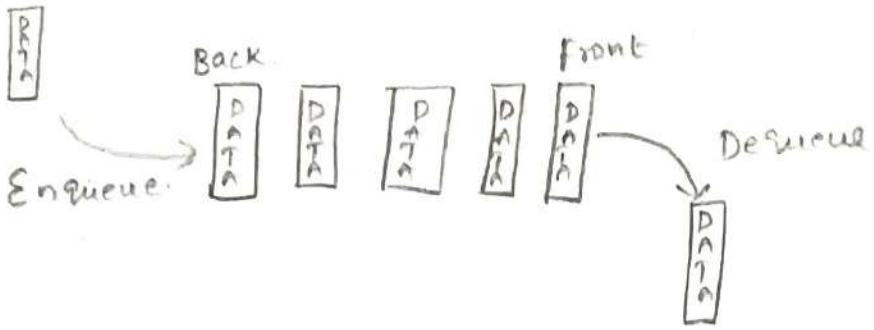


fig: Representation of FIFO Queue.

The main difference between Stack and Queue are as follows:

(1) In stack, insertion & deletion are at one End of stack which is fixed; i.e one side from the list; called TOS
(2) In Queue, insertion & deletion are from both Ends i.e Front & Rear.
(3) In Stack, it Follows LIFO, whereas Queue follows FIFO
(4) In stack it uses PUSH & POP operations, whereas Queue uses Enqueue & dequeue operations.
(5) Stack is used in Recursion problem Solving, whereas Queue is used in Sequential procesing problems.

* Component Of Instructions :-

(i) Logic Instructions
(ii) Shift & Rotate Instructions

(i) Logic Instructions :-

Logical operations are AND, OR and NOT applied to each and every individual bits. These operations are basic building blocks of digital circuit. These instructions are useful to able to perform logic operations in Software.

For Example :- (1) AND Operation :- AND Source, Destination

Ex(1) :- AND R0, R1

In the above Example contents in R0 & R1 are AND and result is Stored in R1 registers.

Suppose R0 = 0010 (4 bit data) & R1 = 0011 (4 bit data

AND R0, R1

means R0 = 0010
R1 = 0011
} perform AND operation

R1 ← 0010

AND

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Operation :-

OR Source, Destination

Ex :- OR R0, R1

In the above Example Contents in R0, R1 procend registers are 'OR' and result is Stored in R1 register.

suppose R0 = 0010
R1 = 0011

R1 ← 0011 (OR)

OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT Operation :-

NOT destination

Ex :- NOT R0

In the above Example Contents in R0 are NOT ; i.e., 0's are to 1's & 1's to 0's

R0 = 0010
NOT R0 ← 1101
} After NOT operation

NOT

A	Y
0	1
1	0

For 1's complement we use `NOT Destination` Instruction

for 2's complement we have add +1 to 1's Complement

for Ex:-

```
NOT R0
Add #1,R0
```

Ex:- R0 = +3 (0011)

After NOT R0 → R0 = 1100

Then Add +1 to R0 = +1

1101 = -3 (2's complement of 3)

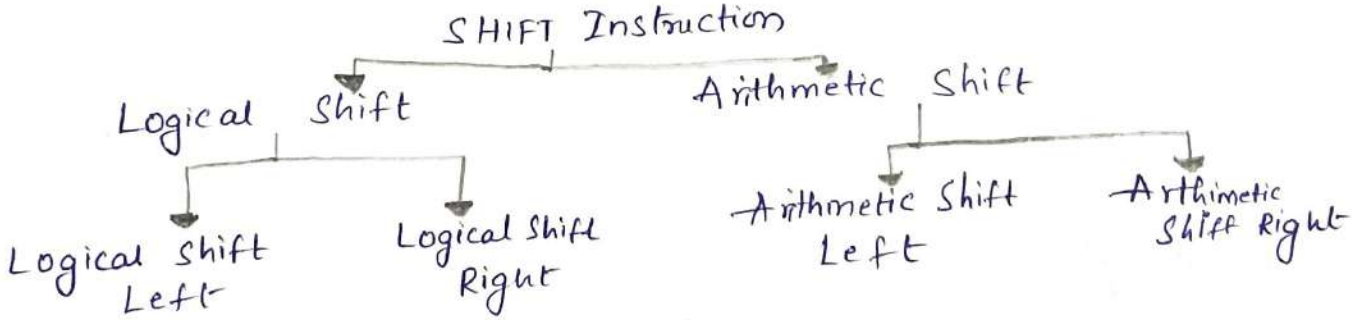
In Some Computers for doing 2's complement, these use a single Instruction is

`Negate R0`

(ii) Shift & ROTATE Instructions :-

(a) SHIFT Instructions :-

In Some applications, the bits of an Operand to be Shifted right or left depending on no. of count. There are basically Two types of Instructions. They are:



Logical Shift Left (LShift L) :-

`Syntax :- LShiftL Count, dst`

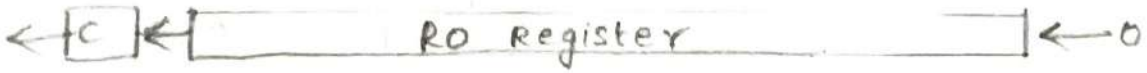
Ex:- (1) LShiftL #1, R0

In the above Example, data in R0 is only '1' shift towards Left.

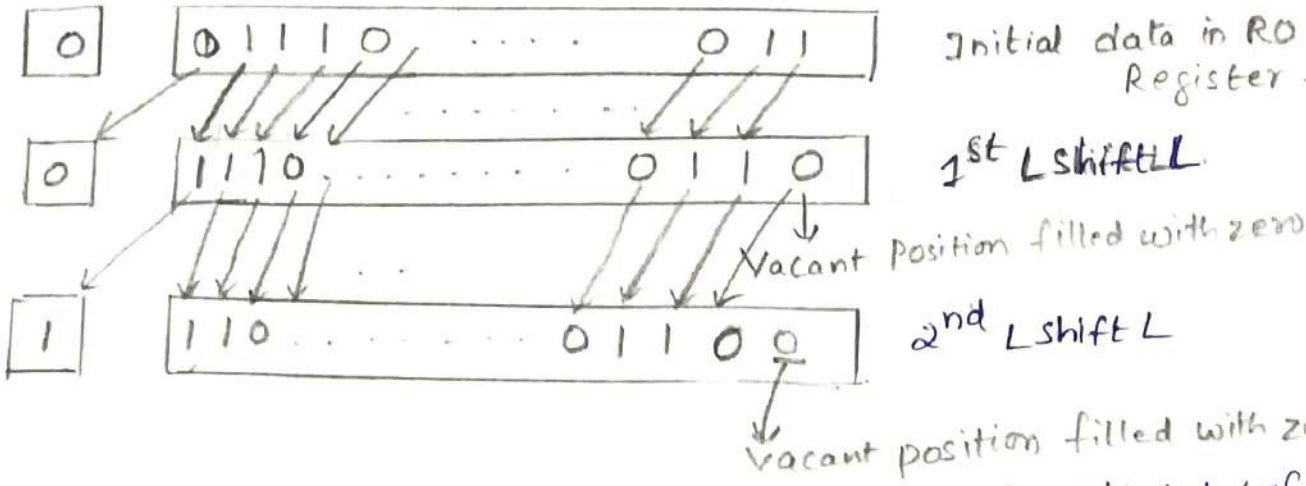
Ex:-(2) LShiftL #2, R0

Here Content in R0 is shifted '2' times left side, and the Vacant position is filled with zero.

Ex:- LShift #2, R0



Ex:-

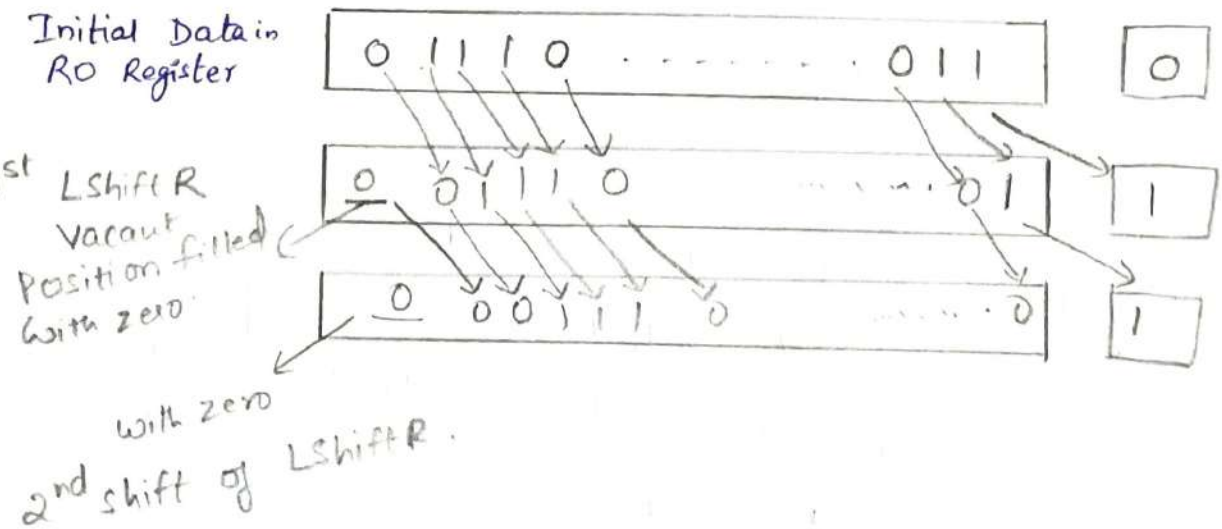


In the above Example, bits in R0 Register is shifted Left and Vacated positions are filled with zero's and the bits shifted out are passed through the Carry flag 'C' and then dropped.

Logic Shift Right (LShiftR) :-

Syntax :- LshiftR count, dst

Ex :- LShiftR #02, R0



In the above Example, it shift the Contents of register R0 right by two bit positions. The vacanted positions are filled with zeros.

Arithmetic shift :- In this we have two types of instructions

(1) Arithmetic shift Left (AshiftL)

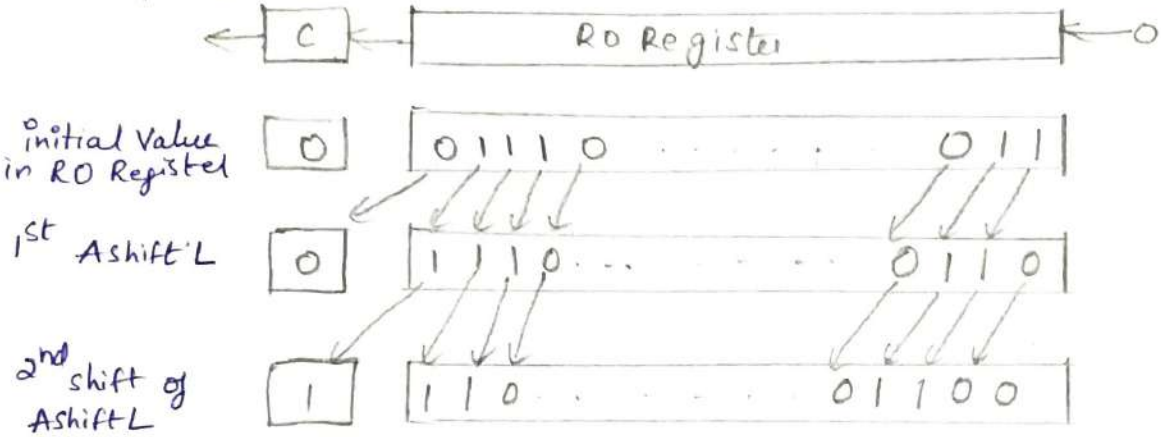
(2) Arithmetic Shift Right (AshiftR)

(1) Arithmetic Shift Left (AShift L) :-

Syntax :- Ashift L Count, dst

Ex :- AshiftL #2, R0

A left Arithmetic shift of a binary number by 2 positions as given in above example. In this Arithmetic shift left, the Empty positions in the LSB (Least Significant Bit) are filled with Zero's. It works same like Logical shift left operation



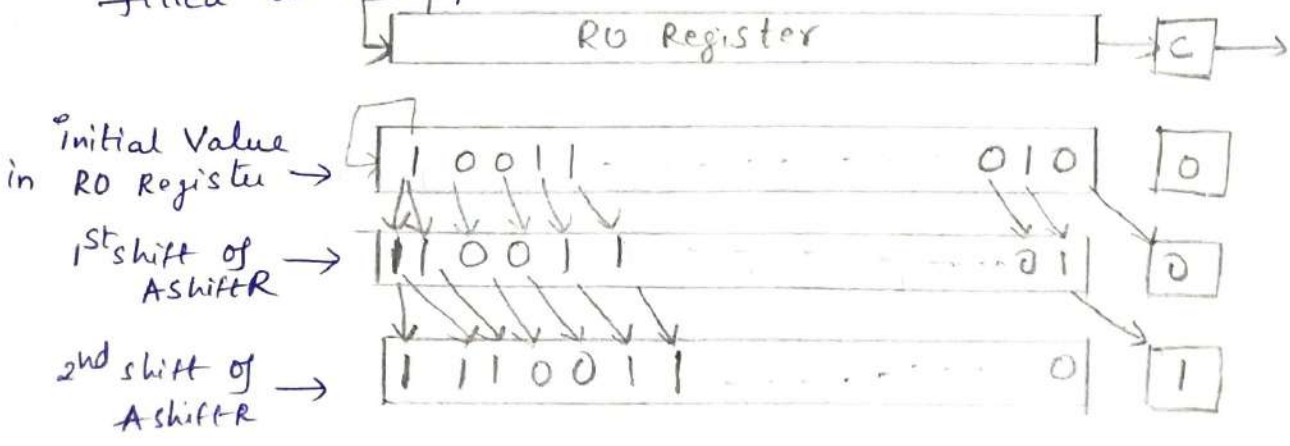
Arithmetic Shift Left

(2) Arithmetic (AshiftR) Shift Right :-

Syntax :- AshiftR count, dst

Ex :- AshiftR #02, R0

A right Arithmetic Shift of a binary number by 2 positions and the Empty positions in the MSB (most Significant Bit) are filled with copies of the original MSB bit.



Arithmetic Shift Right

ROTATE Instructions :-

In the Shift operations, the bits shifted out of the operand are lost, except for the last bit shifted out which is retained in the carry flag 'C'. The rotate operations on the other hand preserves all bits. Rotate instructions move the bits that are shifted out of one end of the operand back into the other end.

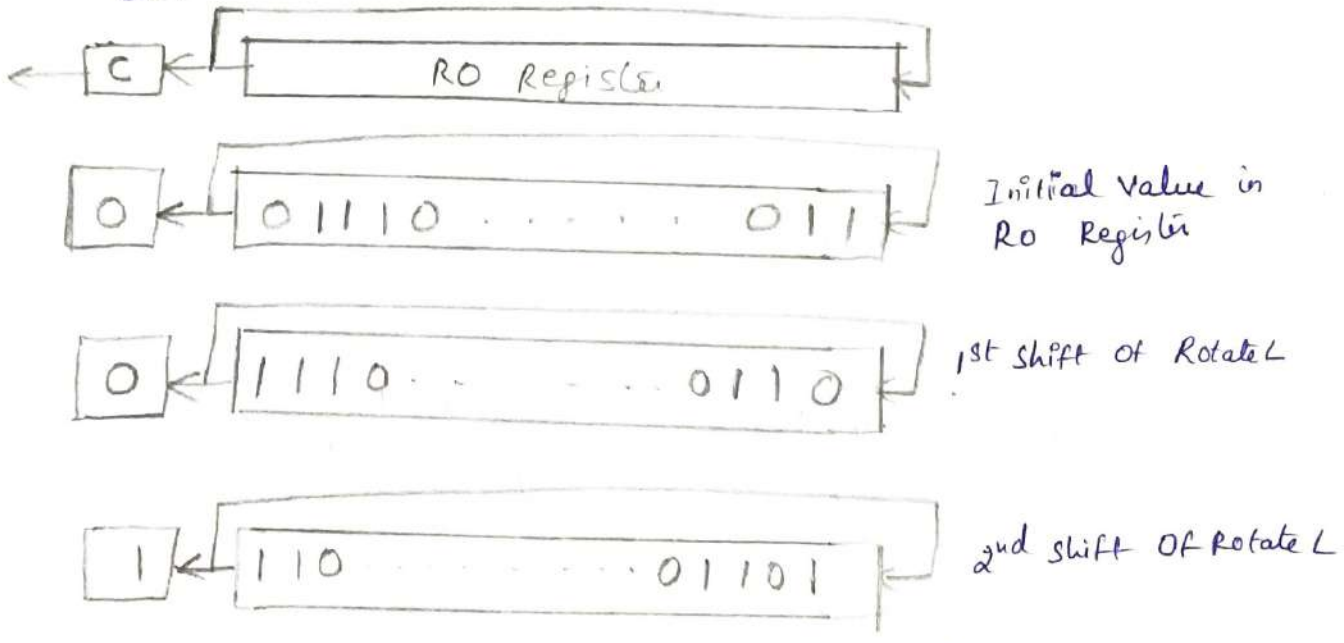
There are four types of Rotate instructions, they are:

(a) Rotate Left without Carry (RotateL)
(b) Rotate Right without Carry (RotateR)
(c) Rotate Left with Carry (RotateLC)
(d) Rotate Right with Carry (RotateRC)

(a) Rotate left without Carry (RotateL) :-

Syntax :- RotateL Count, dsn

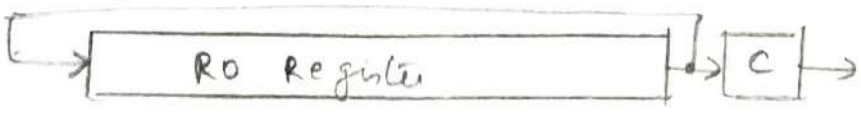
Ex :- RotateL #2, R0

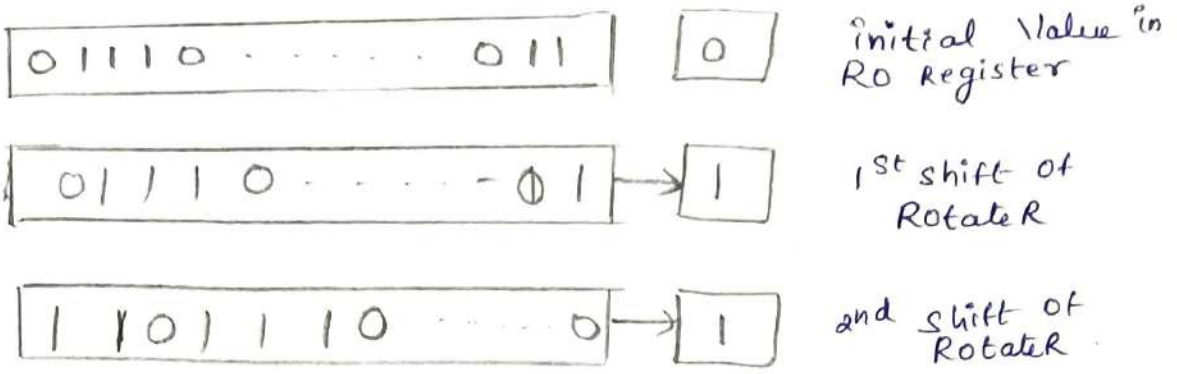


(b) Rotate Right without Carry (RotateR) :-

Syntax :- RotateR Count, dsn

Ex :- RotateR #2, R0

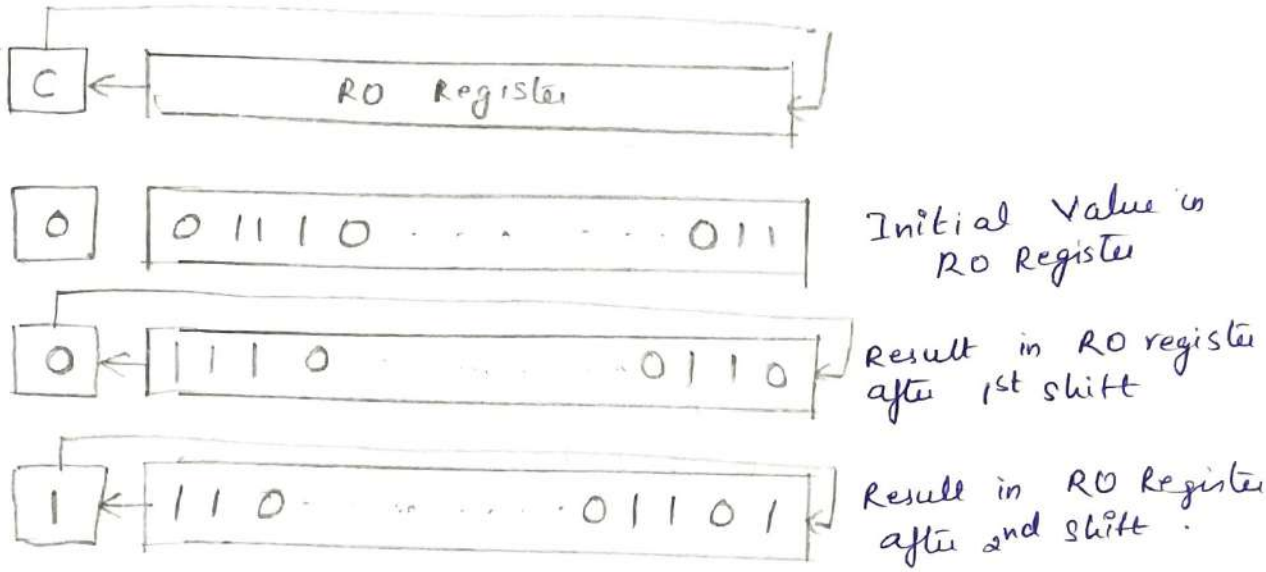




(c) Rotate Left with Carry (RotateLC) :

Syntax :- RotateLC #count, dsn

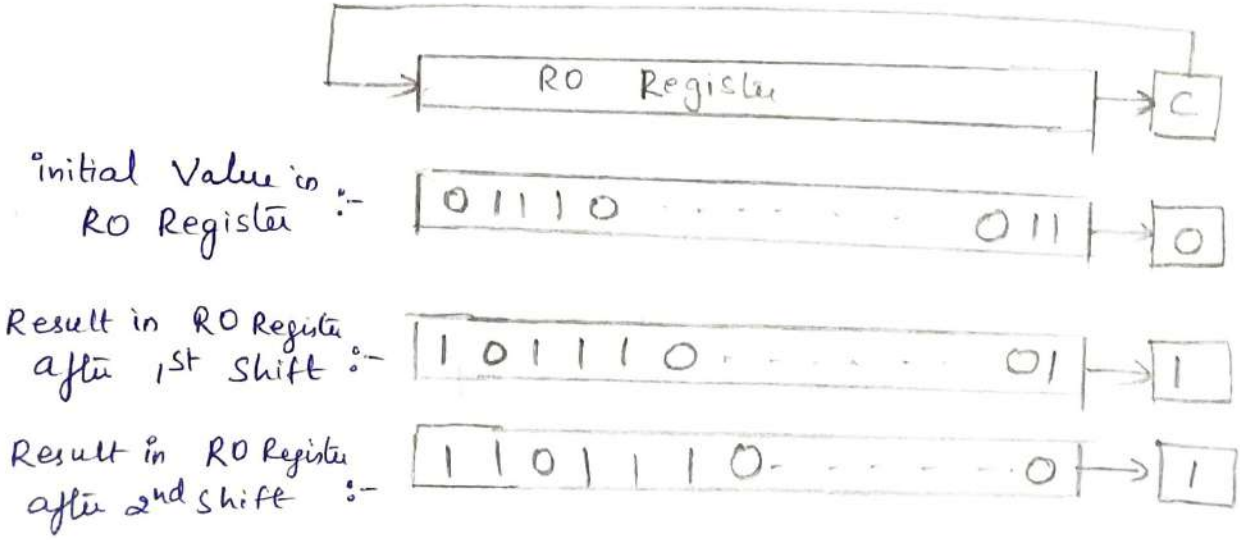
Ex :- RotateLC #02, R0



(d) Rotate Right with Carry (RotateRC) :-

Syntax :- RotateRC Count, dsn

Ex :- RotateRC #02, R0



* ARM Processor :-

ARM is nothing but Advanced RISC machine It was invented by Acorn computers Company in 1980's. It is mainly used because it is low power and low cost.

Applications :- (1) mobile telephones, (2) modems (3) digital hand-cams (4) Automotive Engine management Systems.

Features :- (1) It is a 32 bit RISC processor.
(2) It has 32 bit address bus.
(3) Memory is byte addressable.
(4) memory is accessed only by LOAD & STORE instructions.
(5) All Arithmetic and logic instructions operate only on data in processor registers.
(6) It has ~~to~~ sixteen 32 bit registers from R0 to R15.
(7) Registers R0 - R14 acts as General purpose Registers (GPR's) and R15 act as Program counter (PC).
(8) The GPR's can hold either memory address or data operands.
(9) CPSR - Current program status Register (or) Status register is a 32 bit.

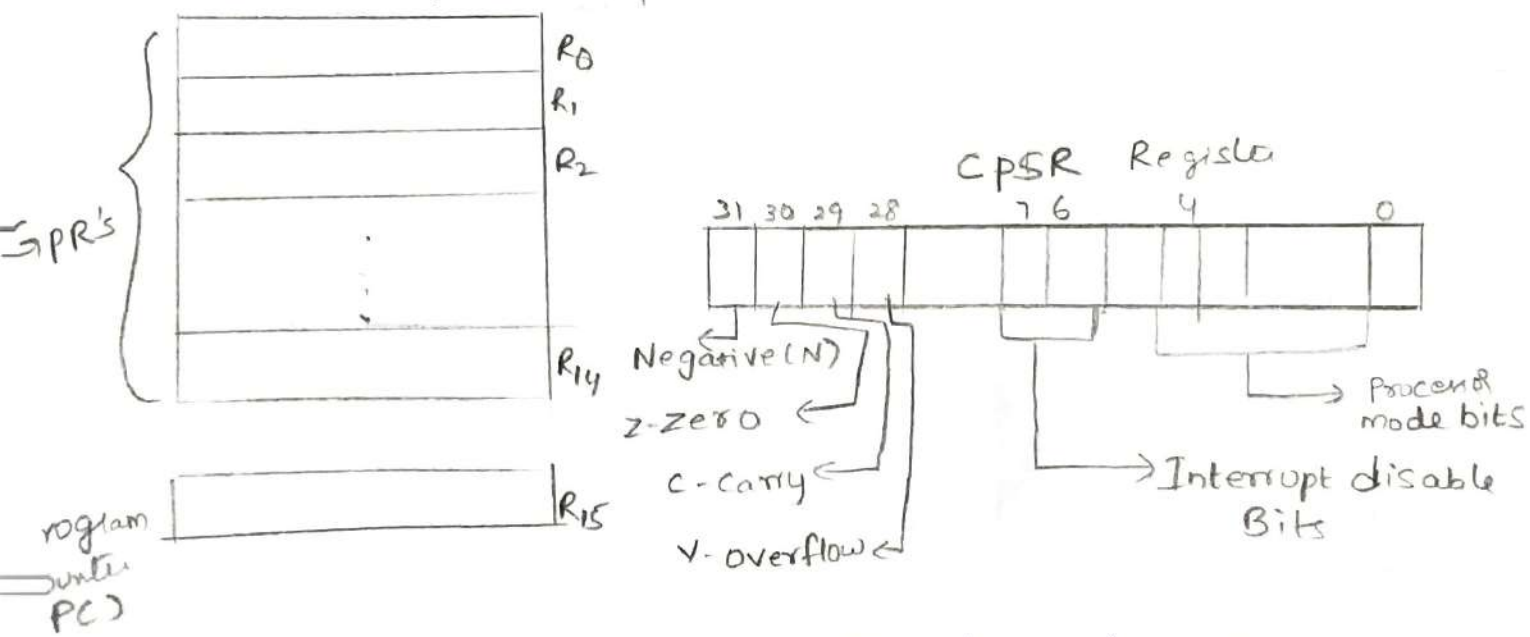


fig :- ARM Register Structure

* Arithmetic & Logic Instructions :-

The ARM instruction set has a number of instructions for Arithmetic & Logic operations.

Arithmetic Instructions :- There are different Operators they are: ADD, SUB, MUL, MLA

(a) **ADD :-** The basic Assembly language Syntax is given as

Syntax :- Opcode R_d, R_n, R_m

where Opcode specifies type of operation to be performed.

R_n, R_m → Source Operands

R_d → destination operands.

Ex :- (1) ADD R_0, R_2, R_4 ($\because R0 \leftarrow [R2] + [R4]$)

(2) ADD R0, R3, #17

means $R0 \leftarrow [R3] + 17$

(3) ADD R0, R1, R5, LSL #4

The above Example (3) is used when Shift (or) rotation instruction is required. The above Example(3) indicates that second operand R5, contents in R5 register is shifted Left 4' times bit positions, and then it is added to the contents of R1 register and last result, Sum is placed in Register R0.

(b) **SUB :-**

Syntax :- Opcode R_d, R_n, R_m

Ex :- SUB R0, R6, R5

Above Example indicates contents in R5 and R6 are Subtracted and result is place in Register R0.

i.e., $R0 \leftarrow [R6] - [R5]$

(c) MUL :- There are two Versions of a multiply instruction

(i) First Version :- In this it multiplies the Contents of two registers and places result in third register (destination register).

Syntax :- MUL R_d, R_n, R_m

Ex:- MUL R0, R1, R2

above Example indicates Contents of R2 & R1 are multiplied and Result is placed in R0 Register

i.e., $R0 \leftarrow [R1] \times [R2]$

(ii) Second Version :- In this fourth register is placed, whose Contents are added to product before Storing the result in the destination register.

Ex:- MLA R0, R1, R2, R3

Here MLA is called Multiply - Accumulate Operation it is used in numerical algorithms for digital Signal Processing

In the above Example

$R0 \leftarrow [R1] \times [R2] + [R3]$

* Logic Instructions :-

The logic operations are AND, OR, XOR and Bit-clear (BIC).

Syntax :- Opcode R_d, R_n, R_m

The above Syntax is Common for logic Operations.

AND :-

Ex:- AND R_0, R_0, R1

The above Example is a bitwise logical AND between the Operands in registers R0 & R1 and result is Saved in R0 Register

$R0 \leftarrow [R0] \wedge [R1]$

(b) OR :-

Ex :- OR R0, R1, R2

Here R0 ← [R1] ∨ [R2]

Bitwise OR operation will be Executed and result is Saved in R0 Register.

(c) XOR [Exclusive OR] :-

Ex :- XOR R0, R1, R2

In this Example R0 ← [R1] ⊕ [R2], and result is Saved in Register R0.

(d) BIC [Bit-clear] :-

This instruction is closely related to the 'AND' Instructions. It complements each bit in Operand Rm before 'AND'ing them with the bits in register Rn.

Syntax :- Opcode R_d, R_n, R_m

Ex :- BIC R0, R0, R1

The above Example indicates

(R0 ← [R0] ∧ (NOT [R1]))

Ex :- If R0 = 02FA62CA
R1 = 0000FFFF

NOT [R1] = R1 = FFFF0000

Now AND Operation R0 = 02FA62CA
R1 = FFFF0000

Result in R0 ← = 02FA0000

(e) Move Negative (MVN) :-

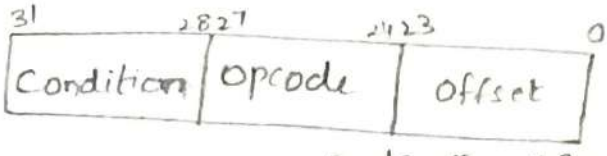
It Complements the bits of the Source Operand and places the result in R_d register.

Ex :- MVN R0, R3

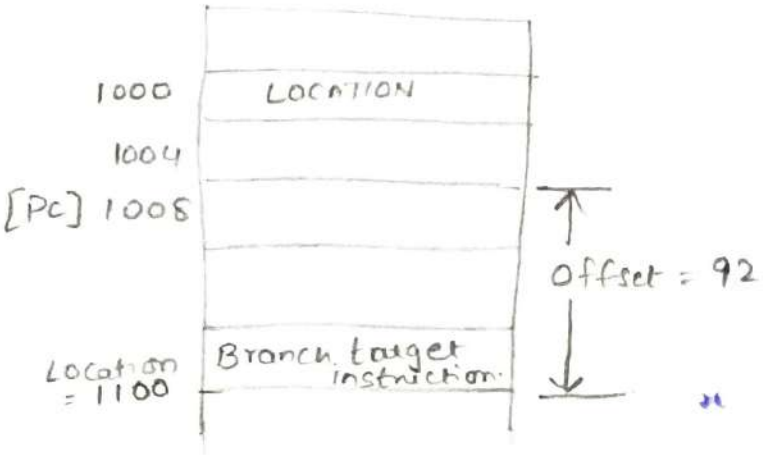
Suppose R3 = 0F0F0F0F, After MVN instruction
R3 = F0F0F0F0, it Complements bits and places the result in R0 Regiter as F0F0F0F0.

* Branch Instructions :-

The Conditional Branch Instructions Contain a signed 2's complement, 24 bit offset that is added to the updated contents of the program Counter (PC) to generate branch target address.



(a) Instruction Format



Branch Target Address
= (PC) + offset
= 1008 + 92
BTA = 1100

fig :- Determination Of a Branch target Address

In the above figure, Address Of PC Value is added to the offset Value then we get the Branch Target Address location, now procesd will Jumps to particular location depending on the condition given in the instruction format b_{28-31}.

ii, Setting Condition Codes :-

For this Compare instruction is given by

`cmp Rn, Rm`

it performs [Rn] − [Rm], have sole purpose of Setting the Conditions code flags based On the result of subtract Operations

→ On the other hand, Arithmetic & logic instructions effect the Condition code flags only if Explicity Specified to do so by a bit in Opcode field

Ex :-
```
ADDS  R0, R1, R2
ADD   R0, R1, R2
```
[Here 'S' suffix is added, where it effects condition code flag]
[It does not effect]

Whereas

* Addressing Modes of ARM processor :-

The way of specifying the address of the operand for a given instruction is called Addressing mode. For a ARM processd we use two instructions, they are (1) LOAD & (2) STORE, these are used to load data values from memory (or) store data values in memory.

Syntax for LOAD & store instructions are as follows :-

(1) LOAD dest register, Source_memory_address
(2) STORE soure_register, dest_memory_address

The addressing modes of ARm processor are as follows :-

(1) Immediate Addressing mode :-

In this, the value or) data of the operand is explicitly mentioned in the instruction itself.

Ex :- LDR #200, R1

(2) Direct. Addressing mode :-

In this, Address of the operand (or) memory location is mentioned in the instruction.

Ex :- LDR 2000, R1

(3) Register Addressing Mode :- The instruction which uses processor Registers to represent operands is the instruction in Register Addressing mode.

Ex :- LDR R1, R2

(4) Index Addressing Mode :-

In this Effective Address is obtained by adding a offset and content in the register that type of Addrening mode is Index Addrenin mode

In this we have three types of Index Addressing mode in ARM processor They are

(1) Pre indexed mode (2) Pre index with write back mode
(3) Post indexed mode

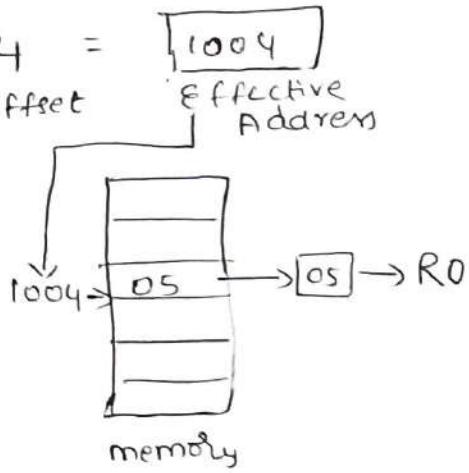
(i) Pre indexed mode :- The effective address of the operand is the sum of the contents of the base register Rn and an offset value.

Syntax :- LDR Rd, [Rn, # Offset]

Ex :- LDR R0, [R1, #4] [∵ R0 ⇐ [[R1] +4]

In the above Example, contents in R1 register is added with offset value, then we get effective address from where data is taken and Loaded (LDR) into R0 register.

Ex :- Suppose R1 [1000] + 4 = [1004]
offset Effective Address



Here the content in R1 register is unchanged after execution also then that mode is called Pre-indexed Mode.

(ii) Pre index with write back mode :-
The effective address of the Operand is generated in same way as pre indexed mode, and then effective address is written back into Rn.

Syntax :- LDR R_d, [Rn, # Offset] !

Ex :- LDR R0, [R1, #4] !

In the above Example [!] indicates write back, here after getting effective address, this address will be write back into R1 register

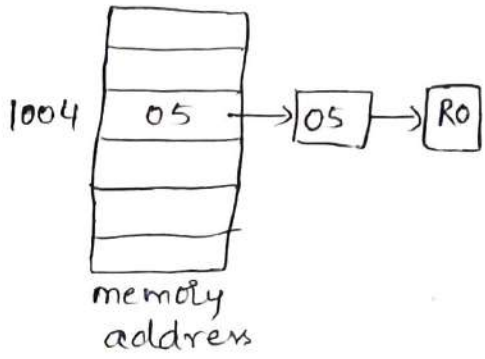
for Ex:- LDR R0, [R1, #4] !

In the above Example, Value in R1 Register is added with offset, then we get effective address, from where data is taken and loaded into R0 Register and at the same time Effective address is loaded in R1 Register i.e., Finally R1 content is changed with Effective addren

Suppose R1 [1000] +4 = [1004] Effective addren
offset

Now after Effective Address is obtained R1 register content is changed with Effective address so R1 [1004] register will have this addren.

∴ R1 register is changes to new location given by Effective addren.



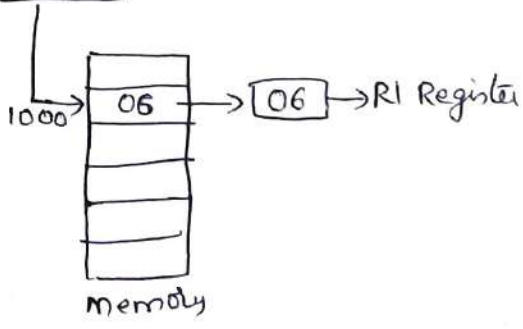
(iii) Post Indexed Addressing Mode :-

The Effective address of the operand is the contents of Rn. The offset is then added to this address and the result is written back into Rn.

Syntax :- LDR Rd, [Rn], #offset

Ex :- LDR R0, [R1], #4

Suppose R1 register has R1 [1000] = Effective Addren



The above Example shows Value in [R1] register is loaded into R0 register, then offset is added, that result is Saved in ~~Result~~ R1 register. i.e, Now R1 [1000] +4 = [1004] → R1 Register
New Value.

∴ R0 ← [R1]

After Loaded R1 = R1 +4
↑offset.

That means R1 changes to new location.

Register Indirect Addressing mode :-

A processor Register is used to hold the address of a memory location where the operand is placed, that type of instruction is called Register Indirect Addressing mode.

Ex:- MOV R1, [R2]

(6) Register Indirect with Scaling Addressing mode :-

Address of the memory operand is given by the sum of two registers, the first acts as a base register and the second is scaled by shifting left/right

(i) Pre Indexed Addressing mode :

Ex:- LDR R0, [R1, R2, LSL #2]

Above Example, Here that Content in R2 register is Logical Left shifted by 2 times, then the content in R2 register sum with content in R1, then we get Effective address, where the data (or) Value is loaded into R0 Register.

The content in R1 Register is not changed

R0 ⇐ data from memory location pointed by (R1+R2 LSL by 2)

(ii) Pre indexed with write back Addressing mode :-

Ex:- LDR R0, [R1, R2, LSL #2]!

In the above Example, content R2 is Logical left shifted by 2 times, the content in R2 register is added with Contents in R1 register, then we get Effective address from where data is taken and loaded into 'R0' Register. After that Effective Address is changed in R1 register.

∴ R1 register is having new value (i.e Effective address).

(iii) Post Index Addressing mode :

Ex:- LDR R0, [R1], R2, LSL #2

In the above Example Contents in R1 is loaded into R0 register later R2 contents is Left shifted 2 times, then value is added with Contents of R1, then final address is stored in R1 register. So R1 register Contents is changed with new value.

(7) Relative Addressing MODE :-

In this, Contents in Program Counter(PC) is added with offset, then Effective address from where data (or) Value is used, this type of Addressing mode is Called as Relative Addressing mode.

Syntax :- LDR R_d, [PC, #offset]

Ex :- LDR R_0, [PC, #04]

Effective Addrss (EA) ← [PC] + 04

The Value (or) data from EA is taken and loaded into R0 Register.

— × —

Parameter	Pre-Index	Pre Index with write back	Post Index
With Immediate Offset			
(1) Syntax	LDR R_d, [Rn, # Offset]	LDR R_d, [Rn, # Offset] !	LDR R_d, [Rn], # Offset
(2) Example	LDR R_0, [R1, # 4]	LDR R_0, [R1, #4] !	LDR R_0, [R1], #4
(3) Effective Addrem	EA = [R1] + 4	EA = [R1] + 4	E.A = [R1]
(4) Function:	R0 ⇐ Data from memory location of E.A Note:- R1 remains unchanged	R0 ⇐ Data from new memory location pointed by R1 R1 contents are changed with E.A.	R0 ⇐ data from new memory location Pointed by R1. After that R1 register has R1 = R1 + 04 R1 contents are changed.
With Scaling (or) magnitude			
(1) Syntax	LDR Rd, [Rn, Rm, Shift, count]	LDR Rd, [Rn, Rm, shift, count] !	LDR R_d, [Rn], Rm shift, count
(2) Example	LDR R_0, [R1, R2, LSL #02]	LDR R_0, [R1, R2, LSL, #02] !	LDR R0, [R1], R_2, LSL #02
(3) Function:	R0 ⇐ Data from memory location pointed by (R1 + R2 LS by 2) Here R1 content is Unchanged	First R1 ⇐ R1 + RS LSL by 2 then R0 ⇐ data from memory location pointed by R1 R1 value is changes	First R0 ⇐ [R1] then R1 ⇐ R1 + R2 LS by 2 Here also R1 value is changes
(4) Effective Addrem	E.A = [R1] + [R2 LSL by 2]	E.A = [R1] + [R2 LSL by 2]	E.A = [R1]

UNIT-3 I/O Organization

Contents :-

- Accessing I/O devices
- Interrupts
  - Interrupt hardware
  - Enabling and Disabling Interrupts
  - Handling Multiple Devices
- Direct Memory Access
- Buses
  - Synchronous Bus
  - Asynchronous Bus
- Interface Circuits
- Standard I/O Interface
  - Peripheral Component Interconnect (PCI) Bus
  - Universal Serial Bus (USB)

* Accessing I/O Devices :- The basic feature of a computer is its ability to Exchange data with other devices. The bus Enables all the device connected to it to Exchange information

A single bus structure is shown in below figure :

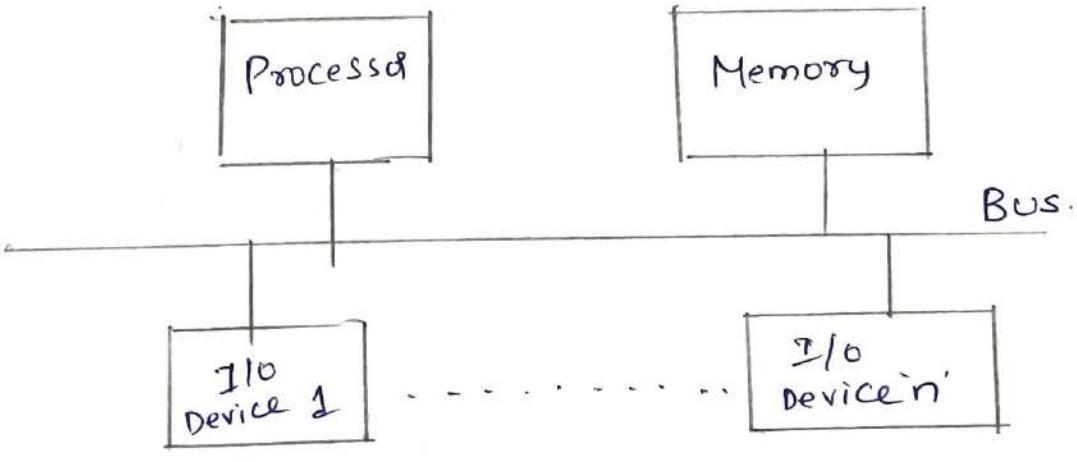


fig :- Single Bus Structure

The bus Consists of 3 Sets of lines

1. Address lines
2. Data lines
3. Control lines.

Each I/O device is assigned a unique set of addresses. When I/O devices and memory share the same address space, then it is called Memory mapped I/O.

I/O Interface For an input device :-

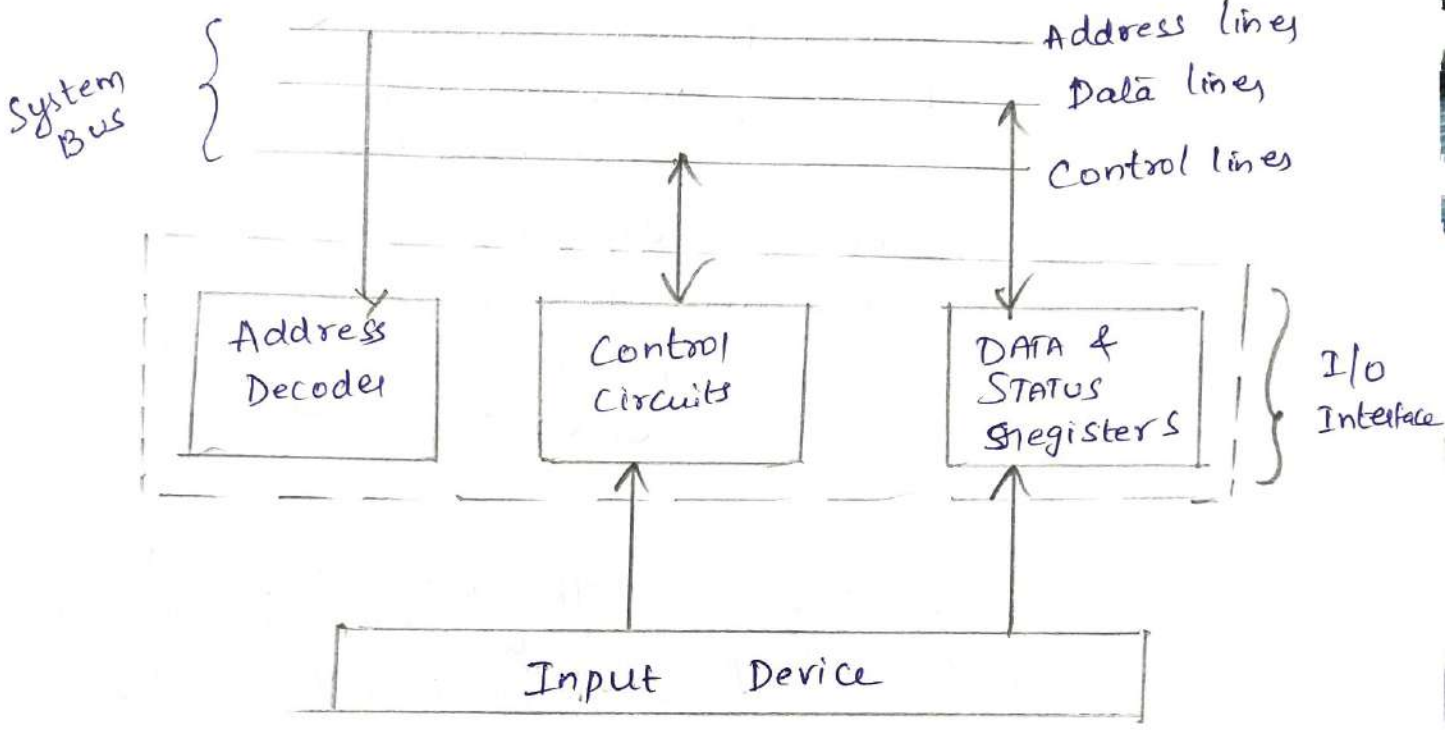


fig :- I/O Interface for an input device

In the I/O interface, we have 3 sections majorly, they are :

(1) Address decoder :- The address decoder Enables the device to recognize its address when this address appears on the address lines.

(2) Data Registers & status Register :-

The data Register holds the data being transferred to or from the processor. The status register Contains the information relevant to the Operation of the I/O device. Both data and status

register are connected to the data bus and assigned unique address.

Control Circuits :- The address decoder, data and status register and the control circuitry required to coordinate I/O transfer constitute the device's interface circuit.

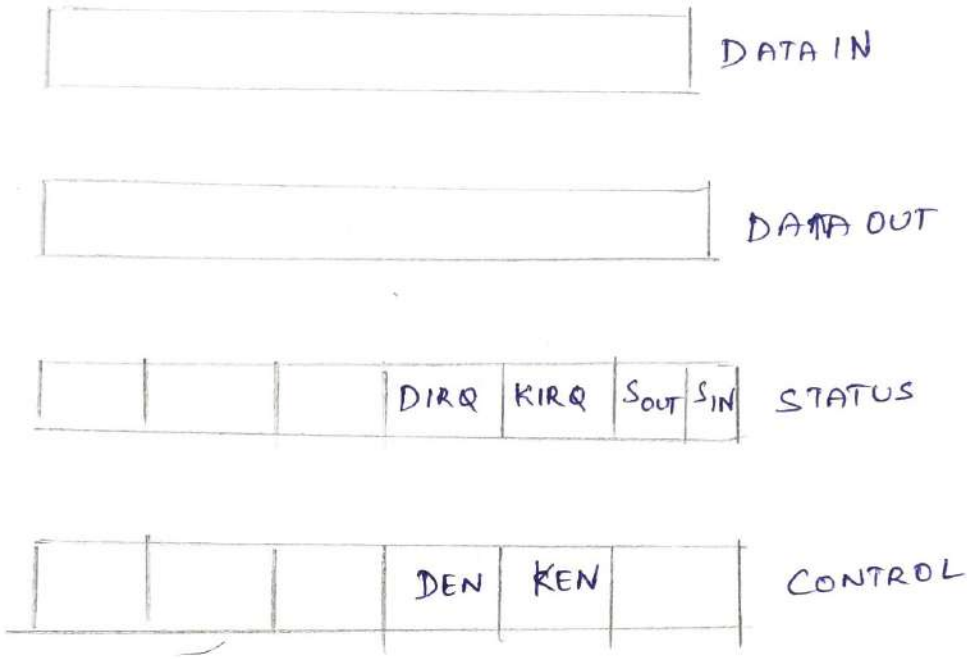


fig :- Registers in keyboard & display Interfaces

The four registers shown in above fogure are used in data transfer operations.

Status Register Contains two control flags S_{IN} and S_{OUT} which provide the status information for the Keyboard and display respectively. The two flags DIRQ (Display Interrupt Request) and KIRQ (Keyboard Interrupt Request) are used for Interrupts.

Data from keyboard are made available in DATAIN registe and data sent to display are stored in DATA OUT register.

Program:-

```
        MOVE    #line, R0       Initialize memory pointer
WAITK   TestBit #0, STATUS      TEST SIN
        Branch=0 WAITK          wait for character to be entered
        MOVE    DATAIN, R1      Read character
WAITD   TESTBIT #1, STATUS      TEST SOUT
        Branch=0 WAITD          wait for display to become Ready.
        Move    R1, DATAOUT     Send character to display
        move    R1, (R0)+       Store character & advance pointer
        Compare #0D, R1         Check if Carriage Return
        Branch≠0 WAITK          If not, get another character
        Move    #0A, DATAOUT    Otherwise send Line Feed.
        Call    PROCESS         Call a subroutine to process the input line.
```

This program reads a line of characters from keyboard and stores it in a memory buffer starting at location LINE. Then it calls subroutine PROCESS to procen input lines as the character is read it is echoed back to the display

Each character is checked to see if it is the carriage return (CR) character. If it is a line feed character it is sent to move the cursor one line down on the display and subroutine "PROCESS" is called. Otherwise the program looks back to wait for another character from the keyboard.

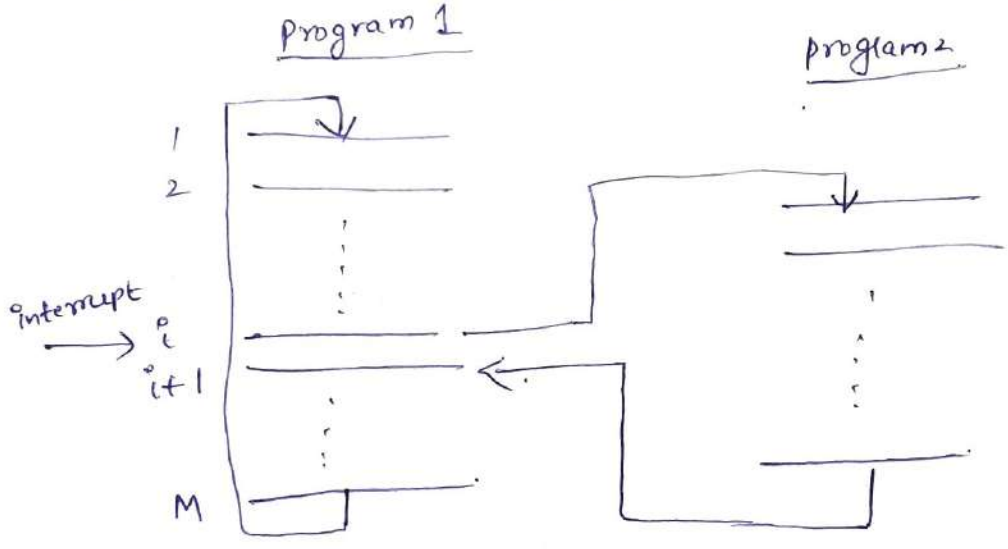
Interrupts :-

An interrupt is a signal to the processor Emitted by hardware (or) Software indicating an event that needs immediate attention.

An interrupt will stop the continous progress of an activity (or) process.

The bus control line, called Interrupt request line is used for interrupts. The interrupt resembles the Subroutine calls

Transfer Of control through the use of interrupts :-



→ when an interrupt occurs, Processor first completes the Execution of i^{th} instructions, then loads the PC with address Of First instruction of ISR.

→ Before going to Execute program(2), Processor saves the Contents of Program Counter & Status Register, saves the Register contents; in stack.

→ After that processor will execute ISR program i.e (program 2) after completion the processor come back to instruction i+1

→ Now processor reloads the PC, all other register contents back, start Execution from instruction i+1.

→ The Time taken for an interrupt request is received and the Start of Execution Of the ISR, this delay is called as Interrupt Latency.

(i) Interrupt Hardware :-

The I/O device requests an interrupt by activating a bus line called "Interrupt request". A single interrupt request line may be used to serve n-devices. An equivalent circuit for an open drain bus used to implement a common interrupt request line is shown in below figure.

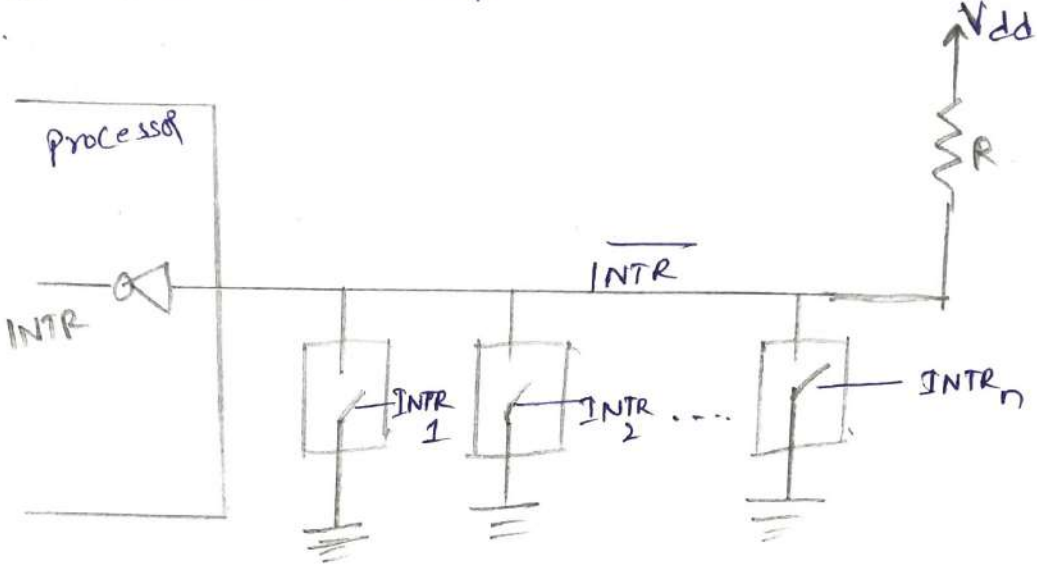


fig:- An equivalent circuit to implement common interrupt request line.

All devices are connected to the line via switches to ground.

To request an interrupt, a device ~~slot~~ closes its associated switch. Thus if all interrupt request signals $INTR_1$ to $INTR_n$ are inactive, i.e., if all switches are open, voltage on interrupt request line will be equal to V_{dd}. This is the inactive state of the line.

Since the closing of one (or) more lines (switches) will cause the line voltage to drop to '0', the value of $\overline{INTR}$ is logical OR of the requests from individual devices i.e.,

$$\overline{INTR} = INTR_1 + INTR_2 + \ldots\ldots + INTR_n$$

$\overline{INTR}$ signal is active, when it is in low voltage state.

(ii) Enabling and disabling Interrupts :-

The sequence of Events involved in handling Interrupt request from a single device are,

→ The device raises an Interrupt request

→ Processor interrupts the program Currently being Executed.

→ Interrupts are disabled by changing the control bit in PS (Processor Status register).

→ ~~processor~~ The device is informed that its request has been recognized & in response, it deactivates the INTR Signal.

→ The actions are enabled & Execution of the interrupted program is resumed.

(iii) Handling Multiple Devices :-

Let us now consider the situation where a number of devices capable of initiating interrupts are connected to the processor. When an Interrupt request is received it is necessary to identify the particular device that raised the request. If two devices raise the interrupt requests at the Same time, it must be possible to break the tie and select one of the two requests for service. The Simplest way to identify Interrupting device is to have the Interrupt Service routine poll all I/o devices in the System. There are three methods to handle, they are

(a) Vectored Interrupts

(b) Interrupt Nesting (or) Priority Interrupt.

(c) Simultaneous Requests (Or) DAISY chain priority

(a) Vectored Interrupts :-

A device requesting an interrupt may identify directly to the Processor. Then the processor can immediately start Executing the Corresponding ISR (Interupt Service Routine), this interrupt handling scheme is Called as Vectored Interrupts.

A commonly used scheme is to allocate permanently an area in the memory to hold the addresses of interrupt Service routines. These addresses are referred as Interrupt Vectors, and they are said to constitute the Interrupt - Vector table.

(b) Priority Interrupt :-

When two or more interrupt requests are arrived Simultaneous to processor, it has some means of deciding which requests to Service first. This can be Solved using the priority Interrupts.

A priority Interrupt is a System that Establishes a Priority over the Various Sources to determine which is to be Serviced first, when two or more requests arrive Simultaneously

Here Each interrupt request line is assigned a different Priority level.

Interrupt requests received over these lines are Sent to a priority arbitration Circuit in the processor as Shown in below figure.

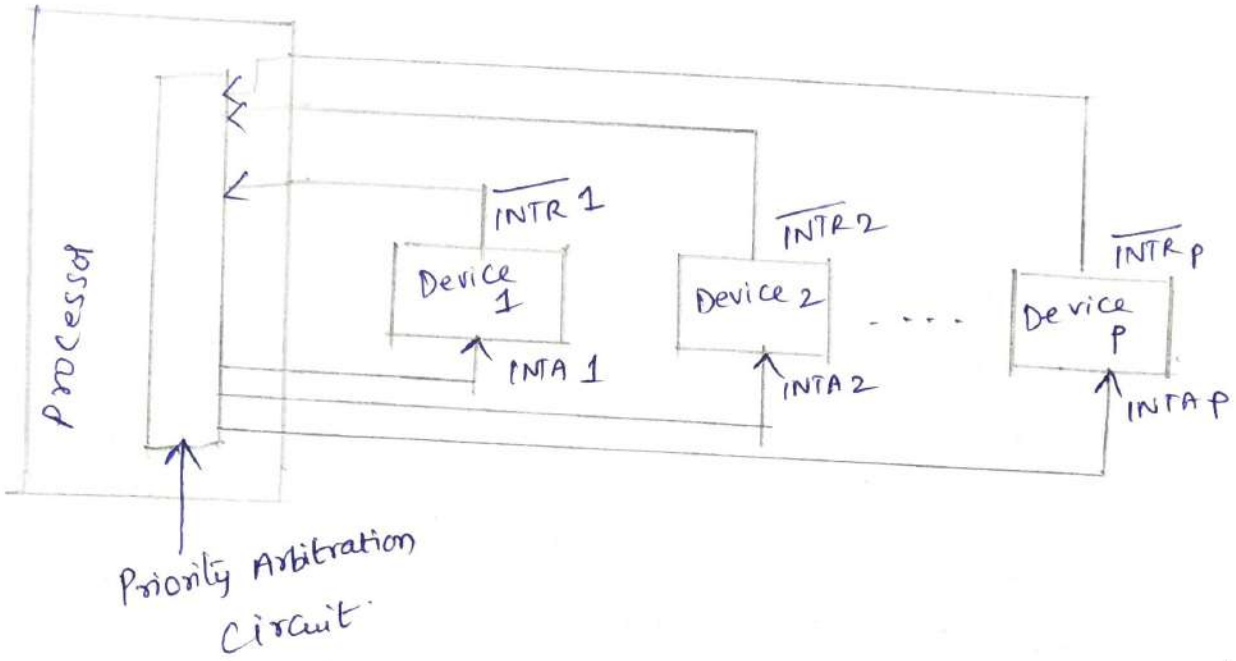


Fig:- Interrupt priority using Individual Interrupt request & Acknowledgement lines

A request is accepted only if it has a higher priority level than that currently assigned to the processor.

(ii) DAISY CHAIN PRIORITY :-

The daisy chaining method of Establishing Priority consists of a serial communication of all devices that request an interrupt.

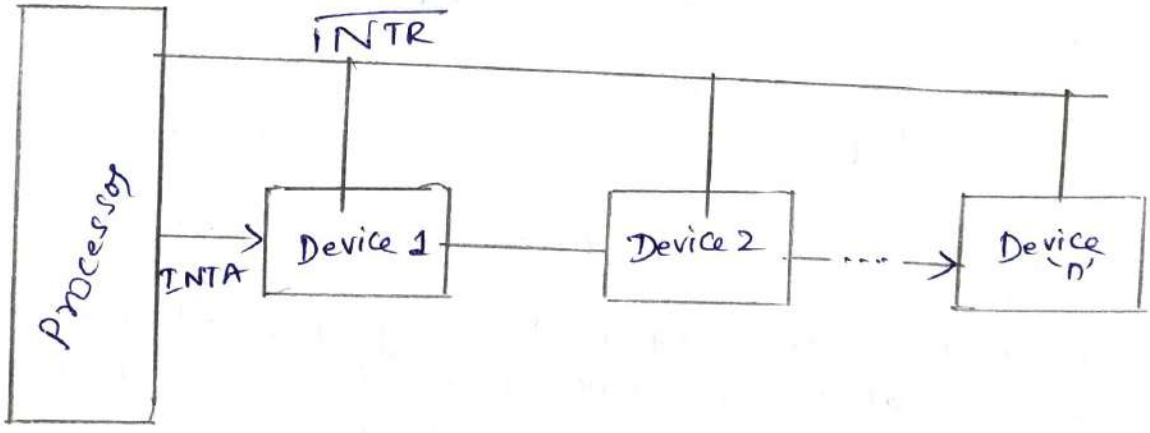


fig :- DAISY CHAIN

The device with highest priority is placed in the first posit followed by lower priority devices upto device with lowest priority which is placed in the last chain.

The Interrupt request line $\overline{INTR}$ is common to all devices. The Interrupt ACKnowledgement line INTA, is connected in daisy Chain Fashion, such that INTA propagates serially through all devices.

When several devices raise interrupt request and $\overline{INTR}$ is activated, the processor responds by setting the INTA line to 1. The signal is received by device 1 which passes on to device 2 only if it does not require any service.

If device 1 has pending requests for interrupt it blocks INTA signal and proceeds to put its identifying code on the data lines.

Therefore in daisy chain the device which is electrically closer to processor has highest priority.

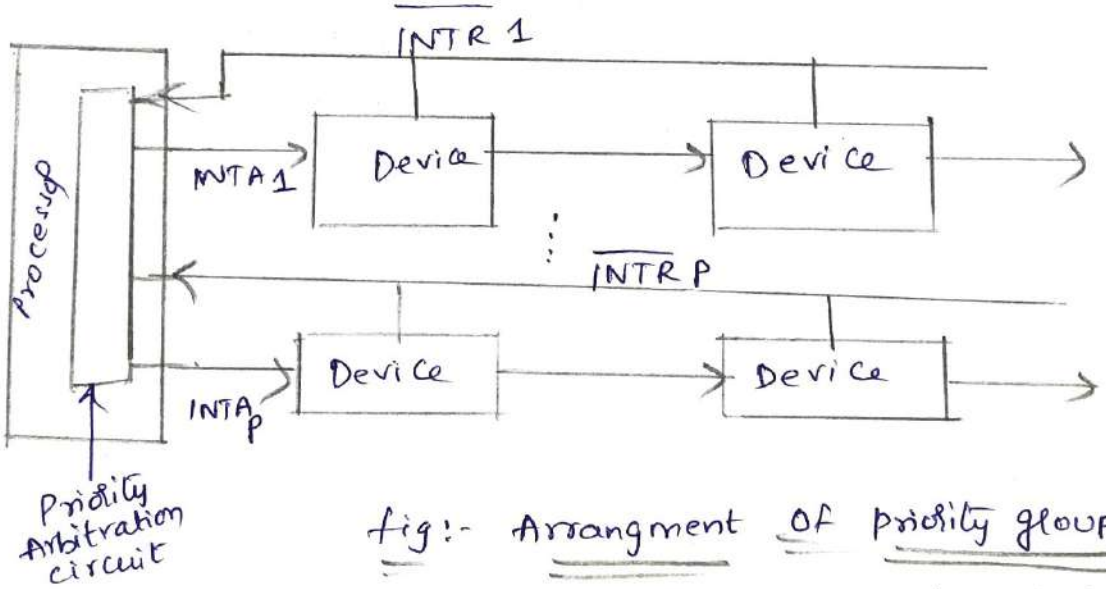


fig:- Arrangment of priority groups.

Here devices are organized in groups, and each group is connected at a different priority level. within a group, devices are connected in a daisy chain. This Organization is used in many computer systems.

* Direct Memory Access :-

A special control unit provided to allow transfer of a block of data directly between an External device and the main Memory, without continous intervention by the processor. This approach is called Direct Memory Access (or) DMA. DMA transfers are performed by a control unit (or) circuit called the "DMA controller".

To transfer of a block of words, the processor sends,

→ starting address
→ Number of words in the block
→ Direction of transfer.

When a block of data is transferred, the DMA controller increment the memory address for successive words, and number of words in the block is decremented. After DMA transfer is completed, the processor returns back to its Program.

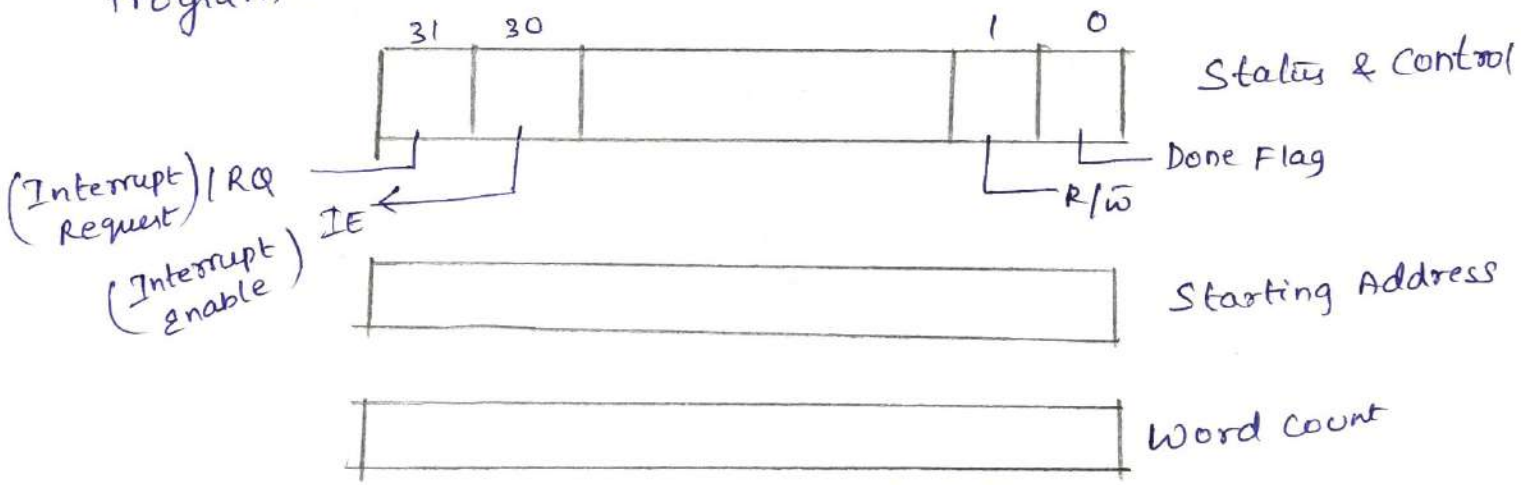


fig :- Registers in DMA Interface

R/$\bar{w}$ ⇒ 1, DMA Controller read data from memory to I/o device
R/$\bar{w}$ = 0, DMA Controller Write Operation
Done Flag = 1, Controller has Completed transferring a block of data and is ready to receive another command.

IE = 1, it causes the controller to raise an interrupt after it has completed transferring block of data

IRQ = 1, it indicates that the Controller has requested an interrupt.

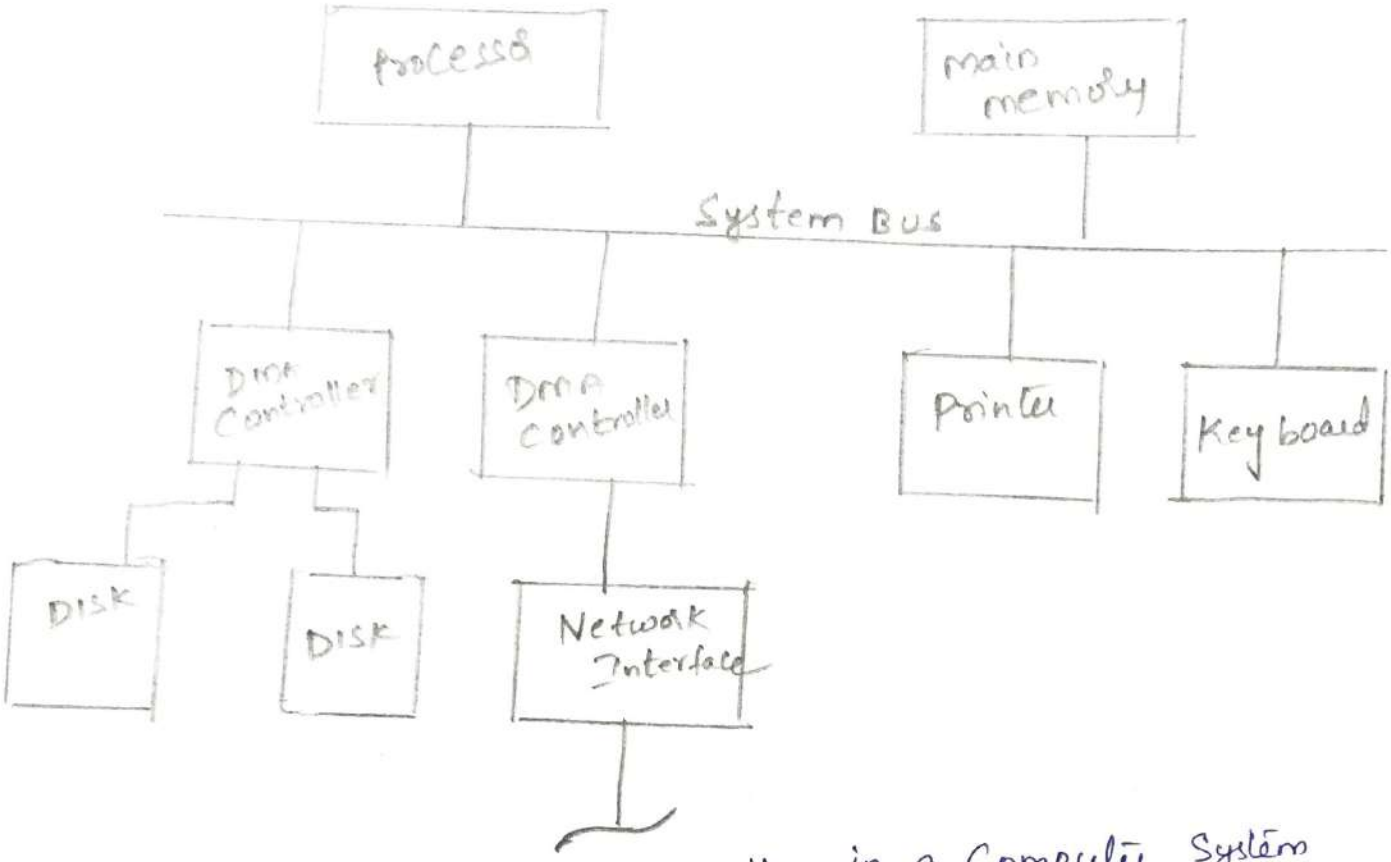


Fig:- DMA Controller in a Computer System

A DMA Controller Connects a high speed network to the Computer bus. Disk Controller, which controls two disks,

→ To start a DMA transfer of a block of data from main memory to one of the disks, a program writes the address and word count information into registers of Corresponding channel of disk Controller.

→ When DMA transfer is Completed, then Status and Control registers of DMA Channel (i.e)., Done Flag bit = 1 and IRQ = 1 and IE = 1.

→ Data transfer between memory and I/O devices using DMA Can be done by accessing the System bus from process so DMAC requesting access of system bus, and get system bus Control from cpu. During DMA transfer cpu can perform Only those operations which do not require System bus.

& Whenever control of System is given to DMAC then do not give it for a longer time to cpu.

Three types of modes of Transfer for DMA; they are

1. Burst mode
2. Cycle Stealing
3. Interleaving DMA.

(1) Burst Mode :- In this mode Entire data (or) burst of blocks is transferred between I/o devices and memory, without interruption. This mode is called Burst mode.
After data transfer, Bus Control is given back to cpu by DMA.

(2) Cycle Stealing :- One word is ready, cpu gives the Control of System Bus to DMAC for 1 cycle in which it will transfer data to memory. During this time cpu keeps control of the buses. DMAC 'steal' memory cycles from cpu, hence interweaving technique is called as cycle stealing.

(3) Interleaving DMA :- Whenever cpu does not require System Bus (doing internal work) then only Control of buses will be given to DMAC.

* Bus Arbitration :-

The device that is allowed to initiate data transfer on the bus at given time is called the "Bus Master". Bus arbitration is the process by which next device to become the bus master is selected and bus mastership is transferred to it.

There are two types of bus Arbitration. They are

(1) Centralized bus Arbitration :- A Single bus arbiter performs required Arbitration.

(2) Distributed Bus Arbitration :- All devices participate in the selection of next bus master.

* Centralized Bus Arbitration :- In this processor is a single arbiter performs required arbitration, below fig shows

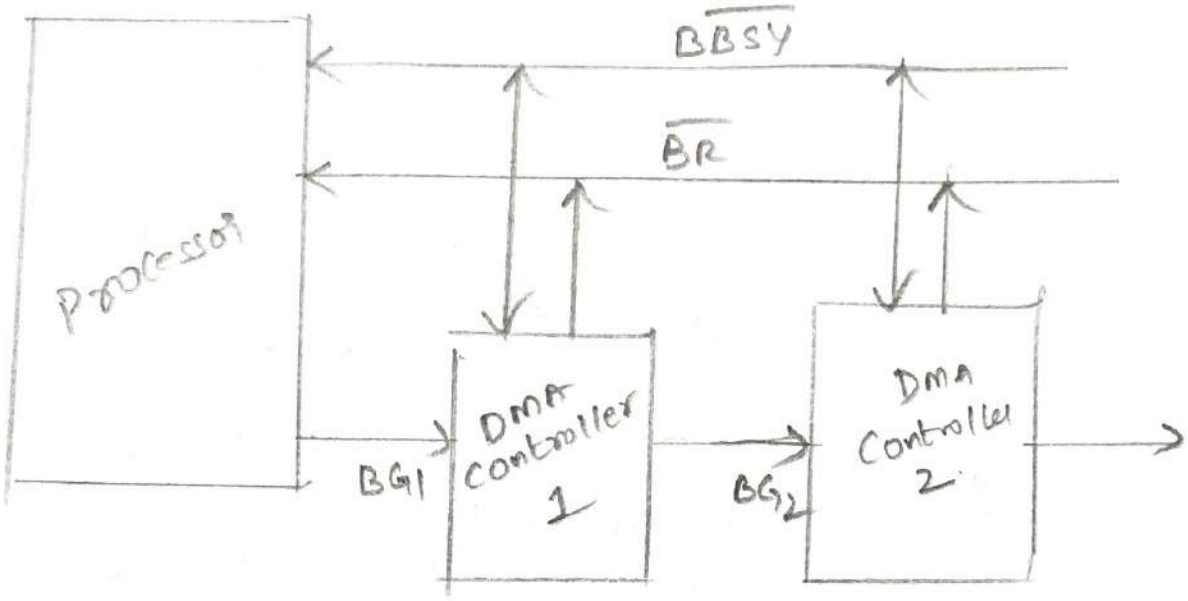


fig :- Centralized Bus Arbitration using a DAISYCHAIN

A DMA controller indicates that it needs to become bus master by activating bus request line $\overline{BR}$, when bus request is activated, Processor activates the Bus Grant signal BG1 to DMA controllers, they can use the bus when it becomes free. This Bus Grant BG Signal is Connected to all DMA controllers using a daisy chain arrangement. Thus if DMA Controller 1 is requesting the bus, it blocks the propogation of the grant signal to other devices. Otherwise it passes the grant downstream by asserting BG2 as shown in above fig. The Current bus master indicates all devices that it is using the bus by activating the signal called bus busy line ($\overline{BBSY}$) signal.

* Distributed Arbitration :- In this All devices waiting to Use the bus have Equal responsibility in carrying out the arbitration process without a Central arbiter, as shown in below figure.

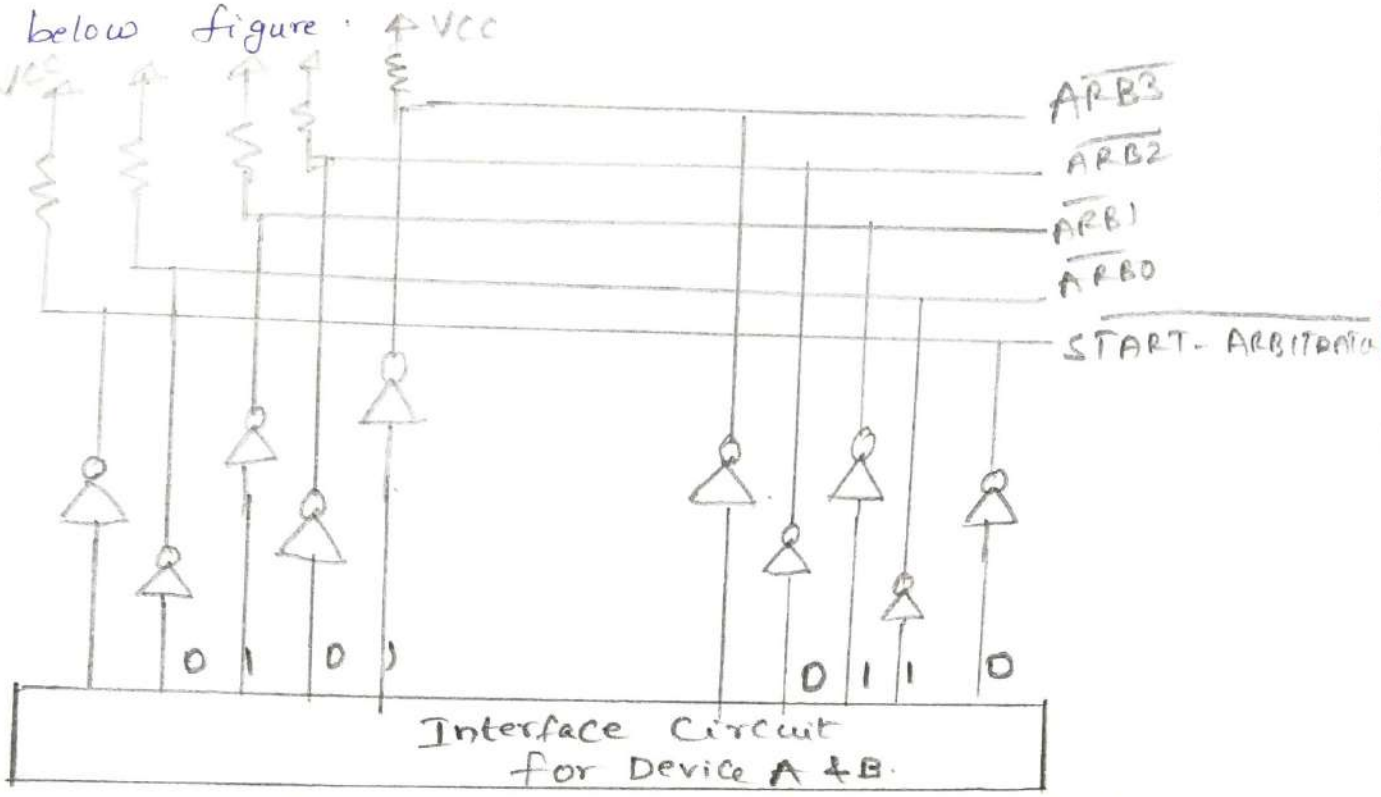


fig :- A Distributed Bus ARBITRATION

Each device on the bus is assigned a 4 bit bus identification number. When one or more devices request the bus, they assert the Start Arbitration signal and Place their four bit id numbers on the four lines $\overline{ARB0}$-$\overline{ARB3}$

→ Let us Assume Device A & B having ID numbers as 5 & 6 respectively are requesting to use of system bus. 'A' transmits 0101 (5) and 'B' transmits 0110 (6), Connection Perform logical 'OR' between them and resultant pattern is '0111' Each device Compares the arbitration lines with ID from MSB If it detects any difference it disables driver at that position and for all lower order bits. In this device 'B' wins the contention.

Decentralization arbitration has advantage of offering higher reliability.

* Buses :- The wire which is used to connect computer components internally and transfer data between them, this is called as a "BUS".

The processor, main memory and I/O devices are interconnected by means of a BUS. It provides communication path for transfer of data.

A Bus protocol is the set of rules that govern the behaviour of various devices connected to the bus, as to when to place information on the bus, when to assert control signals, etc

Bus lines may be grouped into three types : They are

(1) Data Bus (2) Address Bus (3) Control Bus

(1) Data Bus :- The Bus which carries data information is called Data Bus.

(2) Address Bus :- The Bus which carries Address information is called Address BUS

(3) Control Bus :- The Bus which carries control signals and whether it is a read or write operation is specified by control Bus.

During the data transfer operation, one device plays the role of "Master", and another is "Slave".

Master (or) initiator :- Device which initiates data transfer by issuing Read/write command on the bus.

Slave (or) Target :- The device addressed by master is called as slave or Target.

Timing Information is indicate when the processor & I/O devices may place data (or) receive data from the Bus.

There are basically two types of Bus. They are

(1) Synchronous Bus

(2) Asynchronous Bus

* Synchronous Bus :-

In synchronous Buses, the steps of data transfer takes place at fixed clock cycles.

→ Everything is synchronized to bus clock. Bus clock is a square wave signal. The clock signals are made available to both master and slave. A transfer may take multiple bus cycles depending on the speed parameters of the bus and two ends of the transfer.

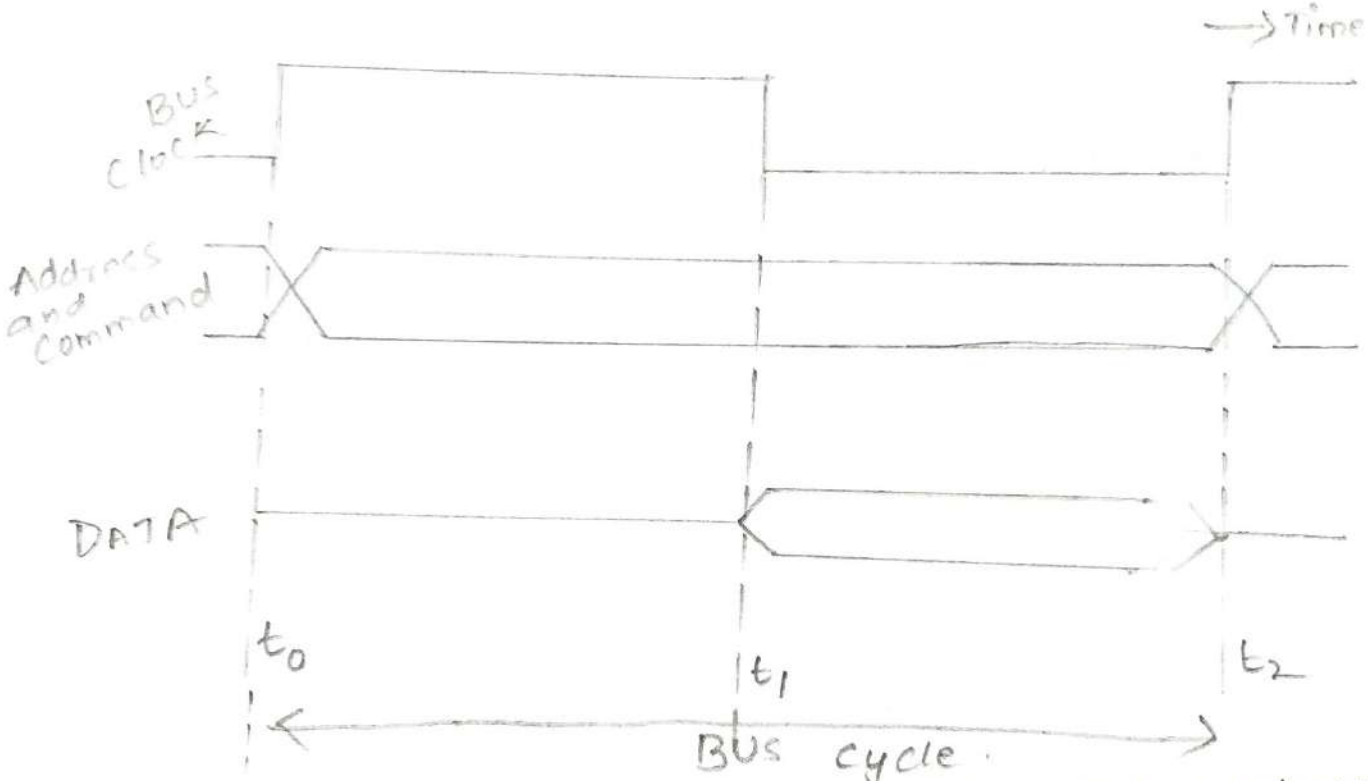


Fig :- Timing of an input transfer on a synchronous bus

The above diagram shows the timing of an input transfer of a synchronous bus. The "crossing points" indicates the time at which the patterns change.

A single line in an indeterminate / high impedance state is represented by an intermediate half way between the low to high signals levels.

→ In this case, the command (or) control signal indicates an input operation (Read) and length of operation to be read.

When Bus clock is applied at time 't0', then address and command lines places Address information and command on control lines.

Now all devices try to decode address and control signals and the corresponding device (slave) can respond at time 't_1'. At time t_1 - Addressed Slave places the data on data lines as shown in timing diagram. At the End of time 't_2', master "Strobes" data into its input buffer. STROBE means capture the value of the data at given instant time and store into buffer. Thus the time $t_0 - t_1 - t_2$ completely called as "one Bus cycle"

* Asynchronous Bus :-

In this data transfer on the bus is based on the use of "Handshake" between master & Slave. The common clock is replaced by two timing control lines. They are :-

(1) Master ready signal
(2) Slave ready signal.

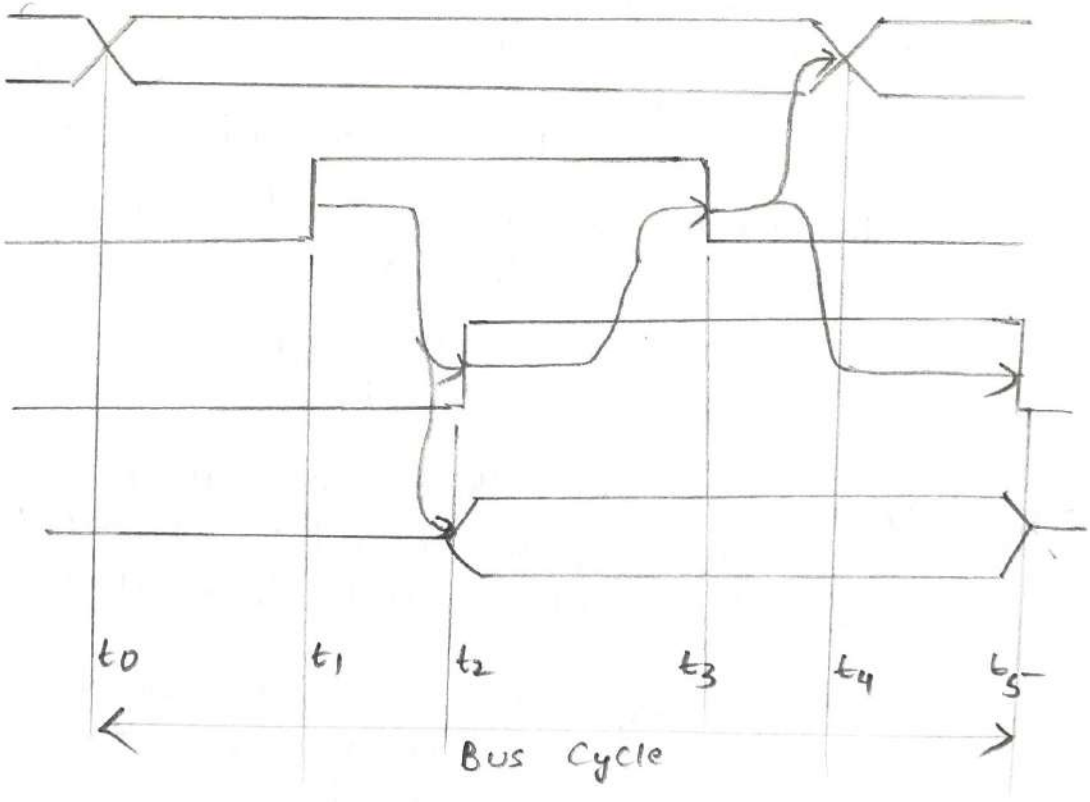


fig :- Handshake control of data transfer during an input operation

The handshake protocol proceed as follows :-

At t0 → Master places address and Command information on the bus and all devices on bus begin to decode information

At t1 → The master sets master Ready Signal line to '1' to inform the I/O devices that the address and Command information is ready.
The signal asserted at 't1' instead of 't0' to allow "Bus skew". Bus skew occurs when two signals transmitted Simultaneously reach the destination at different times. This may occur because different bus lines may have different speeds.

At t2 → Addressed Slave places the data on the bus and asserts the "Slave ready signal".
The period t2 - t1 depends on the distance between the master and slave and delay by slave's circuitry.

At t3 → Slave ready signal arrives at the master, indicates input data are available on the bus.
The master should wait for maximum bus skew plus the Setup time of its input buffer and then strobes the data. It also deactivates the master ready signal to indicates that it has received the data.

At t4 → master removes the address and command information from the bus.

t4-t3 → Allows for bus skew. once master-ready signal is set to '0', it should reach all the devices before the address and Command information is removed from Bus.

At t5 → slave receives the transition of master ready Signal from 1 to 0. It removes data and slave-ready signal from the bus.

A change of state in one signal by a change in other signal is called as "FULL Handshake". It has high Flexibility and Reliability.

* Interface Circuits :-

An I/O interface consists of the circuitry required to connect an I/O device to a compu Bus. On one side of the interface, we have the bus signals for address, data and control. On the other side we have a data path with its associated controls to transfer data between the interface and the I/O devices so I/O device side of the interface is called as a "PORT" This port is classified as two types:

(1) Parallel port

(2) Serial port

Parallel port transfers data in the form of number of bits, normally 8 or 16 to or from the device.

Serial port transfers and Receives data one bit at a time The Conversion from parallel to serial port Vice versa takes place inside the interface circuit.

→ Parallel port use a multiple pin connector between the device and the processor.

A Cable with as many wires as the number of bits transferred Simultaneously hence it is suitable for devices that are physically close to the Computer.

A Serial port uses a single pin connector between the device and the processor hence it is useful for devices that are at a longer distance.

*. PARALLEL PORT :-

In parallel port all bits are transferred at a time. Here a Keyboard (Input device), is connected to a Interface Circuit which is connected to a 32 bit Processor that uses a asynchronous bus protocol. Processor address I/O devices using memory mapped I/O.

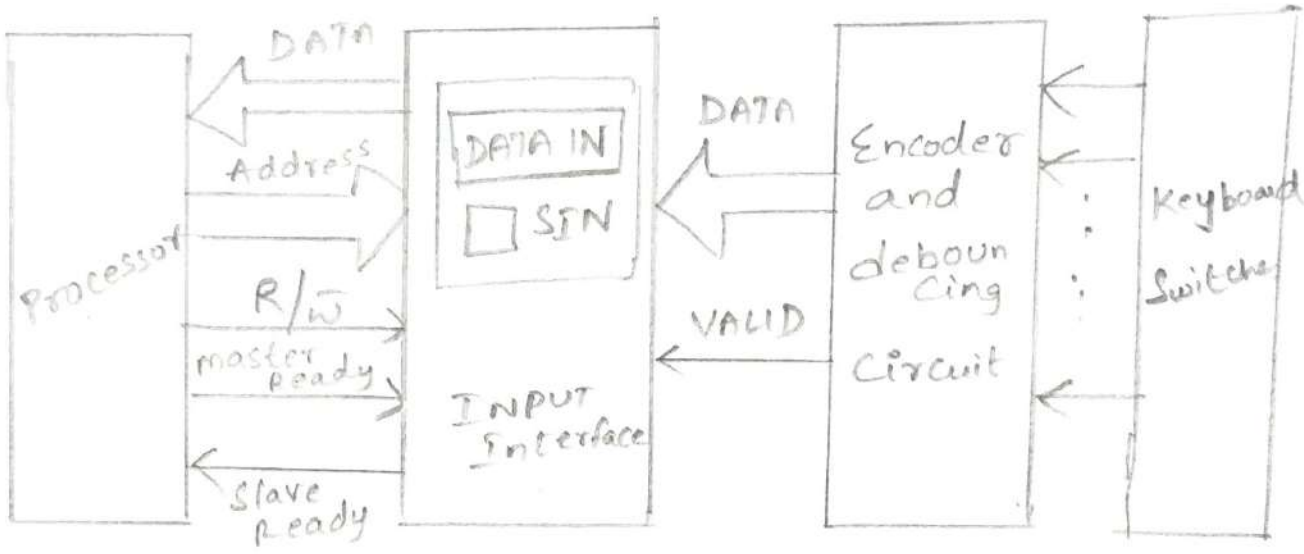


Fig :- Keyboard to Processor Connection

The above diagram shows a keyboard to processor Connection.

When a key is pressed, a signal is produced and given to the Encoder & debouncing circuit, where Encoder circuit generate the ASCII code for corresponding character and if any bouncing are present, they can be Eliminated by using debounicing circuit.

Then the data sent to Input Interface Circuit, which contain a data register, DATAIN, and a status flag SIN so when data is placed in DATAIN, automatically SIN is set to 1.

As the processor is connected in asynchronous bus protocol so first processor place Address, then mode of operation Control Signal (Read) is placed and ~~given to~~ Master-ready Signal is also active and given to Interface Circuit

→ As data is placed in DATA IN register, so it sends Slave ready signal active, Later data is placed on Data lines and given to the processor. Thus handshake control is activated. so master-ready and Slave-ready are the handshake control lines on the processor bus side

* Serial port :-

In Serial port the data transmission is one bit at a time. The key feature of Interface Circuit for a Serial port is that it is capable of Communicating in a bit-serial fashion on device side and bit-parallel fashion on bus side.

→ The Conversion from Parallel & Serial formats can achieved with Shift Registers as shown in below figure.

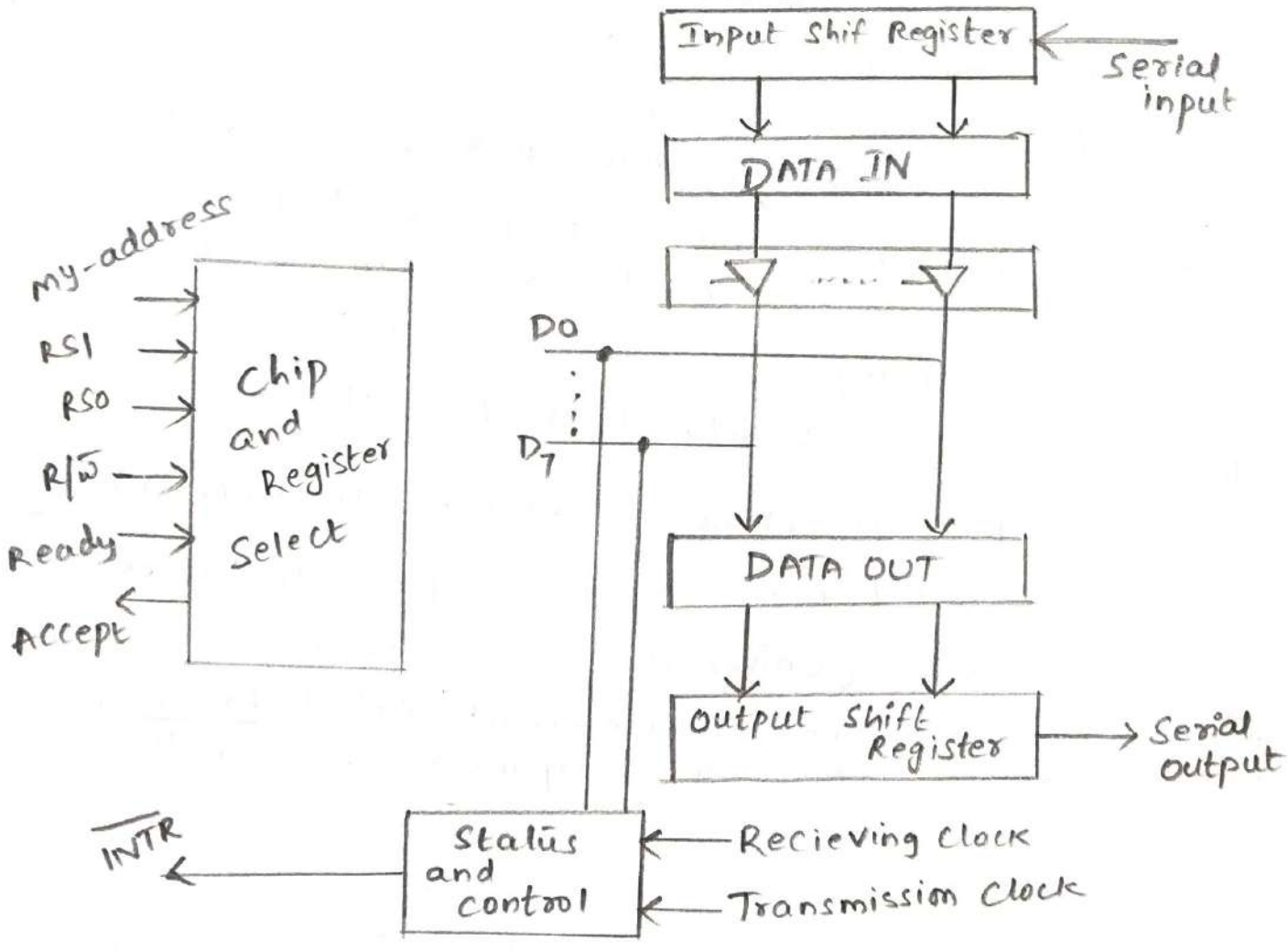


fig :- A Serial Interface

In the above block diagram of Serial Interface, input shift register accepts bit serial input from Input/Output device.

→ When all 8 bits of data have been received, contents of this shift register are loaded into Parallel into the DATAIN register and set SIN flag bit as '1'.

→ From where 8 bit data is placed on Data lines of D_0-D_7.

→ Similarly output data in the DATAOUT register is loaded into output shift register, from which the bits are shifted out and sent to the I/O device.

→ .

* STANDARD I/O Interfaces :-

Input/output device is connected to a computer using an interface circuit.

→ I/O devices fitted with an interface circuit suitable for one computer may not be usable with other computer.

→ So, a different interface may have to be designed for every combination of I/O device and computer, resulting in many different interfaces.

→ There are Three widely used bus standards :-

(1) PCI (Peripheral Component Interconnect)

(2) SCSI (Small Computer System Interfaces)

(3) USB (Universal Serial Bus).

Two Buses are interconnected by a circuit called Bridge.

→ PCI Standard defines an Expansion bus on the motherboard.

→ SCSI & USB are used for connecting additional devices, both inside & outside the computer box.

* PCI [Peripheral Component Interconnect] Bus :-

PCI is introduced in 1992.

→ PCI is developed as a low cost bus that is truly Processor independent.

→ It supports high speed disk, graphics and video devices

→ PCI has plug and play capability for connecting I/O devices.

DATA TRANSFER :-

The data transfer involve a burst of data rather than just one word. PCI supports burst mode of operation. PCI supports read/write operation.

→ PCI has three Address spaces. They are :

(a) memory address space

(b) I/O address space

(c) Configuration Address space.

I/O Address space is intended for use with processor. Configuration Space is intended to give PCI, its plug and play capability.

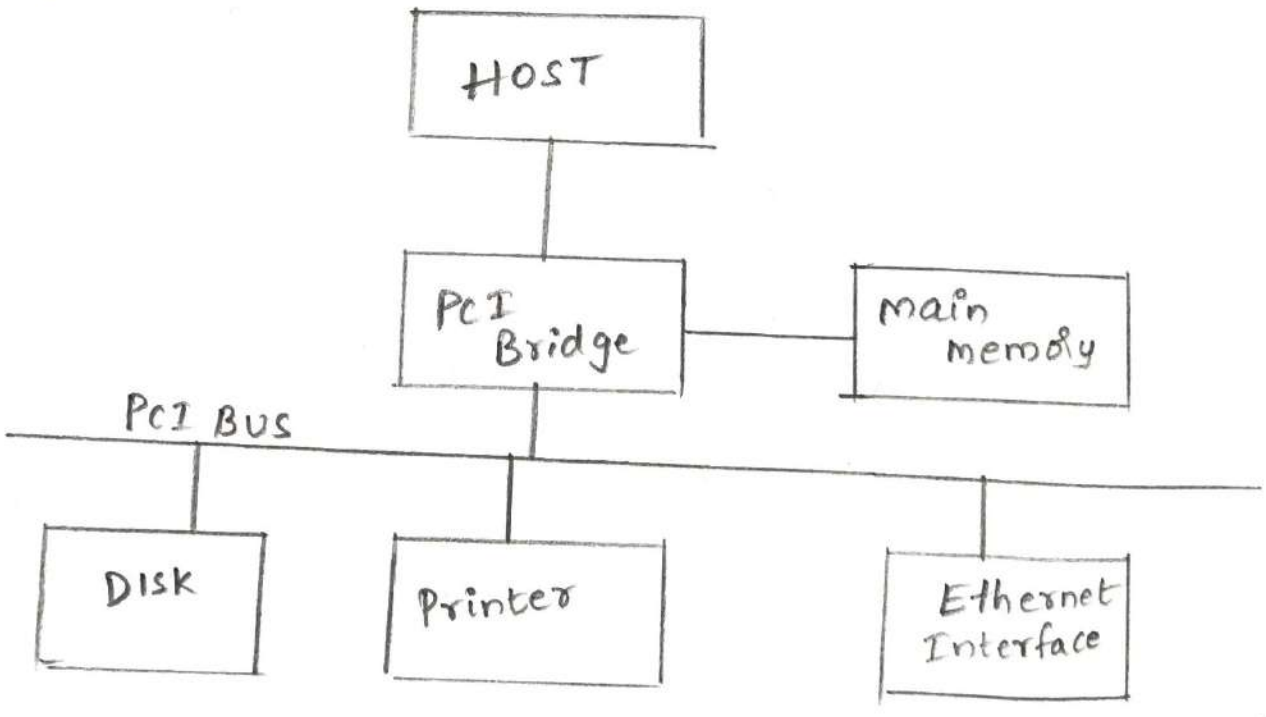


fig :- Use of a PCI bus in a Computer System

The above figure shows PCI bus in computer system. The PCI bridge provides a separate physical connection to main memory. Bridge, which translates the signals and protocols of one bus into another.

→ At any time, only one device will acts as a bus master (or) initiator, always processor (or) DMA will act as initiator.

→ Master maintains the address information on the bus until data transfer is completed

→ The addressed device that responds to read & write Commands is called a target. So a complete transfer operation on the bus, involving an address and brust of data is called a "transaction"

* Data Transfer Signals on PCI Bus :

1. CLK :- PCI bus is a synchronous bus. The operating clock frequency is 33 MHz or 66 MHz

2. FRAME # :- Sent by the initiator to indicate the duration of data transfer.

3. AD :- It is a 32-bit Address and data bus. It can extent to 64 bit

4. C/BE # :- 4 Command / Byte Enable signals (8 for a 64 bit bus)

5. IRDY#, TRDY # :- Initiator ready signal and Target ready signals.

6. DEVSEL # :- A response from device indicating that is has recognized its address and it is ready for a data transfer transaction.

7. IDSEL # :- Initialization Device Select.

Timing diagram of a read Operation on the PCI bus :-

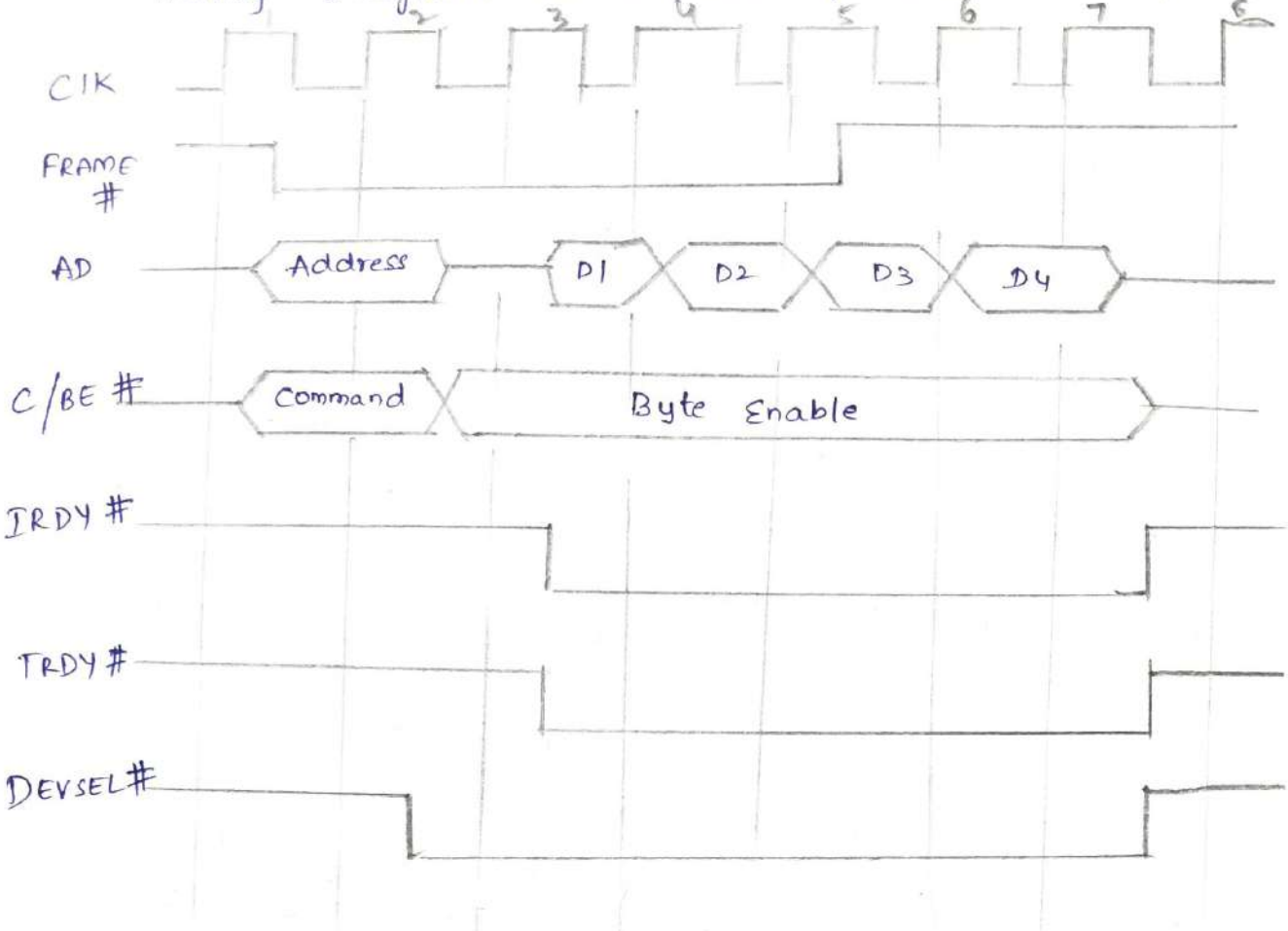


fig :- A Read Operation on the PCI bus

All signal transitions are triggered by the rising Edge of the clock.

Clock cycle 1 :- processor asserts FRAME #, indicate beginning and address on AD lines and a command on C/BE # lines.

clock cycle 2 :- processor disconnect (or) removes address from AD lines, and selected target enables on AD lines, and fetches data to place on bus and DEVSEL # asserts until End Of transaction.

Clock Cycle 3 :- IRDY # asserts signal, indicate it is ready to receive data.

TRDY # asserts signal, and sends word of data.

Clock cycles 4 to 6 :- The target sends three more words of data during clock cycles 4 to 6.

Clock cycle 7 :- The target disconnects its drivers and negates DEVSEL # at the beginning of Clock cycle 7.

* SCSI Bus :- [Small Computer System Interface]

The SCSI bus defined by the ANSI (American National Standards Institute), under the designation

→ A SCSI bus may have 8 data lines, in which case it is called a narrow bus and transfers data one byte at a time.

→ A wide SCSI bus has 16 data lines and transfers data 16 bits at a time.

→ The SCSI bus standard has undergone many revisions, and its data transfer capability has increased very rapidly, almost doubling every two years.

→ SCSI-2, SCSI-3 have been defined, and each has several options.

→ The different SCSI bus signals are :

- DB (Data lines)
- DB (P) [Parity bit for data lines)
- BSY - (Busy)
- SEL (selection)
- C/D (control / Data)
- MSG (message)
- REQ (Request)
- ACK (Acknowledge)
- I/O (Input / output)
- ATN (Attention)
- RST (Reset)

* USB (Universal Serial Bus) :-

A simple, low cost mechanism to Connect the devices to the Computer is possible using USB

USB Supports three speeds of operation They are

(1) Low speed (1.5 mb/s)
(2) Full speed (12 mb/s)
(3) High speed (480 mb/s)

The USB has been designed to meet the Key objectives. They are:

(1) It provide a simple, low cost & Easy to use interconnection structure that overcomes the difficulties due to limited number of I/o ports available on a Computer.

(2) It accommodate a wide range of data transfer Characteristics for I/o devices including telephone & Internet Connections.

(3) Enchance User Convenience through a "plug and play" mode of operation.

Device Characteristics :- The Devices that may be Connected to a Computer cover a wide range of functionality. The plug and play feature means that a new device can be connected at any time while the System is operating. The system should detect Existence of new device automatically and any other facilities needed to Service that device.

*. USB Architecture :-

To accommodate a large number of devices that Can be added or removed at any time, the USB has the tree structure Shown in below figure. Each node of tree has a hub device called a hub Hub acts as an intermediate control point between the host and the I/o devices.

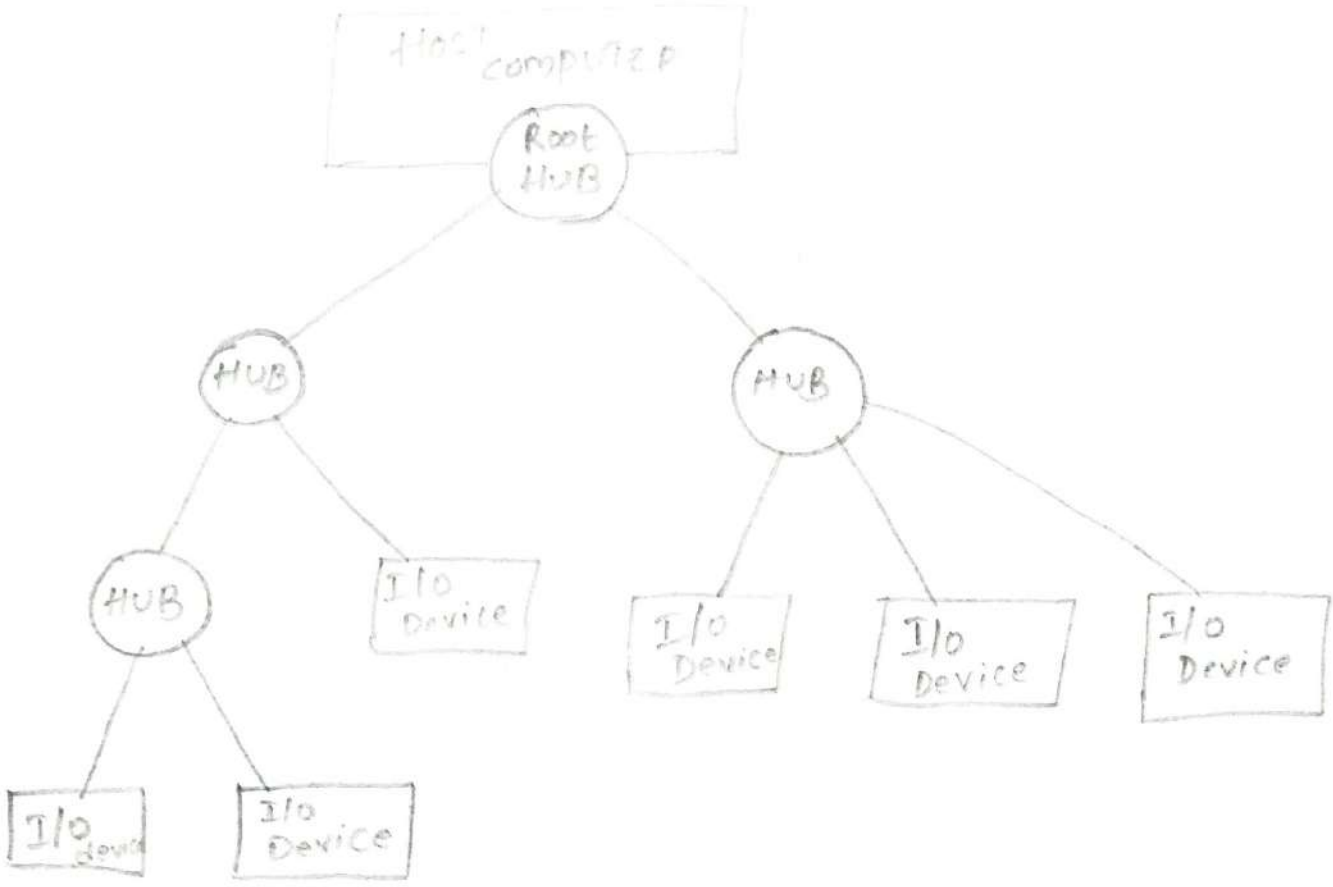


fig (a): Universal Serial Bus Tree Structure

A Root Hub Connects the Entire tree to the host computer. The leaves of the tree are I/o devices (ex:- Keyboard, Speaker, digital TV etc) which are called as functions in USB Terminology.

In normal operation, hub copies a message from host computer and sends to all I/o devices, but only the addressed device will respond to that message.

USB has a Serial bus format which satisfies the low cost & flexibility requirements.

USB protocols :- All information transferred over the USB is organized in packets, where a packet consists of one (or) more bytes of information.

The information transferred on the USB can be divided into two broad categories. They are

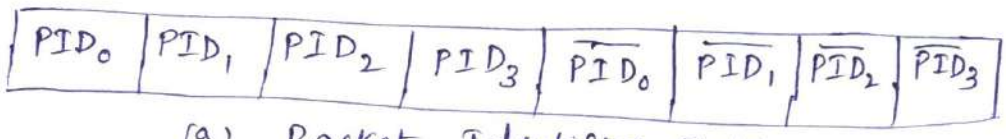
(1) Control packets
(2) DATA packets

→ Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly (or) indicating Error.

→ Data packets carry information that is delivered to a device. A packet contains one or more fields with different kind of information.

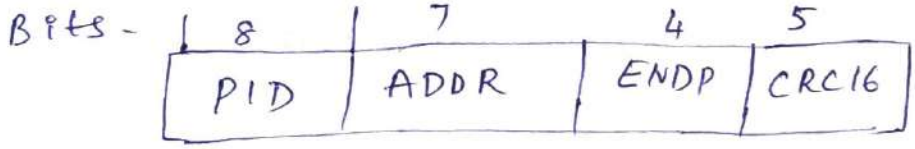
The first field of any packet is called the packet Identifier PID, which identifies type of that packet.

USB packet FORMATS :-



(a) Packet Identifier Field.

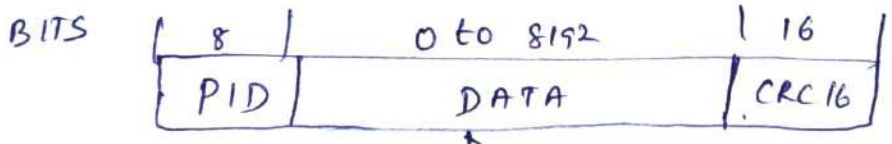
In the above fig (a), Four bits of information in this field, but they are transmitted twice. The first time they are sent with their true values, second time with each bit complemented as shown in fig (a). This enables the receiving device to verify the PID byte has been received correctly.



(b) Token Packet, IN (or) OUT

The fig (b) shows token packet format. Control packets used for controlling data transfer operations are called token packets.

PID - packet Identifier (8 bit)
ADDR - Address of a device (7 bit)
ENDp - End point number within that device (4 bits)
CRC16 - Cyclic Redundancy Check (CRC) for Error checking (5 bits).



(C) DATA PACKET.

Above fig (c) shows data packets which carry input and output data.

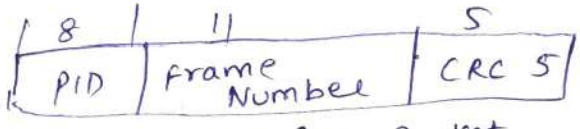
* **Isochronous Traffic on USB :-** One of the key objective of USB is to support transfer of Isochronous data such as sample Voice in a simple manner.

→ Device that generate (or) receive information (data) require a time reference to control the Sampling process.

→ TO Provide reference, USB is divided into frames of Equal length. Frame is 1ms long for full & slow speed data.

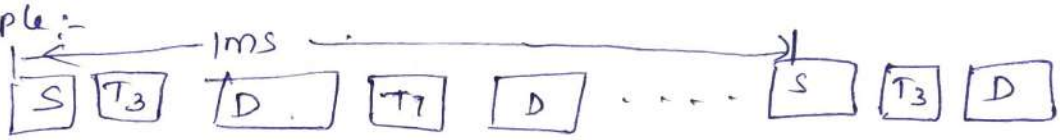
→ The root hub generates a start of frame (SOF) control packet Once Every 1ms to mark the beginning of a new frame.

USB Frames :-



(a) SOF Packet.

Frame Example :-



S = Start of frame ; D = Data packet

T_n = Token packet, address = n.

* **Electrical Characteristics :-**

The cables used for USB connections Consists of four wires.

- Two are used to carry power, ±5V and Ground
- The other two wires are used to carry data.

UNIT-4

MEMORY SYSTEM

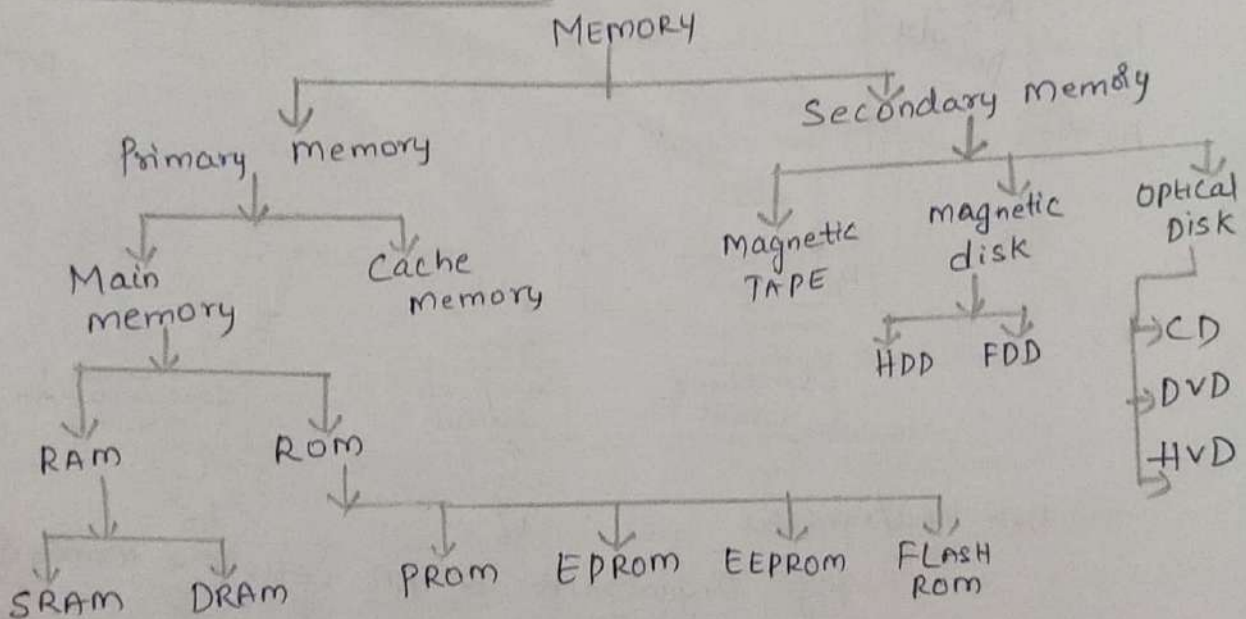
*. **MEMORY :-** A Computer memory is a storage device/place to store data (or) information.

Two types of Memory. They are (1) primary memory and (2) Secondary memory.

Primary memory :- It is also known as Main memory. The memory unit that communicate directly with the cpu are called as main memory (or) primary memory.

Secondary memory :- It is also known as auxiliary memory. The memory unit which provides backup storage called as Secondary memory

Classification of Memory :-



* Internal Organization of Memory Chips:

Memory Cells are arranged in the form of an array, each cell is capable of storing one bit of data. One row is one memory word. All cells of a row are connected to a common line known as "word line". word line is connected to Address decoder as shown in below figure. Data Input / output lines of the memory chip are connected to Sense / write circuits.

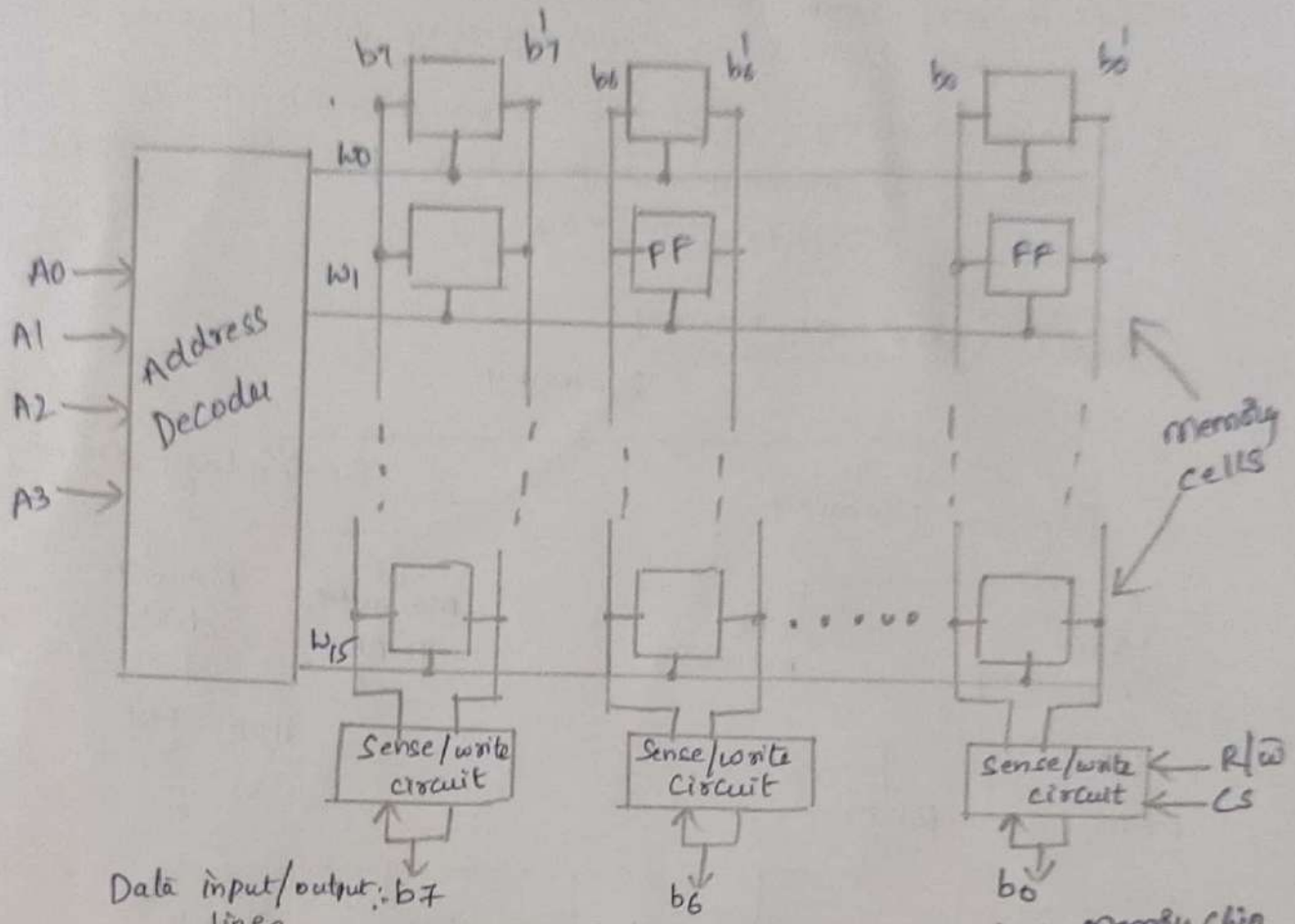


Fig:- Organization of BIT CELLS in a Memory Chip

In the above diagram, memory Chip Consisting of 16 words of 8 bits each, so it is referred as 16x8 organization. Two control lines R/w̄ (Read / write) and cs (chip select) line to the Sense / write circuit. This circuit has 16 external Connections for address, data & control lines (i.e., A_0-A_3, b_0-b_7, cs, R/w̄) and power supply & ground.

Whenever a processor sends a Address, on to address lines (A_0-A_3) then address decoder decodes the addres, corresponding

word line will be Selected, from their data is given to Sense/write circuit through column data lines on to data lines of (b_0-b_7) from where Corresponding data (b_0-b_7) bits are collected.

Organization of 1K X 1 memory Chip :-

When a large memory circuit is Consider i.e 1K X 1 means 1K (1024) memory cells & 1 bit O/P line. The below figure shows organization of 1K x 1 memory Chip.

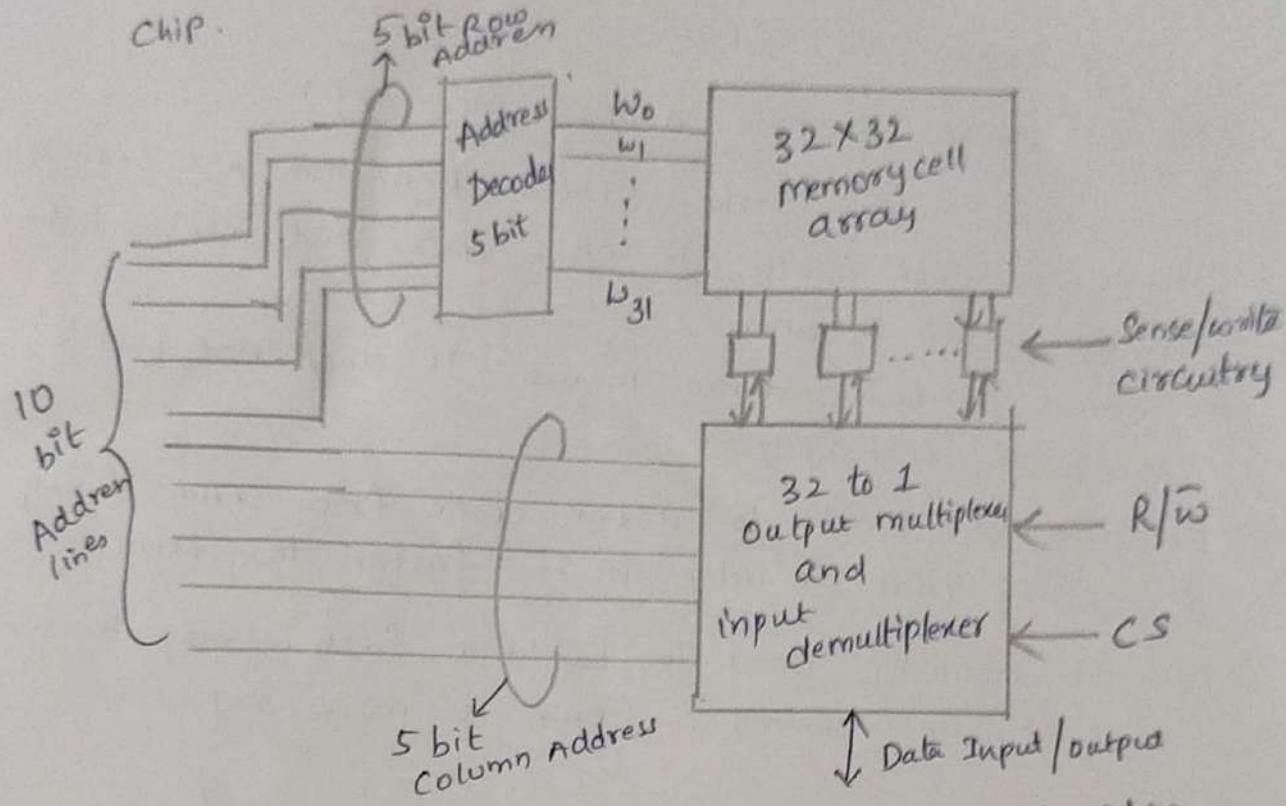


Fig :- Organization of a 1K X 1 memory Chip

Here we use 10 Address lines, 1 data line, resulting 15 External connections. The 10 Address line is divided into two groups of 5 bits Each to form the row & Column addresses for the Cell array.

A row address Selects a row of 32 cells of which are accened in parallel.

A Column address, only one of these cells is connected to External data line by the output multiplexer and input demultiplexer. So only one bit is Selected from demultiplexer and given as Output data.

* RAM [RANDOM ACCESS MEMORY] :-

RAM stands for Random Access memory. It is internal memory of CPU for storing data, program and results of program.

RAM is a Read/Write memory. RAM is also called as Main memory (or) primary memory. data in RAM can be accessed Randomly. RAM is a Volatile memory means content are lost when power is turned off. RAM is of two types

(1) SRAM (2) DRAM

SRAM [STATIC RANDOM ACCESS MEMORY] :-

The static RAM has a 6 transistor circuit in each cell, to store data and retains data until power is on.

As name indicates static means, it does not need to frequent recharging.

CPU does not wait to access data from SRAM during processing that is why it is faster than DRAM.

SRAM is normally used to build a fast memory known as "CACHE MEMORY". SRAM is more expensive.

DRAM [Dynamic RANDOM Access Memory] :-

DRAM is made up of capacitors and transistors. The below figure shows a single transistor dynamic memory cell.

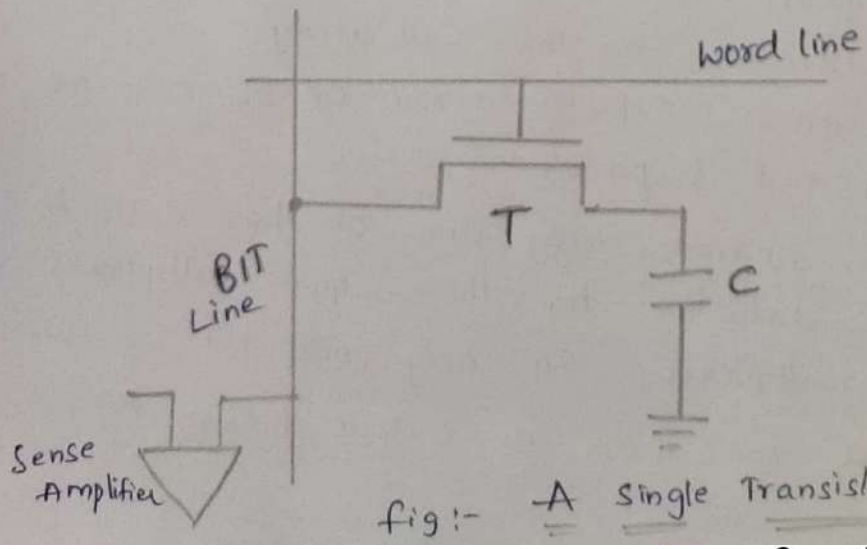


fig :- A single Transistor dynamic memory CELL

Above diagram consists of a Capacitor 'C' and a Transistor 'T' which is a single memory cell to store Either '1' (or) '0' bit. Write and Read Operation can be performed by DRAM.

Read Operation :- we have two lines, word line and bit line; Connected to transistor.

when word line = 1, nmos transistor 'Gate' terminal is given high voltage, then it get short circuit and Capacitor starts discharges through bit line, a sense amplifier connected to bit line detects the voltage (Read) the data.

Write Operation :- when word line = 1, during write operation bit line = 5V or '1', then it get short circuit and capacitor starts Charging and it stores the data.

* Differences between SRAM and DRAM.

SRAM	DRAM
It is Complex	It is simplex
In this 6 Transistor requires to store data	It requires one transistor
In this there is no leakage of Charge	In this there is charge leakage through Capacitor
Power Consumption is High	Power Consumption is low
Cost is high	Cost is low
It requires less time to access data	It require more time to access data.
It is used as Cache memory	It is used as Virtual memory

* ROM [Read Only Memory] :-

ROM is a non Volatile memory i.e permanent

The below figure shows the a ROM Cell

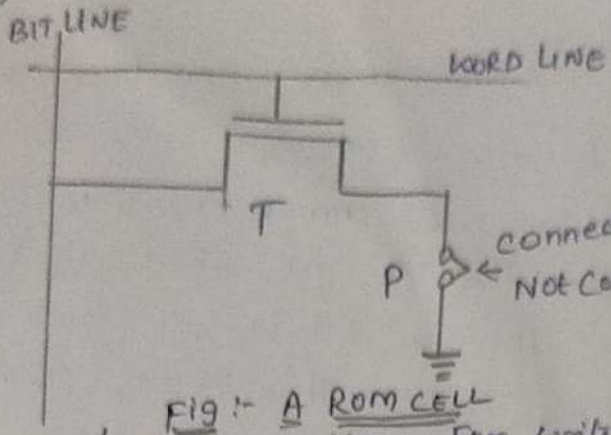


Fig :- A ROM CELL

Rom can do read / write operation. For write operation, to store logic value '0' in the cell, transistor is connected to ground at point 'P'; to store logic value '1' in the cell, transistor bit line is connected through a resistor to power supply.

Read Operation: To read state of cell, word line is activated (WL=1), then transistor switch is closed and voltage on bit line drops to zero, there is a connection between transistor and ground.

If there is no connection to ground, bit line remain at high voltage indicating Logic '1'.

A Sense Amplifier at end of bit line reads the output value.

There are various types of Rom's are available. They are

(1) PROM

(2) EPROM

(3) EEPROM

(4) FLASH MEMORY

* PROM [Programmable Read only Memory] :-

PROM first developed by Texas Instrument

PROM memory is a blank memory contains all 0's. The user can insert 1's at the required locations by burning out the fuses at these locations using pulses. So user can write data onto a PROM Chip. The data stored in it cannot be modified and therefore it is also known as one time programmable device. PROM can store fixed programs and data because it is irreversibl

* EPROM :-

EPROM stands for Erasable programmable ROM. It is different from PROM, in this we can Erase the program and rewrite another program in EEPROM Chip. It is flexible. EPROM can Erase the program by Exposing the Chip to Ultra Violet rays of some specific wavelength fall's onto Chip's glass panel. So fuses are reConstituted and thus new things can be written on the memory. EPROM Eliminates the problem faced by PROM.

* EEPROM :-

EEPROM Stands for Electrically Erasable Programmable ROM. These are also Erasable like EPROM, but Erasing is done by Exposing the Chip to Electric Current. Thus it provides Ease of Erasing it Even if the memory is positioned in the Computer. EEPROM is possible to Erase the Cell contents Selectively.

Disadvantage: Different Voltages are needed for Erasing, writing and reading the Stored data.

* FLASH MEMORY :-

Its technology is similar to EEPROM

In EEPROM, it is possible to read and write contents of a single cell, but in flash memory it is possible to read the contents of single cell, but only possible to write entire block of cells.

Flash memory have greater density, which leads to higher capacity and a lower cost per bit.

It requires a single power supply voltage and consumes less power in their operations.

Applications :- Hand held Computers, Cell phones, digital cameras and mp3 music players etc.,

Larger memory modules consists number of chips, to implement such modules, there are two types they are (1) FLASH CARDS and (2) FLASH DRIVERS.

(1) Flash Card :- It is a small storage devices, to store data on portable or remote computing devices

The card is simply plugged into a conveniently accenible slot. Its memory size are of 8 32 64 MB

(2) Flash Drives :- The flash drives are designed to fully Emulate the hard disk. It is a storage device. Larger flash memory module Can be developed by replacing the hard disk drive. They have shorter seek a access time which results in faster response.

The capacity of flash drive is less than < 1GB than hard disk (>1GB)

* Memory System Consideration :-

The RAM chip for a given application is selected depending on several factors like cost, speed power dissipation, and size of the chip.

SRAM is used when very fast operation is required so it is used for cache memories.

DRAM's are best for main memories because less number of components are required.

To Reduce the number of pins, the dynamic memory chips use multiplexed address inputs.

MEMORY CONTROLLER :-

DRAM's are used for multiplexing address inputs. The address is divided into two parts. The higher order bits which select a row in the cell array, are provided first and latched into memory chip under control of RAS (Row Address Select) signal.

The lower order bits, which select a column, are provided on the same address pins and latched using the CAS (Column Address Select) signal.

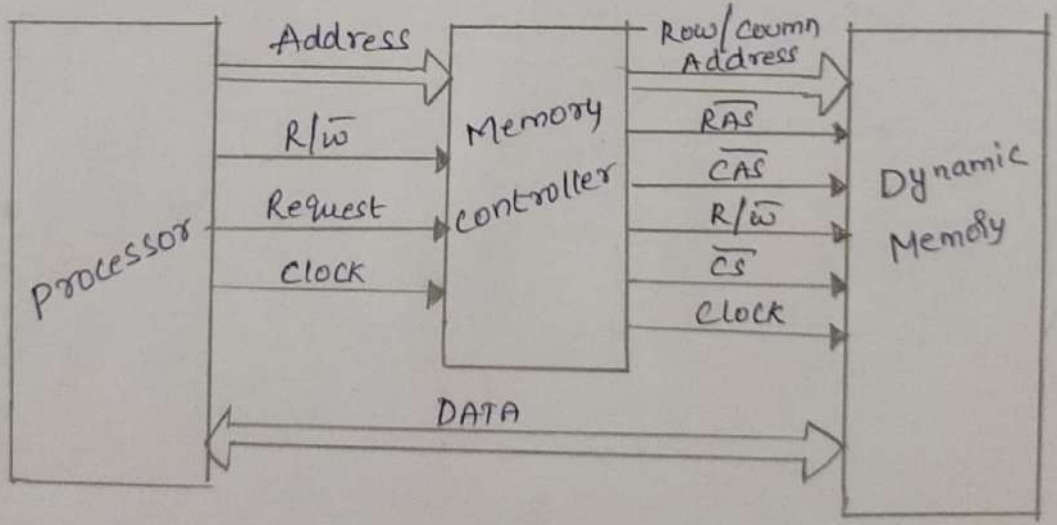


fig:- Use of a Memory Controller

→ Processor issues all bits of an address at the same line. The required multiplexing of address bits is usually performed by a "memory controller circuit".

→ It is placed in between the processor and the Dynamic memory as shown in figure.

→ Controller accepts complete Address and R/$\bar{W}$ signal from processor under control of a 'Request' signal, which indicates memory access operation is needed.

→ memory controller then forward Row and Column portions of address to the memory by $\overline{RAS}$ and $\overline{CAS}$ signals.

→ It also sends R/$\bar{W}$ and $\overline{CS}$ (chip select) which is active low signal is send to memory.

→ Data lines are connected directly between the Processor and the memory.

*. Cache Memory :-

The cache is a small and very fast memory, interposed between processor and the main Memory. Cache memory holds frequently requested data and instructions so that they are immediately available to cpu when needed.

It reduces the average time to access data from the main memory.

The effectiveness of cache mechanism is based on a property of computer programs called "Locality of Reference".

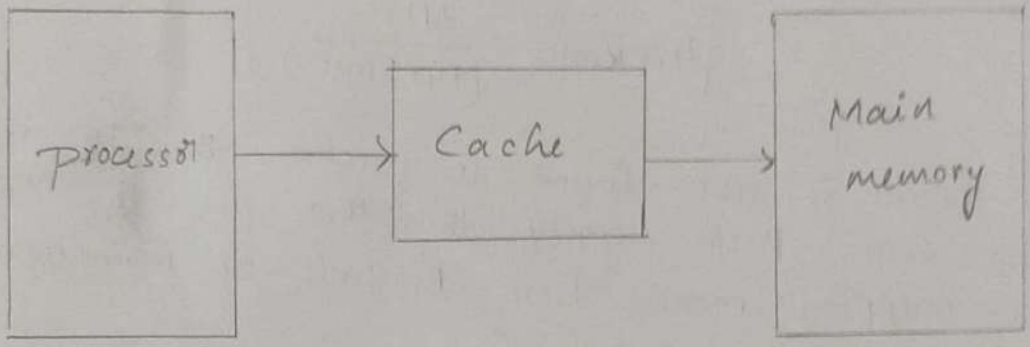


fig: Cache Memory.

In the above Cache memory figure, when a processor issues a Read request, contents of a block of memory words containing the location specified are transferred into the Cache.

→ When the program references any of the locations in this block, desired contents are read from cache directly.

→ Cache memory can store a reasonable number of blocks at any given time, but this number is small compared to main memory.

Block means a set of contiguous address locations of some size

→ When the cache is full and a memory word (data) that is not in cache and now we have store that word in Cache, now we have remove and create space for new block, for doing this process we use replacement algorithms.

→ The data transfer between cache and main memory it is always done in terms of blocks. The data transfer between cache and processor is done in terms of words.

→ The performance of cache memory is measured in terms of a quantity called HIT RATIO.

→ When the cpu requests any word in cache, if it is available in cache it is called as "HIT"

→ If the word is not available in cache, then it is called as "Miss".

→ HIT RATIO is defined as number of hits to the total cpu references to memory.

$$\text{HIT RATIO} = \frac{\text{HIT}}{(\text{HIT} + \text{MISS})}$$

when word is not found in cache, then it collect the data from main memory; then that process is called Mapping. Cache mapping means how contents of main memory are brought into cache.

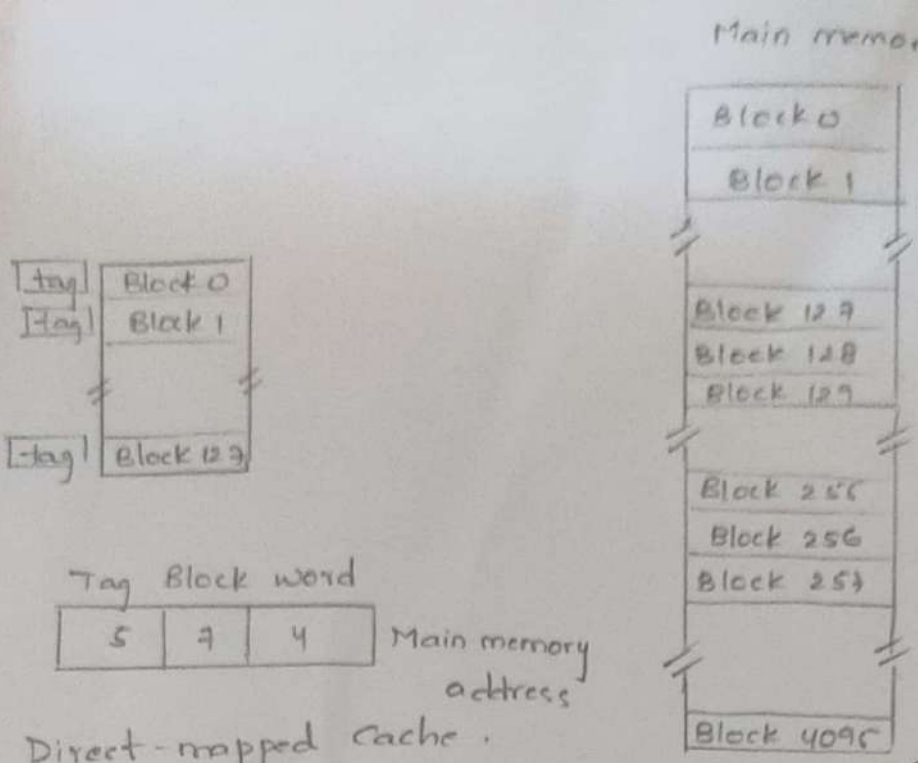
There are different ways of mapping functions available. There are three types of mapping functions:

1. Direct mapping
2. Associative mapping
3. Set Associative mapping.

(1) Direct Mapping:-

The simplest way to determine cache locations in which to store memory blocks is the direct mapping technique.

→ In this technique, block j of the main memory maps onto block j modulo 128 of the cache is shown in below fig.,



Direct-mapped Cache.

Thus whenever one of the main memory blocks 0, 128, 256.... is loaded into the cache, it is stored in cache block 0. Blocks 1, 129, 257.... are stored in cache block 1 and so on

→ The memory address can be divided into three fields The lower order 4 bits select one of 16 words in a block.

→ When a new block enters the cache, the 7 bit cache block field determines the cache position in which this block must be stored.

→ The high order 5 bits of the memory address of the block are stored in 5 tag bits associated with its location in the Cache.

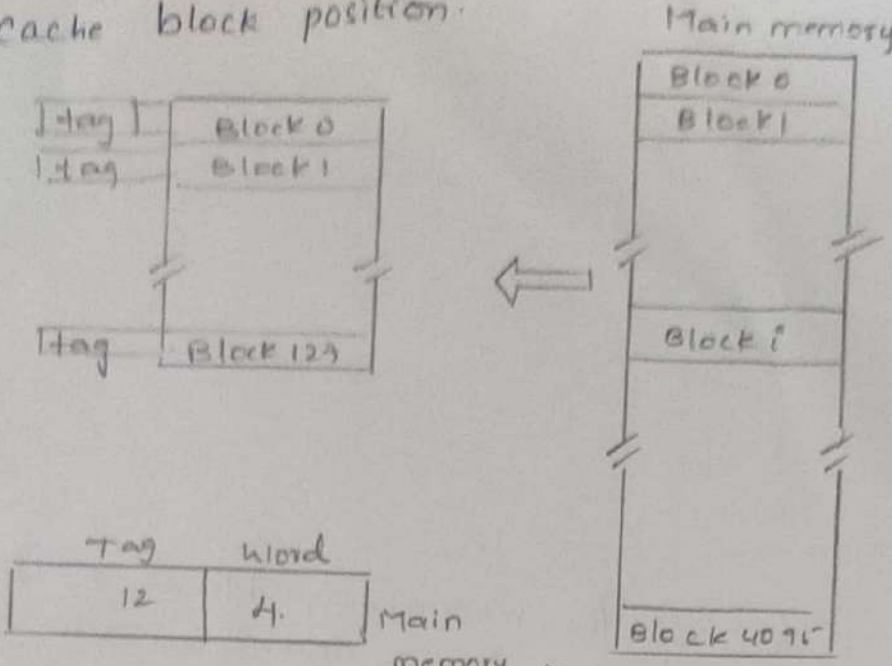
→ The tag bits identify which of the 32 main memory blocks mapped into this cache position is currently resident in the Cache.

→ As Execution proceeds, the 7 bit cache Block field of each address generated by the processor points to a particular block location in the cache.

→ The higher order 5 bits of the address are compared with the tag bits associated with that cache location. If they match, then the desired word in that block of cache

→ If there is no match, then the block containing the required word must first be read from the main memory and loaded into the Cache.

* Associative Mapping :- The most flexible mapping method, in which a main memory block can be placed into any cache block position.

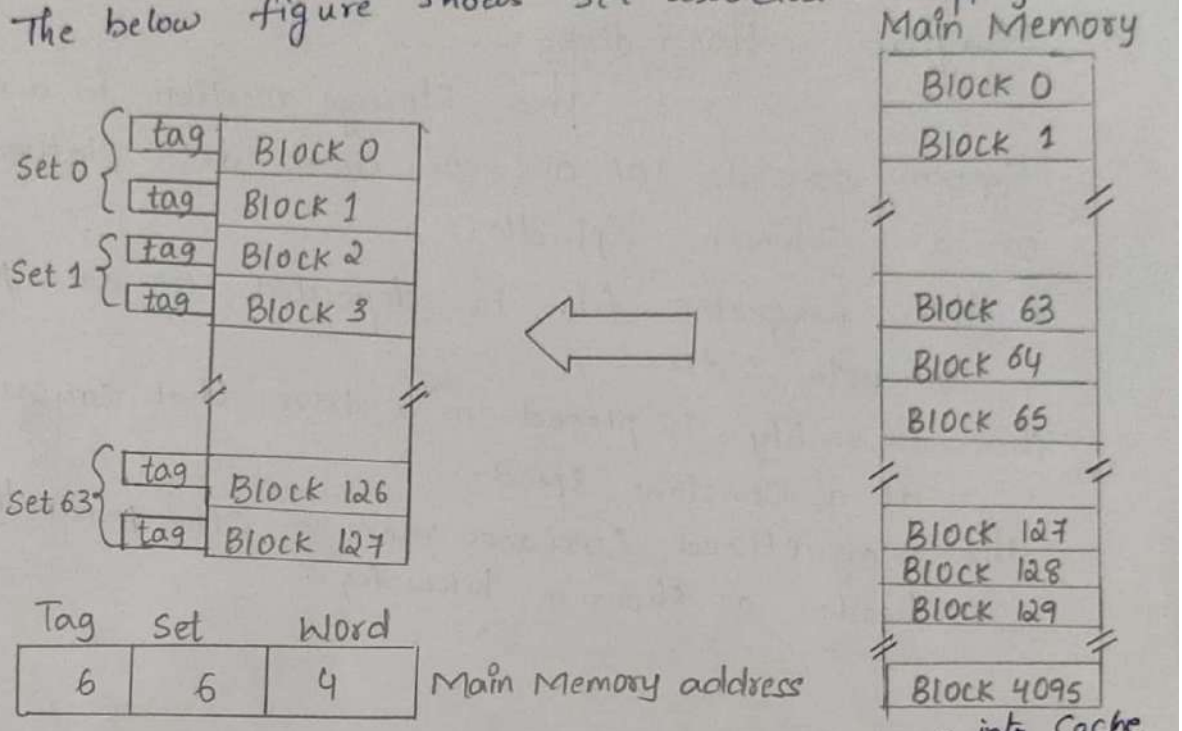


Associative-mapped cache.

In this case, 12 tag bits are required to identify a memory block when it is resident in the cache.
The tag bits of an address received from processor are compared to tag bits of each block of cache to see if desired block is present (or) not, this is called Associative mapping technique.
It can choose any ~~mem~~ location in cache to place memory block. when a new block is brought into the cache, it replaces an exisiting block only if cache is full.
The complexity of an associative cache is higher than that of a direct mapped cache, because of the need to search all 128 tag patterns to determine whether a given block is in the cache.
To avoid a long delay, tags must be searched in parallel this type of search is called Associative search.

* Set Associative Mapping :-

The combination of direct & associative mapping techniques are used. Blocks of the cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of a specific set. The below figure shows set associative mapping technique



In this case, memory blocks 0, 64, 128 .... 4032 map into cache Set 0, and they can occupy either of two block positions within this set.

→ Having 64 sets means that the 6 bit set field of the address determines which set of the cache might contain desired block. The tag field of address must then be associatively compared to tags of two blocks of the set to check if the desired block is present. This two way associative search is simple.

A cache that has k blocks per set is referred to as a k-way set associative mapping.

* Secondary storage :

The Large storage requirements of most Computer Systems are economically realized in form of magnetic and optical disks, which are known as Secondary Storage devices.

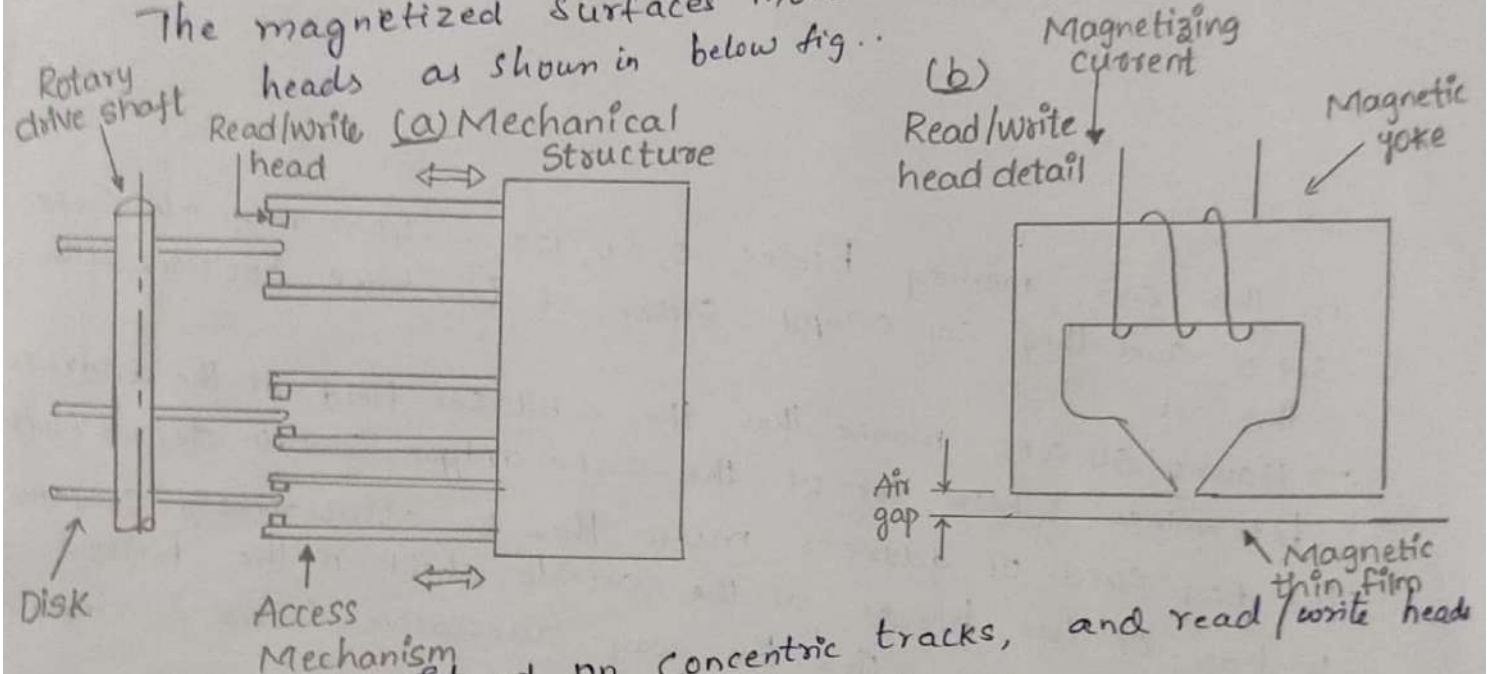
Magnetic Hard disks :-

The storage medium in a magnetic disk System consists of one (or) more disk platters mounted on a Common Spindle.

A thin magnetic film is deposited on each platter, usually on both sides.

The assembly is placed in a drive that causes it to rotate at a Constant speed.

The magnetized surfaces move in close proximity to read/write heads as shown in below fig..



Data are stored on concentric tracks, and read/write heads move radially to access different tracks.

Organization and Accessing of Data on a Disk :-

Each surface is divided into Concentric tracks, and each track is divided into sectors. The set of Corresponding tracks on all surfaces of a stack of disks forms a logical Cylinder.

- All tracks of a cylinder can be accessed without moving the read/write heads.

Data are accessed by specifying the surface number, the track number and the sector number.

Read and write operations always start at sector boundaries.

Data bits are stored serially on each track, each sector may contain 512 (or) more bytes.

The data are preceeded by a sector header that contains identification (addressing) information used to find the desired sector on the selected track.

→ there is a small inter-sector gap that enables the disk control circuitry to distinguish easily between two consecutive sectors.

- Access Time :-

There are two components involved in the time delay between receiving an address and the beginning of actual data transfer.

Seek Time :- Time required to move the read/write head to the proper track is called as seek time.

Rotational delay (or) Latency time :- The time taken to reach the addressed sector after the read/write head is positioned over the correct track.

Access Time :- The sum of these two (seek time + latency time) delays is called as disk Access time.

Disk Controller :-

Operation of a disk drive is controlled by a disk controller circuit which also provides an interface between the disk drive and the rest of the computer system. The disk controller may be used to control more than one drive.

* Optical Disks :-

Storage devices can also be implemented using optical means.

The familiar compact disk (CD) was first practical applications of this technology used in audio systems in mid 1980's by Sony & philips Companies.

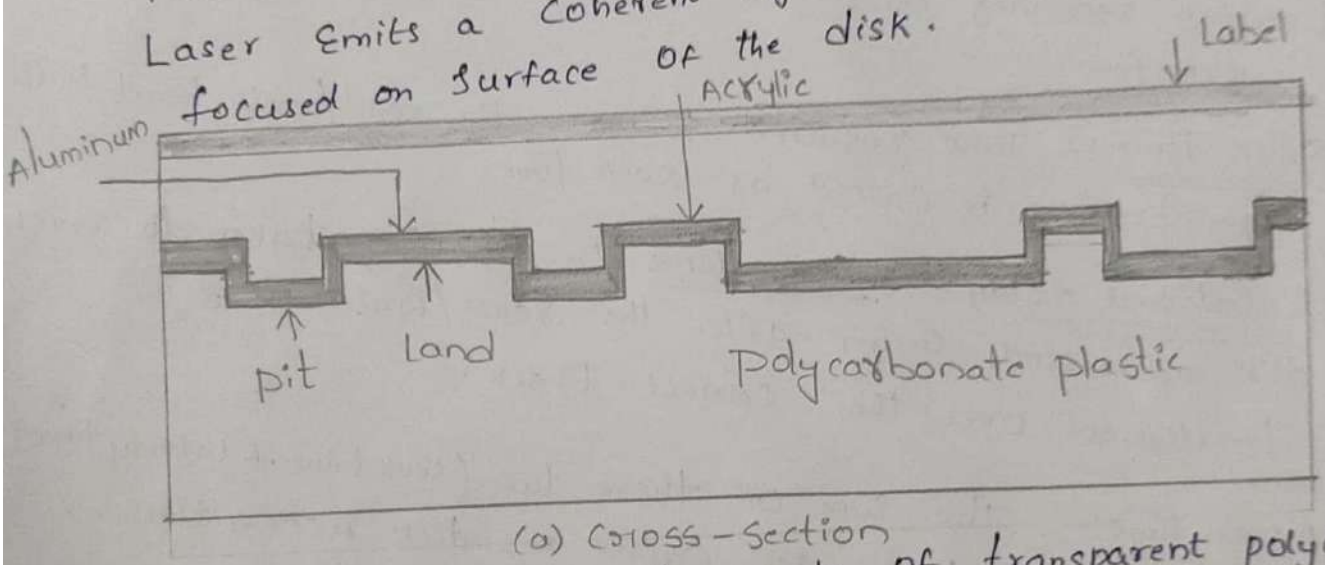
CD Technology :-

The optical technology that is used for CD systems make use of the fact that laser light can be focused on a very small spot.

→ A laser beam is directed onto a spinning disk, with tiny indentations arranged to form a long spiral track on its surface.

→ The indentations reflect the focused beam towards a photo detector, which detects the stored binary patterns.

Laser emits a coherent light beam that is sharply focused on surface of the disk.



(a) Cross-section

The bottom layer is made of transparent polycarbonate plastic, which serve as a clear glass base.

The surface of this plastic is programmed to store data by indenting it with pits.

The unintended parts are called Lands.

A thin layer of Aluminium (reflecting) material is placed on top of a programmed disk.

Finally topmost layer is deposited and stamped with a 'Label'.

The total thickness of the disk is 1.2mm, almost all of it contributed by the polycarbonate plastic.

When a laser beam scans across the disk and encounters a transition from a pit to land is shown in below fig.

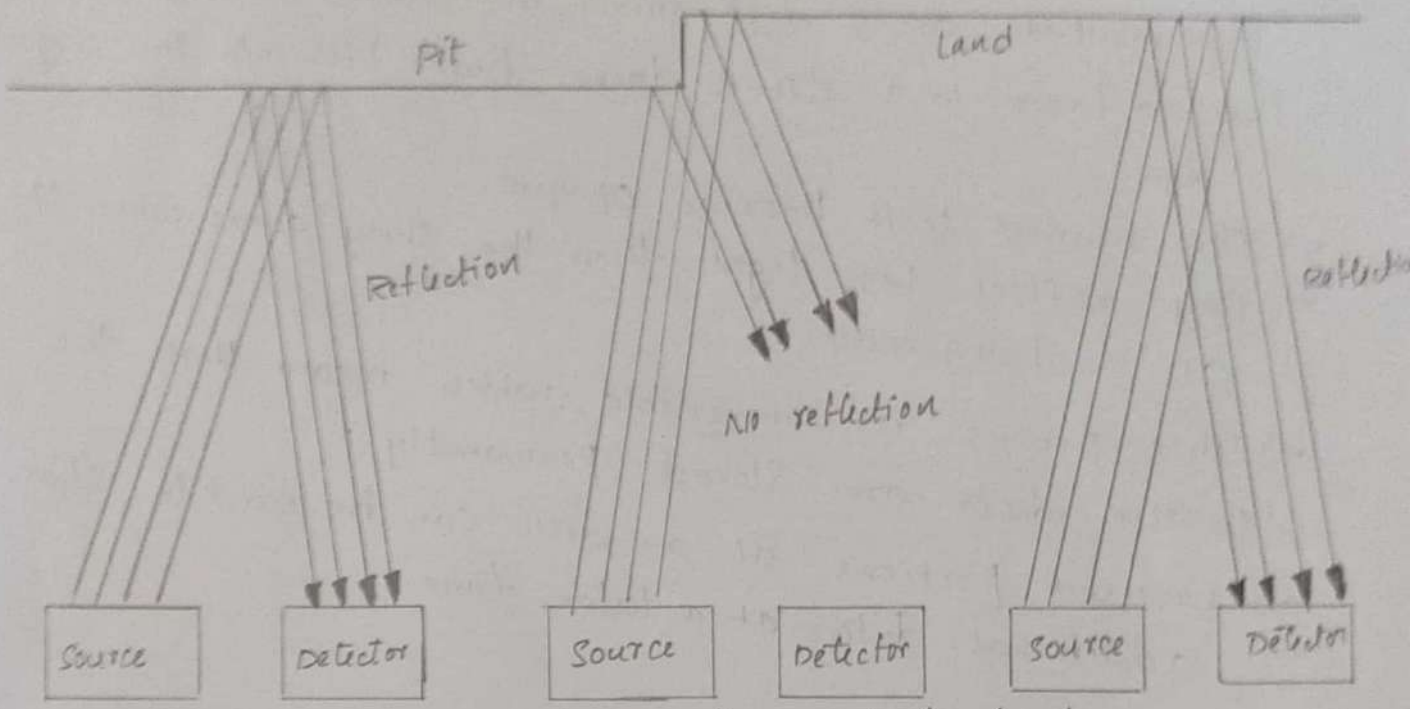


fig. Transition from pit to Land

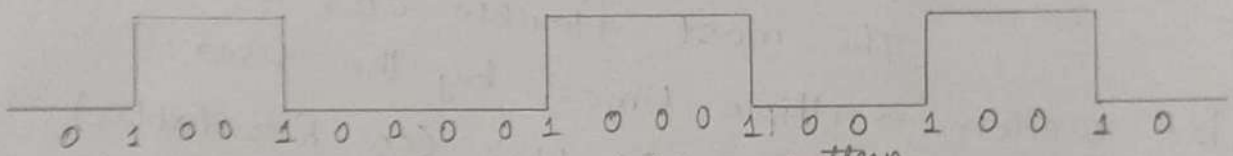


fig. Stored binary pattern

When the light reflects solely from the pit, (or) solely from the Land, detector will see the reflected beam as a bright spot. So at this point data stored as '0'.

But when the beam moves through the edge where the pit changes to the Land and viceversa, i.e., pit-Land and Land-pit transitions the detector will not see a reflected beam and will detect a dark spot. So at this point data stored as '1'. as shown in above fig (c) stored binary pattern.

CD-Recordable :-

A new type of CD was developed in late 1990's on which data can be easily recorded by a computer user. It is known as CD-Recordable (CD-R).

→ A shiny spiral track covered by an organic dye is implemented on a disk during the manufacturing process.

→ Then a laser in a CD-R drive burns pits into the organic dye

→ The burned spots become opaque.

→ They reflect less light than the shiny areas when the CD is being read.

→ This process is irreversible, which means that the written data are stored permanetly.

→ Unused portions of a disk can be used to store additional data at a later time.

CD-Rewritable :-

The most flexible CD's are those that can be written multiple times by the user. They are known as CD-RW's (CD-Rewritables). The basic structure of CD-RW's is similar to the structure of CD-Rs.

→ Instead of using an organic dye in the recording layer, an alloy of silver, indium, antimony and tellurium is used.

→ This alloy has interesting and useful behaviour when it is healed and cooled.

DVD technology:- The success of CD technology and the continuing quest for greater storage capability has led to the development of DVD (Digital Versatile Disk) technology.

→ The first DVD standard was defined in 1996 by a consortium companies, with the objective of being able to store a full length movie on one side of a DVD disk.

→ The physical size of a DVD disk is same as that of CD's. The disk is 1.2mm thick, and it is 120mm in diameter.

→ Using these improvements leads to a DVD capacity of 4.7G bytes.

→ Access time for DVD drivers are similar to CD drives.

→ Rewritable versions of DVD devices have also been developed, providing large storage capacities.

CD-ROM:- The CD's used to store computer data are called CD-ROM's because like semiconductor Rom chips, their contents can only be read.

→ Stored data are organized on CD-ROM tracks in the form of blocks called sectors.

→ There are several different formats for a sector one format known as Mode 1, uses 2352-byte sectors.

→ There is a 16 byte header that contains a synchronization field used to detect the beginning of the sector and addressing information used to identify the sector.

→ This is followed by 2048 bytes of stored data

The basic speed known as 1X is 75 sectors per second. This provides a data rate of 153,600 bytes/s (150K bytes/s) using the mode 1 format.

Higher speed CD-ROM drives are identified in relation to the basic speed.

UNIT-5

PROCESSING UNIT & MICRO PROGRAMMED CONTROL

Syllabus:

Processing Unit
- → Fundamental concepts
- → Register Transfers
- → Performing an Arithmetic (or) Logic operation
- → Fetching a word from memory
- → Execution of a Complete Instruction
- → Hardwired Control

Microprogrammed Control
- Micro Instructions
- Micro program Sequencing
- Wide branch Addressing
- MicroInstructions with next Address field.

* Fundamental Concepts:

To Execute a program, the processor fetches one instruction at a time and performs the operations Specified.

Instructions are fetched from Successive memory locations until a branch (or) Jump instructions is Encountered.

→ The processor keeps track of the address of the memory locations containing the next instruction to be fetched using the program Counter (PC).

→ Another key register in the Processor is the Instruction Register (IR).

→ Suppose that each instruction comprises 4 bytes and that it is stored in one memory word.

To Execute an instruction, the processor has to Perform the following three steps:

(1) Fetch the contents of the memory location pointed to by the PC. They are loaded into the IR

$$IR \leftarrow [[PC]]$$

(2) Assuming that the memory is byte addressable, increment the contents of the PC by 4

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

(3) Carry out the actions specified by instruction in the IR.

Here first two steps represents fetch phase, third step represents Execution phase.

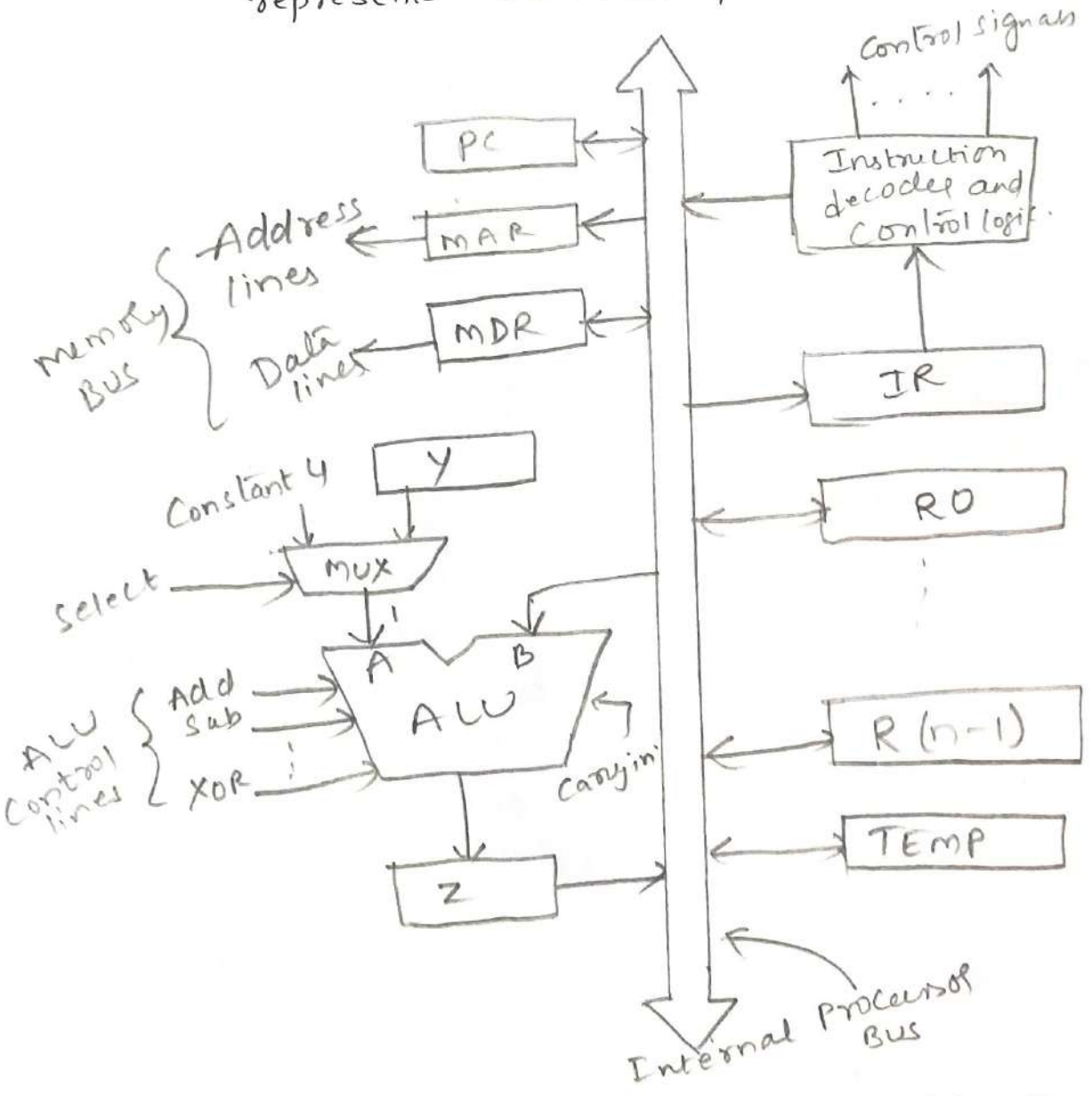


fig: Single bus organization of datapath inside a Processor.

The above diagram shows organization in which the Arithmetic and logic unit (ALU) and all registers are interconnected via a single common bus.

→ Data may be loaded into MDR either from memory bus or from the internal processor bus.

→ The input of MAR is connected to the internal bus and its output is connected to External bus.

→ The multiplexer MUX selects either the output of register Y (or) a constant value of 4 to be provided as input 'A' of the ALU.

→ Three registers Y, Z, and TEMP are used by the processor for temporary storage during execution of some instructions.

→ The registers, the ALU and the interconnecting bus are collectively referred to as the datapath.

* Register Transfers :-

Instruction Execution involves a sequence of steps in which data are transferred from one register to another. For each register, two control signals are used to place the contents of that register on the bus or to load the data on the bus into the register; as shown in below fig.

The input and output of register R_i are connected to bus via switches controlled by the signals R_{in} and R_{iout}.

when R_{in} is set to 1, the data on the bus are loaded into R_i register

Similarly, when R_{iout} is set to 1, contents of Register R_i are placed on the bus.

while R_{iout} is '0', bus can be used for transferring data from other registers

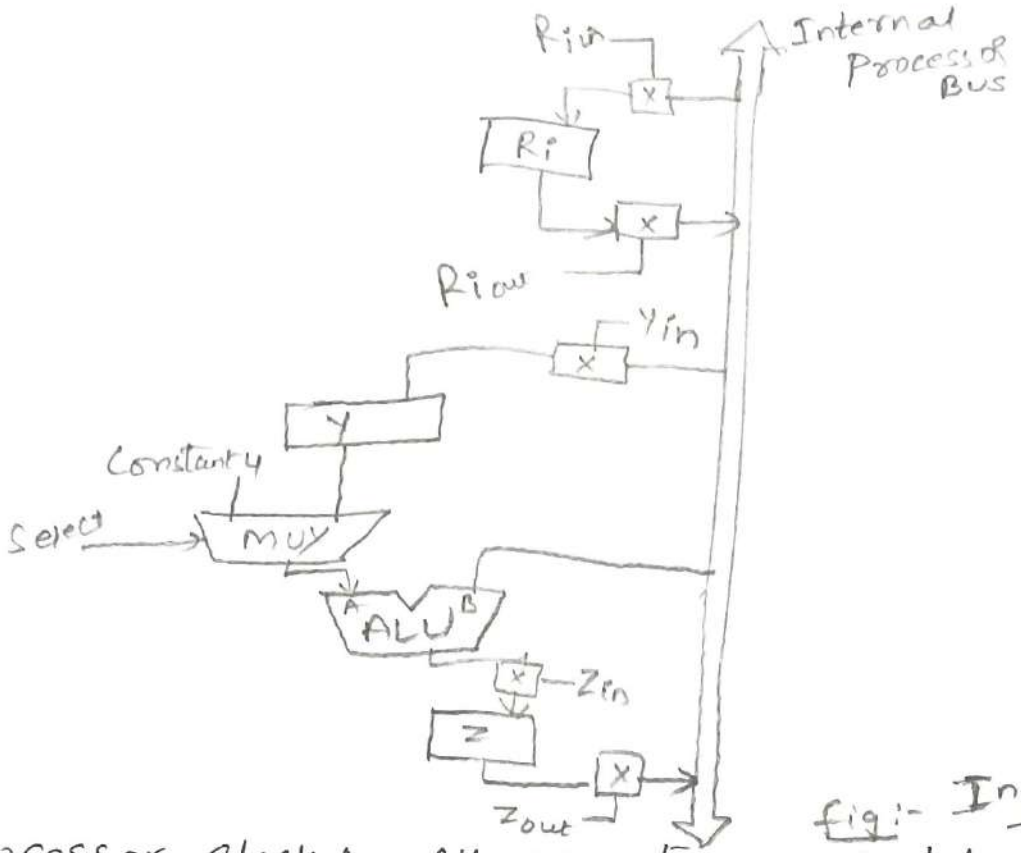


fig:- Input output gating for the registers

Processor Clock :- All operations and data transfers within the processor take place within time periods is called as processor clock.

Multiphase Clocking :- The registers consist of edge-triggered flipflops, but when edge triggered flipflops are not used, two (or) more clock signals may be needed to guarantee proper transfer of data. This is known as multiphase clocking.

For Example :- To Transfer the Contents of register R1 to register R4.

Step 1 :- Enable the output of register R1 by Setting R_{1out} to 1, this places the contents of R1 on the Processor bus.

- Enable the input of register R4 by setting R_{4in} to 1, this loads data from the processor bus into register R_4.

Performing an Arithmetic (or) logic operation :-

→ The ALU is a Combinational Circuit that has no internal storage.

→ It performs Arithmetic and logic operations on the two operands applied to its A and B inputs.

→ one of the operands is the output of multiplexer(MUX) and other Operand is Obtained directly from the bus.

→ The result produced by the ALU is stored temporarily in register 'Z'

→ Therefore, a sequence of operations to add the contents of register R1 to those of register R2 and store the result in register R3 is,

1. R_{1out}, Y_{in}
2. R_{2out}, select Y, Add, Z_{in}
3. Z_{out}, R_{3in}

* Fetching a word From Memory :-

To fetch a word of information from memory, the processor has to specify the address of memory locations where this information is stored and request a Read operation.

→ This applies whether the information to be fetched represents an instruction in a program (or) an operand specified by an instruction.

→ The process transfers the required address to the MAR whose output is Connected to the address lines of the memory bus.

→ when the requested data are received from the memory they are stored in register MDR, from where they can be transferred to other registers in the processor.

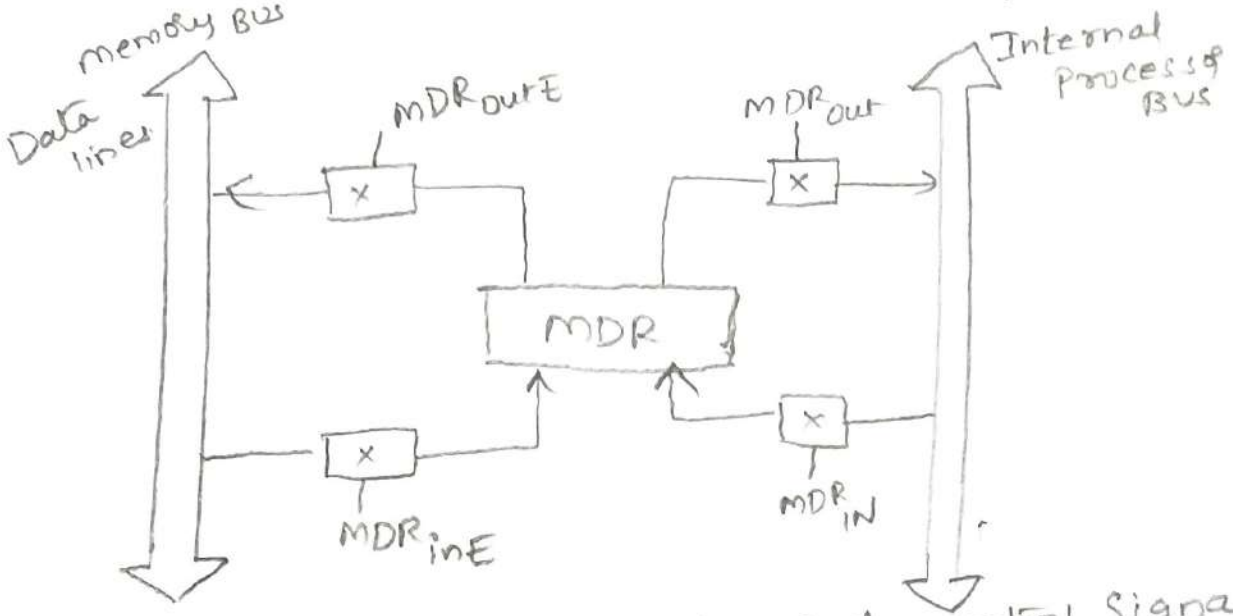


fig:- Connection and control signals for register MDR.

The above diagram shows connection and control signals for register MDR. It has 4 control signals

They are: MDR_{in} and MDR_{out} control the connection to the internal bus

MDR_{inE} and MDR_{outE} control the connection to the External Bus.

The control signal called memory Function Completed (MFC) is used for this purpose.

Ex:- Read operation, consider instruction

MOVE (R1), R2

Actions needed to Execute this instruction are,

1. MAR ← [R1]
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from memory bus
5. R2 ← [MDR]

Timing diagram of memory Read operation

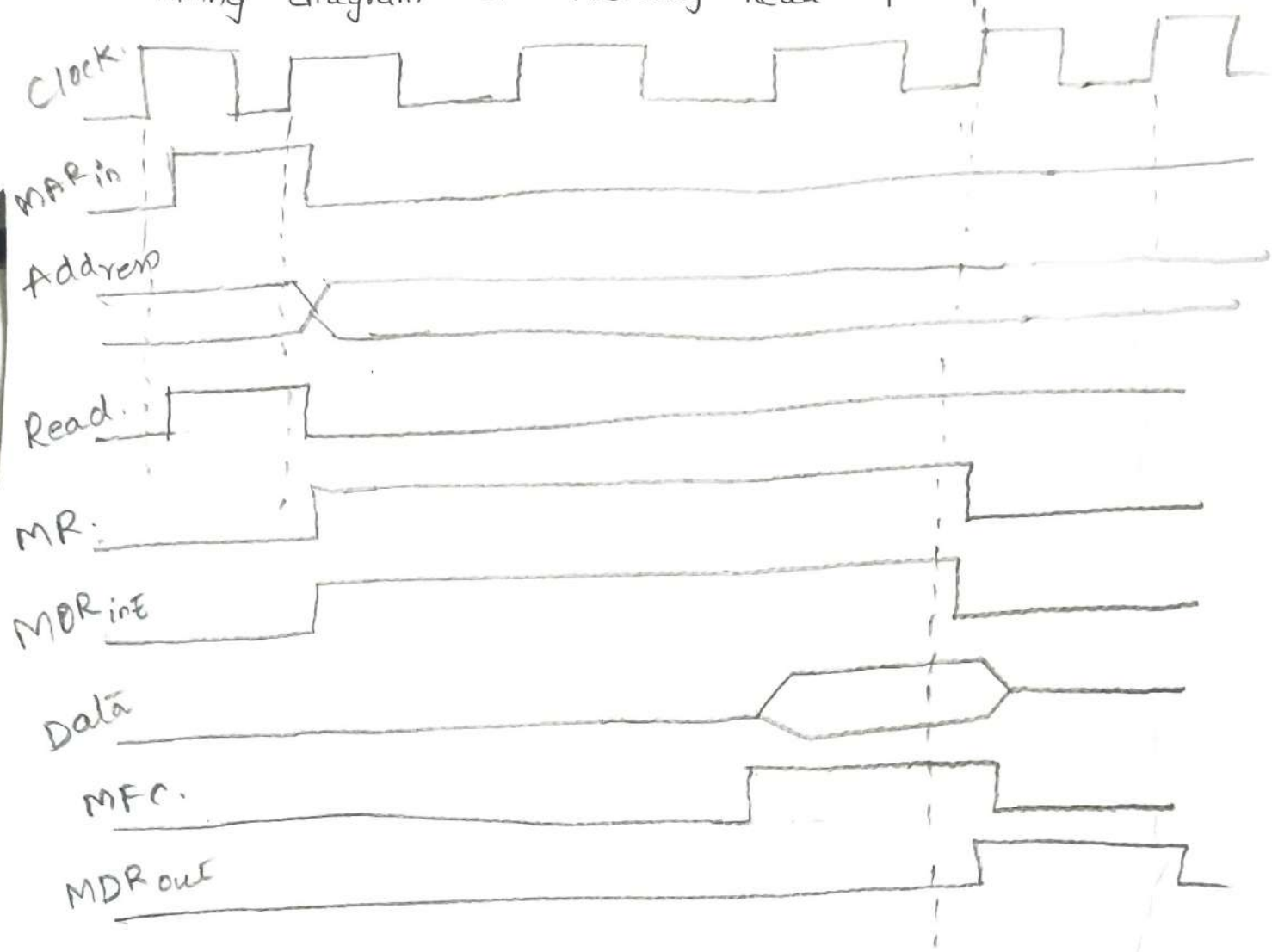


fig:- Timing of a memory Read operation

The memory read operations requires three steps, the actions are as follows:

1. $R1_{out}$, MAR_{in}, Read
2. MDRinE, WMFC
3. MDR_{out}, $R2_{in}$

where WMFC (wait memory function controlled) is the Control Signal that causes Processor's Control Circuitry to wait for arrival of MFC signal.

*. Execution of a Complete Instruction :-

To Execute one instruction, the sequence of elementary operations are required. Let us Consider an instruction

Add (R3), R1

above instruction adds the contents of a memory location pointed to by R3 to register R1. For Executing this instruction requires the following actions :

→ Fetch the Instruction
→ Fetch the first operand (the contents of the memory location pointed by R3)
→ Perform the addition
→ Load the result into R1.

Step	Actions
1.	PC_{out} , MAR_{in} , Read, Select4 , Add, Z_{in}
2.	Z_{out} , PC_{in} , Y_{in} , WMFC
3.	MDR_{out} , IR_{in}
4.	$R3_{out}$, MAR_{in} , Read
5.	$R1_{out}$, Y_{in} , WMFC
6.	MDR_{out} , Select Y , Add , Z_{in}
7.	Z_{out} , $R1_{in}$, End

→ In step 1, the Instruction fetch Operations is initiated by loading the Contents Of the PC into the MAR and Sending a Read request to the memory.

→ The Select signal is set to Select4, which causes multiplexer MUX to select the Constant 4.

→ This Value is added to the operand at input B, which is the contents of the PC, and the result is stored in register Z.

→ The updated Value is moved from register Z back into the PC during Step 2, while waiting for the memory to respond.

→ In Step 3, the word fetched from the memory is loaded into the IR.

→ From Step 1 to Step 3 represents Instruction fetch phase

→ In step 4, the Contents of register R3 are transferred to the MAR and a memory read operation is initiated

→ In step 5, the Contents of R1 are transferred to register 'Y' to prepare the addition operations.

→ ~~when the~~

→ In step 6 :- when the read operation is Completed, the memory operand is available in register MDR, and the addition operations is performed.

→ In step 7 :- The Sum is stored in register Z, then transferred to R1.

From Step 4 to step 7 represents Execution phase

* HARD WIRED CONTROL :-

To Execute instructions, processors must have some means of generating the control signals needed in the proper sequence. There are two categories. They are

(1) Hardwired Control
(2) Micro programmed Control.

The control unit organization is shown in below figure.

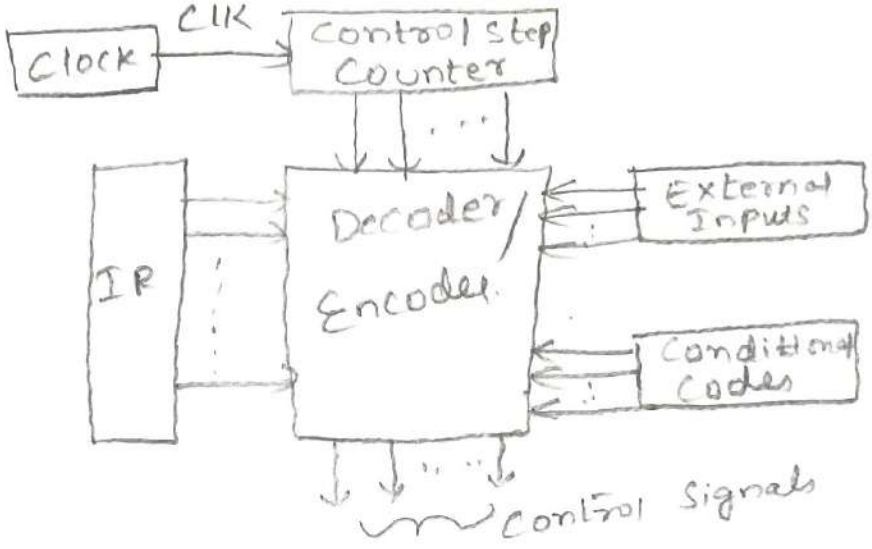


fig :- control unit organization

→ Consider the sequence of control signals.
→ Each step in this sequence is completed in one clock period.
→ A Counter may be used to keep track of the control steps as shown in above figure.
→ Each State (or) Count of this Counter corresponds to one control step.
→ The required control signals are determined by following information:
(1) Contents of the control step counter
(2) Contents of the instruction register.

(3) Contents of the Condition Code flags.
(4) External input signals, such as MFC and interrupt requests.

The below figure shows detail block diagram for hardwired Control unit.

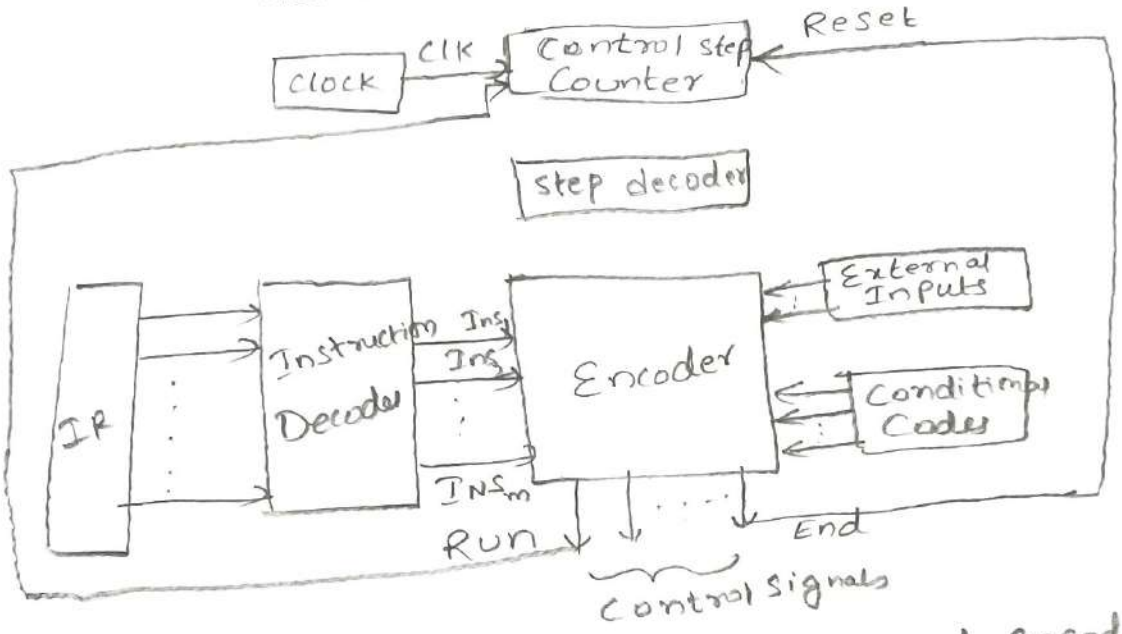


fig:- Seperation of decoding and encoding functions.

The Encoder generates Signal for Single bus processor organization.

Instruction decoder :- It decodes the instruction loaded in the IR.

→ If IR is an 8 bit register then Instruction decoder generates 28 (256 lines) - one for each instruction.

→ According to code in the IR, only one line amongst all output lines of decoder goes high (set to 1 and all other lines are set to 0).

Step decoder :-

It provides a separate Signal line for each Step or time slot, in a control sequence.

Encoder :- It gets in the input from Instruction decoder, step decoder, External inputs and Condition Codes. It uses all these inputs to generate the individual Contro

Signals.
After Execution of each instruction end signal is generated this resets control Step Counter and make it ready for generation of control step for next instruction.

* MICROPROGRAMMED CONTROL :

The Control Signals are generated by a program similar to machine language programs is called as microprogrammed Control.

→ Inside the processor Control Signals can be generated using a Control Step Counter and a decoder/encoder Circuit.

→ A Control word (CW) is a word whose individual bits represent the Various Control Signals.

→ A Sequence of CW's Corresponding to the Control Sequence of a machine Instruction Constitutes the microroutine for that instruction, and the individual Control words in this microroutine are referred to as microinstructions

→ The microroutine for all instructions in the Instruction Set of a Computer are stored in a Special memory Called the Control Store.

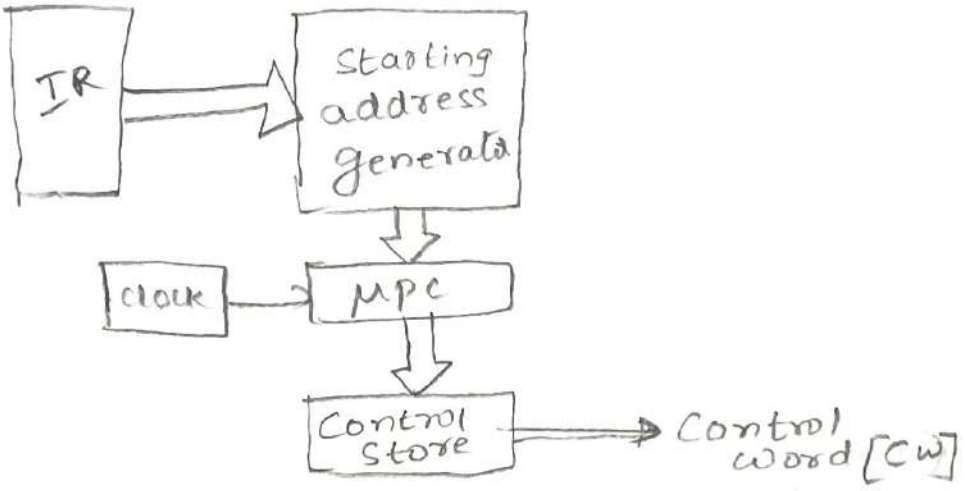


fig :- (a) Basic Organization of a microprogrammed Control unit

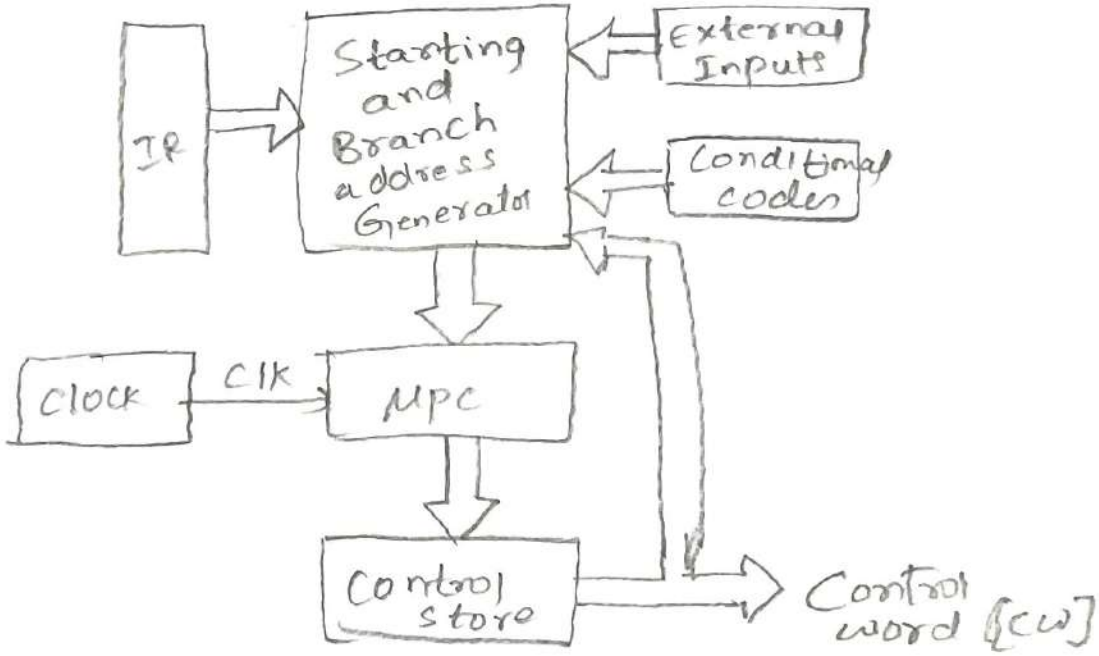
→ To read the control words sequentially from the control store, a micro program Counter (μpc) ~~are~~ is used.

→ Everytime a new instruction is loaded into the IR, register the output of the block labeled "starting address generator" is loaded into the μpc.

→ The μpc is then automatically incremented by the clock, causing successive microinstructions to be read from the Control Store.

→ Hence the Control Signals are delivered to various parts of the processor in the correct sequence.

· For branch instructions, these microinstructions specify external inputs, conditional codes as shown in below figure.



fig(b) Organization of control unit to allow conditional branching in the microprogram.

The following Actions will be taken place for microinstructions

Address	microinstruction
0	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
1	Z_{out}, PC_{in}, Y_{in}, WMFC
2.	MDR_{out}, IR_{in}
3.	Branch to Starting address of appropriate microroutine
...	...
25	If N=0, then branch to microinstruction '0'
26	Offset field of IR_{out}, Select Y, Add, Z_{in}
27.	Z_{out}, PC_{in}, End.

The instruction Branch <0 now be implemented by a microroutine such as shown in above Actions. After loading this instruction into IR, a branch micro instruction transfers control to the corresponding microroutine, which is assumed to start at location 25 in the control Store. This address is the output of the starting address generator block as shown in figure (a).

The microinstruction at location 25 tests the 'N' bit of the condition codes, if this bit is equal to '0', a branch takes place to location '0' to fetch a new machine instruction otherwise, the microinstruction at location 26 is executed to put the branch target address into register Z, microinstruction in location 27 loads this address into the PC.

→ To implementation of a Conditional branch, inputs to this block consists of External inputs and conditional codes as shown in figure(b)

* Micro instructions:

The Control word possess Certain instruction Usually referred to as Microinstructions.

Each microinstruction specifies the microoperations for the System.

There are two types of microinstruction formats. They are

(i) Horizontal microinstruction format
(ii) Vertical microinstruction format.

(i) Horizontal microinstruction format:-

Internal CPU control signals	System bus Control Signals	Jump Conditions (Indirect bit, Zero overflow, Unconditional)	Micro Instruction Address

It supports long formats, Express (or) result to high degree of parallelism

→ low degree of Encoding of Control Information.

(ii) Vertical microinstruction format:-

Function codes	Function codes	Jump Condition	Micro Instruction Address

→ Vertical microinstruction format Supports Less degree of parallelism in Case of microoperations.

– Subsequently high Encoding in Case of Control Information

→ Relatively Short formats.

* MicroInstruction for Add (Rsrc)+, Rdst :-

A microprogram is a set of microinstructions. Microinstructions are Executed in the Sequential Order.

Let us Consider a microinstruction for Execute the Instruction

Add (Rsrc)+, Rdst

Addition of Source operand is accessed in the auto increment mode with destination operand and store result in destination register.

Where Rsrc and Rdst are general purpose registers in Processor. The below table shows the Complete microroutine for fetching and Executing the instruction.

Address (octal)	MicroInstruction
000	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
001	Z_{out}, PC_{in}, Y_{in}, WMFC
002	MDR_{out}, IR_{in}
003	μBranch $\{\mu pc \leftarrow 101$ (from instruction decoder); $\mu PC_{5,4} \leftarrow [IR_{10,9}]$; $\mu PC_3 \leftarrow [\overline{IR_{10}}] \cdot [\overline{IR_9}] \cdot [IR_8]\}$
121	$RSrc_{out}$, MAR_{in}, Read, Select4, Add, Z_{in}
122	Z_{out}, $Rsrc_{in}$
123	μBranch $\{\mu pc \leftarrow 170$; $\mu PC_0 \leftarrow [\overline{IR_8}]\}$, WMFC
170	MDR_{out}, MAR_{in}, Read, WMFC
171	MDR_{out}, Y_{in}
172	$Rdst_{out}$, Select Y, Add, Z_{in}
173	Z_{out}, $Rdst_{in}$, End

* Wide Branch addressing :-

Generating branch addresses means becomes more difficult as the number of branches increases. In such situations programmable logic Array can be used to generate the required branch addresses. This simple and inexpensive way of generating branch addresses is known as wide branch addressing.

Here the opcode of a machine instruction is translated into the starting address of the corresponding micro-routine (or) micro program.

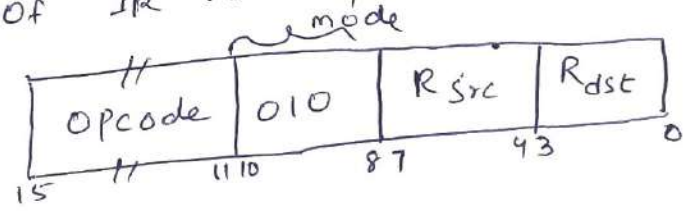
→ It is possible to issue a wait for MFC (memory function controlled) command in branch microinstruction.

→ The WMFC signal means that the microinstruction may take several clock cycles to complete.

Ex:- Add (Src)+, R_{dst}

Format of IR is depicted as:

Contents in IR :-



* MICROINSTRUCTIONS WITH NEXT-ADDRESS FIELD :-

The purpose of branch microinstructions is to find the address of the next microinstruction to be fetched; that means, they do not perform any useful operation in the datapath, they are needed only to determine the address of the next instruction. Thus they reduce the Operating ~~Syst~~ speed of the processor.

→ So to overcome this problem, we provide special address field for branch addresses as shown in below figure.

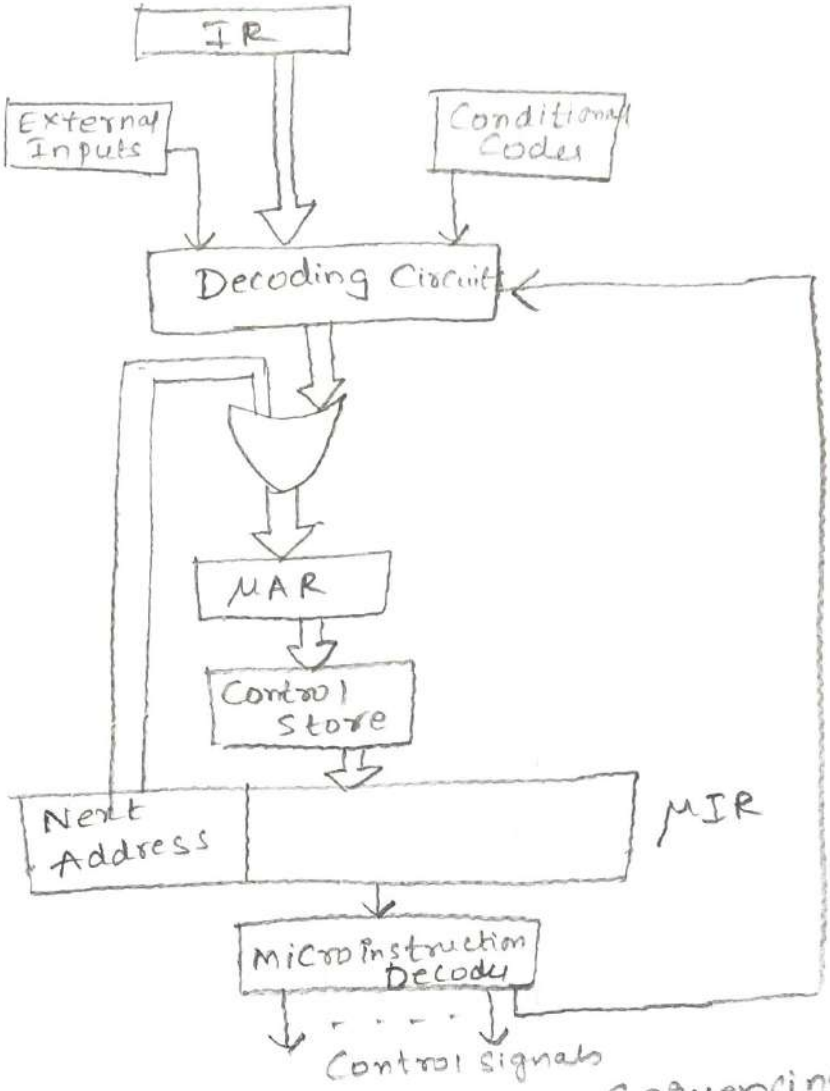


fig:- Micro instruction

Sequencing organization

There is a special address field called "Next-address field" in each microinstruction in order to specify the address of the next microinstructions.

→ As a result, each microinstructions generates the effect of a branch microinstruction and also performs its intended functions.

→ So there is no need for a seperate μpc to store the address of next instruction.

Difference between Hardwired Control and microprogrammed Control unit

Microprogrammed Control unit	Hardwired control unit
1. Its Control functions are implemented in Software	(1) Its Control functions are Implemented in hardware
2. It has Slower Execution speed	(2) It has faster Execution speed.
3. It can easily accomodate changes such as new system Specifications (or) new instruction redesign.	(3) It is not flexible towards any changes.
4. Its design process is Systematic	(4) Its design process is complicated
5. It usually supports more than 100 instructions	(5) It supports less than 100 instructions.
6. Ability to support Operating Systems and diagnostics fealises are easy	(6) It is difficult to support O.S & diagnostics features.
7. Chip area efficiency uses more area	(7) It uses less area of chip.