COMPUTER ORGANIZATION UNIT-1

[SIR C R REDDY COLLEGE OF ENGINEERING]

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING.

Dr DEEPAK NEDUNURI ELURU.

Computer Organization (CO)

UNIT-I

JNTUK
R16
2-2-CSE

Syllabus: Basic structure of Computers:

- Functional Units
- Basic Operational concepts
- Bus structures
- System Software
- Performance
- The history of computer development

TEXT-BOOK:

1. Computer Organization, Carl Hamacher, Zvonks Vranesic, Safea Zaky, 5th Edition, McGraw Hill

(1) Computer types:

→ Computer is a fast electronic calculating machine that accepts digitized input information, processes it according to a list of internally stored instructions and produces the resulting output information.

→ The list of instructions is called a Computer program, and the internal storage is called computer memory.

→ The different computer types are,

(i) Desktop computers
  - personal computer
(ii) notebook Computers
(iii) WorkStations
(iv) Enterprise Systems
(v) Servers
(vi) Super Computers.

→ Desktop Computers have 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 ~~most~~ storage media includes,
- hard disks
- CD-ROMs etc.

→ The most common form of desktop computers is personal computer, which has found wide use in homes, schools, and business offices.

→ Portable notebook computers are a compact version of the personal computer with all of these components packaged into a single unit the size of a thin briefcase.

→ Workstations have more computational power than personal computers, used in engineering applications, especially for interactive design work.

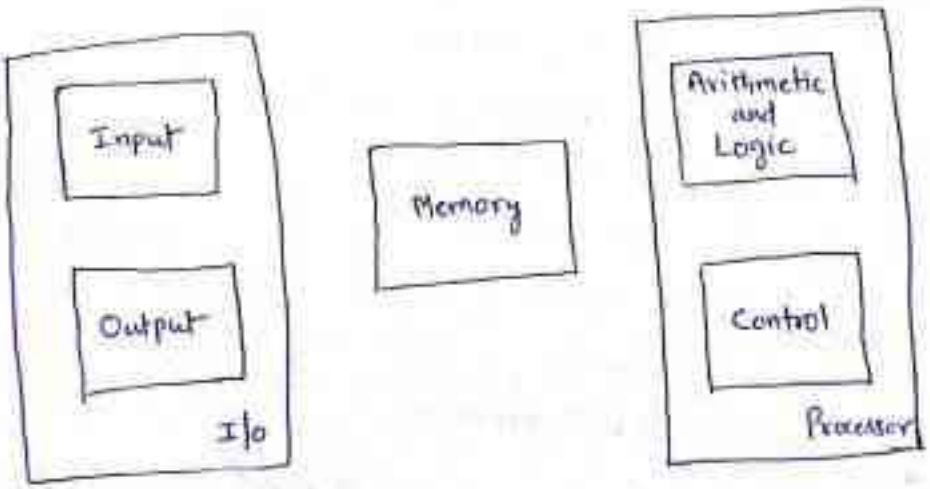
→ Enterprise Systems (or) Mainframes, are used for business data processing in medium to large corporations that require much more computing power and storage capacity than workstations can provide.

→ Servers contain sizable database storage units and are capable of handling large volumes of requests to access the data.

→ Super computers are used for the large-scale numerical calculations required in applications such as weather forecasting and aircraft design and simulation.

(2) Functional Units:

→ Basic Functional Units of a Computer is depicted as,



ASR - CO - 1.3

→ A computer consists of five functionally independent main parts
- input
- memory
- Arithmetic and Logic
- output and
- control unit.

→ Instructions (or) Machine Instructions, are explicit commands that
- govern the transfer of information within a computer as well as between the computer and its I/O devices.
- Specify the arithmetic and logic operations to be performed.

(i) 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,
- Mouse
- Joystick
- Scanner
- Touch Screen
- Light Pen, .. etc

(ii) Memory Unit:

→ The function of the memory unit is to store programs and data.
→ There are two classes of storage
- Primary
- Secondary
→ Primary storage 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 Access Memory)
→ The Secondary storage is used when large amounts of data and many programs have to be stored, particularly for information that is accessed infrequently.
→ The different Secondary storage devices are,
- HDD (Hard disk drive)
- FDD (Floppy disk drive)
- Magnetic disks and tapes
- Optical disks (CD-ROM) - compact disk Read Only Memory

(iii) Arithmetic and Logic Unit :

→ Most computer operations are executed in the arithmetic and Logic Unit (ALU) of the Processor.

→ Consider an example, Suppose two numbers located in the memory are to be added.

→ They are brought into the processor, and the actual addition is carried out by the ALU.

→ The sum may then be stored in the memory (or) retained in the processor for immediate use.

(iv) Output Unit:

→ The output unit is the counter part of the input unit.

→ Its function is to send processed results to the outside world.

→ The different output devices are,

- Monitor
- Printer
- Plotter

(v) Control Unit:

→ All activities inside the machine are directed and controlled by the control unit.

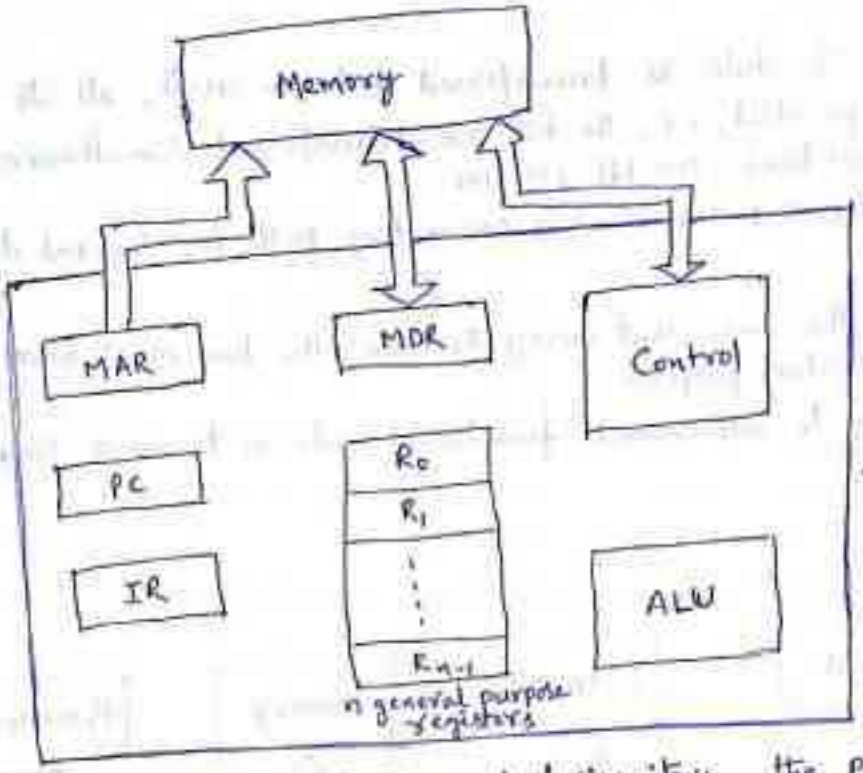
→ The control unit is effectively the nerve center that sends control signals to other units and senses their states.

(3) Basic Operational Concepts

→ Transfers between the memory and the processor are started 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 connections between the Processor and the memory is depicted as,



→ In addition to the ALU and the control circuitry, the processor contains a number of registers used for several different purposes.

→ The Instruction Register (IR) holds the instruction that is currently being executed.

→ Its output is available to the control circuits, which generates the timing signals that control the various processing elements involved in executing the instruction.

→ The Program Counter (PC) keeps track of the execution of a program and it contains the memory address of the next instruction to be fetched and executed.

→ During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed.

→ The PC points to the next instruction that is to be fetched from memory.

→ It has also n-general purpose registers R_0 through R_{n-1}

→ Finally, two registers facilitate communication with the memory

→ These are the Memory Address Register (MAR) and Memory Data Register (MDR)

→ The MAR holds the address of the locations to be accessed.

→ The MDR contains the data to be written into (or) read out of the addressed location.

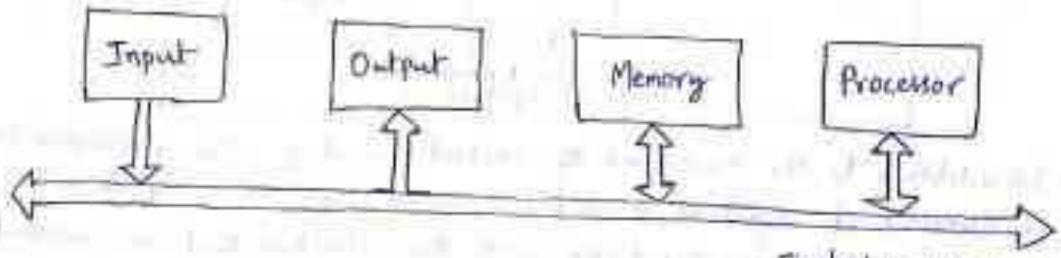
④ Bus Structures:

→ 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, 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 the data, the bus must have lines for address and control purposes.

→ The simplest way to interconnect functional units is to use a single bus is depicted as,



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.

→ The main virtue of the single-bus structure is its low cost and its flexibility for attaching peripheral devices.

→ Systems that 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.

⑤ Software: (System Software)

→ In order 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 functions such as,

- Receiving and interpreting user commands.
- Entering and Editing application programs and storing them as files in secondary storage devices.
- Managing the storage and retrieval of files in secondary storage devices

- Running standard application programs such as processors, spreadsheets, (or) games, with data supplied by the user.
- Controlling I/O units to receive input information and produce output results.
- Translating programs from source form prepared by the user into object form consisting of machine instructions.
- Linking and running user-written application programs with existing standard library routines, such as numerical computation packages.

→ System Software is thus responsible for the coordination of all activities in a computing System.

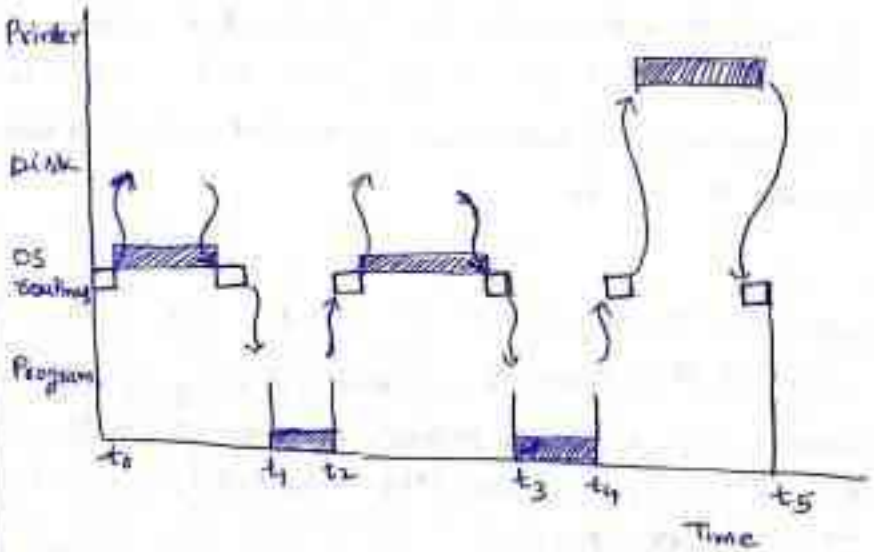
→ Application Programs are usually written in a high-level programming language, such as C, C++, Java, in which the programmer specifies mathematical (or) text-processing operations.

→ Operating System 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.

→ A System program that all programmers use is a text editor. It is used for entering and editing application programs.

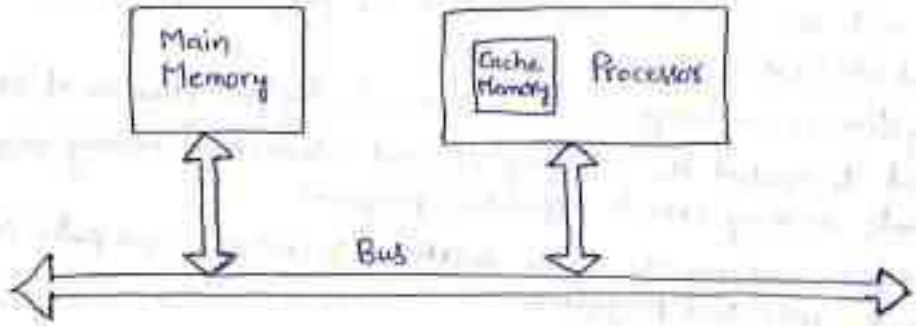
→ User Program and OS routine sharing of the processor is depicted as,



→ The Operating System manages the concurrent execution of several application programs called as multiprogramming (or) multitasking

(6) Performance:

→ The most important measure of the performance of a computer is how quickly it can execute programs.
→ The speed with which a computer executes programs is affected by the design of its hardware and its machine language instructions.
→ To represent the performance of the processor, we should consider only the periods during which the processor is active.
→ These are the periods labeled Program and OS routines. The sum of these periods as the processor time needed to execute the program.
→ The processor cache is depicted as



→ 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, the data are fetched and a copy is placed in the cache.
→ Later, if the same instruction or data item is needed a second time, it is read directly from the cache.

(i) Processor clock:

→ Processor circuits are controlled by a timing signal called a clock.
→ The clock defines regular time intervals, called clock cycles.
→ To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps, such that each step can be completed in one clock cycle.
→ The length P of one clock cycle is an important parameter that affect processor performance.

→ Its inverse is the clock Rate, $R = 1/P$, 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.

→ The 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 basic 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.

→ Consider the instruction

Add R_1, R_2, R_3

- which adds the contents of registers R_1 and R_2, and places the sum into R_3.
- The contents of R_1 and R_2 are first transferred to the inputs of the ALU.
- After the add operation is performed, the sum is transferred to R_3.
- The processor can read the next instruction from the memory while the addition operation is being performed.
- Then if that instruction also uses the ALU, its operands can be transferred to the ALU inputs at the same time that the result of the Add instruction is being transferred to R_3.

→ A higher degree of concurrency can be achieved if multiple instruction pipelines are implemented in the processor.
→ This means that multiple functional units are used, creating parallel paths through which different instructions can be executed in parallel.
→ With such an arrangement, it becomes possible to start the execution of several instructions in every clock cycle. This mode of operation is called Superscalar execution.

(iv) Clock Rate:

→ There are two possibilities for increasing the clock rate, R.
- 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 the clock rate R, to be increased.
- 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 and RISC are used for complex instructions.
→ CISC – Complex Instruction Set Computers
RISC – Reduced Instruction Set Computers.

(vi) Compilers:

→ A compiler translates a high-level language program into a sequence of machine instructions.
→ To reduce N, 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 NXS, which is the total number of clock cycles needed to execute a program.

(vii) 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 comparisons possible, standardized programs must be used.
→ The performance measure is the time it takes a computer to execute a given benchmark.
→ The accepted practice today is to use an agreed-upon selection of real application programs to evaluate performance.
→ A nonprofit organization called SPEC (System Performance Evaluation Corporation) selects and publishes representative application programs for different application domains, together with test results for many commercially available computers.
→ 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

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} \text{SPEC}_i\right)^{1/n}$$

where n is the number of programs in the suite.

(7) The History of Computer Development:

→ Computers as we know them today have been developed over the past 60 years
→ A long slow evolution of mechanical calculating devices preceded the development of computers
→ Development of the technologies used to fabricate the processors, memories, and I/O units of computers has been divided into four generation.

(i) First Generation (1945-55)
(ii) Second Generation (1955-65)
(iii) Third generation (1965-75)
(iv) Fourth Generation (1975 - present)

(i) The First Generation:

→ The key concept of a stored program was introduced by John von Neumann
→ Programs and their data were located in the same memory, as they are today.
→ Assembly language was used to prepare programs and was translated into machine language for execution.
→ Basic arithmetic operations were performed in a few milliSeconds using vacuum tube technology to implement logic functions.
→ Magnetic core memories and magnetic tape storage devices were also developed.

(ii) The Second Generation:

→ The transistor was invented at AT&T Bell Laboratories in the late 1940s and quickly replaced the vacuum tube.
→ This basic technology shift marked the start of the second generation.
→ Magnetic core memories and magnetic drum storage devices were more widely used in the second generation.
→ High-level languages such as FORTRAN were developed.
→ Compilers were developed to translate these high-level language programs into a corresponding assembly language program.
→ Separate I/O processors were developed.
→ IBM became a major computer manufacturer during this time.

(iii) The Third Generation:

→ The ability to fabricate many transistors on a single silicon chip, called Integrated Circuit (IC) technology developed.
→ Integrated circuit memories began to replace magnetic core memories.
→ Other developments included the introduction of microprogramming, parallelism, and Pipelining.
→ Operating System software allowed efficient sharing of a Computer System by several user programs.
→ Cache and virtual memories were developed.
→ Cache memory makes the main memory appear faster than it really is, and virtual memory makes it appear larger.
→ System 360 mainframe computers from IBM and the line of PDP mini-computers from Digital Equipment Corporation were dominant commercial products of the third generation.

(iv) The Fourth Generation:

→ Tens of thousands of transistors could be placed on a single chip, and the name Very Large Scale Integration (VLSI) was coined to describe this technology.

→ VLSI technology allowed a complete processor to be fabricated on a single chip called Microprocessor.

→ Companies such as Intel, National Semiconductor, Motorola, Texas Instruments, and Advanced Micro devices, were the driving forces of this technology.

→ Organizational concepts such as concurrency, pipelining, caches, and virtual memories evolved to produce the high-performance computing systems of today.

→ Portable Notebook computers, desktop PCs, and workstations, interconnected by local area networks (LAN), WAN (wide area networks), and the Internet.

→ Centralized computing on mainframes is now used primarily for business applications in large companies.

COMPUTER ORGANIZATION UNIT-2

SIR C R REDDY COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Dr DEEPAK NEDUNURI ELURU.

Computer Organization

ASR-CO-2.1

UNIT-2

JNTUK
R16
2-2 CSE

Syllabus : Machine Instruction and Programs

- Instructions and Instruction Sequencing
  - Register Transfer Notation
  - Assembly Language Notation
  - Basic Instruction types
- Addressing Modes
- Basic Input Output Operations
- The Role of Stacks and Queues in Computer Programming Equation
- Component of Instructions
  - Logic Instructions
  - Shift and Rotate Instructions.

Text Book :

1. Computer Organization, Carl Hamacher, Zvonko Vranesic, Safea Zaky, 5th Edition, McGraw Hill

(1) Instructions and Instruction Sequencing:

→ A computer must have instructions capable of performing four types of operations.

- Data transfers between the memory and the processor registers
- Arithmetic and Logic operations on data
- Program Sequencing and Control.
- I/O Transfers.

(i) Register Transfer Notation:

→ We need to describe the transfer of information from one location in the computer to another.

→ Possible locations that may be involved in such transfers are memory locations, processor registers, (or) registers in the I/O subsystem.

→ Names for the addresses of memory locations may be LOC, PLACE, A, VAR2
Processor register names may be R_0, R_5 and
I/O register names may be DATAIN, OUTSTATUS, and so on.

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

Example: $R1 \leftarrow [LOC]$

- means that the contents of memory location LOC are transferred into processor register R1.

Example: $R3 \leftarrow [R1] + [R2]$

- means the operation that adds the contents of registers R1 and R2 and then places their sum into register R3.

→ This type of notation is known as Register Transfer Notation (RTN)

→ The right-hand side of RTN is always denotes a value and the left-hand side is the name of a location where the value is to be placed.

(ii) Assembly Language Notation:

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

Example: MOVE LOC, R1

- Here data transferred from memory location LOC to processor register R1
- The contents of LOC are ~~not~~ unchanged by the execution of this instruction, but the old contents of Register R1 are overwritten.

Example: Add R1, R2, R3

- Here, adding two numbers contained in processor registers R1 and R2 and placing their sum in R3 can be specified by the assembly language statement

~~Add~~ R3,

(iii) Basic Instruction Types:

→ There are 4 types of Instructions available

a) Three-address Instructions
b) Two-address Instructions
c) One-address Instructions
d) Zero-address Instructions

a) Three-address Instructions:

→ An instruction have three operands

Syntax: opcode source1, source2, Destination

Example: Adding two numbers

ADD A,B,C (∵ C ← [A] + [B])

- where A,B are called Source operands, C is called destination operand.
- opcode means operation code.

b) Two-address Instructions:

→ An instruction having two operands

Syntax: opcode Source, Destination

Example: MOVE B,C (∵ C ← [B])
ADD A,C (∵ C ← [A] + [C])

c) One-address Instructions:

→ An instruction having only one operand
→ When a second operand is needed, as in the case of an ADD instruction, a processor register, called Accumulator is used.

Example: LOAD A (∵ AC ← [A])
ADD B (∵ AC ← [AC] + [B])
STORE C (∵ C ← [AC])

d) Zero-address Instructions:

→ An instruction having zero operands.
→ It uses stack operations PUSH and POP to perform operations.

Example:

```
PUSH A    (∵ TOS ← [A])
PUSH B    (∵ TOS ← [B])
ADD       (∵ TOS ← [A] + [B])
POP C     (∵ C ← [TOS])
```

Another Example: Evaluate X = (A+B) * (C+D)

Three Address:

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

Two Address:

```
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:

```
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:

```
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])
```

(2) Addressing Modes:

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

→ The different Generic Addressing Modes is given as

S.NO	Name	Assembler Syntax	Addressing function
1	Immediate	#value	Operand = value
2	Register	R_i	EA = R_i
3	Absolute (Direct)	LOC	EA = LOC
4	Indirect	(R_i) (LOC)	EA = $[R_i]$ EA = [LOC]
5	Index	$X(R_i)$	EA = $[R_i] + X$
6	Base with Index	(R_i, R_j)	EA = $[R_i] + [R_j]$
7	Base with index and offset	$X(R_i, R_j)$	EA = $[R_i] + [R_j] + X$
8	Relative	$X(PC)$	EA = $[PC] + X$
9	Auto Increment	$(R_i)+$	EA = $[R_i]$; Increment R_i
10	Auto Decrement	$-(R_i)$	Decrement R_i EA = $[R_i]$

(i) Implementation of Variables and Constants:

→ In Assembly language, a variable is represented by allocating a register (or) a memory location to hold its value.

→ There are two addressing modes to access variables

a) Register mode

b) Absolute mode.

*) Register Mode:

->The operand is the contents of a processor Register

Example: MOVE R1, R2

- Here R_1, R_2 are registers.

*) Absolute (Direct) Mode:

→ The operand is in a memory location

Example: ADD A,B

→ Address and Data constants are represented in Assembly language using the Immediate mode.

*) Immediate Mode:

→ The operand is given explicitely in the instruction.

Example: MOVE #200, R0

→ places the value 200 in Register R0

→ # Sign indicates that this value is to be used as an immediate operand.

Example:

```
MOVE  B, R1
ADD   #7, R1
MOVE  R1, A
```

(ii) Indirection and pointers:

→ In Addressing modes, the instruction does not give the operand (or) its address explicitly.

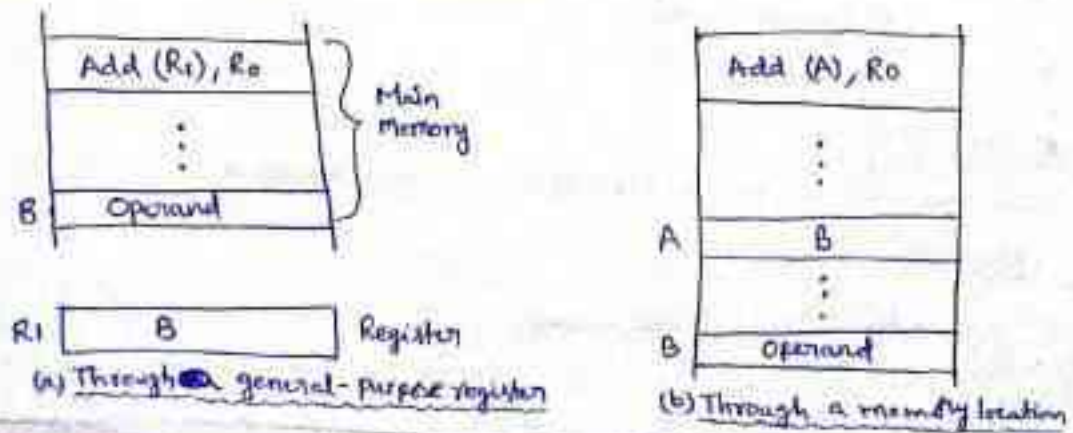
→ Instead, it provides information from which the memory address of the operand can be determined. This address is called as Effective Address(EA)

*) Indirect Mode:

→ The effective address of the operand is the contents of a register (or) memory location whose address appears in the instruction

→ We denote indirection by placing the name of the register (or) the memory address given in the instruction in parantheses.

→ Indirect Addressing is depicted as



(a) - To execute the Add instruction, the processor uses the value of B, which is register R1, as the effective address of the operand.
- It requests a read operation from the memory to read the contents of location B.
- The value read is the desired operand, which the processor adds to the contents of register R0.

(b) - In Indirect Addressing through a memory location, the processor first reads the contents of memory location A, then requests a second read operation using the value B as an address to obtain the operand.
- The register (or memory location that contains the address of an operand is called a pointer.

→ Consider the C-language statement

A = *B;

- where B is a pointer variable
- This statement may be compiled into

MOVE B, R1
MOVE (R1), A

- Using indirect addressing through memory,

MOVE (B), A

(iii) Indexing and Arrays:

→ It is useful in dealing with lists and Arrays

*) Index Mode:

→ The effective address of the operand is generated by adding a constant value to the contents of a register
→ The register used may be either a special register (or) general-purpose register called Index register.
→ Index mode Symbolically represented as,

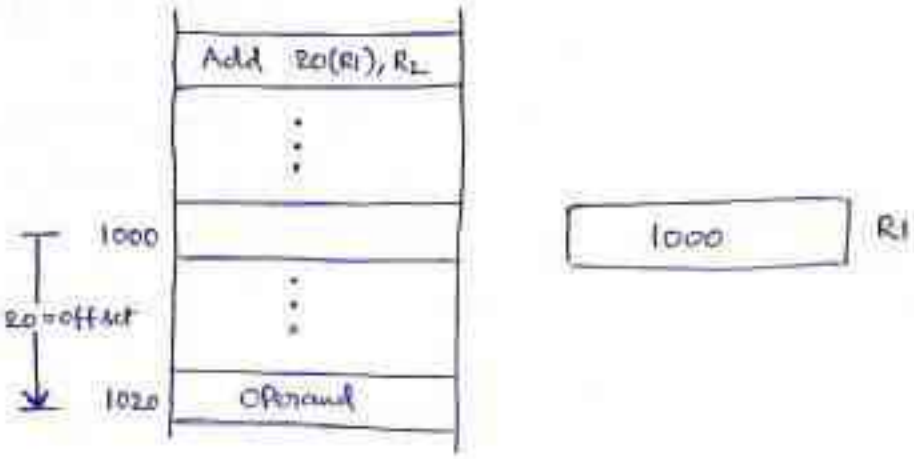
X(Ri)

where X denotes the constant value contained in the instruction
Ri denotes the name of the register involved.
→ The effective address of the operand is given by

EA = X + [Ri]

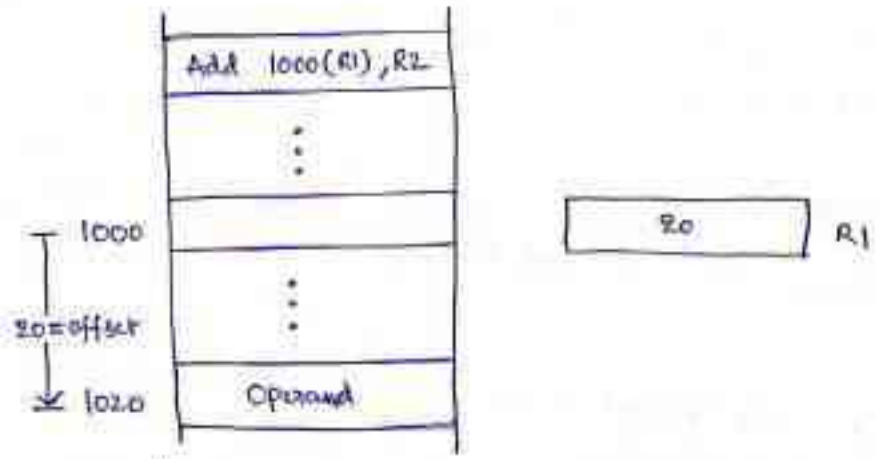
→ The contents of the index register are not changed in the process of generating the effective address.

→ Two ways of using Index mode (Index Addressing) is depicted as,



(a) Offset is given as a Constant

→ The index register R1 contains the address of a memory location, and the value X defines an offset (or) displacement) from this address to the location where the operand is found.



(b) offset is in the Index register

→ Here, the constant X corresponds to a memory address, and the contents of the index register define the offset to the operand.

→ In either case, the effective address is the sum of two values
- one is given explicitly in the instruction, and
- the other is stored in a register.

(iv) Relative Addressing:

→ Index mode is used for general-purpose processor registers

→ In Relative Addressing, the program Counter (PC) is used instead of a general purpose register.

*) Relative Mode:

→ The effective address is determined by the Index mode using the Program Counter in place of general purpose register Ri

→ Relative mode Symbolically represented as,

$$X(PC)$$

→ The effective address of the operand is given by

$$EA = X + [PC]$$

→ This mode can be used to access data operands

(v) Additional Modes:

(a) Auto increment Mode

(b) Autodecrement Mode

*) Autoincrement Mode:

→ The effective address of the operand is the contents of a register specified in the instruction

→ After Accessing the operand, the contents of this register are automatically incremented to point to the next item in the list.

→ Symbolically represented as

$$(Ri)+$$

- It normally increments one, but in byte-sized operands (or) byte-addressable memory

  1 for 8-bit operands
  2 for 16-bit operands
  4 for 32-bit operands.

→ The effective address of the operand is

$$EA = [Ri];$$
Increment Ri

(*) Autodecrement Mode:

→ The contents of a register specified in the instruction are first automatically decremented and are then used as the effective address of the operand

→ Symbolically represented as,

$$-(R_i)$$

→ The effective address of the operand is

Decrement R_i;
$EA = [R_i]$

(3) Basic Input/Output Operations:

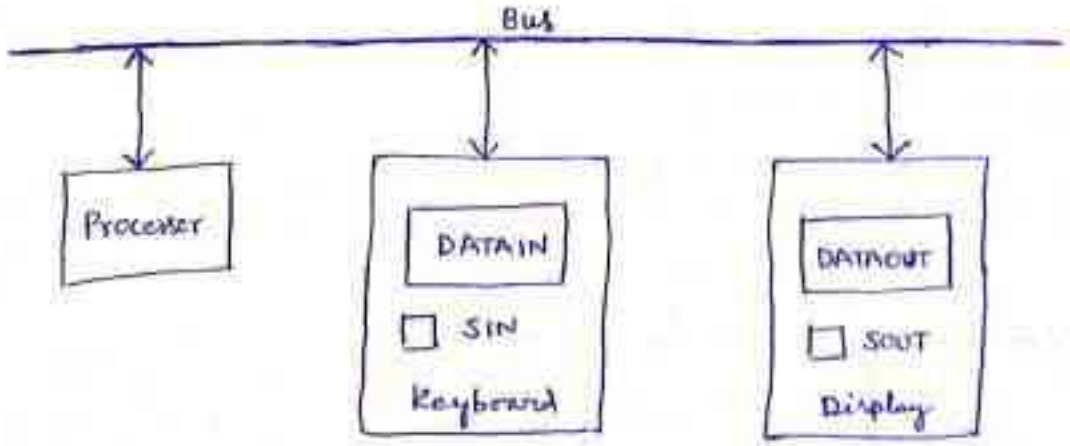
→ I/O operations are essential, and the way they are performed can have a significant effect on the performance of the computer.

→ Consider a task that reads in character input from a keyboard and produces character output on a display screen. A simple way of performing such I/O tasks is to use a method known as Program-Controlled I/O

→ The rate of data transfer from the keyboard to a computer is limited by the typing speed of the user, which is unlikely to exceed a few characters per second.

→ The rate of output transfers from the computer is determined by the rate at which characters can be transmitted over the link between the computer and the display device, typically several thousand characters per second.

→ Bus connection for processor, keyboard, and display are depicted as,



→ The Keyboard and the display are separate devices.
→ Consider the problem of moving a character code from the Keyboard to the processor.
→ Striking a key stores the corresponding character code in an 8-bit buffer register associated with the Keyboard.
→ Let us call this register DATAIN
→ To inform the processor that a valid character is in DATAIN, a status control flag, SIN, is set to 1.
→ A program monitors SIN, and when SIN is set to 1, the processor reads the contents of DATAIN
→ When the character is transferred to the processor, SIN is automatically cleared to 0.
→ If the second character is entered at the keyboard, SIN is again set to 1 and the process repeats.
→ When characters are transferred from the processor to the display, a buffer register, DATAOUT, and the status control flag, SOUT, are used for this transfer.
→ The buffer registers DATAIN and DATAOUT and the status flags SIN and SOUT are part of circuitry commonly known as a device Interface.

Example:

→ The processor can monitor the keyboard status flag SIN and transfer a character from DATAIN to Register R1 by the following sequence of operations

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

→ Sequence of operations used for transferring output to the display is given as,

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

→ The contents of the keyboard character buffer DATAIN can be transferred to Register R1 in the processor by the instruction

```
MOVEBYTE   DATAIN, R1
```

→ Similarly, the contents of Register R1 can be transferred to DATAOUT by the instruction

```
MOVEBYTE   R1, DATAOUT
```

(4) The Role of STACKS and QUEUES in Computer Programming Equations

→ In order to organize the control and information linkage between the main program and the subroutine, a data structure called a stack is used.

→ A stack is a list of data elements, words (or) bytes, with the accessing restriction that elements can be added (or) removed at one end of the list only.

→ It uses a pointer variable called Top of the Stack.

Example :- pile of Trays in a cafeteria

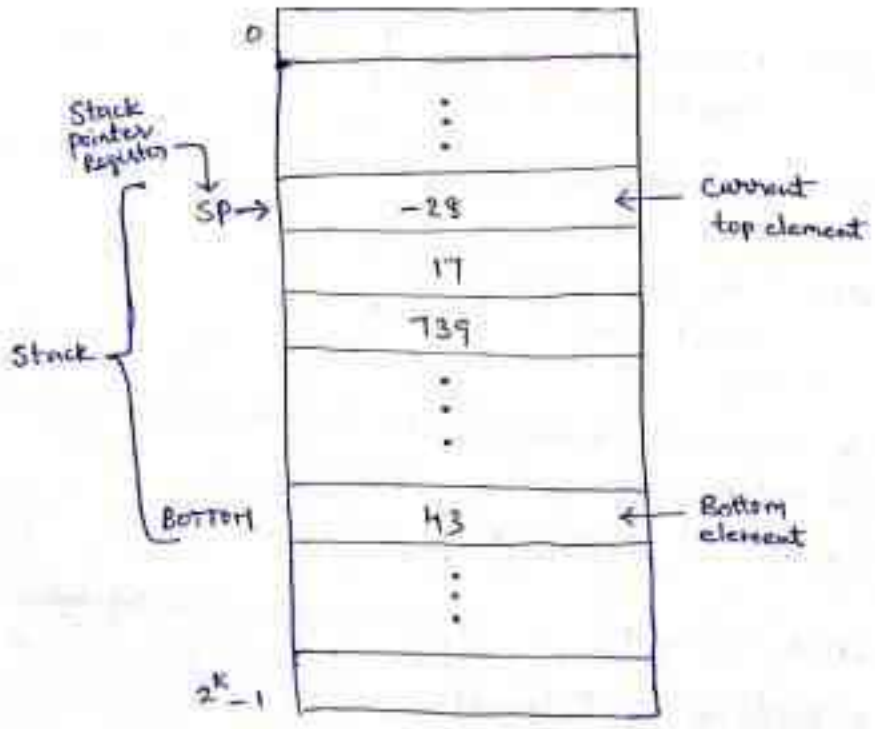
- Customer pick up new trays from the top of the pile, and clean trays are added to the pile by placing them onto the top of the pile.

→ Stack uses LIFO (Last In First Out) mechanism, which describes the last data item placed on the stack is the first one removed when retrieval begins

→ PUSH and POP are the operations of stack, which describes placing the new item on the stack and removing the top item from the stack, respectively

→ Data stored in the memory of a computer can be organized as a stack, with successive elements occupying successive memory locations.

→ A stack of words in the memory is depicted as



→ It contains numerical values, with 43 at the bottom and -28 at the top.
→ A processor register is used to keep track of the address of the element of the stack that is at the top at any given time, called Stack pointer (SP)
→ In a byte-addressable memory with a 32-bit word length, the PUSH operation can be implemented as,

```
SUB  #4, SP
MOVE NEWITEM, (SP)
```

→ The POP operation can be implemented as

```
MOVE (SP), ITEM
ADD  #4, SP
```

→ If the processor has the Autoincrement and Autodecrement addressing modes, then the PUSH operation can be performed by the single instruction

```
MOVE NEWITEM, -(SP)
```

→ The pop operation can be performed by the single instruction

```
MOVE (SP)+, NEWITEM
```

→ Another useful datastructure that is similar to the stack is called Queue
→ Data are stored in and retrieved from a queue on a First In First Out (FIFO) basis.
→ Here, new data are added at the back (high-address end) and retrieved from the front (low-address end) of the queue.
→ It uses two pointers FRONT and REAR.
→ INSERT and DELETE are the operations of queue.

⑤ Components of Instructions:

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

(i) Logic Instructions:

→ Logical operations such as AND, OR, and NOT, applied to individual bits, are the basic building blocks of digital circuits

→ It is also useful to be able to perform logic operations in software

Example: 1's complement

```
NOT DST
```

- Complements all bits contained in the destination operand, changing 0's to 1's and 1's to 0's

Example: 2's Complement

```
NOT  R0
ADD  #1, R0
```

→ Many computers have a single instruction for 2's complement

```
NEGATE R0
```

→ Logical operators AND, OR, NOT represented as bits in a table as,

AND

Operand1	Operand2	AND
0	0	0
0	1	0
1	0	0
1	1	1

OR

Operand1	Operand2	OR
0	0	0
0	1	1
1	0	1
1	1	1

NOT

Operand	NOT
0	1
1	0

(ii) Shift and Rotate Instructions:

(a) Shift Instructions:

→ There are many applications that require the bits of an operand to be shifted right (or) left some specified number of bit positions.

→ There are 2 types of Shift Instructions

(*) Logical shift Instructions

(*) Arithmetic Shift Instructions

(*) Logical Shift Instructions:

→ There are 2 Logical shift Instructions

- Logical Shift Left (LShiftL)
- Logical shift Right (LShiftR)

→ These instructions shift an operand over a number of bit positions specified in a count operand contained in the instruction.

• Logical Shift Left (LShiftL):

→ The general form of Logical Shift Left Instruction is

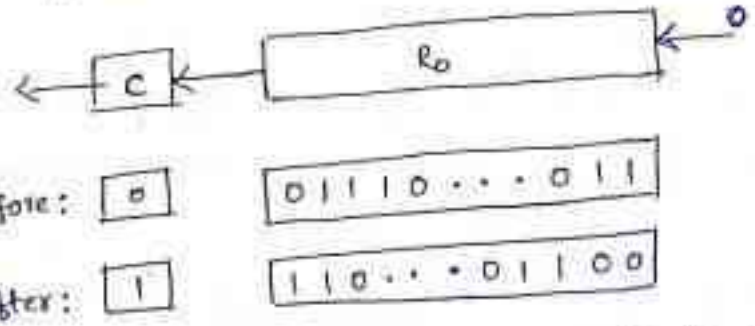
```
LShiftL  count, DST
```

→ The count operand may be an immediate operand (or) it may be contained in a processor register.

Example:

```
LShiftL  #2, R0
```

→ This is depicted as,



Logical Shift Left

→ It shifts the contents of Register R0 left by two positions

→ Vacated positions are filled with 0's.

- Logical Shift Right:

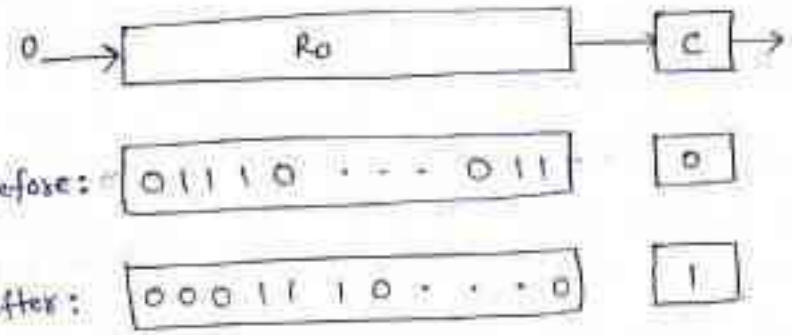
→ The general form is,

LShiftR Count, DST

Example:

LShiftR #2, R0

→ This is depicted as,



Logical Shift Right

→ It shifts the contents of register R0 right by two bit positions.
→ Vacated positions are filled with zeros.

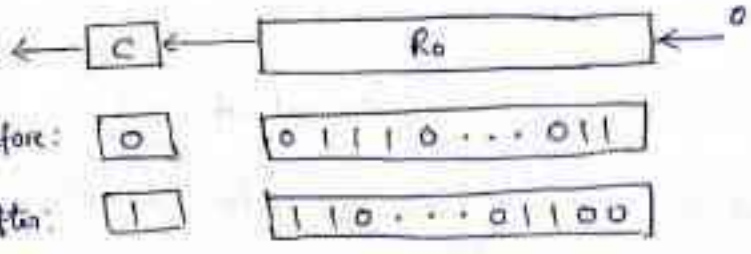
(*) Arithmetic Shift Instructions:

→ There are two types in Arithmetic Shift Instructions
- Arithmetic Shift Left (AShiftL)
- Arithmetic Shift Right (AShiftR)

- Arithmetic Shift Left:

AShiftL #2, R0

→ A Left Arithmetic shift of a binary number by 2 positions
→ The empty positions in the LSB (Least Significant Bit) are filled with zeros.



Arithmetic Shift Left

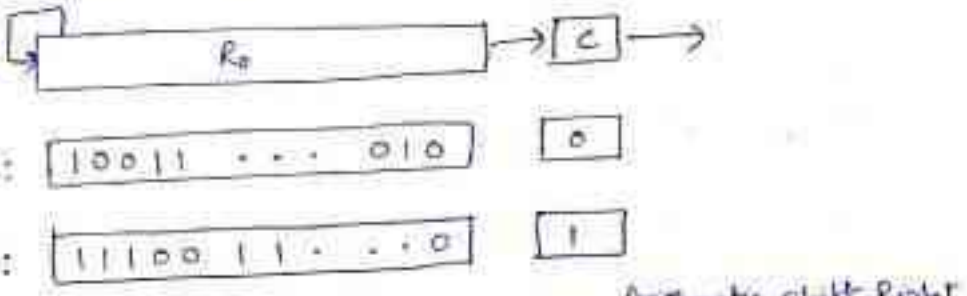
→ It works same like Logical shift Left operation

• Arithmetic Shift Right:

AShiftR #2, R0

→ A right arithmetic shift of a binary number by 2 positions.

→ The empty positions in the MSB (Most Significant Bit) are filled with copies of the original MSB bit.



Arithmetic shift Right

(b) 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.

→ To preserve all bits, a set of rotate instructions can be used.

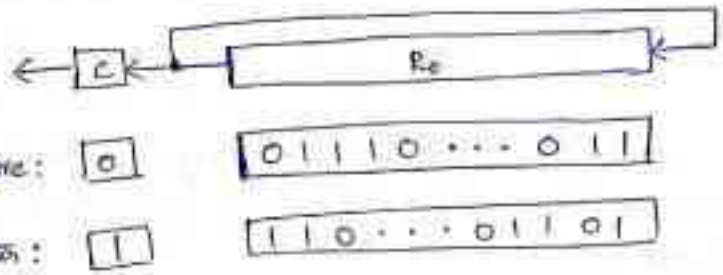
→ They move the bits that are shifted out of one end of the operand back into the other end.

→ The different Rotate Instructions are,

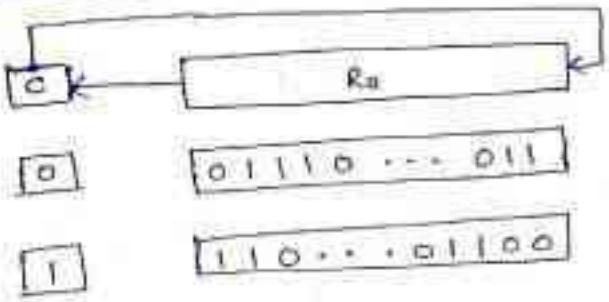
* Rotate Left without Carry (RotateL)
* Rotate Left with Carry (RotateLC)
* Rotate Right without Carry (RotateR)
* Rotate Right with Carry (RotateRC

→ The different Rotate Operations are depicted as,

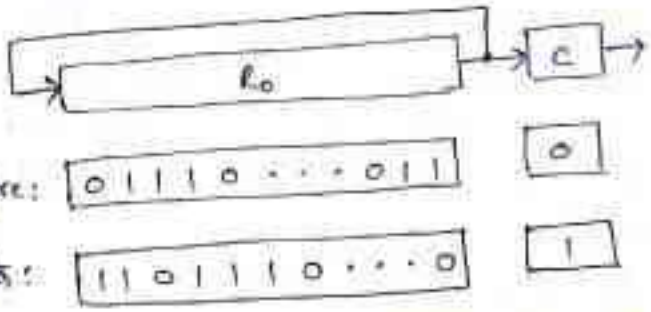
(*) Rotate Left without carry : RotateL #2, R0



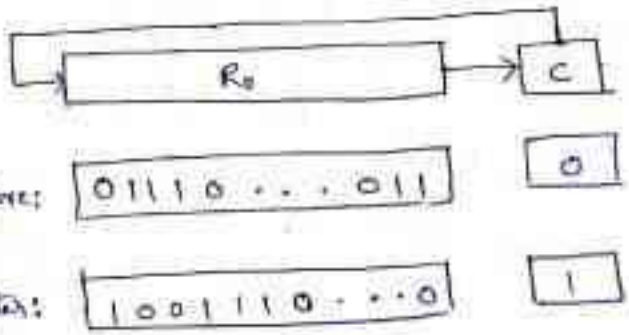
(*) Rotate Left with carry : RotateLC #2, R0



(*) Rotate Right without carry : RotateR #2, R0



(*) Rotate Right with carry : RotateRC #2, R0



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING.

SIR C R REDDY COLLEGE OF ENGINEERING

Dr DEEPAK NEDUNURI

ELURU.

UNIT-3 : Types of Instructions

JNTUK
CSE
2-2
R16

Syllabus :

- Arithmetic and Logic Instructions
- Branch Instructions
- Input/Output operations
- Addressing Modes

Text Book :

1. Computer Organization, Carl Hamacher, Zvonko Vranesic, Safwa Zaky, 5th Edition, McGrawHill.

① Arithmetic and Logic Instructions :

(i) Arithmetic Instructions
(ii) Logic Instructions

(i) Arithmetic Instructions :

* ADD
* SUB
* MUL
* MLA

(*) ADD :

→ The basic Assembly Language expression for arithmetic instruction is

`OPCode Rd, Rn, Rm`

- where the operation specified by the OPCode is performed using the operands in general-purpose registers R_n and R_m.
- The result is placed in Register R_d

Example : ADD R_0, R_2, R_4 ($\because R_0 \leftarrow [R_2] + [R_4]$)

→ Instead of being contained in Register R_m, the second operand can be given in the immediate mode

Example : ADD $R_0, R_3, \#17$ ($\because R_0 \leftarrow [R_3] + 17$)

→ The second operand can be shifted (or) rotated before being used in the Operation.

→ When a shift (or) rotation is required, the instruction is given as

`ADD R0, R1, R5, LSL #4`

- Here, the second operand, which is contained in Register R5, is shifted left 4 bit positions, and it is then added to the contents of Register R1
- the sum is placed in Register R0

(*) SUB:

SUB R_0, R_6, R_5 ($\because R_0 \leftarrow [R_6] - [R_5]$)

(*) MUL:

→ Two versions of a Multiply instruction are provided:

(i) The first version multiplies the contents of two registers and places the low-order 32-bits of the product in a third register
The high-order bits of the product, if there are any, are discarded.

Example:

MUL R_0, R_1, R_2 ($\because R_0 \leftarrow [R_1] \times [R_2]$)

(ii) The Second version specifies a fourth register whose contents are added to the product before storing the result in the destination register.

Example: MLA R_0, R_1, R_2, R_3 ($\because R_0 \leftarrow [R_1] \times [R_2] + [R_3]$)

- This is called Multiply-Accumulate Operation
- It is used in numerical algorithms for digital signal processing.

(ii) LOGIC Instructions:

→ The different Logic Instructions are,

* AND
* OR
* XOR (exclusive OR)
* BIC (Bit-Clear)

→ Logic Instructions have the same format as the Arithmetic Instructions

AND R_d, R_n, R_m ($\because R_d \leftarrow [R_n] \wedge [R_m]$)

- which is a bitwise logical AND between the operands in Registers R_n and R_m

Example:

AND R_0, R_0, R_1 ($\because R_0 \leftarrow [R_0] \wedge [R_1]$)

- If register R_0 contains the hexadecimal pattern 02FA62CA and R_1 contains the pattern 0000FFFF, then the result 0000 62CA being placed in register R_0.

Examples:

(i) OR R_0, R_1, R_2 ($\because R_0 \leftarrow [R_1] \vee [R_2]$)

(ii) XOR R_0, R_1, R_2 ($\because R_0 \leftarrow [R_1] \oplus [R_2]$)

→ The BIC (Bit-clear) instruction is closely related to the AND instruction
→ It complements each bit in operand R_m before ANDing them with the bits in register R_n

BIC R_0, R_0, R_1 ($\because R_0 \leftarrow [R_0] \wedge (\text{NOT}[R_1])$)

- If R_0 is 02FA62CA and R_1 is 0000FFFF, then the result is 02FA0000 being placed in R_0.

→ The MOVE NEGATION instruction, with the Op-code mnemonic MVN, Complements the bits of the source operand and places the result in R_d

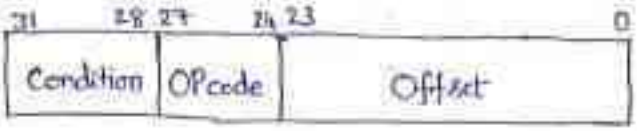
Example: MVN R_0, R_3

→ If the contents of R3 are the hexadecimal pattern 0F0F0F0F, then it places the result F0F0F0F0 in register R0

(2) Branch Instructions.

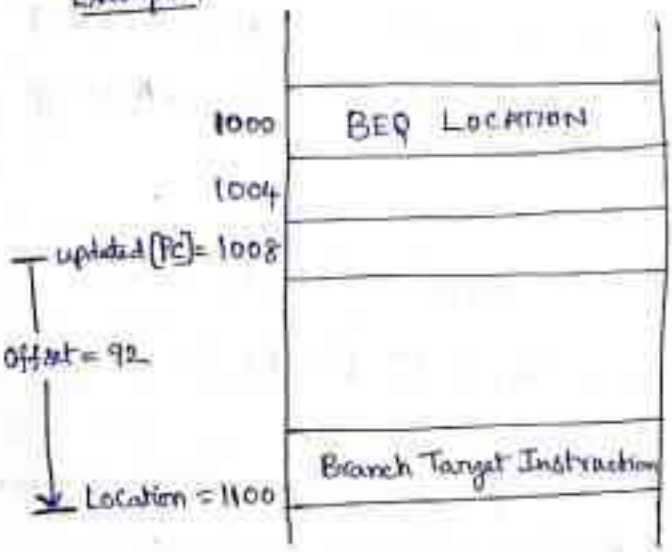
→ Conditional branch instructions contain a signed, 2's complement, 24-bit offset that is added to the updated contents of the program counter (PC) to generate the branch target address

→ The format for the branch instructions is depicted as,



Instruction Format

Example:



Determination of a Branch Target Instruction

→ The BEQ (Branch if Equal to 0) causes a branch if the Z flag is set to 1

→ The condition to be tested to determine whether (or) not branching should take place is specified in the high-order 4-bits, b_{31-28}, of the instruction word.

→ A Branch instruction is executed in the same way as any other ARM instruction, that is, it is executed only if the current state of the condition code flags corresponds to the condition specified in the condition field of the instruction

→ In Example, If the Branch instruction is at address location 1000, and the branch target address is 1100, then the offset has to be 92 because

the contents of the updated PC will be 1000+8 = 1008 when the address 1100 is computed.

(i) Setting Condition Codes:

→ Some instructions, such as COMPARE, given by

CMP R_n, R_m

- which performs the operation,

$[R_n] - [R_m]$

- have the sole purpose of setting the Condition code flags based on the result of the subtract operation.

→ On the other hand, the arithmetic and Logic instructions affect the Condition Code flags only if explicitly specified to do so by a bit in the OP code field.

→ This is indicated by appending the suffix S to the assembly language OP-code mnemonic

Example:

→ The instruction

ADDS R_0, R_1, R_2

sets the condition code flags, but

ADD R_0, R_1, R_2

does not.

③ I/O Operations:

→ The ARM architecture uses memory-mapped I/O

→ Reading a character from a keyboard (or) sending a character to a display can be done using program-controlled I/O

→ Suppose that bit 3 in each of the device status registers INSTATUS (Keyboard) and OUTSTATUS (display) contains the respective control flags SIN and SOUT

→ Also assume that the keyboard DATAIN and display DATAOUT registers are located at addresses INSTATUS +4 and OUTSTATUS +4, immediately following the status register locations.

→ The read and write wait loops can then be implemented as,

Example:

- Assume that address INSTATUS has been located into Register R_1
- The instruction sequence given as,

```
READWAIT  LDR   R3, [R1]
          TST   R3, #8
          BEQ   READWAIT
          LDRB  R3, [R1, #4]
```

- It reads a character into register R3 when a key has been pressed on the keyboard
- The test (TST) instruction performs the bitwise logical AND operation on its two operands and sets the condition code flags based on the result.
- The immediate operand 8 has a single one in the bit 3 position.
- Therefore the result of the TST operation will be zero if bit 3 of INSTATUS is zero and will be non zero if bit 3 is one, signifying that a character is available in DATAIN.
- The BEQ instruction branches back to READWAIT if the result is zero, looping until a key is pressed, which sets bit 3 of INSTATUS to One.

- Assuming that address OUTSTATUS has been loaded into register R2.

The instruction sequence is given as,

```
WRITEWAIT  LDR  R4, [R2]
           TST  R4, #8
           BEQ  WRITEWAIT
           STRB R3, [R2, #4]
```

- It sends the character in Register R3 to the DATAOUT register when the display is ready to receive it.

④ Addressing Modes:

→ The 68000 processor addressing modes is depicted in a table as,

Name	Assembler Syntax	Addressing Modes
Immediate	#value	Operand = Value
Absolute Short	Value	EA = Sign Extended .WValue
Absolute Long	Value	EA = Value
Register	R_n	EA = R_n (∴Operand = $[R_n]$)
Register Indirect	(A_n)	EA = $[A_n]$
Indexed basic	WValue (A_n)	EA = WValue + $[A_n]$
Indexed Full	BValue $(A_n, R_K.S)$	EA = BValue + $[A_n]$ + $[R_K]$
Relative basic	WValue (PC) (or) Label	EA = WValue + [PC]
Relative Full	BValue $(PC, R_K.S)$ (or) Label (R_K)	EA = BValue + [PC] + $[R_K]$
Autoincrement	$(A_n)+$	EA = $[A_n]$ Increment A_n ;
Autodecrement	$-(A_n)$	Decrement A_n EA = $[A_n]$;

→ The 68000 processor have several addressing modes.

(1) Immediate Mode:

- The operand is contained in the instruction
- Four sizes of operands can be specified, Byte, Word, Long-word, OP-code word.
- The fourth size consists of very small numbers that can be

included directly in the OP-code word of some instructions.

(ii) Absolute Mode : (Direct Mode)

- The absolute address of an operand is given in the instruction, following the OP code.
- There are two versions of this mode - long and short.
- In the long mode, a 24-bit address is specified explicitly.
- In the short mode, a 16-bit value is given in the instruction to be used as the low-order 16-bits of an address.
- The sign bit of this value is extended to provide the high-order eight bits of the address.
- Since the sign bit is either 0 (or) 1, it follows that in the short mode only two pages of the addressable space can be accessed.
- These are the 0 page and the FF8 page, each containing 32K bytes.

(iii) Register Mode :

- The operand is in a processor register specified in the instruction.

(iv) Register Indirect mode:

- The effective address of the operand is in an address register specified in the instruction.

(v) Basic Index Mode:

- A 16-bit signed offset and an address register, An, are specified in the instruction
- The sum of this offset and the contents of An is the effective address of the operand.

(vi) Full Index Mode:

- An 8-bit signed offset, an address register An, and an index register R_K (either an address (or) a data register) are given in the instruction.
- The effective address of the operand is the sum of the offset and the contents of registers An and R_K
- Either all 32-bits (or) the sign-extended low-order 16 bits of R_K are used in the derivation of the address.

(vii) Basic Relative Mode:

→ This is same as the basic index mode except that the program Counter is used instead of an address register, An.

(viii) Full Relative Mode:

→ This is same as the full index mode except that the program Counter is used instead of an address register, An.

(ix) Autoincrement Mode:

- The effective address of the operand is in an address register, An, specified in the instruction
- After the operand is accessed, the contents of An are incremented by 1, 2, (or) 4, depending on whether a byte, a word, (or) a Long-word operand, respectively, is involved.

(x) Autodecrement Mode:

- The contents of an address register, An, specified in the instruction are decremented by 1, 2, (or) 4, depending on whether a byte, a word, (or) a long-word operand, respectively, is involved.
- The effective address of the operand is the decremented contents of An.

ER ORGANIZATION UNIT-4

SIR C R REDDY COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Dr DEEPAK NEDUNURI ELURU.

Computer Organization (CO)

UNIT-4

I/O Organization

Syllabus:

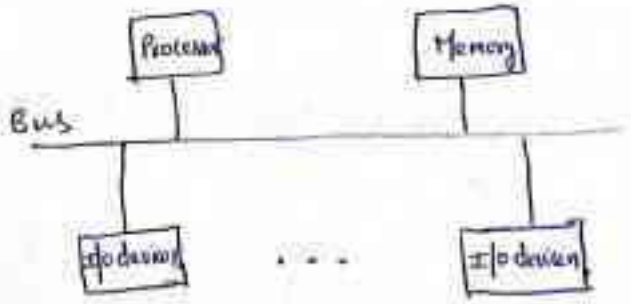
- Accessing I/O devices
- Interrupts
  - Interrupt Hardware
  - Enabling and Disabling Interrupts
  - Handling Multiple devices. ✓
- Direct Memory Access ✓
- Buses
  - Synchronous Bus
  - Asynchronous Bus ✓
- Interface Circuits ✓
- Standard I/O Interface ✓
  - Peripheral Component Interconnect (PCI)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

① Accessing I/O devices:

→ The basic feature of a computer is its ability to exchange data with other devices

→ The bus enables all the devices connected to it to exchange information.

→ A single bus structure is depicted as



→ The bus consists of 3 sets of lines

- Address lines
- Data lines
- Control lines

- Each I/O device is assigned a unique set of addresses
- when I/O device and the memory share the same address space, the arrangement is called Memory-mapped I/O.

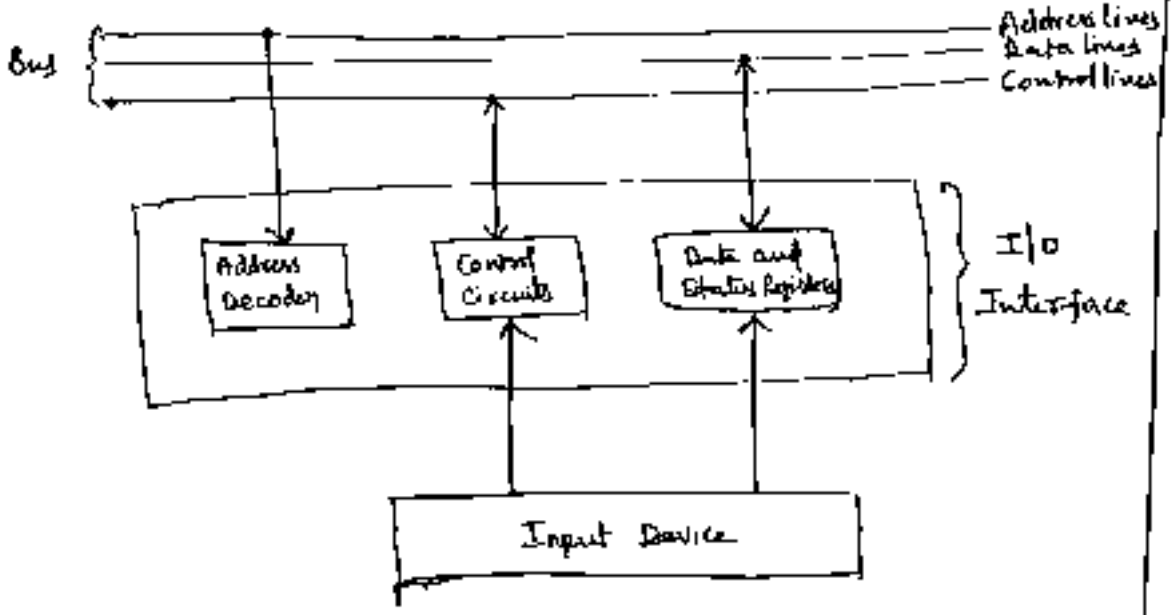
Memory-mapped I/O:

- With Memory-mapped I/O, any machine instruction that can access memory can be used to transfer data to (or) from an I/O device.

Example:

```
MOVE  DATAIN, R0  (Input Device to Processor Register)
MOVE  R0, DATAOUT (Processor Register to Output Device)
```

- The I/O Interface for an Input device is depicted as



- Here, the address decoder enables the device to recognize its address when this address appears on the address lines.
- The data register holds the data being transferred to (or) from the processor
- The Status register contains information relevant to the operation of the I/O device.

② Interrupts :

→ An Interrupt will stop the continuous progress of an activity (or) process.

→ An Interrupt is a signal to the processor emitted by hardware (or) software Indicating an event that needs immediate attention.

→ Alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing.

→ The bus control line, called interrupt-request line is used for interrupts.

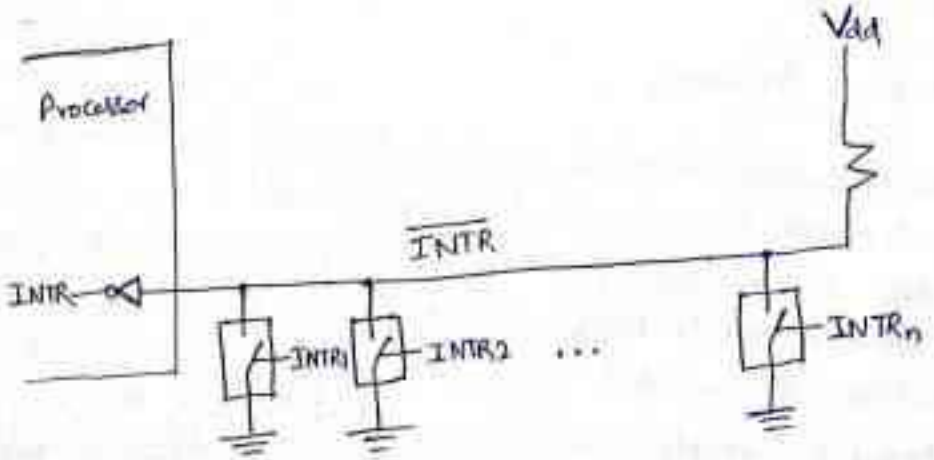
i) Interrupt Hardware :

→ An I/O device requests an Interrupt by activating a bus line called interrupt-request.

→ Most Computers are likely to have several I/O devices that can request an interrupt.

→ 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 depicted as



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

→ To request an Interrupt, a device closes its associated switch.

→ Thus, if all interrupt-request signals $INTR_1$ to $INTR_n$ are inactive, i.e, if all switches are open, the voltage on the interrupt-request line will be equal to V_{dd}.

→ This is the inactive state of the line.

→ Since the closing of one (or) more switches will cause the line voltage to drop to 0, the value of INTR is the logical OR of the requests

from individual devices, i.e,

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

→ The $\overline{INTR}$ signal is active when in the 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.
- The processor interrupts the program currently being executed.
- Interrupts are disabled by changing the control bits in the PS (Processor Status register)
- The device is informed that its request has been recognized, and in response, it deactivates the interrupt request signal
- The action requested by the interrupt is performed by the interrupt-service routine
- Interrupts are enabled and execution of the interrupt program is resumed.

(iii) Handling Multiple devices:

→ Consider the situation where a number of devices capable of initiating interrupts are connected to the processor.
→ Because these devices are operationally independent, there is no definite order in which they will generate interrupts.
→ This situation is handled by 3 methods.

(a) Vectored Interrupts
(b) Interrupt Nesting
(c) Simultaneous Requests

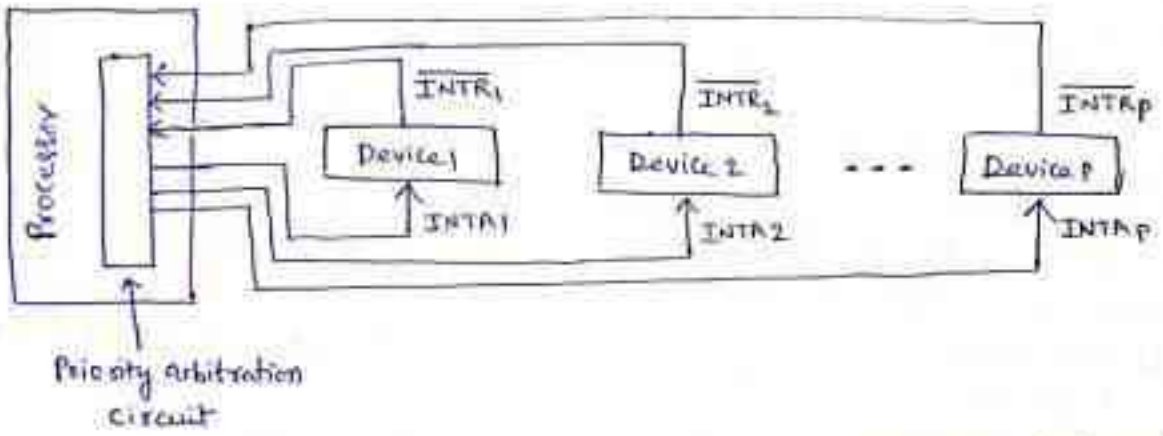
a) Vectored Interrupts:

→ A device requesting an interrupt may identify directly to the processor
→ Then the processor can immediately start executing the corresponding interrupt-service routine.
→ This interrupt handling scheme is known 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.
→ For example, 128 bytes may be allocated to hold a table of 32 interrupt vectors.

b) Interrupt Nesting:

→ Implementation of Interrupt priority using individual interrupt-request and Acknowledge lines is depicted as,

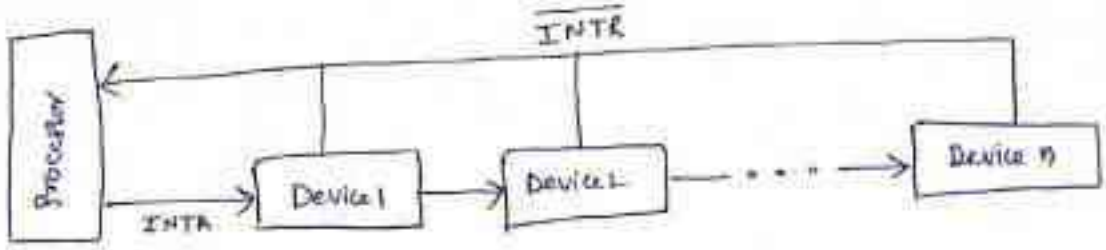


→ Here, Each of the interrupt-request lines is assigned a different Priority level.
→ Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor.
→ A request is accepted only if it has a higher priority level than that currently assigned to the processor.

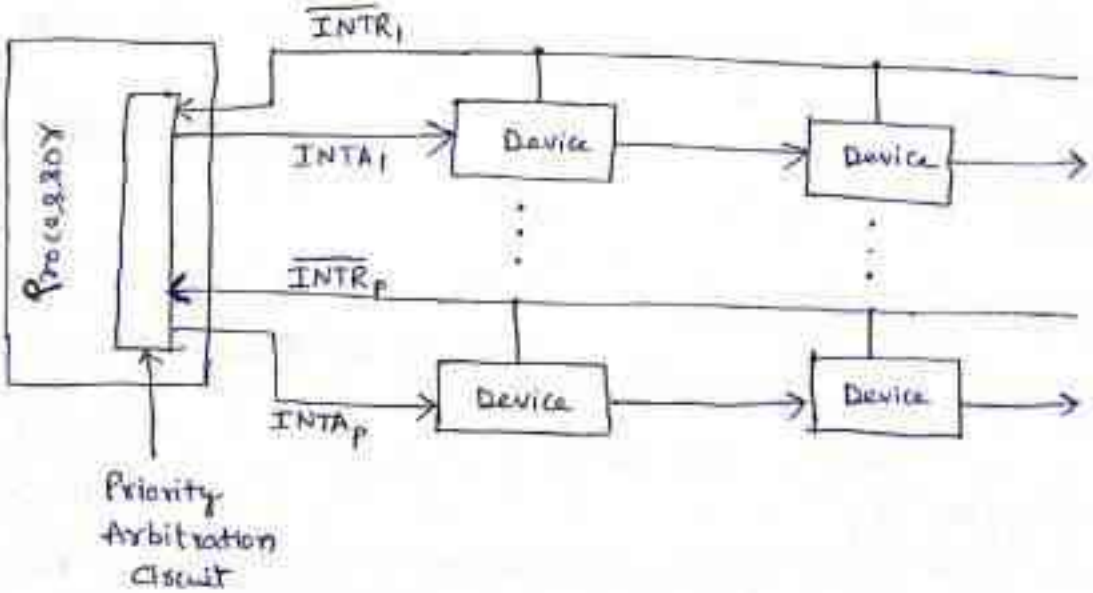
c) Simultaneous Requests:

→ Consider the problem of simultaneous arrivals of interrupt requests from two (or) more devices.
→ The processor must have some means of deciding which request to service first.
→ The processor simply accepts the request having the highest priority.
→ Priority is determined by the order in which the devices are polled, i.e, the polling the status registers of the I/O devices.

→ A Daisy chain priority interrupt Scheme is depicted as,



→ The Interrupt-request line $\overline{INTR}$ is common to all devices.
→ The Interrupt-Acknowledge line INTA, is connected in a daisy-chain fashion.
→ The INTA Signal propagates serially through the devices.
→ When several devices raise an interrupt request and the $\overline{INTR}$ line is activated, the processor responds by setting the INTA line to 1.
→ This Signal is received by device 1.
→ Device 1 passes the Signal on to device 2 only if it doesnot require any Service.
→ In daisy-chain arrangement, the device that is electronically closest to the processor has the highest priority.
→ The second device along the chain has second highest priority, and so on.
→ The arrangement of priority groups is depicted as,



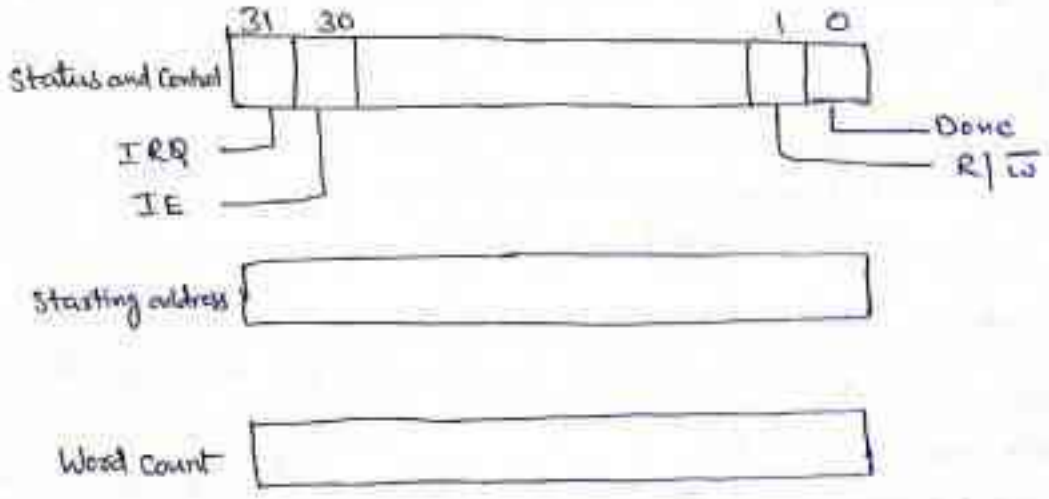
→ Here, Devices are organized in groups, and each group is connected at a different priority level.
→ Within a group, devices are connected in a daisy-chain
→ This organization is used in many Computer Systems.

(3) DMA : (Direct Memory Access)

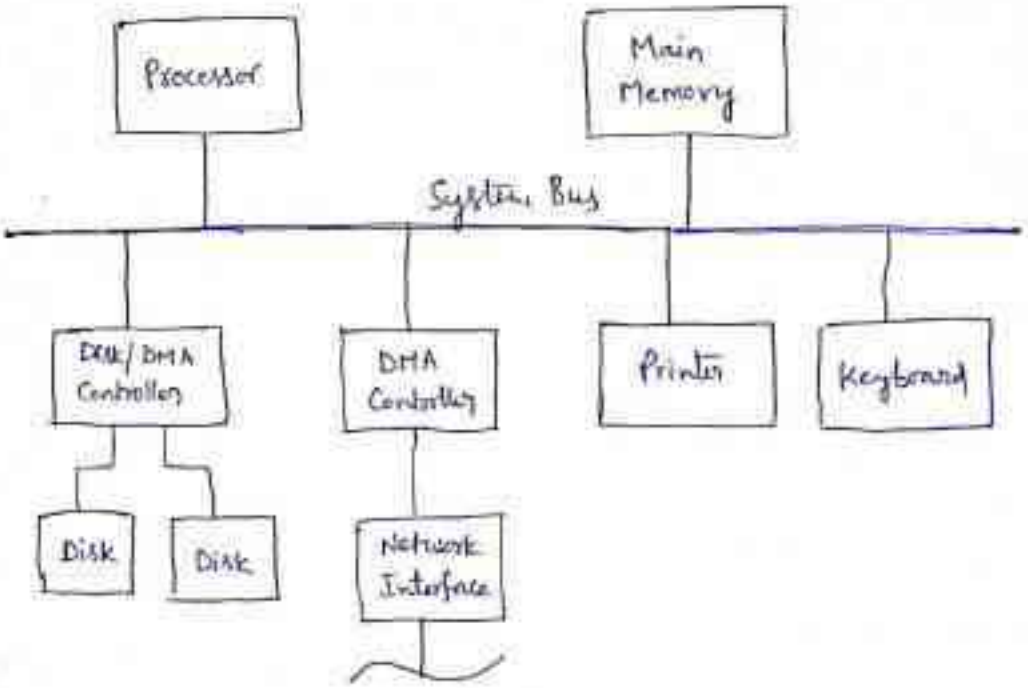
→ To transfer large blocks of data at high speed, a special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor.
→ This approach is called Direct Memory Access (DMA)
→ DMA transfers are performed by a control unit that is part of the I/O device interface, called DMA controller.

(*) DMA Controller:

→ The DMA controller performs the functions that would normally be carried out by the processor when accessing the main memory.
→ For each word transferred, it provides the memory address and all the bus signals that control data transfer.
→ Although the DMA controller can transfer data without intervention by the processor, its operation must be under the control of a program executed by the processor.
→ The Registers used in DMA Controller are depicted as,



→ The use of DMA Controllers in a Computer System is depicted as,



(*) Bus Arbitration:

→ The device that is allowed to initiate data transfers on the bus at any given time is called the Bus Master.

→ Bus arbitration is the process by which the next device to become the bus master is selected and bus mastership is transferred to it

→ The selection of the bus master must take into account the needs of various devices by establishing a priority system for gaining access to the bus.

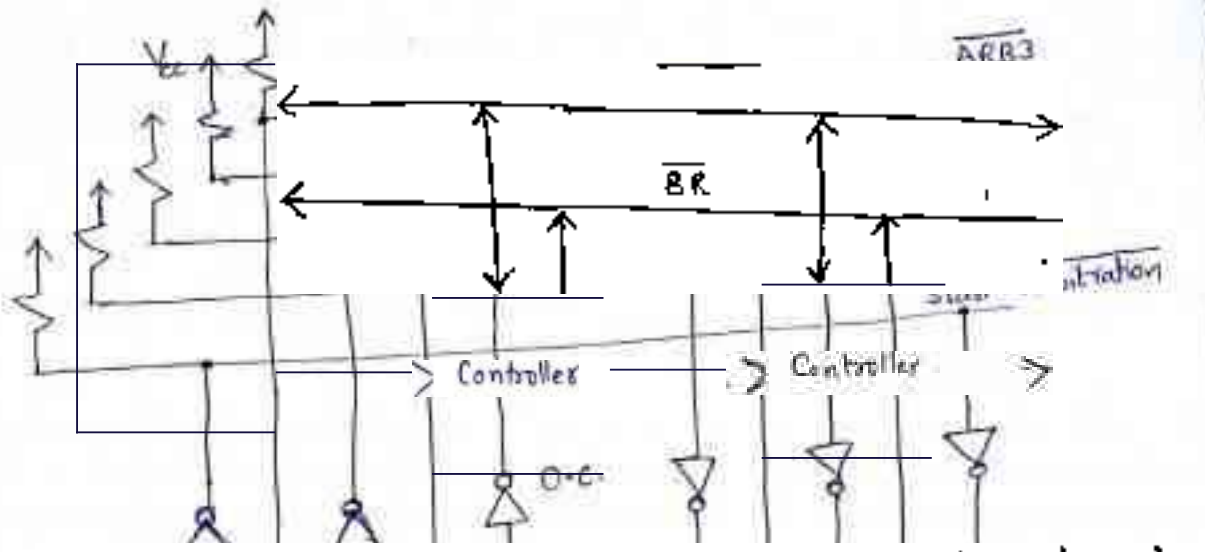
→ There are two approaches to bus arbitration
  (i) Centralized Arbitration
  (ii) Distributed Arbitration

(i) Centralized Arbitration:

→ In this approach, a single bus arbiter performs the required arbitration.

→ A simple arrangement for bus arbitration using a daisy chain is depicted as

ASR-CO-4.10



→ In this case, the processor is normally the bus master unless it grants

by activating the Bus-Request line, BR

→ When Bus-Request is activated, the processor activates the Bus-Grant Signal by activating another OC

→ Hence, after receiving the Bus-Grant signal, a DMA controller waits for Bus-Busy to become inactive, then assumes mastership of the bus.

→ The ... At this time, it activates Bus-Busy to prevent other devices from using

→ Each ... when the bus at the same time.

Start-Arbitration signal and place their 4-bit ID numbers on four open-collector lines, ARB0 through ARB3

→ A winner is selected as a result of the interaction among the signals transmitted over these lines by all contenders.

→ The → Distributed arbitration means that all devices waiting to use the bus request that has the highest ID number.

④ BUSES :

→ The processor, main memory, and I/O devices can be interconnected by means of a common bus.

→ The Bus primary function is to provide a communication path for the transfer of data.

→ The bus includes the lines needed to support interrupts and arbitration.

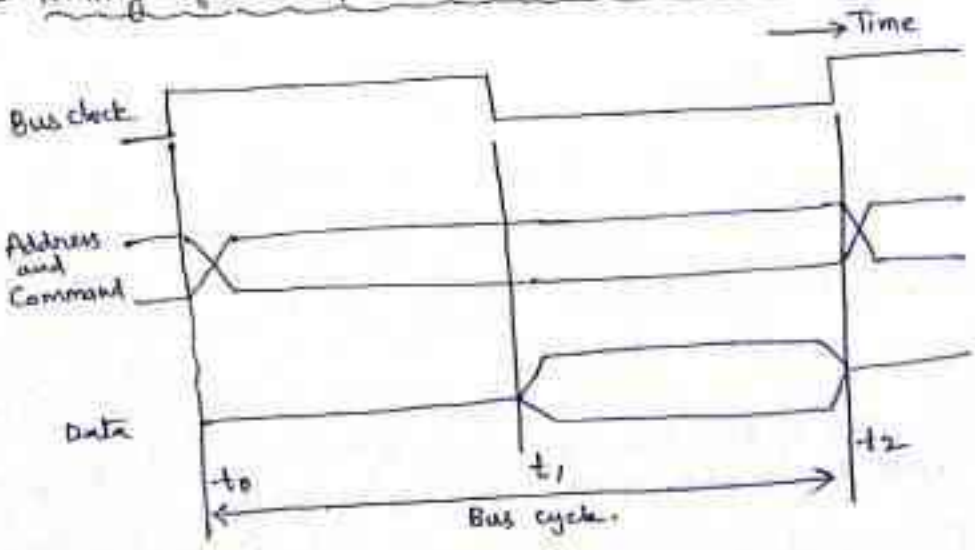
→ The bus lines used for transferring data may be grouped into three types.

- data lines
- address lines
- Control lines

- In any data transfer operation, one device plays the role of a master.
- This is the device that initiates data transfers by issuing read (or) write commands on the bus, hence, it may be called an initiator.
- Normally, the processor (or) devices with DMA capability become bus masters.
- The device addressed by the master is referred to as a slave (or) target

(i) Synchronous Bus :

- In a synchronous bus, all devices derive timing information from a common clock line.
- Equally spaced pulses on this line define equal time intervals.
- In the simplest form of a synchronous bus, each of these intervals constitutes a bus cycle during which one data transfer can take place.
- Timing of an input transfer on a Synchronous bus is depicted as

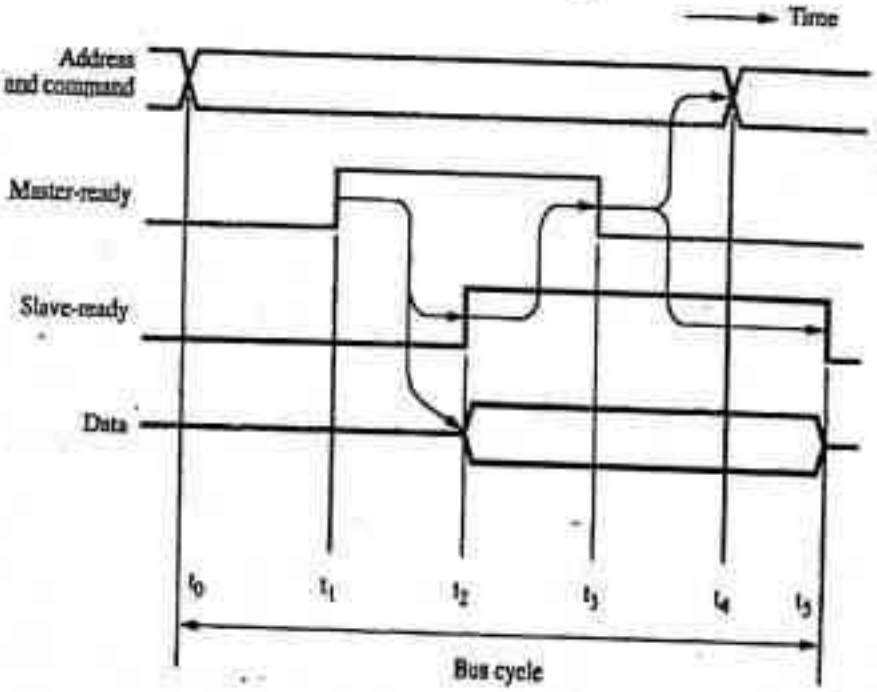


→ The address and datalines are high and low at the same time.
→ This is a common convention indicating that some lines are high and some low, depending on the particular address (or) data pattern being transmitted.
→ The crossing points indicate the times at which these patterns change.
→ A signal line is an indeterminate (or) high-impedance state is represented by an intermediate level half-way between the low and high signal levels.

(ii) Asynchronous Bus:

→ An alternative scheme for controlling data transfer on the bus is based on the use of a handshake between the master and the slave.

→ Handshake control of data transfer during an input operation is depicted as,



→ The common clock is replaced by two timing control lines
- Master-Ready
- Slave-Ready

→ The first is asserted by the master to indicate that it is ready for a transaction, and the second is a response from the slave.

→ In principle, a data transfer controlled by a handshake proceeds as,

- The master places the address and command information on the bus.
- Then it indicates to all devices that it has done so by activating the Master-ready line.
- This causes all devices on the bus to decode the address, data and control.

→ - The master waits for Slave-ready to become asserted before to transfer data between the interface and the I/O

→ This is called a port.

→ It can be classified as

(i) Parallel Port
(ii) Serial Port

→ A parallel port transfers data in the form of a number of bits, typically 8 or 16, simultaneously to or from the device.

→ A serial port transmits and receives data one bit at a time

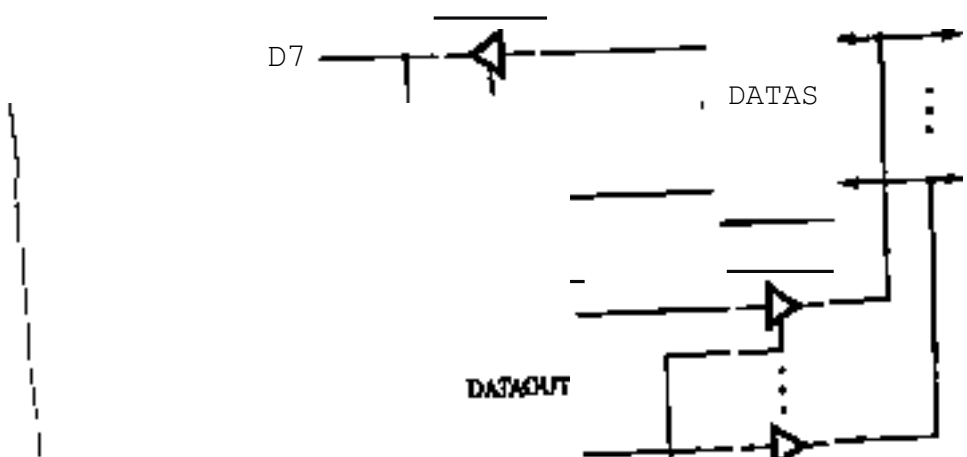
→ Communication with the bus is the same for both formats, the conversion from the parallel to the serial format, and vice versa, takes place inside the interface circuit.

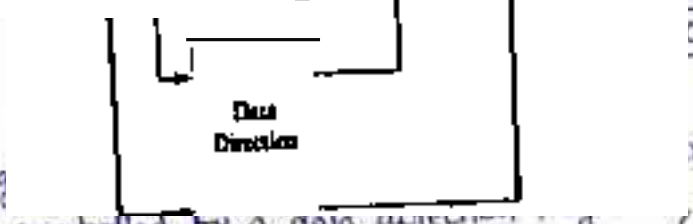
(i) Parallel Port:

→ The selected slave strobes the data into its output buffer when it receives peripherals.

→ The name refers to the fact that the data [illegible] the Slave-ready signal to 1.

→ The remainder of the cycle is identical to the input operation.


D7
DATAS
DATAOUT



→ The DATAOUT reg ... ree-state drivers that are controlled by a data ... DDR.

→ For a given bit, if the DDR value is 1, the corresponding data line acts as an output line, otherwise, it acts as an input line.

→ Two lines C1 and C2, are provided to control the interaction between the interface circuit and the I/O device it serves.

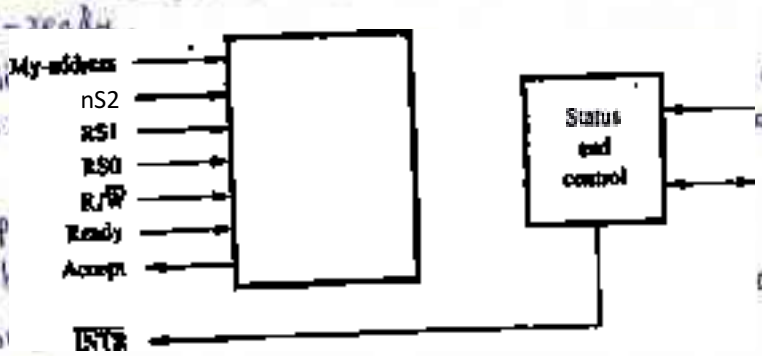
→ Line C2 is bidirectional to provide several different modes of signaling, including the handshake.

→ The Ready and Accept lines are the handshake control lines on the processor bus side, and hence would be connected to Master-ready and Slave-ready.

→ The input si... output of an address ... el to the interface.

→ An interrup...

→ It should ... on the Computer bu...



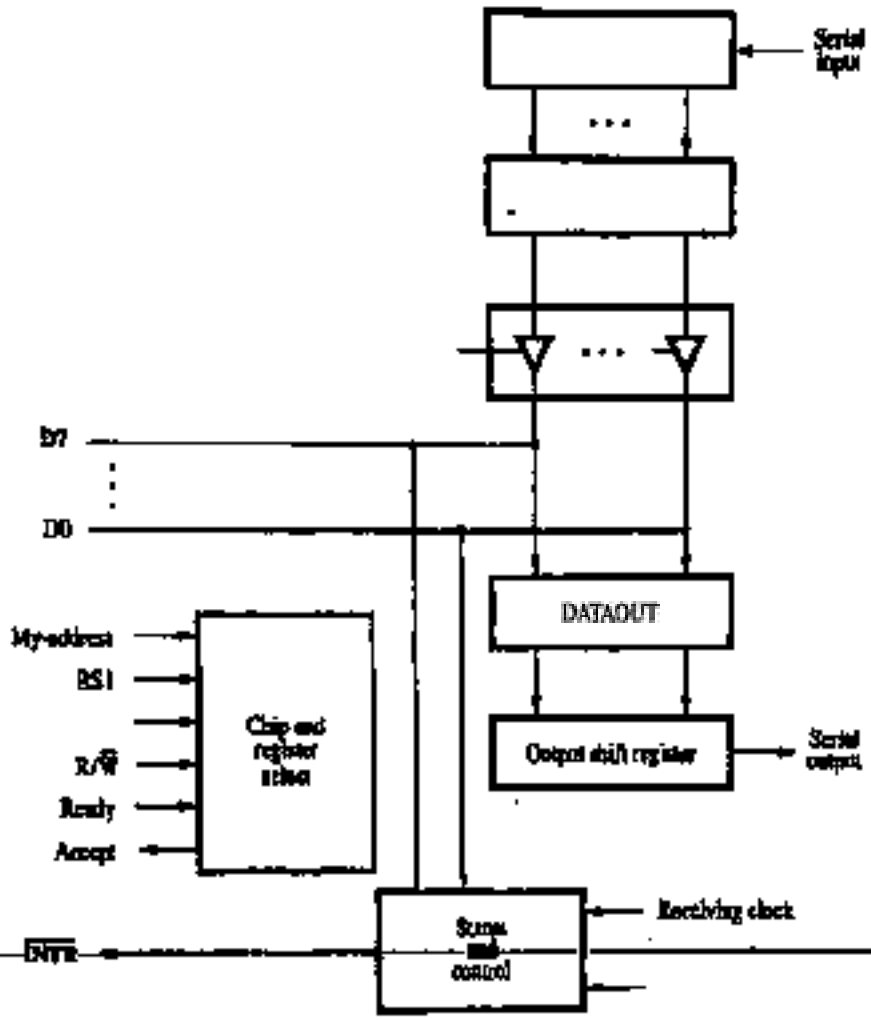
(ii) Serial Port :

→ A serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time.

→ The key feature of an interface circuit for a serial port is that it is capable of communicating in a bit-serial fashion on the device side and in a bit-parallel fashion on the bus side.

→ The transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability.

→ A Serial interface is depicted as,



→ It includes the familiar DATAIN and DATAOUT registers.

→ When all 8 bits of data have been received, the contents of this IN register.

ice.

⑥ Standard I/O Interfaces:

→ A different interface may have to be de
of I/O device and computer, resulting

→ ISA – Industry Standard Architecture
IDE – Integrated Device Electronics

→ The PCI standards defines an expansion bus on the motherboard.

→ The two buses are interconnected by a circuit, which we will call a bridge, that translates the signals and protocols of one bus into

(i) PCI (Peripheral Component Interconnect) Bus:

→ The PCI bus is a good example of a system bus that grew out of the need for standardization.

→ [illegible] processor bus but in a standardized

→ [illegible] as if they

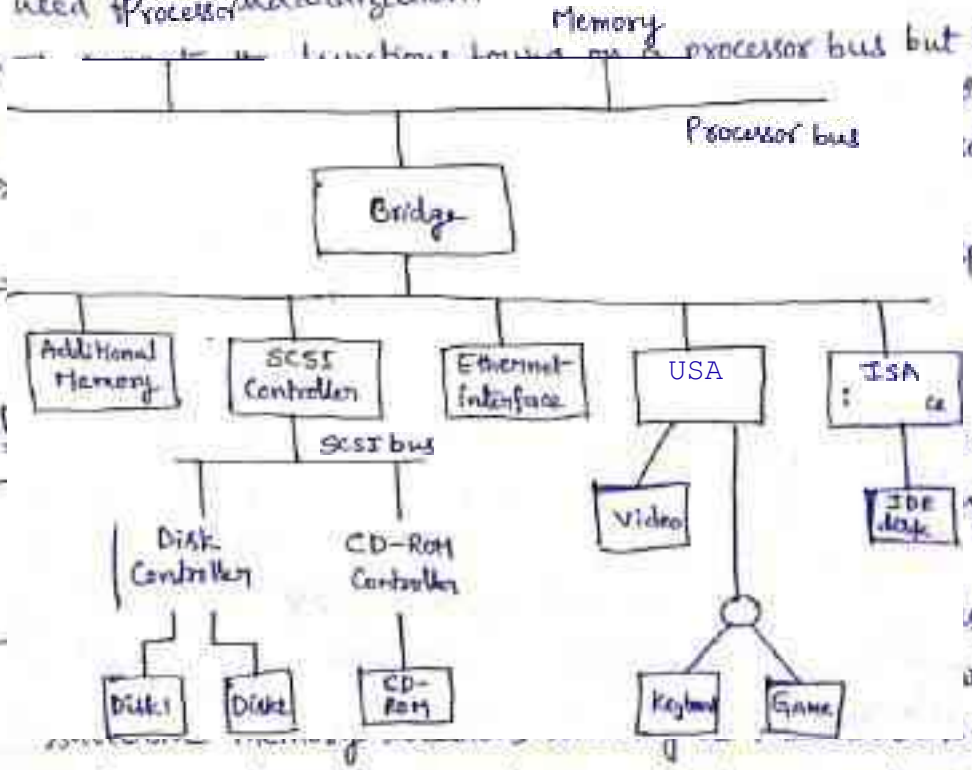
→ [illegible] pace of the

→ [illegible] a read operation

→ [illegible] a sequence

→ [illegible] a memory

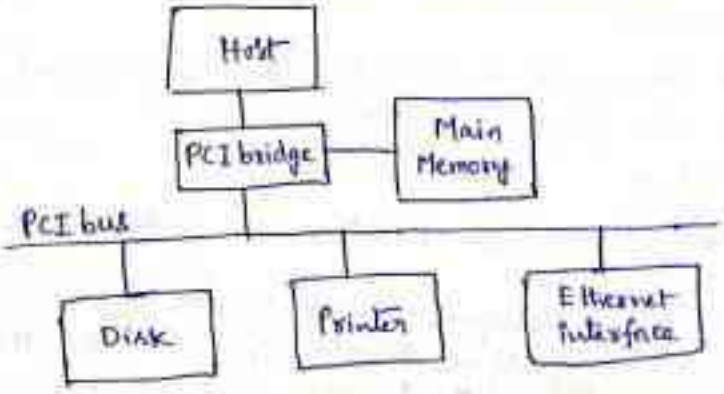
→ [illegible] written in



→ The PCI is designed primarily to support this mode of operation.

→ A read (or) write operation involving a single word is simply treated as a burst of length one.

→ The use of a PCI bus in a Computer System is depicted as,



(ii) SCSI Bus :

→ SCSI stands for Small Computer System Interface.
→ It refers to a standard bus defined by the ANSI (American National Standards Institute) under the designation X3.131
→ In the original specifications of the standard, devices such as disks are connected to a computer via a 50-wire cable, which can be upto 25 meters in length and can transfer data at rates up to 5 megabytes/s
→ 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 the data bus)
- BSY (Busy)
- SEL (Selection)
- C/D (Control/Data)
- MSG (Message)
- REQ (Request)
- ACK (Acknowledge)
- I/O (Input/Output)
- ATN (Attention)
- RST (Reset)

(ii) USB (Universal Serial Bus):

→ A simple, low-cost mechanism to connect the devices to the computer is possible using USB.

→ The USB supports two speeds of operation

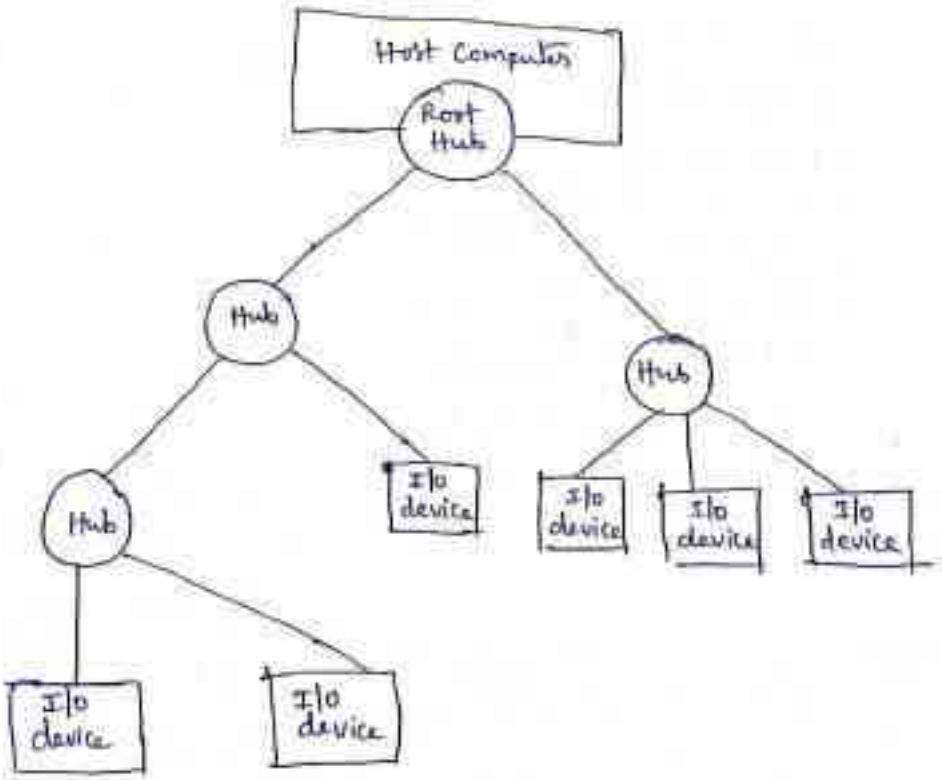
* Low-speed (1.5 Mb/s)
* Full-speed (12 Mb/s)

→ The recent development is USB 2.0

→ The USB has been designed to meet several key objectives.

- Provide a simple, low-cost, and easy to use interconnection system.
- Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections.
- Enhance user convenience through a "plug-and-play" mode of operation.

→ USB Tree structure is depicted as,



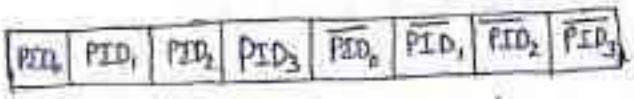
→ To accommodate a large number of devices that can be added (or) removed at any time, the USB has the tree structure.
→ Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O devices.
→ At the root of the tree, a root hub connects the entire tree to the host computer.
→ The leaves of the tree are the I/O devices being served (for example, keyboard, Internet connection, speaker (or) digital TV), which are called functions in USB terminology.

USB protocols:

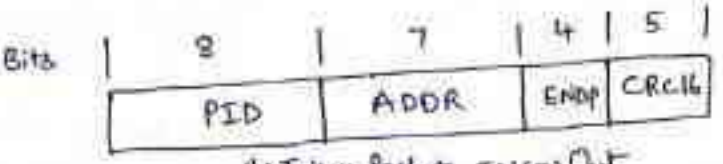
→ All information transferred over the USB is organized in packets, where a packet consists of one (or) more bytes of information.
→ The information ~~directed~~ transferred on the USB can be divided into two broad categories.

- Control
- Data

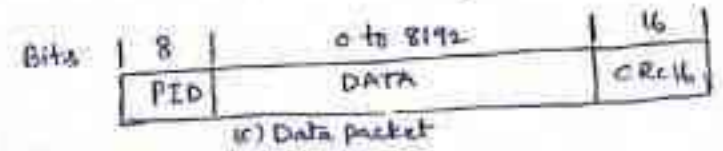
→ Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, (or) indicating error.
→ Data packets carry information that is delivered to a device.
→ A packet contains one or more fields with different kinds of information.
→ The first field of any packet is called the packet identifier, PID, which identifies the type of that packet.
→ USB packet formats is depicted as,



(a) Packet Identifier field



(b) Token Packet, IN (or) Out



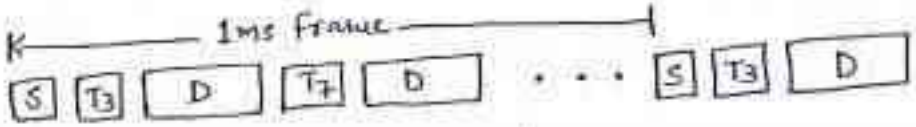
(c) Data packet

→ ADDR – Address
ENDP – Ending Packet
CRC – Cyclic Redundancy Check

Isochronous Traffic on USB:

→ One of the key objectives of the USB is to support the transfer of Isochronous data, such as sample voice, in a simple manner.
→ Devices that generate (or) receive Isochronous data require a time reference to control the sampling process.
→ To provide this reference, transmission over the USB is divided into frames of equal length.
→ A frame is 1ms long for full and low speed data.
→ The root hub generates a Start Of Frame (SOF) control packet precisely once every 1 ms to mark the beginning of a new frame.
→ USB frames depicted as,

(a) SOF Packet



(b) Frame Example

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

CO-UNIT4 - Short Answer Questions

① Discuss about Single bus structure.

page 4.1

② Explain about Memory-mapped I/O.

page 4.2

③ What is programmed I/O?

Programmed I/O:

→ Programmed I/O is a method of transferring data between the CPU and a peripheral.

Advantages:

→ I/O command is issued by the processor to the respective I/O module.
→ Requested I/O instruction is executed at the earliest.
→ I/O module switches to the next task as soon as it completes the task.

Disadvantages:

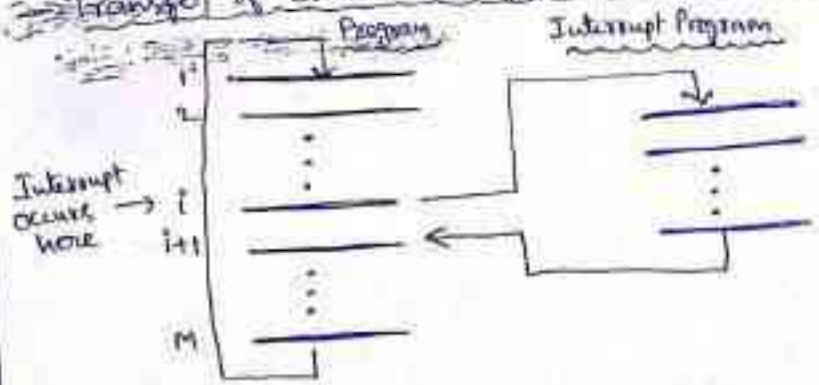
→ Processor has to wait until the I/O module is ready for performing data transfer.
→ Processor have to interrogate several times regarding the status of I/O module.

④ What is interrupt-initiated data transfer?

Interrupt-initiated Data transfer:

→ When the I/O device is ready to transfer data, instead CPU keep asking, does not check the flag.
→ When the flag in the interface is set, the interface will initiate an interrupt, and the CPU will drops what it is doing and do the I/O transfer.
→ Until the I/O transfer is done, it returns to what it was doing

→ Transfer of control through the use of Interrupts is depicted as,



(5) Discuss Daisy-chain priority Interrupt

page 4.6

(6) Explain a) Interrupt b) Exception

a) page 4.3

b) Exception:

→ An exception is an abnormal (or) unusual termination

→ An exception is a condition that results from software and prevents the processor from executing the current instruction stream.

→ It affects the normal execution of an instruction.

(7) What is DMA?

page 4.7

(8) What is the need for Bus Arbitration?

page 4.8

(9) Discuss handshaking (or) Asynchronous Data transfer (or) Asynchronous Bus

page 4.12

(10) Explain PCI bus.

page 4.19

(11) Explain SCSI bus.

page 4.21

(12) Explain USB.

page 4.22

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING.

SIR C R REDDY COLLEGE OF ENGINEERING

Dr DEEPAK NEDUNURI

ELURU.

COMPUTER ORGANIZATION
UNIT-5: THE MEMORY SYSTEMS

Syllabus:

- Basic memory concepts
- Memory System Consideration
- Read- Only Memory
  - ROM
- PROM
- EPROM
- EEPROM
- Flash Memory
- Cache Memories
  - Mapping Functions
- INTERLEAVING
- Secondary Storage
  - Magnetic Hard Disks
  - Optical Disks

Text Book to Follow:

1. Computer Organization, Carl Hamacher, ZvonksVranesic, SafeaZaky, 5th Edition, McGraw Hill

1. BASIC MEMORY CONCEPTS:

- A memory unit is the collection of storage units or devices together.
- The memory unit stores the binary information in the form of bits.
- Memory/ Storage is classified into 2 categories:
  - i) Volatile Memory
  - ii) Non Volatile Memory

i) Volatile Memory:

- This loses its data, when power is switched off.

ii) Non-Volatile Memory:

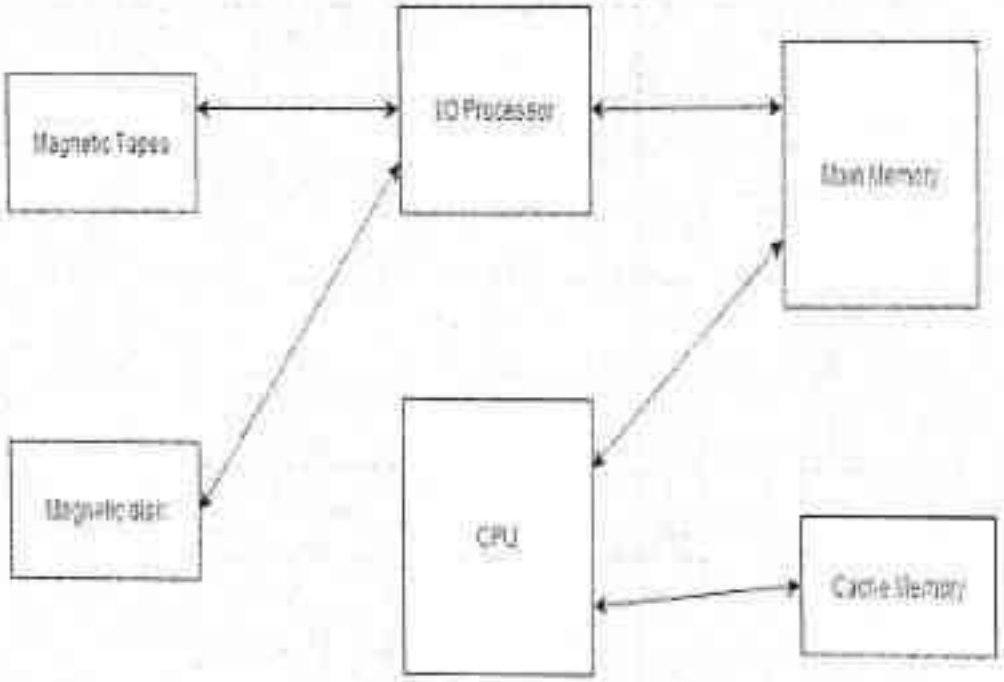
- This is a permanent storage and does not lose any data when power is switched off.

Memory Access Methods:

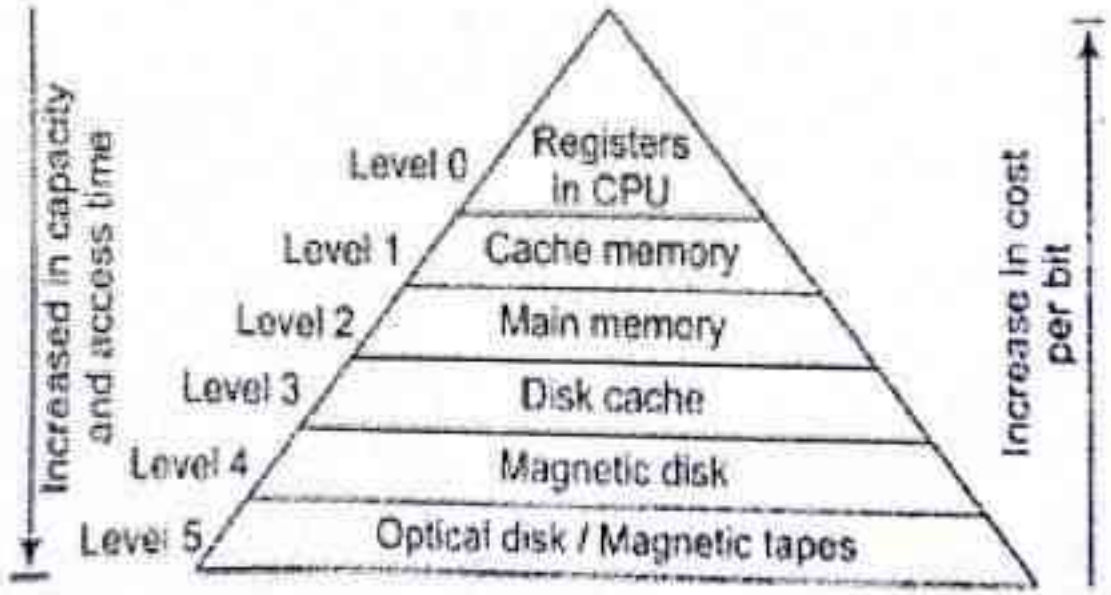
i) Random Access:

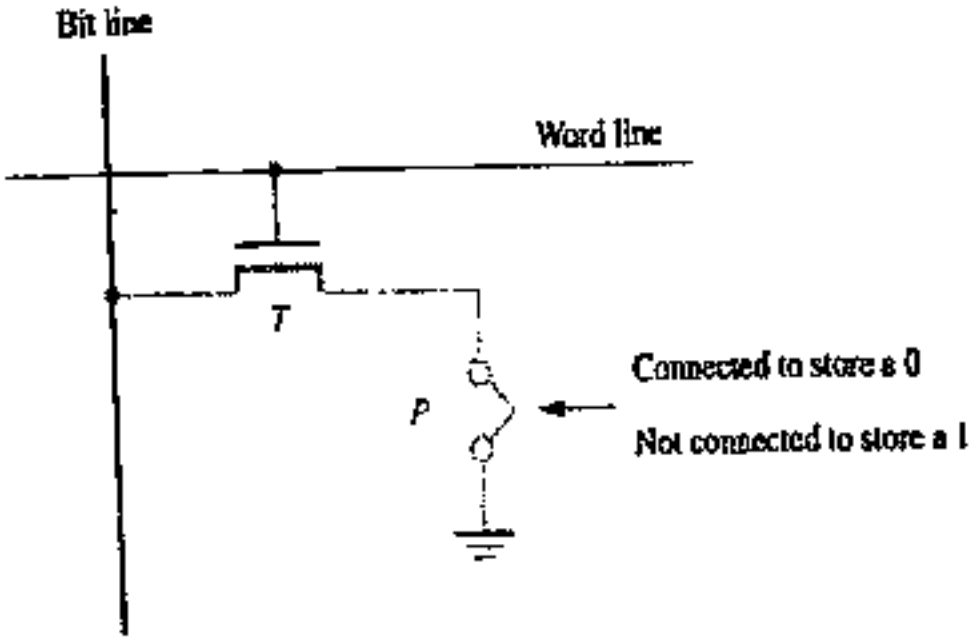
Memory Hierarchy:

- The Memory Hierarchy is depicted as,



- The comparisons in the Memory Hierarchy is depicted as,





- A logic value 0 is stored in the cell if the transistor is connected to ground at point P. otherwise a 1 is stored.
- The bit line is connected through a resistor to the power supply
-To read the state of the cell, the word line is activated
- Thus the transistor switch is closed and the voltage on the bit line drops to near zero if there is a connection between the transistor and ground.
- If there is no connection to ground, the bit line remains at the high voltage, indicating a 1.
- A sense circuit at the end of the bit line generates the proper output value.
- Data are written into a ROM when it is manufactured.

PROM:

- It stands for Programmable Read Only Memory.
- It was first developed in 70s by Texas Instruments.
- It is made as a blank memory.
- A PROM programmer or PROM burner is required in order to 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.

EPROM:

- It stands for Erasable Programmable ROM.
- It is different from PROM as unlike PROM the program can be written on it more than once.
- This comes as the solution to the problem faced by PROM.
- The bits of memory come back to 1, when ultra violet rays of some specific wavelength falls into its chip's glass panel.
- The fuses are reconstituted and thus new things can be written on the memory.

EEPROM:

- It stands for Electrically Erasable Read Only Memory.

These are also erasable like EPROM, but the same work of erasing is performed with electric current. Thus, it provides the ease of erasing it even if the memory is positioned in the computer.

It stores computer system's BIOS. Unlike EPROM, the entire chip does not have to be erased for changing some portion of it.

- Thus, it even gets rid of some biggest challenges faced by using EPROMs.

FLASH ROM:

It is an updated version of EEPROM.

In EEPROM, it is not possible to alter many memory locations at the same time. However, Flash memory provides this advantage over the EEPROM by enabling this feature of altering many locations simultaneously.

It was invented by Toshiba and got its name from it capability of deleting a block of data in a flash.

Flash Cards:

- One way of constructing a larger module is to mount flash chips on a small card.
- Such flash cards have a standard interface that makes them usable in a variety of products.
- A card is simply plugged into a conveniently accessible slot.
- Flash cards come in a variety of memory sizes.
- Typical sizes are 8, 32, and 64 Mbytes.

Flash Drives:

- Larger flash memory modules have been developed to replace hard disk drives.

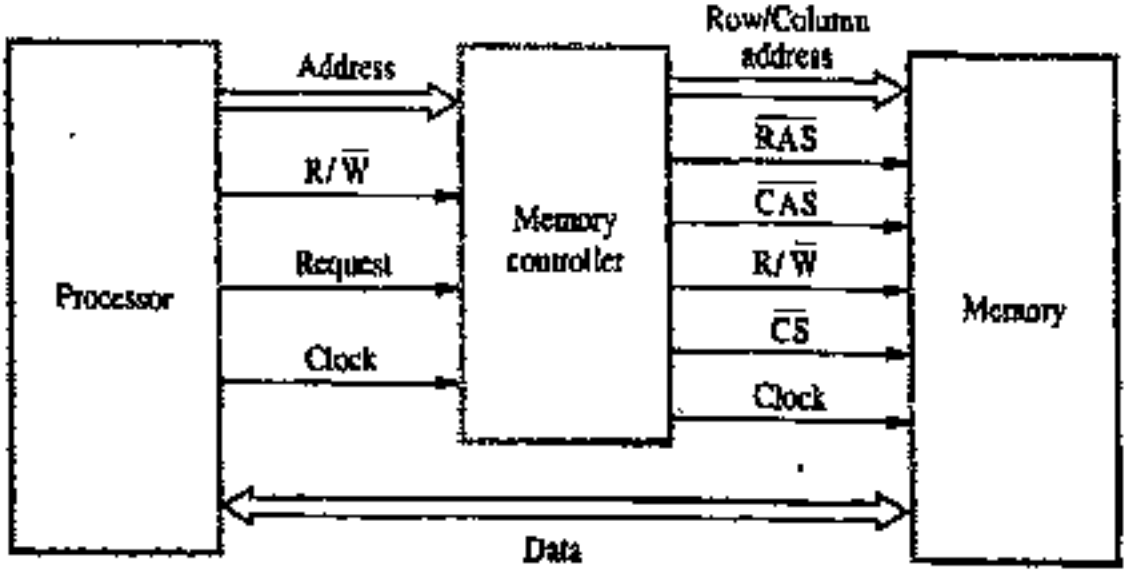
- These flash drives are designed to fully emulate the hard disks, to the point that they can be fitted into standard disk drive bays.
- However, the storage capacity of the flash drives is significantly lower.
- Currently, the capacity of flash drives is less than one Giga Byte.

3. MEMORY SYSTEM CONSIDERATIONS:

- The choice of a RAM chip for a given application depends on several factors such as, cost, speed, power dissipation, and size of the chip.

Memory controller:

- The address is divided into two parts.
- The high-order address bits, which select a row in the cell array, are provided first and latched into the memory chip under control of the RAS signal.
- Then the lower-order address bits, which select a column, are provided on the same address pins and latched using the CAS signal.
- A typical procedure issues all bits of an address at the same time.
- The required multiplexing of address bits is usually performed by a Memory Controller Circuit, which is interposed between the processor and the dynamic memory
- Use of a Memory Controller is depicted as,

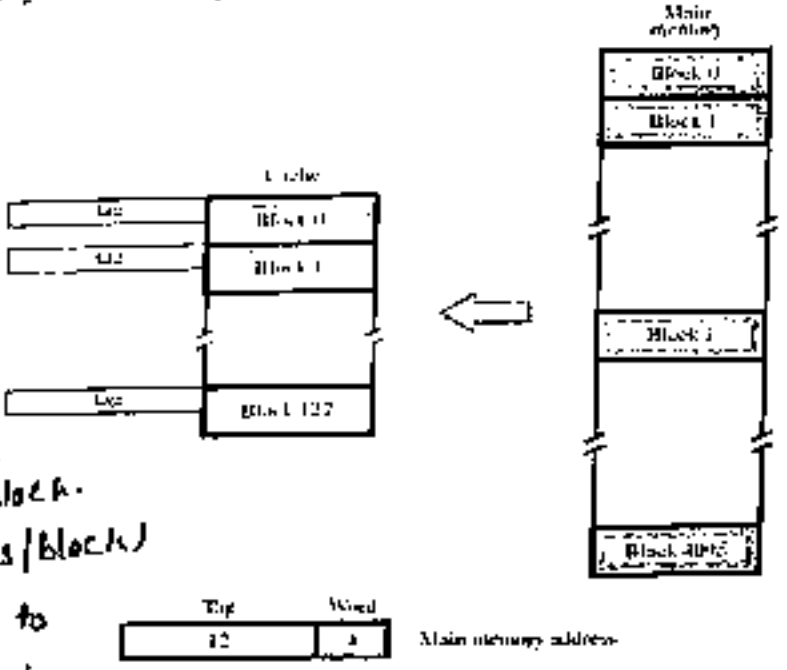


- The controller accepts a complete address and the R/W signal from the processor, under control of a request signal which indicates that a memory access operation is needed.
- The controller then forwards the row and column portions of the address to the memory and generates the RAS and CAS signals.

- Thus, whenever one of the main memory blocks 0, 128, 256, . . . is loaded into the cache, it is stored in cache block0.
- Blocks 1, 129, 257, . . . are stored in cache block 1, and so on.
- The memory address can be divided into *three* fields
- The low-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 high-order 5 bits of the address are compared with the tag bits associated with that cache location. If they match, then the desired word is in that block of the 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.

iii) Associative Mapping:

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



Main memory address

- 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 the processor are compared to the tag bits of each block of the cache to see if the desired block is present.
- This is called the *associative-mapping technique*.
- It gives complete freedom in choosing the cache location in which to place the memory block, resulting in a more efficient use of the space in the cache.
- When a new block is brought into the cache, it replaces (ejects) an existing block only if the 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, the tags must be searched in parallel.
- A search of this kind is called an *associative search*.

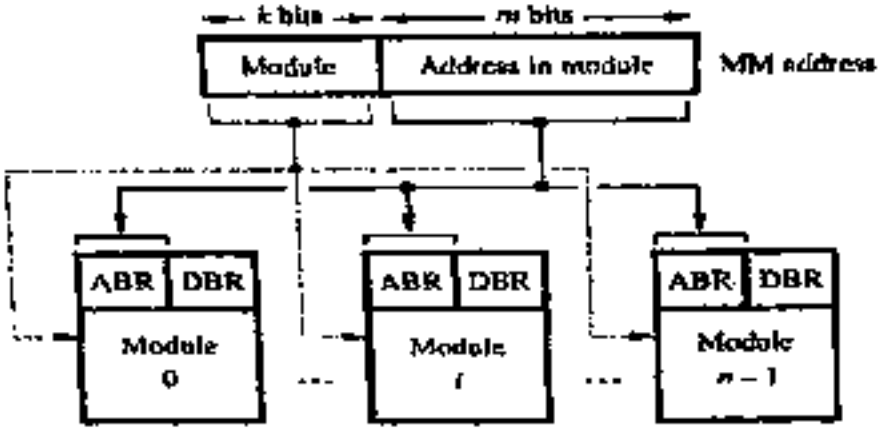
iii) Set-Associative Mapping:

- Another approach is to use a combination of the direct- and associative-mapping techniques.
- The 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.
- Hence, the contention problem of the direct method is eased by having a few choices for block placement.
- At the same time, the hardware cost is reduced by decreasing the size of the associative search.
- An example of this set-associative-mapping technique is shown in Figure 8.18 for a cache with two blocks per set.
- In this case, memory blocks 0, 64, 128, . . . , 4032 map into cache set 0, and they can occupy either of the two block positions within this set.
- Having 64 sets means that the6-bit set field of the address determines which set of the cache might contain the desired block.
- The tag field of the address must then be associatively compared to the tags of the two blocks of the set to check if the desired block is present.
- This two-way associative search is simple to implement.
- The number of blocks per set is a parameter that can be selected to suit the requirements of a particular computer.
- For the main memory and cache sizes, four blocks per set can be accommodated by a 5-bit set field, eight blocks per set by a 4-bit set field, and so on.

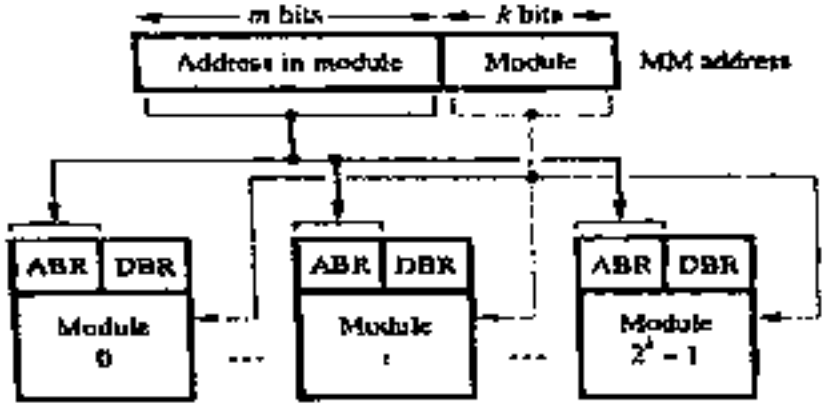
3. MEMORY INTERLEAVI NG:

- If the main memory of a computer is structured as a collection of physically separate modules. each with its own address buffer register (ABR) and data buffer register (DBR). memory access operations may proceed in more than one module at the same time.
- Thus the aggregate rate of transmission of words to and from the main memory system can be increased.
- How individual addresses are distributed over the modules is critical in determining the average number of modules that can be kept busy as computations proceed.

- Two methods of address layout, i.e., Addressing Multiple-module Memory Systems are depicted as,



(a) Consecutive words in a module



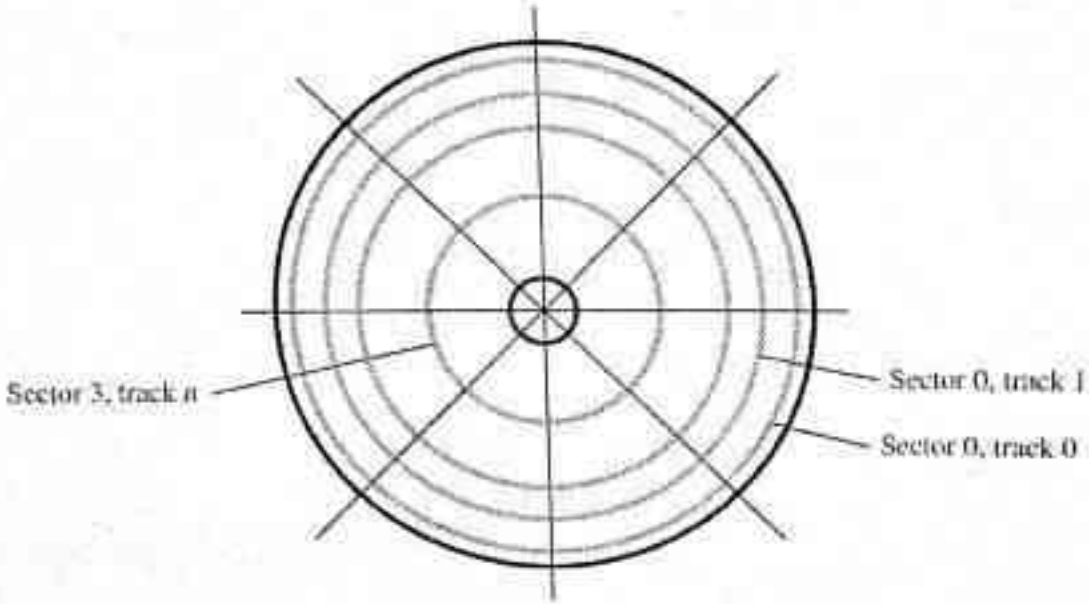
(b) Consecutive words in consecutive modules

- In the first case, the memory address generated by the processor is decoded.
- The high-order k-bits name one of n modules, and the low-order m-bits name a particular word in that module.
- When consecutive locations are accessed, as happens when a block of data is transferred to a cache, only one module is involved.
- At the same time, however, devices with direct memory access (DMA) ability may be accessing information in other memory modules.
- The second and more effective way to address the modules is called Memory Interleaving.

- In this scheme, changes in magnetization occur for each data bit.
- Clocking information is provided by the change in magnetization at the midpoint of each bit period.
- The drawback of Manchester encoding is its poor bit-storage density.
- The space required to represent each bit must be large enough to accommodate two changes in magnetization.
- In most modern disk units, the disks and the read/write heads are placed in a sealed, air-filtered enclosure.
- This approach is known as Winchester technology.

Organization and Accessing of Data on a Disk:

- The organization of data on one surface of a disk is depicted as,



- 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 preceded by a *sector header* that contains identification (addressing) information used to find the desired sector on the selected track.
- Following the data, there are additional bits that constitute an *error-correcting code* (ECC).

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING.

SIR C R REDDY COLLEGE OF ENGINEERING

Dr DEEPAK NEDUNURI

ELURU.

UNIT-6: Processing Unit & Micro-programmed Control

JNTUK R16 2-2 CSE

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 ✓

→ Micro-programmed Control
  → Micro Instructions
  → Microprogram Sequencing ✓
  → Wide-branch Addressing
  → MicroInstructions with Next-Address Field ✓

① 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 instruction 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.

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

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

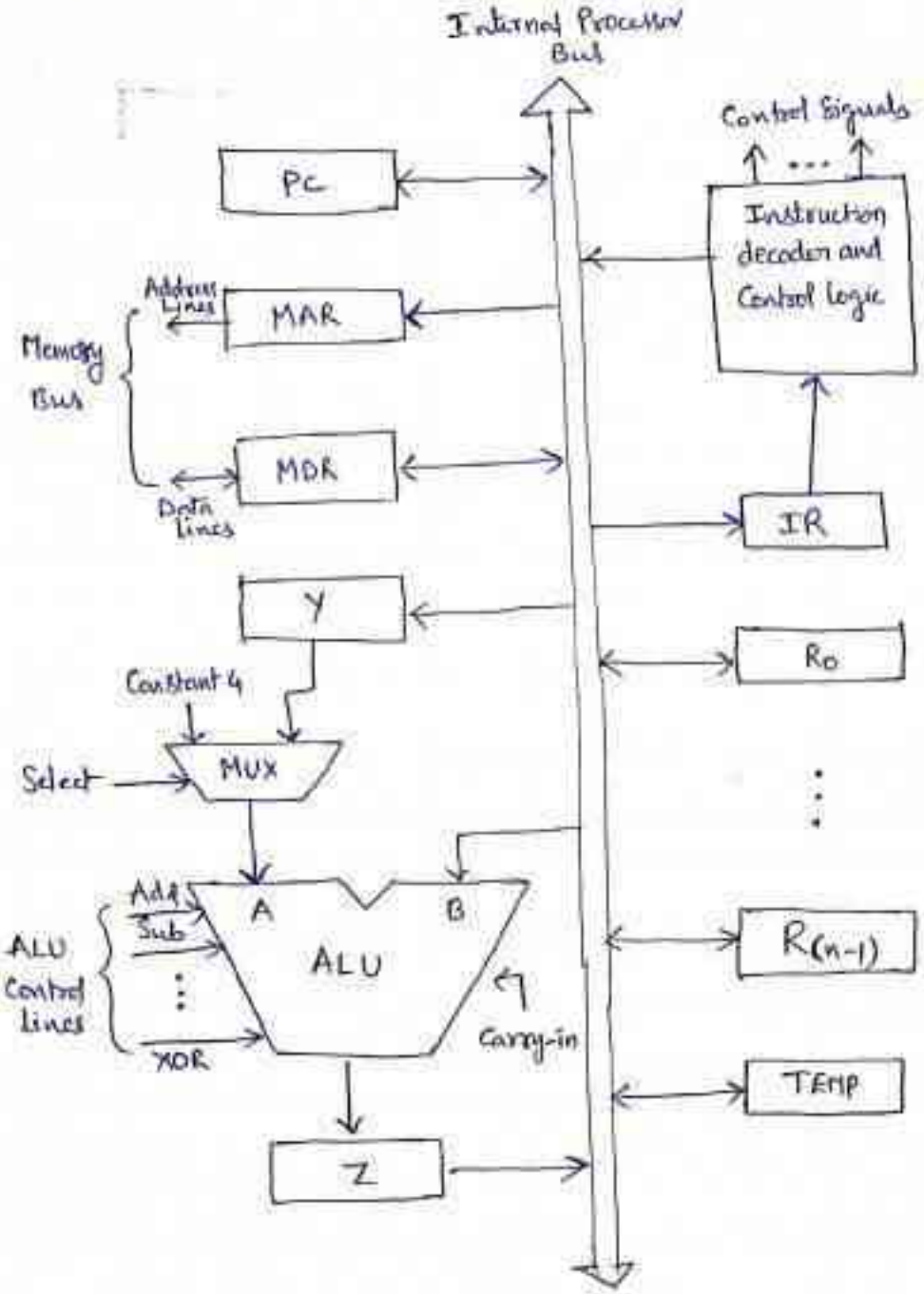
* Assuming that the memory is byte addressable, increment the contents of the PC by 4

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

* Carry out the actions specified by the instruction in the IR.

→ Here, first two steps represents fetch phase, Third step represents execution phase.

→ Single-bus organization of the datapath inside a processor is depicted as,



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

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

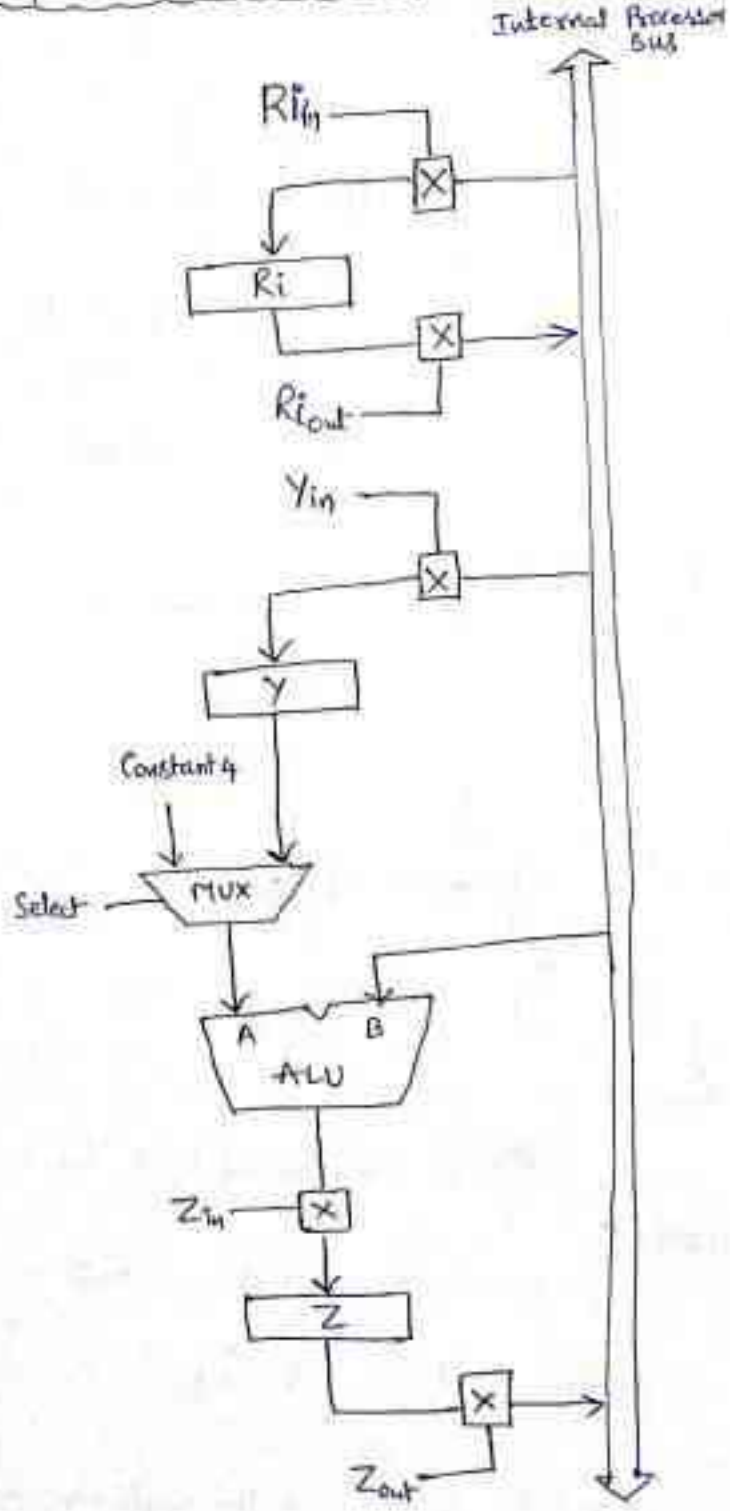
→ The multiplexer MUX selects either the output of register Y (or) a constant value 4 to be provided as input A of the ALU.

(a) Register Transfer :

→ 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) load the data on the bus into the register.

→ Input and output gating for the registers is depicted as,



→ The input and output of register R_i are connected to the bus via switches controlled by the signals Ri_{in} and Ri_{out}, respectively.

→ When Ri_{in} is set to 1, the data on the bus are loaded into R_i

→ Similarly, when Ri_{out} is set to 1, the contents of Register R_i are placed on the bus.

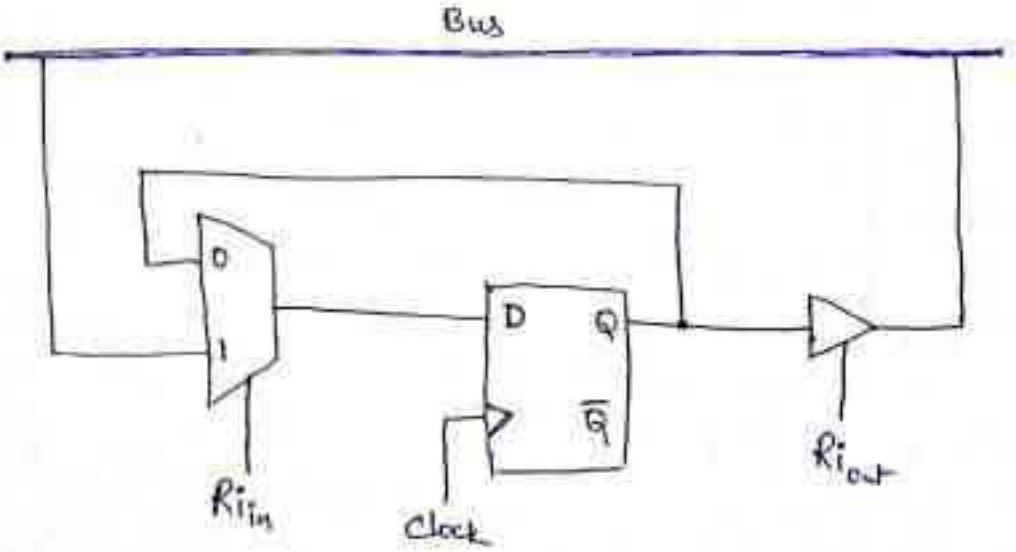
→ While Ri_{out} is equal to 0, the bus can be used for transferring data from other registers

Example:

→ To transfer the contents of register R_1 to register R_4,

- Enable the output of register R_1 by setting $R1_{out}$ to 1, This places the contents of R_1 on the processor bus.
- Enable the input of register R_4 by setting $R4_{in}$ to 1, This loads data from the processor bus into register R_4.

→ Input and Output gating for one register bit is depicted as,



→ A two-input multiplexer is used to select the data applied to the input of an edge-triggered D-flipflop

→ When the control input Ri_{in} is equal to 1, the multiplexer selects the data on the bus.

→ When Ri_{in} equal to 0, the multiplexer feeds back the value currently stored in the flip-flop.

→ When Ri_{out} is equal to 0, the gate's output is in the high-impedance (electrically disconnected) state.

→ When $Ri_{out} = 1$, the gate drives the bus to 0 (or) 1, depending on the value of Q.

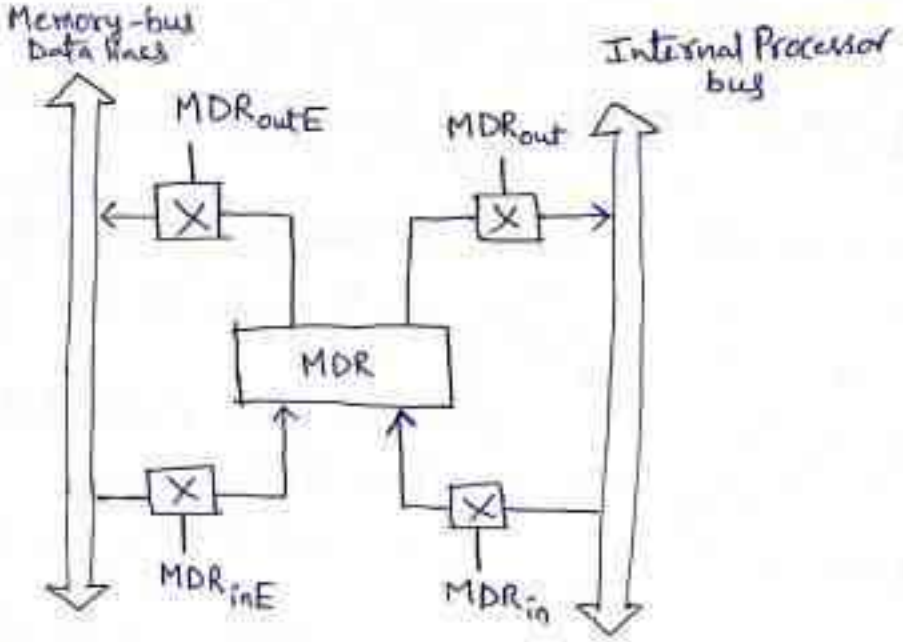
(b) Performing an Arithmetic (or) Logic Operation:

→ 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 the multiplexer MUX and the 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 R_1 to those of register R2 and store the result in register R3 is,

1. $R1_{out}$, Y_{in}
2. $R2_{out}$, Select Y, Add, Z_{in}
3. Z_{out}, $R3_{in}$

(c) Fetching a word from memory:

→ To fetch a word of information from memory, the processor has to specify the address of the memory location 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 processor 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.
→ The connection and control signals for register MDR is depicted as,



→ It has 4 control signals
- MDR$_{in}$ and MDR$_{out}$ control the connection to the internal bus.
- MDR$_{inE}$ and MDR$_{outE}$ control the connection to the external bus

→ As an example of a read operation, consider the instruction

MOVE (R1), R2

- The 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 the memory bus
5. R2 ← [MDR]

② Execution of a Complete Instruction :

→ Consider the instruction

`Add (R3), R1`

which adds the contents of a memory location pointed to by R3 to register R1.

→ Executing this instruction requires the following actions.

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

→ The Control Sequence for execution of the instruction Add (R3), R1 is given as

Step	Action
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}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

→ In Step 1, the instruction fetch operation 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 the 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 through 3 represents instruction fetch phase.

→ The contents of register R3 are transferred to the MAR in Step 4, and a memory read operation is initiated.

→ Then the contents of R1 are transferred to register Y in Step 5, to prepare the addition operation.

→ When the Read operation is completed, the memory operand is available in register MDR, and the addition operation is performed in Step 6.

→ The sum is stored in register Z, then transferred to R1 in Step 7.

→ From Step 4 through 7 represents execution phase.

(ii) BRANCH Instructions:

→ A branch instruction replaces the contents of the PC with the branch target Address.

→ This address is usually obtained by adding an offset X, which is given in the branch instruction, to the updated value of the PC

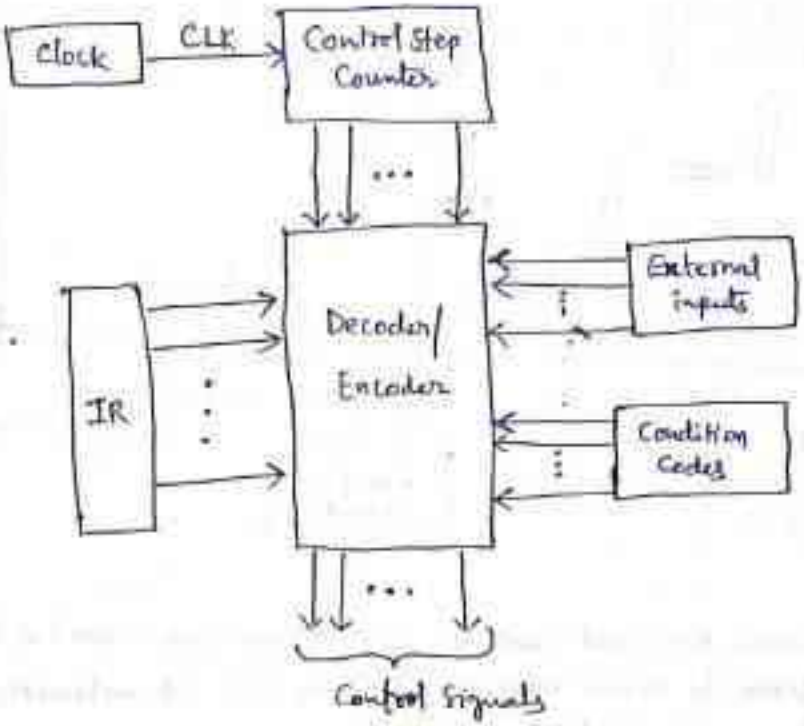
→ Control Sequence for an unconditional Branch instruction is given as

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in}, End

③ Hardwired Control :

→ To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence.
→ This approach has two Categories
* Hardwired Control
* Micro-programmed Control

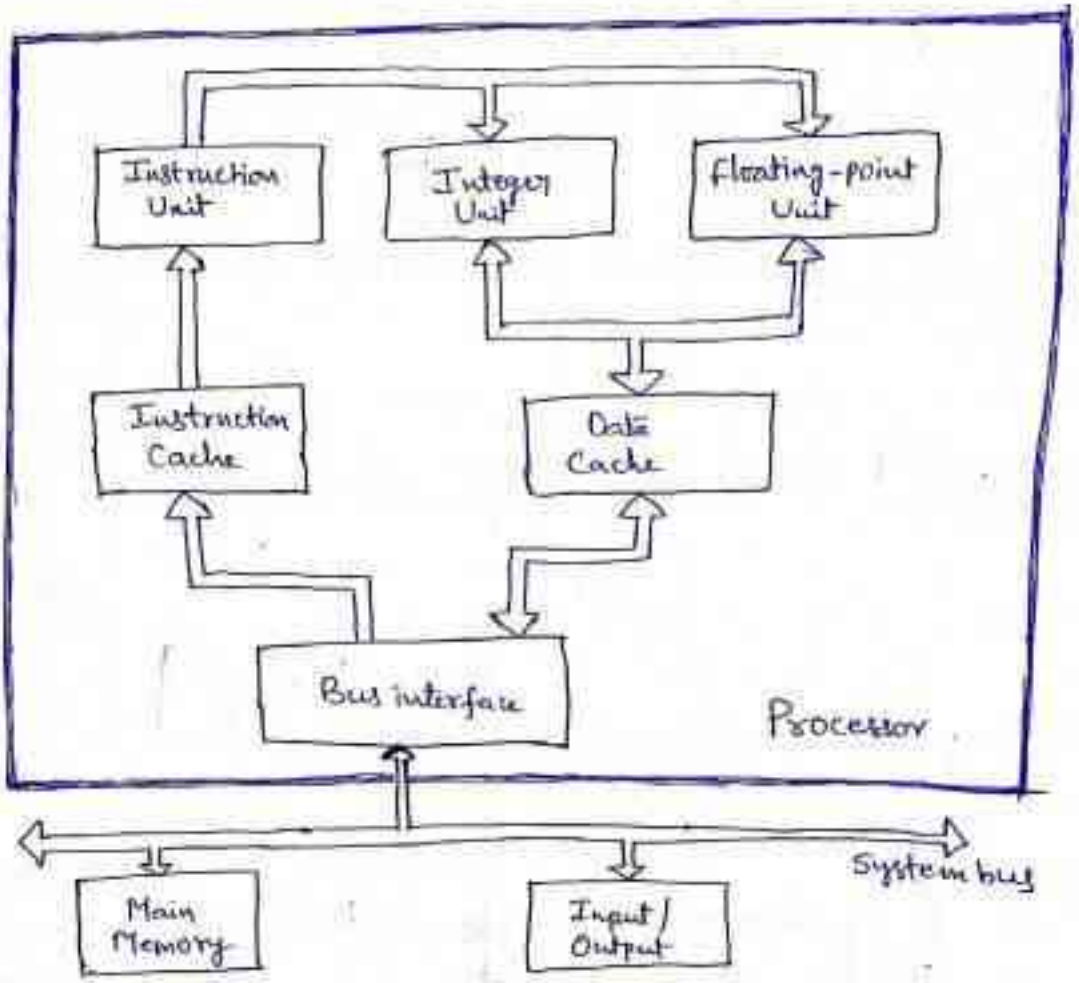
→ The control unit organization is depicted as



→ 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.
→ Each state, or count, of this counter corresponds to one control step.
→ The required control signals are determined by the following information.
- Contents of the control step counter
- Contents of the instruction register
- Contents of the condition code flags
- External input signals, such as MFC and interrupt requests

(i) A Complete Processor :

→ The Block diagram of a Complete Processor is depicted as



→ It has an instruction unit that fetches instructions from an Instruction Cache (or) from the main memory when the desired instructions are not already in the Cache.

→ It has seperate processing units to deal with integer data and floating-point data.

→ Using seperate caches for instruction and data is common practice in many processors today.

→ The processor is connected to the System bus and, hence, to the rest of the Computer, by means of a bus interface.

④ Microprogrammed Control:

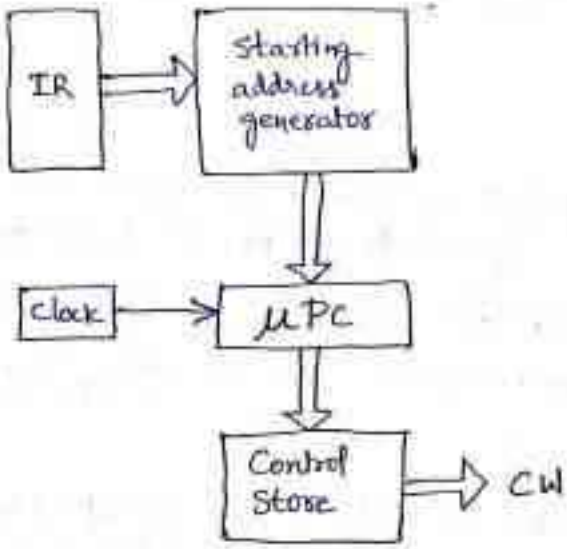
→ The control signals are generated by a program similar to machine language programs represents a scheme called Microprogrammed Control

→ A control word (CW) is a word whose individual bits represent the various control signals

→ A sequence of CWs 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 microroutines for all instructions in the instruction set of a computer are stored in a special memory called the control store.

→ The Basic organization of a microprogrammed control unit is depicted as,



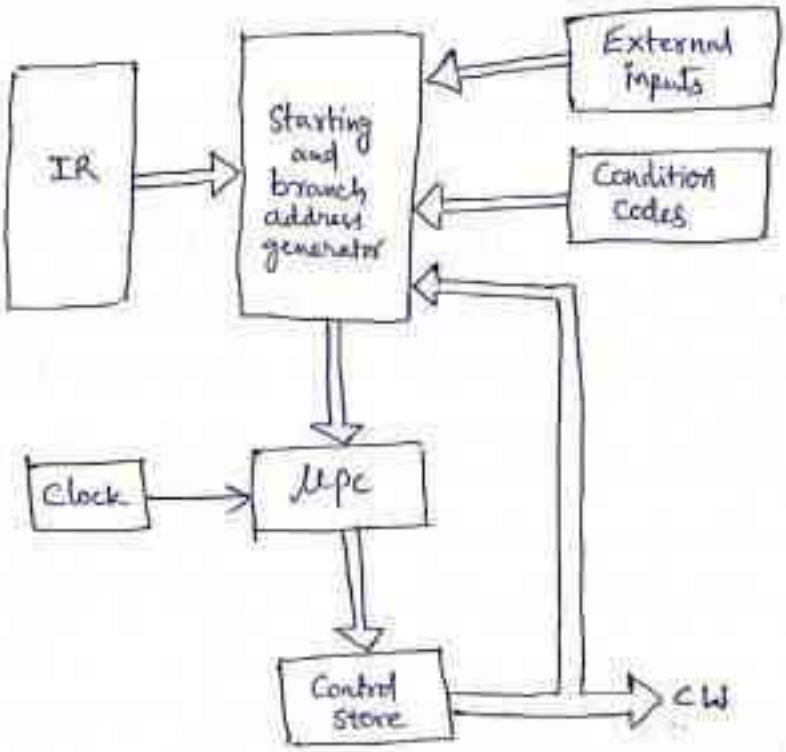
→ To read the control words sequentially from the control store, a micro-program counter (μPC) is used.

→ Everytime a new instruction is loaded into the IR, 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.

→ Organization of the control unit to allow conditional branching in the microprogram is depicted as



(i) Microinstructions :

→ The control word belonging to a control unit possess certain instructions, usually referred to as Microinstructions.

→ Each Microinstruction specifies the Microoperations for the System

→ A microinstruction can be structured in many ways. Some of them are,

- Assign a bit position to each control signal. This scheme is not good enough, as it results in long instructions. And also because of only few bits are set to 1, it does not use the bit space properly.
- Encode the microinstructions using bits, with assumptions.
- Group the mutually exclusive control signals into fields.
- Enumerate the patterns of required signals in all microinstructions.

→ There are two types of microinstruction formats.

(a) Horizontal Microinstruction Format
(b) Vertical Microinstruction Format

a) Horizontal Microinstruction format:

Internal CPU Control Signals	System bus Control Signals	Jump Condition (Indirect bit, Zero overflow, Unconditional)	Microinstruction address

→ Horizontal Microinstruction Format Supports
- Long formats
- expression recent to high degree of parallelism
- Low degree of encoding of control information

b) Vertical Microinstruction Format:

Function codes	Function Codes	Jump Condition	Microinstruction Address

→ Vertical Microinstruction Format supports
- Less degree of parallelism in case of microoperations.
- Subsequently high encoding in case of control information.
- Relatively short format

(ii) Microprogram Sequencing

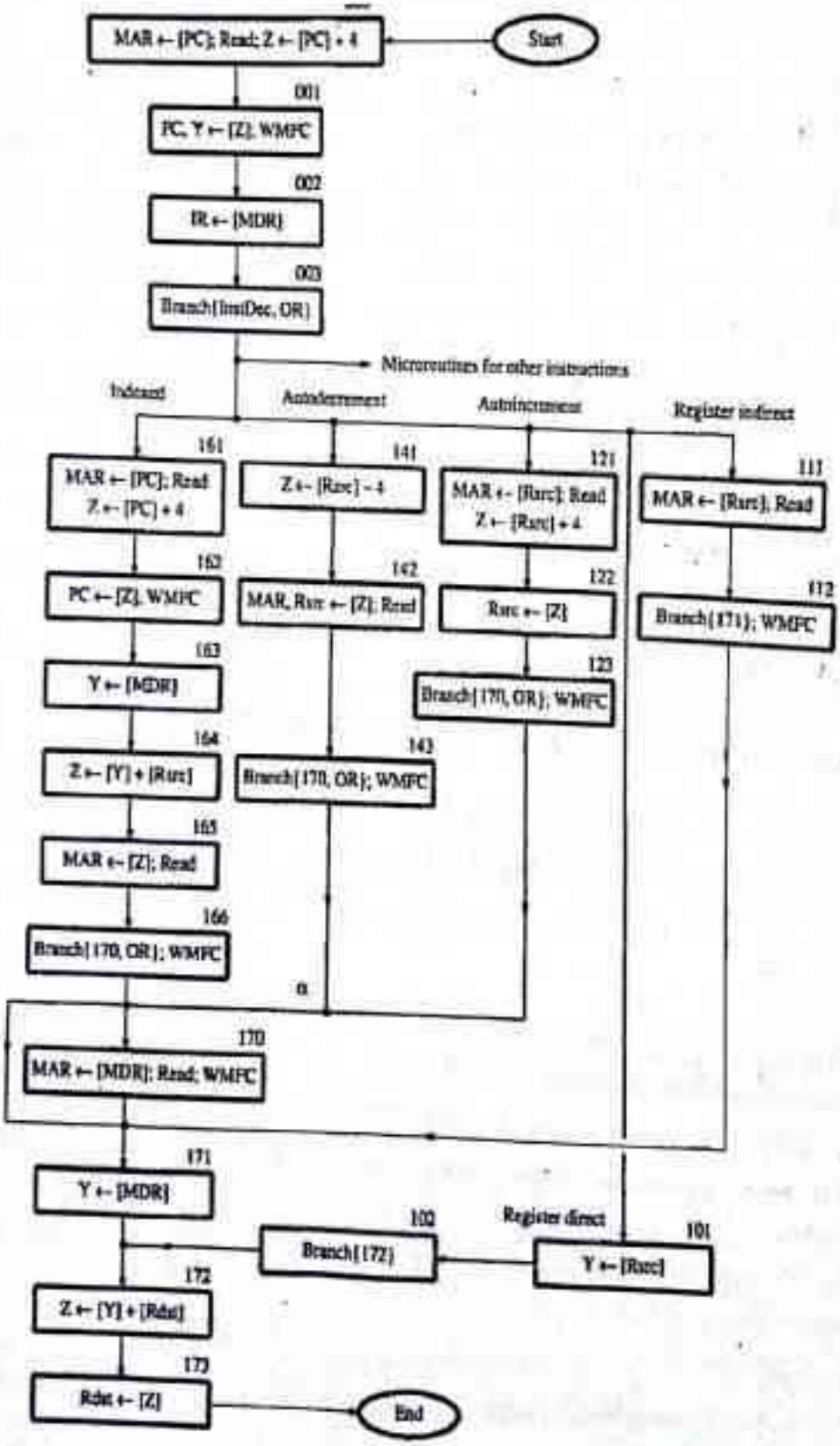
→ A microprogram is a set of microinstructions
→ In microprogram sequencing, microinstructions are executed in the sequential order.
→ The Microprogram Sequencer generates the order of executing microinstructions from the Control Store.

Example:
→ Consider a machine instruction that adds R_s and R_d and stores the result in R_d

Add R_s, R_d

→ Where, R_s is the source operand register and R_d is the destination operand register

→ Flowchart of a microprogram for the Add Rsrc, Rdst instruction is depicted as,



A-SR-10-6.15

(iii) Wide-Branch Addressing:

→ Generating branch addresses means that the circuitry becomes more complex

→ Example, the machine instruction fetch is completed, and an appropriate microroutine should be selected according to addressing modes

→ Here, the OPcode of a machine instruction is translated into a starting address.

→ It is possible to issue a wait for MFC (WMFC) Command in a branch micro-instruction.

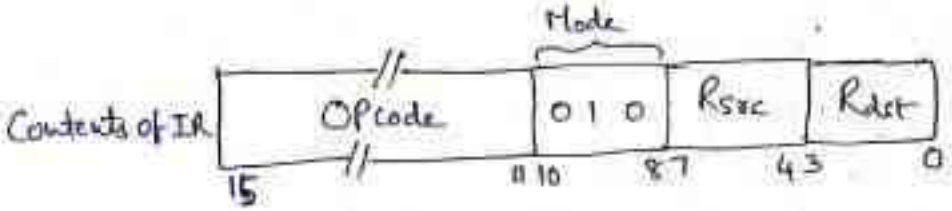
→ The WMFC signal means that the microinstruction may take several clock cycles to complete.

Example:

→ For the instruction

Add (Rsrc)+, Rdst

→ The Format of IR is depicted as



(iv) 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 data path.

→ This effects the operating speed of the System.

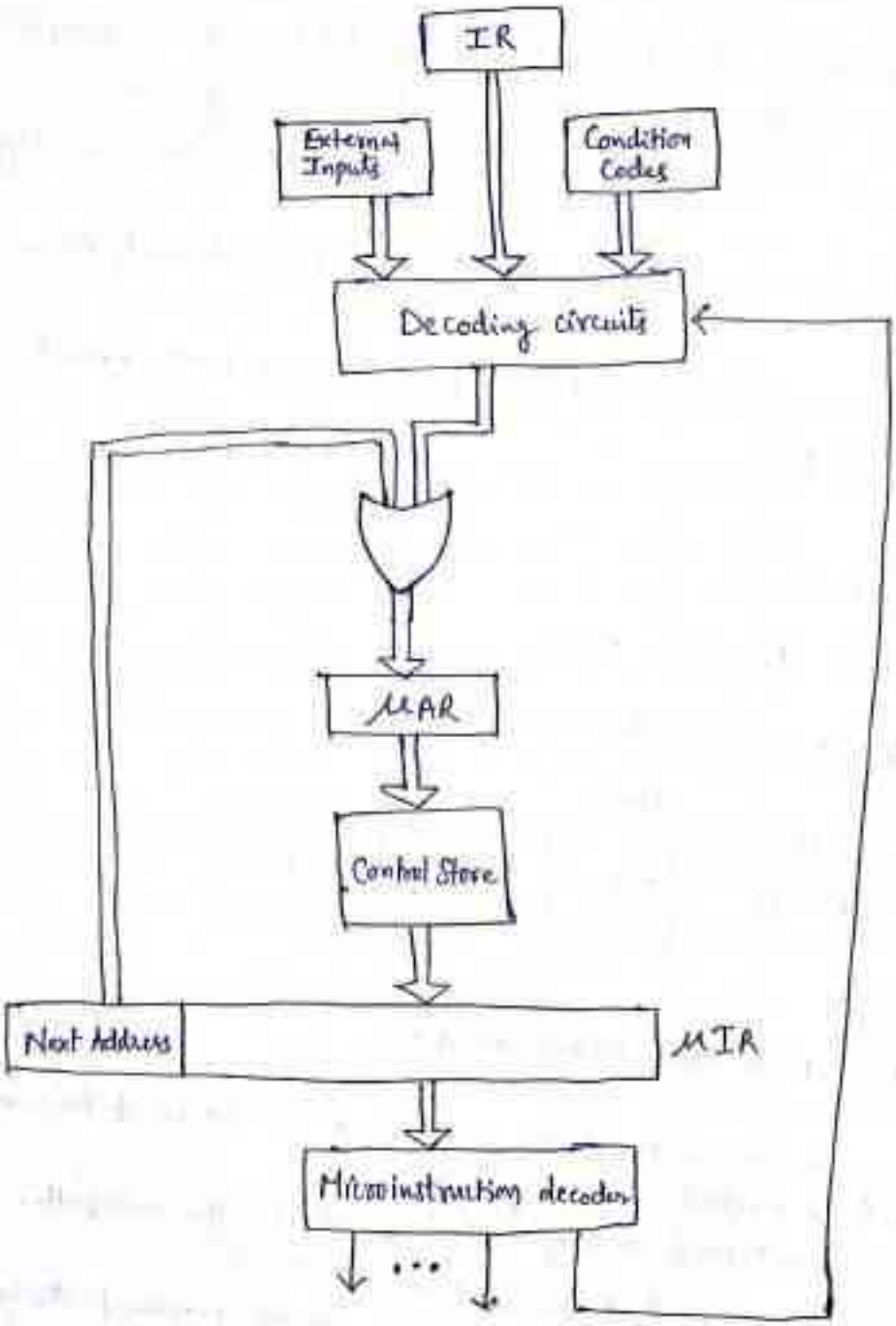
→ The situation becomes worse when the processor shares common parts in microinstructions using branch microinstruction in order to execute sequential instructions.

→ Thus, execution of one simple instruction needs several branch instructions, which badly affects the System's Operating System (OS).

→ One way to overcome the above problem is to modify the sequencing technique based on incrementable μpc.

→ An alternative is to include a special address field called "Next-address field" in each microinstruction in order to specify the address of the next microinstruction

→ The Microinstruction-Sequencing Organization is depicted as,



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

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

CO - UNIT 6 - Short Answer Questions

(1) What is register transfer? Discuss..

page 6.3

(2) Briefly discuss unconditional branch instructions.

page 6.8

(3) Explain Hardwired Control

page 6.9

(4) Explain Microprogrammed Control unit

page 6.11

(5) Define a) Micro-operation b) Micro-program c) Micro-instruction

a) Microoperation:

→ A Microoperation refers to a Simple (or) a Complex operation that produces certain output, which is stored in a memory location (or) register.

b) Microprogram:

→ Microprogram refers to a sequence of microinstructions.
→ It is usually stored in Control memory.
→ A Microprogram is executed by a micro programmed Control unit.

c) MicroInstruction:

→ The Control word belonging to a Control unit possess certain instructions, usually referred to as microinstructions.
→ Each Microinstruction Specifies microoperations for the System

(6) Write Micro instruction formats.

(or)

Write about a) Horizontal Microinstruction Format b) Vertical Microinstruction format

page 6.13

(7) Differentiate between Hardwired Control and Microprogrammed Control.

Microprogrammed control Unit	Hardwired control unit
1. It has slower execution speed	1. It has faster execution speed
2. Its control functions are implemented in Software	2. Its control functions are implemented in hardware
3. It can easily accommodate changes such as new system specifications (or) new instructions 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. It can easily support operating systems and diagnostic features	6. It cannot easily support OS and diagnostic features
7. It can easily handle Large/ Complex instruction sets	7. It cannot easily handle Large/ Complex instruction sets.
8. It is used in Mainframes and Microprocessors	8. It is used in RISC microprocessors