UNIT I

Basic Structure of Computers

Functional units, Basic operational concepts, Bus structures, Software performance, Memory locations and addresses, Memory operations, Instruction and instruction sequencing, Addressing modes, Assembly language, Basic I/O operations.

1.1 Basic Structure of Computers

Q1. Define Computer

Ans :

Computer

Computer is a fast electronic calculating machine that accepts digitized input information, processing it according to a list of internally stored instructions and produces the resulting output information. The list of instructions is called as a Computer program and the internal storage is called as Computer memory.

Q2. Write about various types of languages used in computer

Ans :

Types of Languages

Just as humans use language to communicate, and different regions have different languages, computers also have their own languages that are specific to them. Different kinds of languages have been developed to perform different types of work on the computer. Basically, languages can be divided into two categories according to how the computer understands them.

Low-Level Languages

A language that corresponds directly to a specific machine. Low-level computer languages are either machine codes or are very close them. A computer cannot understand instructions given to it in high-level languages or in English. It can only understand and execute instructions given in the form of machine language i.e. binary. There are two types of low-level languages:

Machine Language

A language that is directly interpreted into the hardware. Machine language is the lowest and most elementary level of programming language and was the first type of programming language to be developed. Machine language is basically the only language that a computer can understand and it is usually written in hex. It is represented inside the computer by a string of binary digits (bits) 0 and 1. The symbol 0 stands for the absence of an electric pulse and the 1 stands for the presence of an electric pulse. Since a computer is capable of recognizing electric signals, it understands machine language.

Advantages

- Machine language makes fast and efficient use of the computer.
- It requires no translator to translate the code. It is directly understood by the computer.

Disadvantages

- All operation codes have to be remembered
- All memory addresses have to be remembered.
- It is hard to amend or find errors in a program written in the machine language.

Assembly Language

A slightly more user-friendly language that directly corresponds to machine language. Assembly language was developed to overcome some of the many inconveniences of machine language. This is another low-level but very important language in which operation codes and operands are given in the form of alphanumeric symbols instead of 0's and 1's.

These alphanumeric symbols are known as mnemonic codes and can combine in a maximum of five-letter combinations e.g. ADD for addition, SUB for subtraction, START, LABEL etc. Because of this feature, assembly language is also known as 'Symbolic Programming Language.

Advantages

- Assembly language is easier to understand and use as compared to machine language.
- It is easy to locate and correct errors.
- It is easily modified.

Disadvantages

- Like machine language, it is also machine dependent/specific.
- Since it is machine dependent, the programmer also needs to understand the hardware.

High-Level Languages

Any language that is independent of the machine. High-level computer languages use formats that are similar to English. The purpose of developing high-level languages was to enable people to write programs easily, in their own native language environment (English).

High-level languages are basically symbolic languages that use English words and/or mathematical symbols rather than mnemonic codes. Each instruction in the high-level language is translated into many machine language instructions that the computer can understand.

Advantages

- High-level languages are user-friendly
- They are easier to learn.
- They are easier to maintain
- A program written in a high-level language can be translated into many machine languages and can run on any computer
- Programs developed in a high-level language can be run on any computer text.

Disadvantages

- A high-level language has to be translated into the machine language by a translator, which takes up time.

1.1.1 Functional Units

Q3. What is digital computer?

Ans :

It is an electronic computer in which the input is discreterather than continuous, consisting of combinations of numbers, letters, and other characters written in an appropriate programming language and represented internally in binary notation, i.e., using only the two digits 0 and 1. By counting, comparing, and manipulating these digits or their combinations according to a set of instructions held in its memory, a digital computer can perform such tasks as to control industrial processes and regulate the operations of machines; analyze and organize vast amounts of business data; and simulate the behaviour of dynamic systems.

Q4. Explain the block diagram of digital computer.

(OR)

Explain the functional units of basic computer.

Ans : **(Imp.)**

Block Diagram of a Computer

All these major Operations of the computer are performed by using the five basic components of the computer, which are interconnected each other, known as block diagram of the computer.

The following are the five major components of the block diagram of a computer.

- Input Unit
- Output Unit
- Arithmetic and Logic Unit
- Control Unit
- Memory Unit

The following figure shows the block diagram of the computer with five major components.

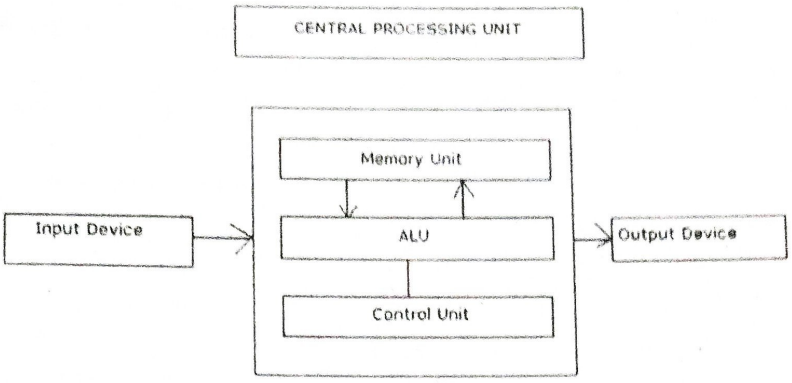


Fig.: Block diagram of digital computer

Input Unit

The Input Unit accepts the instructions and data from outside world through input devices like keyboard, mouse etc. Then it converts these instructions and data from user understandable form to electronic signals, which are understood by the computer. Then it supplies these signals to the Central Processing Unit for further processing.

Central Processing unit

Once the data is accepted from the input unit, it fed to the CPU for further processing before the output is generated. This is known as an electronic brain.

CPU carries out instructions and tells the rest of the computer system what to do. This is done by the Control Unit of the CPU which sends command signals to the other components of the computer

It performs Arithmetical and Logical Calculations with the help of ALU, which is known as Computer calculator. Holds data and instructions which are in current use. These are kept in the Main Memory.

All these operations are done by the following three major components of the CPU.

- Memory Unit
- Control Unit
- Arithmetic and Logic Unit

Memory Unit

It is also known as Primary Memory. It can store the data in terms of programs and files. The data stored can be accessed and processed whenever is needed by the CPU is known as Primary memory. The memory is measured as BYTES, KB, MB , GB etc.

The Primary memory is sub divided into,

- **RAM**: This is the main store and is the place where the programs and software we load gets stored.When the Central Processing Unit runs a program, it fetches the program instructions from RAM
- **ROM**: The CPU can only fetch or read instructions from ROM. ROM comes with instructions permanently stored in it.

Arithmetic and Logic Unit

Whenever calculations are required, the control unit transfers the data from storage unit to ALU Once the computations are done, the results are transferred to the storage unit by the control unit and then it is send to the output unit for displaying results.

The ALU is again sub divided into two functional units:

- **Arithmetic Unit** :The arithmetic unit executes arithmetic operations. Which includes addition, subtraction, multiplication, division
- **Logical Unit** : The logic unit executes logical operations. Which includes comparisons like greater than, less than, equals to etc. It also performs logical operations like AND, OR and NOT.

Control Unit

- It controls all other units in the computer. The control unit must communicate with both the arithmetic logic unit and main memory. The control unit instructs the arithmetic logic unit which arithmetic operations or logical is to be performed.
- The control unit instructs the input unit, where to store the data after receiving it from the user.It controls the flow of data and instructions from the storage unit to ALU.
- It also controls the flow of results from the ALU to the storage unit.

Out Put Unit

It produces the results or the information after the computation to the outside world through the output devices like monitor, printer etc.

It can also store the output to the secondary storage devices like floppy disks, hard disks, CD etc. The control Unit instructs the output unit to produce the output when ever it is necessary to the user.

1.1.2 Basic Operational Concepts

Q5. Explain basic operational concepts of a computer.

Ans :

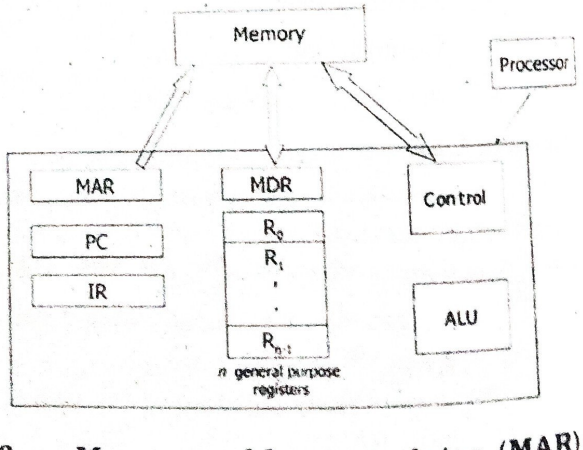
1. The program contains of a list of instructions is stored in the memory.
2. Individual instructions are brought from the memory into the processor, which execute the specified operations.
3. Data to be used as operands are also stored in the memory.

Add R1,R2,R3

In This instruction add is the operation perform on operands R1,R2 and place the result stored in R3.

The top level view of the computer is as follows:

1. **Instruction register (IR)**
   - i) The instruction register holds the instruction that is currently being executed.
   - ii) Its output is available to the control circuits, which generate the timing signals that control the various processing elements involved in executing the instruction.
2. **Program counter (PC)**
   - i) The program counter is another specialized register.
   - ii) It keeps track of the execution of a program.
   - iii) It contains the memory address of the next instruction to be fetched and executed.
   - iv) 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.



3. **Memory address register (MAR) & Memory data register(MDR)**
   - i) These two registers facilitate communication with the memory.
   - ii) The MAR holds the address of the location to be accessed.

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

4. Operating steps for Program execution (or) Instruction Cycle

i) Execution of the program (stored in memory) starts when the PC is set to point to the first instruction of the program.

ii) The contents of the PC are transferred to the MAR and a Read control signal is sent to the memory.

iii) The addressed word is read out of the memory and loaded into the MDR. Next, the contents of the MDR are transferred to the IR. At this point, the instruction is ready to be decoded and executed.

iv) If the instruction involves an operation to be performed by the ALU, it is necessary to obtain the required operands.

v) If an operand resides in memory (it could also be in a general purpose register in the processor), it has to be fetched by sending its address to the MAR and initiating a Read cycle.

vi) When the operand has been read from the memory into the MDR, it is transferred from the MDR to ALU.

vii) After one or more operands are fetched in this way, the ALU can perform the desired operation.

viii) If the result of the operation is to be stored in the memory, then the result is ent to the MDR.

ix) The address of the location where the result is to be stored is sent to the MAR, and a write cycle is initiated.

x) At some point during the execution of the current instruction, the contents of the PC are incremented so that the PC pints to the next instruction to be executed.

xi) Thus, as soon as the execution of the current instruction is completed, a new instruction fetch may be started.

xii) In addition to transferring data between the memory and the processor, the computer accepts data from input devices and sends data to output devices. Thus, some machine instructions with the ability to handle I/O transfers are provided.

1.1.3 Bus Structures

Q6. What is bus? Write about the types of buses used in computer. **(Imp.)**

Ans :

BUS

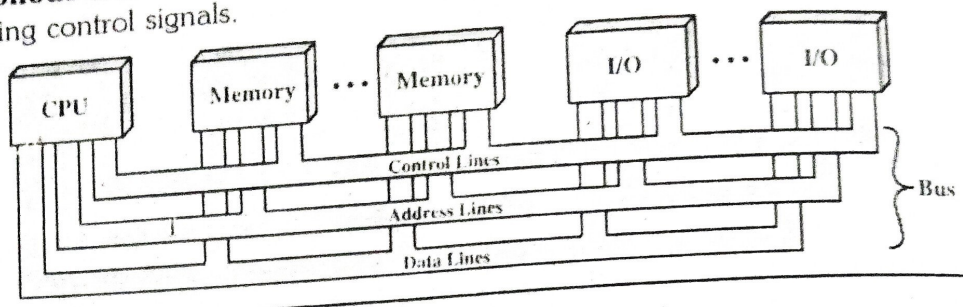
A group of lines(wires) that serves as a connecting path for several devices of a computer is called a bus.

- The Data bus Carries(transfer) data from one component (source) to other component (destination) connected to it. The data bus consists of 8, 16, 32 or more parallel signal lines. The data bus lines are bi-directional. This means that CPU can read data on these lines from memory or from a port, as well as send data out on these lines to a memory location.
- The Address bus is the set of lines that carry(transfer) address information about where in memory the data is to be transferred to or from. It is an unidirectional bus. The address bus consists of 16, 20, 24 or more parallel signal lines. On these lines CPU sends out the address of the memory location.

The Control Bus carries the Control and timing information. Including these three the following are various types of busses. They are,

- **System Bus:** A System Bus is usually a combination of address bus, data bus, and control bus respectively.
- **Internal Bus:** The bus that operates only with the internal circuitary of the CPU.
- **External Bus:** Buses which connects computer to external devices is nothing but external bus.

- **Back Plane:** A Back Plane bus includes a rowpf connectors into which system modules can be plugged in.
- **I/O Bus:** The bus used by I/O devices to communicate with the CPU is usually reffered as I/O bus
- **Synchronous Bus:** While using Synchro-nous bus, data transmission between source and destination units takes place in a **given timeslot** which is already known to these units.
- **Asynchronous Bus:** In this case the data transmission is governed by a special concept. That is handshaking control signals.



1.1.4 Software

Q7. What is software? Explain various types of software. (Imp.)

Ans :

A total computer system includes both software and Hardware .

1. Hardware consists of physical components and all associated equipment.
2. Software refers to the collection programs that are written for the computer and writing a program for a computer consists of specifying, directly or indirectly a sequence of machine instructions.
3. The computer software consists of the instructions and data that the computer manipulates to perform various data processing tasks.

Types

1. Application software,
2. System software

1. **Application software:** Application software allows end users to accomplish one or more specific (not directly computer development related) tasks. Its usually written in high level languages, such as c ,c++, java.

Typical applications include:

- Word processing
- spreadsheet
- computer games
- databases
- industrial automation
- business software
- quantum chemistry and solid state physics software
- telecommunications (i.e., the internet and everything that flows on it)

- educational software
- medical software
- military software
- molecularmodeling software
- image editing
- simulation software
- Decision making software

2. **System software:** System software is used to run application software.

System software is a collection of programs that are executed as needed to perform functions such as :

1. Receiving and interpreting user commands.
2. Entering and editing application programs and sorting them as files in secondary storage devices.(Editor)
3. Managing the storage and retrieval of files in secondary storage devices.
4. Running standard application programs such as word processors, spread sheets, or games, with data supplied by the user.
5. Controlling I/O units to receive input information and produce output results.
6. Translating programs from high level language to low level language. (Assemblers)
7. Linking and running user-written application program with existing standard library routines, such as numerical computation packages. (Linker)

The following are the types of system software:

- **Compiler:** A compiler is a computer program (or set of programs) that transforms source code written in a computer language (the source language) into another computer language (the target language, often having a binary form known as object code). The most common reason for wanting to transform source code is to create an executable program. The name "compiler" is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language or machine code). A program that translates from a low level language to a higher level one is a decompiler. A program that translates between high-level languages is usually called a language translator, source to source translator
- **Linker:** Linker is a program in a system which helps to link a object modules of program into a single object file. It performs the process of linking. Linker are also called link editors. Linking is process of collecting and maintaining piece of code and data into a single file. Linker also link a particular module into system library. It takes object modules from assembler as input and forms an executable file as output for loader. Linking is performed at both compile time, when the source code is translated into machine code and load time, when the program is loaded into memory by the loader. Linking is performed at the last step in compiling a program.
- **Assembler:** An assembler is a program that converts assembly language into machine code. It takes the basic commands and operations from assembly code and converts them into binary code that can be recognized by a specific type of processor. Assemblers are similar to compilers in that they produce executable code. However, assemblers are more simplistic since they only convert low-level code (assembly language) to machine code. Since each assembly language is designed for a specific processor, assembling a program is performed using a simple one-to-one mapping from assembly code to machine code.
- **Loader:** A loader is a major component of an operating system that ensures all necessary programs and libraries are loaded, which is essential during the startup phase of running a program. It places the libraries and programs into the main memory in order to prepare them for execution.

1.1.5 Performance

Q8. What is computer performance? Explain various measures of the performance of a computer. **(Imp.)**

Ans :

Performance

The most important measure of the performance of a computer is how quickly it can compute 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 a processor, we should consider only the periods during which the processor is active.

At the start of execution, all program instructions and the required data are stored in the memory as shown below. 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 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.

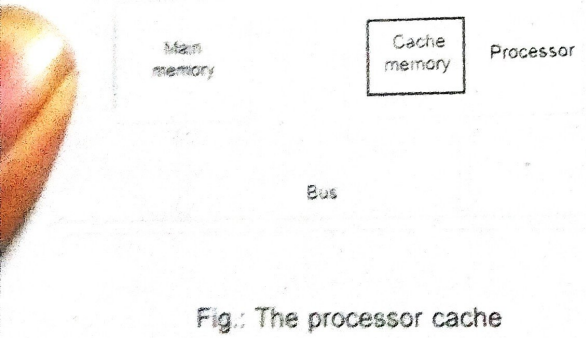


Fig.: The processor cache

Computer performance is often described in terms of clock speed (usually in MHz or GHz).

This refers to the cycles per second of the main clock of the CPU. Performance of a computer depends on the following factors.

a) Processor Clock

1. Processor circuits are controlled by a timing signal called a clock. A clock is a microchip that regulates speed and timing of all computer functions.
2. Clock Cycle is the speed of a computer processor, or CPU, which is the amount of time between two pulses of an oscillator. Generally speaking, the higher number of pulses per second, the faster the computer processor will be able to process information.
3. CPU clock speed, or clock rate, is measured in Hertz - generally in gigahertz, or GHz. A CPU's clock speed rate is a measure of how many clock cycles a CPU can perform per second
4. 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.
5. The length P of one clock cycle is an important parameter that affects processor performance.
6. Its inverse is the clock rate, R = 1/P, which is measured in cycles per second.
7. If the clock rate is 500(MHz) million cycles per second, then the corresponding clock period is 2 nanoseconds.

b) Basic performance equation

The Performance Equation is a term used in computer science. It refers to the calculation of the performance or speed of a central processing unit (CPU).

Basically the Basic Performance Equation [BPE] is an equation with 3 parameters which are required for the calculation of "Basic Performance" of a given system. It is given by

$$T = (N*S)/R$$

Where 'T' is the processor time [Program Execution Time] required to execute a given program written in some high level language .The compiler generates a machine language object program corresponding to the source program.

'N' is the total number of steps required to complete program execution. 'N' is the actual number of instruction executions, not necessarily equal to the total number of machine language instructions in the object program. Some instructions are executed more than others (loops) and some are not executed at all (conditions).

'S' is the average number of basic steps each instruction execution requires, where each basic step is completed in one clock cycle. We say average as each instruction contains a variable number of steps depending on the instruction.

'R' is the clock rate [In cycles per second]

c) Pipelining and Super scalar operation

1. A substantial improvement in performance can be achieved by overlapping the execution of successive instructions, using a technique called pipelining.
2. Consider the instruction
3. Add R1, R2, R3
4. Which adds the contents of registers R1 and R2, and places the sum into R3
5. The contents of R1 and R2 are first transferred to the inputs of the ALU.
6. After the add operation is performed, the sum is transferred to R3.
7. Processor can read the next instruction from the memory while the addition operation is being performed.
8. 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 add instruction is being transferred to R3.
9. Thus, pipelining increases the rate of executing instructions significantly.

d) Super scalar operation

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

e) Clock rate

1. There are two possibilities for increasing the clock rate, R.
2. First, improving the Integrated Circuit technology makes logic circuit faster, which reduces the needed to complete a basic step. This allows the clock period, P, to be reduced and the clock rate, R, to be increased.
3. Second, reducing the amount of processing done in one basic step also makes it possible to reduce the clock period, P.

f) Instruction set: CISC and RISC

1. The terms CISC and RISC refer to design principles and techniques.
2. RISC: Reduced instruction set com puters.
3. Simple instructions require a small number of basic steps to execute.
4. For a processor that has only simple instructions, a large number of instructions may by need to perform a given programming task. This could lead to a large value of N and a small value for S.
5. It is much easier to implement efficient pipelining in processors with simple instruction sets.
6. CISC: Complex instruction set computers.
7. Complex instructions involve a large number of steps.
8. If individual instructions perform more complex operations, fewer instructions will be needed, leading to a lower value of N and a larger value of S.
9. Complex instructions combined with pipelining would achieve good performance.

g) Optimizing Compiler

1. A compiler translates a high-level language program into a sequence of machine instructions.
2. To reduce N, we need to have a suitable machine instruction set and a compiler that makes good use of it.
3. An optimizing compiler takes advantage of various features of the target processor to reduce the product N * S.
4. The compiler may rearrange program instructions to achieve better performance.

h) Performance measurement

1. SPEC rating.
2. A nonprofit organization called" System Performance Evaluation Corporation" (SPEC) selects and publishes representative application programs for different application domains.
3. The SPEC rating is computed as follows.
4. SPEC rating = Running time on the reference computer Running time on the computer under test.
5. Thus SPEC rating of 50 means that the computer under test is 50 times faster than the reference computer for these particular benchmarks.
6. The test is repeated for all the programs in the SPEC suite, and the geometric mean of the results is computed.
7. Let SPEC, be the rating for program 'i' in the suite.

The overall SPEC rating for the computer is given by,

$$\text{SPEC rating} = \prod_{i=1}^{n} (\text{SPECL}_1)$$

Where n is the number of programs in the suite.

1.1.6 Memory Locations and Addresses

Q9. Explain about the memory locations and addresses.

Ans :

Memory-Locations and Addresses

- Memory consists of many millions of storage cells (flip-flops).
- Each cell can store a bit of information i.e. 0 or 1 (Figure).
- Each group of n bits is referred to as a word of information, and n is called the word length.
- The word length can vary from 8 to 64 bits.
- A unit of 8 bits is called a byte.
- Accessing the memory to store or retrieve a single item of information (word/byte) requires distinct addresses for each item location. (It is customary to use numbers from 0 through 2^k-1 as the addresses of successive-locations in the memory).
- If 2^k = no. of addressable locations; then 2^k addresses constitute the address-space of thecomputer. For example, a 24-bit address generates an address-space of 2^24 locations (16 MB)
- To access the memory location either you must know the memory location by its unique name or it is required to provide a unique address to each memory location.
- The memory locations are addressed from 0 to 2^{K-1} i.e. a memory has 2^K addressable locations. And thus the address space of the computer has 2^K addresses. Let us try some suitable values for K.

2^{10} = 1024 = 1K (Kilobyte)

2^{20} = 1,048,576 = 1M (Megabyte)

2^{30} = 1073741824 = 1G (Gigabyte)

2^{40} = 1.0995116e+12 = 1T (Terabyte)

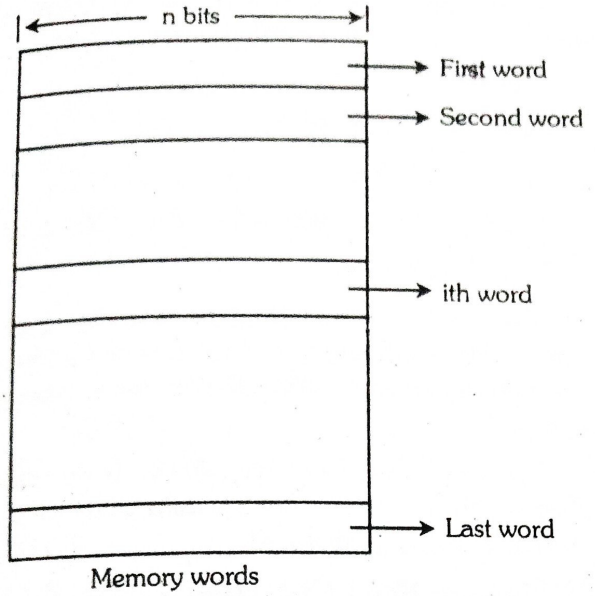


Fig.-1

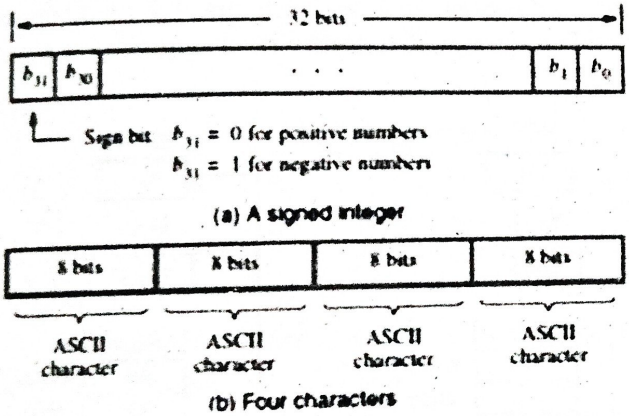


Fig. 2: Examples of encoded information in a 32-bit word

Q10. Explain, how to assign byte addresses in the memory.

(OR)

Write about big-endian or little-endian assignment process for byte addresses.

(Imp.)

Ans :

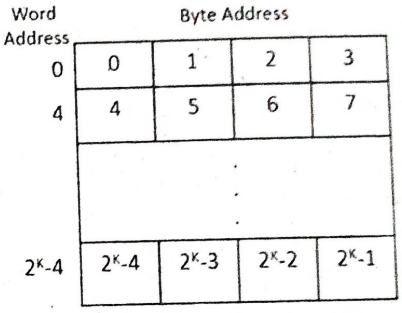
Byte-Addressability

A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits.

- In byte-addressable memory, successive addresses refer to successive byte locations in the memory.
- Byte locations have addresses 0, 1, 2. . . . .
- If the word-length is 32 bits, successive words are located at addresses 0, 4, 8. . with each word having 4 bytes.

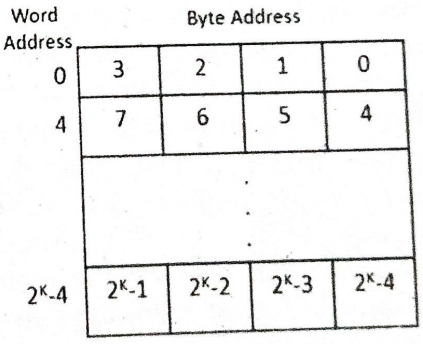
ig-Endian and Little-Endian Assignments in Byte Addresses

The big-endian and little-endian are two methods of assigning byte addresses across the words in the memory. In the big-endian assignment, the lower byte addresses are used for the leftmost bytes of the word. Observe the word 0 in the image below, the leftmost bytes of the word have lower byte addresses.



Big-Endian Assignment

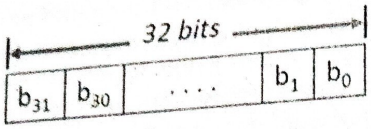
In the little-endian assignment, the lower byte addresses are used for the rightmost bytes of the word. Observe the word 0 in the image below the rightmost bytes of word 0 has lower byte addresses.



Little-Endian Assignment

The leftmost bytes of the word are termed as most significant bytes and the rightmost bytes of the words are termed as least significant bytes.

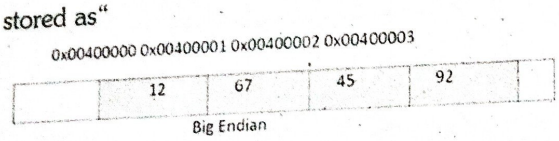
Thus the big-endian and little-endian specify the ordering of bytes inside a word. Similarly, the bits must be labelled inside the byte or a word and the most common way of labelling bits in a byte or word is as shown in the figure below i.e. labelling the bits as $b_7, b_6, \ldots\ldots, b_1, b_0$ from left to write as we do in little-endian assignment.



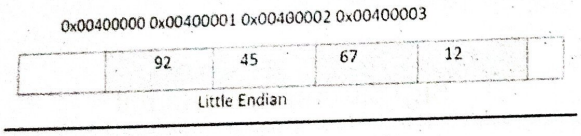
Labelling bits within a Byte

Example

0x12674592 in 32-bit representation can be stored as"



Big Endian



Little Endian

1.1.7 Memory Operations

Q11. Write about the operations that can perform on memory.

Ans : **(Imp.)**

A memory unit stores binary information in groups of bits called words. Data input lines provide the information to be stored into the memory, Data output lines carry the information out from the memory. The control lines Read and write specifies the direction of transfer of data. Basically, in the memory organization, there are n memory locations indexing from 0 to 2^l-1 where l is the address buses. We can describe the memory in terms of the bytes using the following formula:

$$N = 2^l$$

Where,

l is the total address buses

N is the memory in bytes

For example, some storage can be described below in terms of bytes using the above formula:

$$1kB = 2^{10} \text{ Bytes}$$

$$64\ kB = 2^6 \times 2^{10} \text{ Bytes}$$

$$= 2^{16} \text{ Bytes}$$

$$4\ GB = 2^2 \times 2^{10}(kB) \times 2^{10}(MB) \times 2^{10}(GB)$$

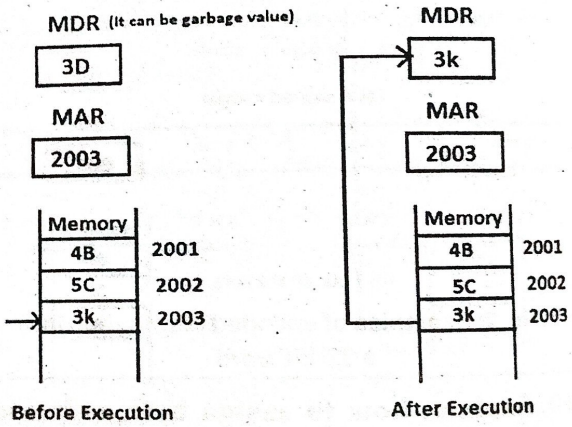
$$= 2^{32} \text{ Bytes}$$

Memory Address Register (MAR) is the address register which is used to store the address of the memory location where the operation is being performed.

Memory Data Register (MDR) is the data register which is used to store the data on which the operation is being performed.

1. Memory Read Operation:

Memory read operation transfers the desired word to address lines and activates the read control line.Description of memory read operation is given below:

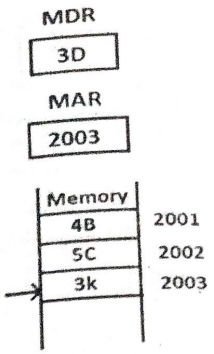


Memory Read Operation

In the above diagram initially, MDR can contain any garbage value and MAR is containing 2003 memory address. After the execution of read instruction, the data of memory location 2003 will be read and the MDR will get updated by the value of the 2003 memory location (3D).

2. Memory Write Operation

Memory write operation transfers the address of the desired word to the address lines, transfers the data bits to be stored in memory to the data input lines. Then it activates the write control line. Description of the write operation is given below:



Memo

In the above
2003 and MDR cont
write instruction 3D
location.

1.1.8 Instruction cing

Q12. List the ty perform on

Ans :

Computer
sequence of sma
numbers, testing f
the character from k
to be displayed on

A computer
of performing four

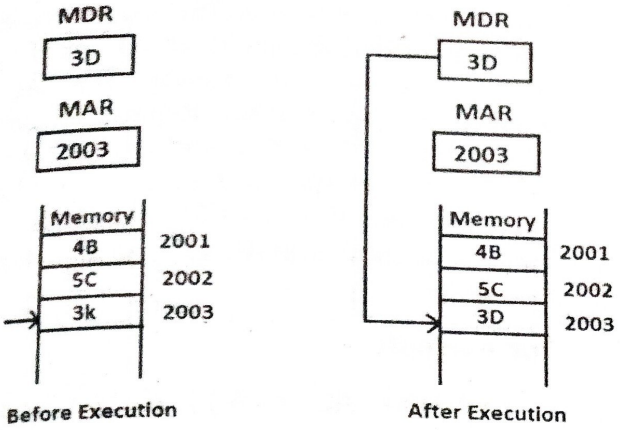
1. Data transfe processor re
2. Arithmetic a
3. Program se
4. I/O transfers

Q13. How to wri Explain wi

Ans :

Register Transfe

It is used t
location to other
RTN, source is alw
side of "→". D
register, specified



Memory Write Operation

In the above diagram, the MAR contains 2003 and MDR contains 3D. After the execution of write instruction 3D will be written at 2003 memory location.

1.1.8 Instruction and Instruction Sequencing

Q12. List the type of operations that can perform on the computer.

Ans :

Computer programming consists of a sequence of small steps, such as adding two numbers, testing for particular condition ,reading the character from keyboard and sending a character to be displayed on screen.

A computer must have instructions capable of performing four types of operations:

1. Data transfers between the memory and the processor registers
2. Arithmetic and logic operations on data
3. Program sequencing and control
4. I/O transfers

Q13. How to write Register Transfer notation? Explain with an example.

Ans : **(Imp.)**

Register Transfer Notation

It is used to transfer information from one location to other location inside the computer. In RTN, source is always a value specified on right hand side of "→". Destination is always a processor register, specified on left hand side.

Syntax :Register → Source

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. Source can be processor register, I/O register, memory location, but destination register is always a processor register. RTN uses square brackets to indicate content of location. These braces are always placed only around the Source.

For example,

1. R3 → R1] + [23]

   This operation that adds the contents of registers R1 and R2, and places their sum into register R3

2. R2 → [LOC], means that the contents of memory location LOC are transferred into processor register R2.

Assembly Language Notation: Assembly Language Notation is a type of notation which is used to represent machine instructions and programs.

For example:

LOAD LOC, R2

A generic instruction that causes the transfer, from memory location LOC to processor register R2, is specified by the statement

The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R2 are overwritten. The name Load is appropriate for this instruction, because the contents read from a memory location are *loaded* into a processor register.

The second example :

ADD R4, R2, R3

Adding two numbers contained in processor registers R2 and R3 and placing their sum in R4 can be specified by the assembly-language statement

In this case, registers R2 and R3 hold the source operands, while R4 is the destination.

An *instruction* specifies an operation to be performed and the operands involved. In the above examples, we used the English words Load and Add to denote the required operations.

In the assembly-language instructions of actual (commercial) processors, such operations are defined by using *mnemonics*, which are typically abbreviations of the words describing the operations. For example, the operation Load may be written as LD, while the operation Store, which transfers a word from a processor register to the memory, may be written as STR or ST. Assembly languages for different processors often use different mnemonics for a given operation.

Q14. Explain basic instruction types with an examples.

Ans :

Basic Instruction Types

An instruction is of various lengths depending upon the number of addresses it contains. Generally CPU organization is of three types on the basis of number of address fields:

- Single Accumulator organization
- General register organization
- Stack organization

1. In first organization operation is done involving a special register called accumulator.
2. In second on multiple registers are used for the computation purpose.
3. In third organization the work on stack basis operation due to which it does not contain any address field. On the basis of number of addresses instructions are classified as:

Three address Instructions: This instruction has three operands(address fields) to specify a register or a memory location.

- Syntax :operation source1, source2,

EX: Add A, B, C [C<-[A]+[B])

Where A, B are called source operands, C is called destination operand.

- **Two address Instructions**: This instruction has two operands(address fields) to specify a register or a memory location.
- **Syntax : operation source, destination.**

For example,

Add A, C (C→[A] + [C])

- One address Instructions: This instruction has one operand (address field) to specify a register or a memory location. This use a implied Accumulator(AC) Register for data manipulation. One operand is in AC and other is in register or memory location. Implied means that the CPU already knows that one operand is in AC so there is no need to specify it.

For example,

LOAD A (AC→[A])

ADD B (AC→[AC] + [B])

STORE C (C→[AC])

- **Zero address Instructions:** This instruction has zero address fields. A stack based computer do not use address field in instruction. It uses stack operations PUSH and POP to perform operations. To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation. For example,

Push A (TOS → [A])

Push B (TOS → [B])

Add (TOS → [A] + [B])

Pop C (C→[TOS])

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

Move A, R1 (R1→[A])

Add B, R1 (R1→[R1] + [B])

Move C, R2 (R2→[C])

Add D, R2 (R2→[R2] + [D])

Mul R1, R2 (R2→[R1] * [R2])

Move 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])

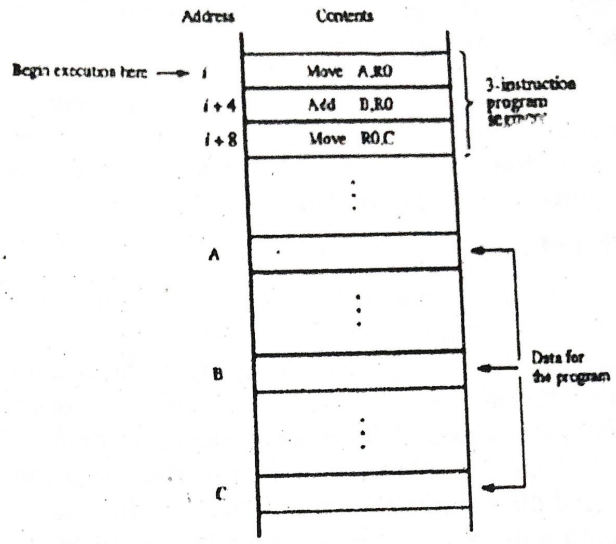
Mul (TOS → ([A] + [B]) * ([C] + [D])

Pop X (X → [A] + [B]) * ([C] + [D])

Q15. Write about instruction execution sequencing or straight line sequencing. (Imp.)

Ans :

Instruction Execution and Straight-Line Sequencing: To begin executing a program, the address of its first instruction (I in our example) must be placed into the PC. Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses. This is called straight-line sequencing. For example, consider the following assembly instruction which add contents of two memory locations. i.e. C à[A] + [B]. The following diagram shows a possible program segment for this task as it appears in the memory of a computer.



A program for C ← [A] + [B].

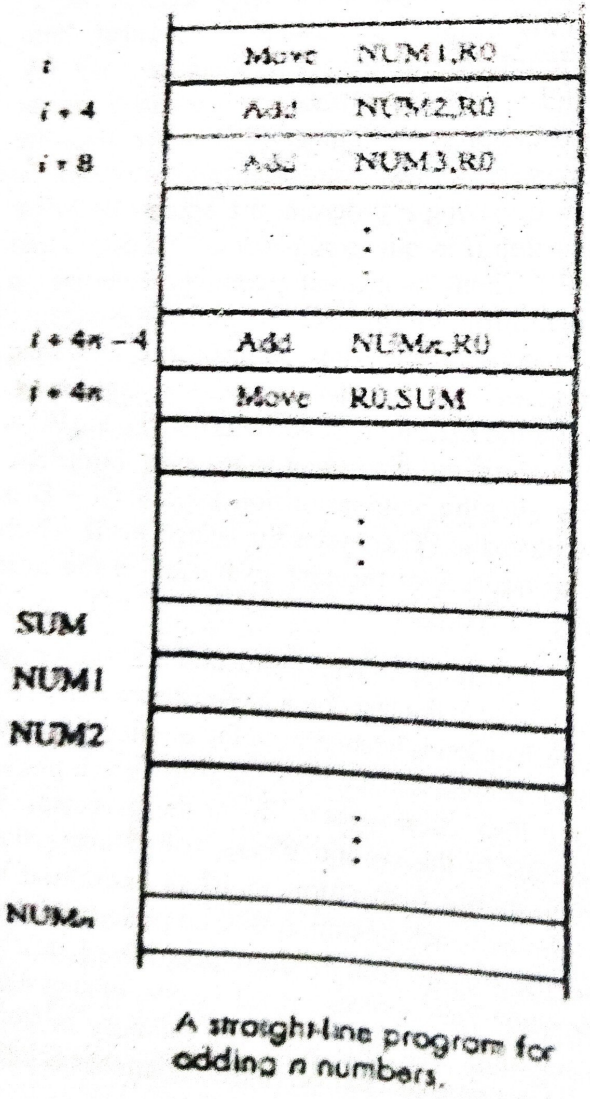
The four instructions of the program are in successive word locations, starting at location *i*. Since, each instruction is 4 bytes long, the second, third, and fourth instructions are at addresses *i* + 4, *i* + 8, and *i* + 12. The processor contains a register called the *program counter* (PC), which holds the address of the next instruction to be executed. To begin executing a program, the address of its first instruction (*i* in our example) must be placed into the PC. Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses. This is called *straight-line sequencing*. During the execution of each instruction, the PC is incremented by 4 to point to the next instruction. Thus, after the Store instruction at location *i* + 12 is executed, the PC contains the value *i* + 16, which is the address of the first instruction of the next program segment.

Executing a given instruction is a two-phase procedure. In the first phase, called *instruction fetch*, the instruction is fetched from the memory location whose address is in the PC. This instruction is placed in the *instruction register* (IR) in the processor. At the start of the second phase, called *instruction execute*, the instruction in IR is examined to determine which operation is to be performed. The specified operation is then performed by the processor. This involves a small number of steps such as fetching operands from the memory or from processor registers, performing an arithmetic or logic

operation, and storing the result in the destination location. At some point during this two-phase procedure, the contents of the PC are advanced to point to the next instruction. When the execute phase of an instruction is completed, the PC contains the address of the next instruction, and a new instruction fetch phase can begin.

Branching

Normally, the instructions executed in linear fashion through the program, and the address of the instructions is obtained from PC in the control unit. This sequence is interrupted when a branch instruction is executed, at such a time the address field of the Branch instruction is inserted into the PC and the process continues. Consider the task of adding a list of *n* numbers. The following diagram shows straight line sequencing program to add list of n numbers.



A straight-line program for adding *n* numbers.

The addresses of the memory locations containing the *n* numbers are symbolically given as NUM1, NUM2, . . . , NUM*n*, and separate Add instructions is used to add each number to the contents of register R0. After all the numbers have been added, the result is placed in memory location SUM.

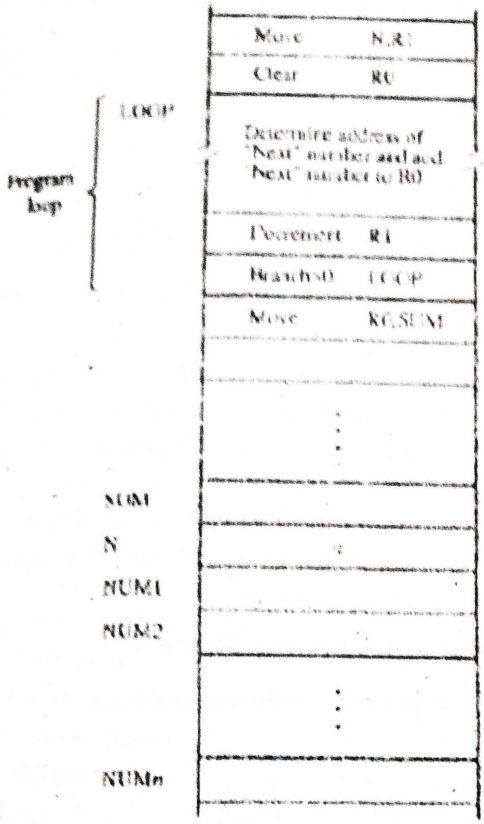
Instead of using a long list of Add instructions, it is possible to implement a program loop in which the instructions read the next number in the list and add it to the current sum. To add all numbers, the loop has to be executed as many times as there are numbers in the list. The following shows the structure of the desired program. The body of the loop is a straight-line sequence of instructions executed repeatedly. It starts at location LOOP and ends at the instruction Branch >0. During each pass through this loop, the address of the next list entry is determined, and that entry is fetched and added to R0.

Assume that the number of entries in the list, *n*, is stored in memory location N, as shown. Register R1 is used as a counter to determine the number of times the loop is executed. Hence, the contents of location N are loaded into register R1 at the beginning of the program. Then, within the body of the loop, the instruction

Decrement R1

Reduces the contents of R1 by 1 each time through the loop. Execution of the loop is repeated as long as the contents of R1 are greater than zero.

Next use branch instruction. This type of instruction loads a new address into the program counter. As a result, the processor fetches and executes the instruction at this new address, called the *branch target*, instead of the instruction at the location that follows the branch instruction in sequential address order. A *conditional branch* instruction causes a branch only if a specified condition is satisfied. If the condition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed.



Using a loop to add n numbers

The instruction Branch> 0 LOOP is a conditional branch instruction that causes a branch to location LOOP if the contents of register R1 are greater than zero. This means that the loop is repeated as long as there are entries in the list that are yet to be added to R0. At the end of the *n*th pass through the loop, the Subtract instruction produces a value of zero in R2, and, hence, branching does not occur. Instead, the Store instruction is fetched and executed. It moves the final result from R0 into memory location SUM.

Condition Codes

The processor keeps track of information about the results of various operations for use by subsequent conditional branch instructions. This is done by recording the required information in individual bits, often called condition code flags. These flags are grouped together in a special processor register called condition code register or status register. Individual condition code flags are set to 1 or cleared to 0, depending on outcome of operation performed.

Four commonly used flags are

- **N (negative):** Set to 1 if the result is negative; otherwise, cleared to 0.
- **Z (zero):** Set to 1 if the result is zero; otherwise, cleared to 0.
- **V (overflow):** Set to 1 if arithmetic overflow occurs; otherwise, cleared to 0.
- **C (carry):** Set to 1 if a carry-out results from the operation; otherwise, cleared to 0.

1.1.9 Addressing Modes

Q16. What is Addressing Mode? How to represent it.

Ans : **(Imp.)**

The operation field of an instruction specifies the operation to be performed.And this operation must be performed on some data.So each instruction need to specify data on which the operation is to be performed.But the operand(data) may be in accumulator, general purpose register or at some specified memory location.So, appropriate location (address) of data is need to be specified, and in computer, there are various ways of specifying the address of data.These various ways of specifying the address of data are known as "Addressing Modes"

So Addressing Modes can be defined as-

"The technique for specifying the address of the operands "

And in computer the address of operand i.e., the address where operand is actually found is known as"Effective Address".

Now, in addition to this, the two most prominent reason of why addressing modes are so important are:

1. First, the way the operand data are chosen during program execution is dependent on the addressing mode of the instruction.
2. Second, the address field(or fields) in a typical instruction format are relatively small and sometimes we would like to be able to reference a large range of locations, so here to achieve this objective i.e., to fit this large range of location in address field, a variety of

addressing techniques has been employed. As they reduce the number of field in the addressing field of the instruction.

Thus, Addressing Modes are very vital in Instruction Set Architecture(ISA).

Now, before discussing various addressing modes, hereare some notations These are:

A = Contents of an address field in the instruction

R = Contents of an address field in the instruction that refers to a register

EA = Effective Address(Actual address) of location containing the referenced operand.

(X) = Contents of memory location x or register X.

Q17. Explain briefly various types of addressing modes.

Ans :

Types of Addressing Modes

1. Implied and Immediate Addressing Modes
2. Direct or Indirect Addressing Modes
3. Register Addressing Modes
4. Register Indirect Addressing Mode
5. Auto-Increment and Auto-Decrement Addressing Modes
6. Displacement Based Addressing Modes

1. Implied and Immediate Addressing Modes:

Although most Addressing modes need the address field of the instruction, but implied and immediate addressing modes are the only addressing modes that need no address field at all.

Implied Addressing Mode:

Implied Addressing Mode also known as "Implicit" or "Inherent" addressing mode is the addressin mode in which, no operand(register or memory location or data) is specified in the instruction. As inthis mode the operand are specified implicit in the definition of instruction.

As an example: The instruction :

"Complement Accumulator" is an Implied Mode instruction because the operand in the accumulator register is implied in the definition of instruction.In assembly language it is written as:

CMA: Take complement of content of AC

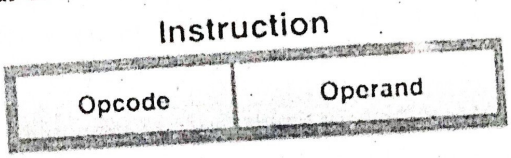
Similarly, the instruction,

RLC: Rotate the content of Accumulator is an implied mode instruction.

In addition to this, all Register-Reference instruction that use an accumulator and Zero-Address instruction in a Stack Organised Computer are implied mode instructions, because in Register reference operand implied in accumulator and in Zero-Address instruction, the operand implied onthe Top of Stack.

Immediate Addressing Mode:

In Immediate Addressing Mode operand is specified in the instruction itself.In other words, an immediate mode instruction has an operand field rather than an address field, which contain actual operand to be used in conjunction with the operand specified in the instruction.That is, in this mode, the format of instruction is:



As an example: The Instruction:

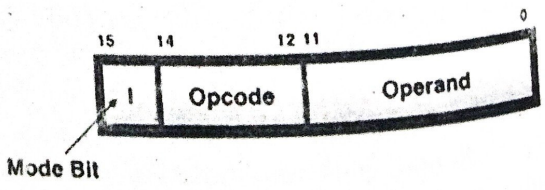
MVI 06 Move 06 to the accumulator

ADD 05 ADD 05 to the content of accumulator

In addition to this, this mode is very useful for initialising the register to a constant value.

2. Direct and Indirect Addressing Modes:

The instruction format for direct and indirect addressing mode is shown below:



Instruction Format

It consists of 3-bit opcode, 12-bit address and a mode bit designated as(I).The mode bit (I) is zero for Direct Address and 1 for Indirect Address.

Direct Addressing Mode

Direct Addressing Mode is also known as "Absolute Addressing Mode".In this mode the address of data(operand) is specified in the instruction itself.That is, in this type of mode, the operand resides in memory and its address is given directly by the address field of the instruction.

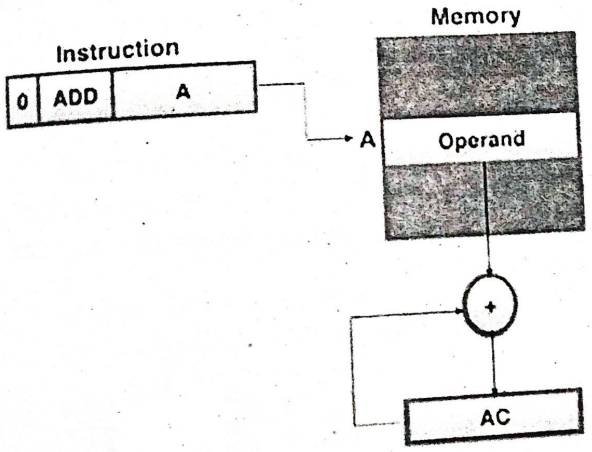
Means, in other words, in this mode, the address field contain the Effective Address of operand

i.e., EA=A

As an example: Consider the instruction:

ADD A Means add contents of cell A to accumulator.

It Would look like as shown below:



Here, we see that in it Memory Address = Operand.

Indirect Addressing Mode

In this mode, the address field of instruction gives the memory address where on, the operand is stored in memory.That is, in this mode, the address field of the instruction gives the address where the "Effective Address" is stored inmemory.

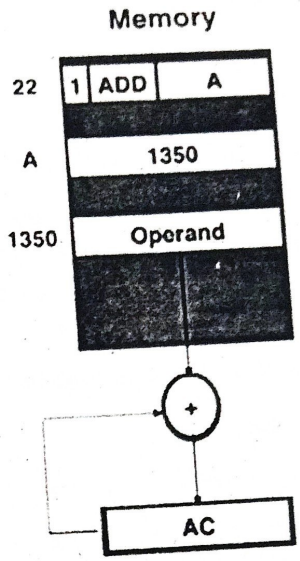
i.e., EA=(A)

Means, here, Control fetches the instruction from memory and then uses its address part to access memory again to read Effective Address.

As an example: Consider the instruction:

ADD (A) Means adds the content of cell pointed to contents of A to Accumulator.

It look like as shown in figure below:



Thus in it, AC <— M[M[A]] [M=Memory]

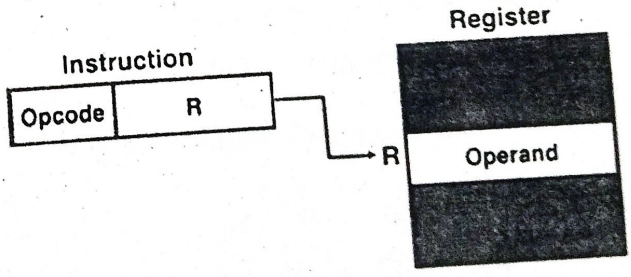
i.e., (A)=1350=EA

3. Register Addressing Mode

In Register Addressing Mode, the operands are in registers that reside within the CPU.That is, in this mode, instruction specifies a register in CPU, which contain the operand.It is like Direct Addressing Mode, the only difference is that the address field refers to a register instead of memory location.

i.e., EA=R

It look like as:



Example of such instructions are:

MOV AX, BX

Move contents of Register BX to AX

ADD AX, BX

Add the contents of register BX to AX

Here, AX, BX are used as register names which is of 16-bit register.

Thus, for a Register Addressing Mode, there is no need to compute the actual address as the operand is in a register and to get operand there is no memory access involved.

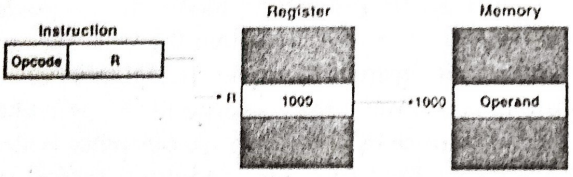
4. Register Indirect Addressing Mode

In Register Indirect Addressing Mode, the instruction specifies a register in CPU whose contents give the operand in memory.In other words, the selected register contain the address of operand rather than the operand itself.That is,

i.e., EA=(R)

Means, control fetches instruction from memory and then uses its address to access Register and looks in Register(R) for effective address of operand in memory.

It look like as:



Here, the parentheses are to be interpreted as meaning contents of.

Example

MOV AL, [BX]

Code example in Register:

MOV BX, 1000H

MOV 1000H, operand

From above example, it is clear that, the instruction (MOV AL, [BX]) specifies a register [BX], and in coding of register, we see that, when we move register [BX], the register contain the address of operand(1000H) rather than address itself.

5. Auto-increment and Auto-decrement Addressing Modes

These are similar to Register indirect Addressing Mode except that the register is incremented or decremented after(or before) its value is used to access memory.

These modes are required because when the address stored in register refers to a table of data in memory, then it is necessary to increment or decrement the register after every access to table so that next value is accessed from memory.Thus, these addressing modes are common requirements in computer.

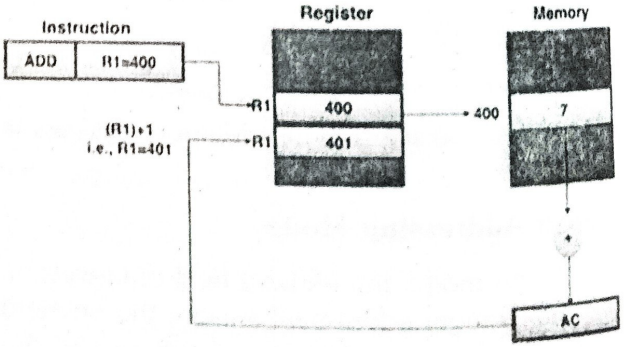
Auto-increment Addressing Mode

Auto-increment Addressing Mode are similar to Register Indirect Addressing Mode except that the register is incremented after its value is loaded (or accessed) at another location like accumulator(AC).

That is, in this case also, the Effective Address is equal to

EA=(R)

But, after accessing operand, register is incremented by 1.



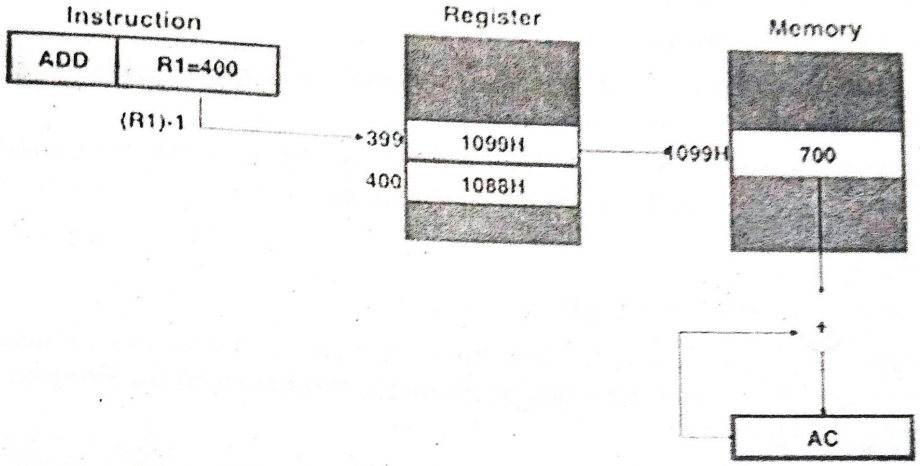
Here, we see that effective address is (R)=400 and operand in AC is 7.

And after loading R1 is incremented by 1.It becomes 401.

Means, here we see that, in the Auto-increment mode, the R1 register is increment to 401 after execution of instruction.

Auto-decrement Addressing Mode

Auto-decrement Addressing Mode is reverse of auto-increment, as in it the register is decrement before the execution of the instruction. That is, in this case, effective address is equal to EA=(R)-1.



Here, we see that, in the Auto-decrement mode, the register R1 is decremented to 399 prior to execution of the instruction, means the operand is loaded to accumulator, is of address 1099H in memory, instead of 1088H.Thus, in this case effective address is 1099H and contents loaded into accumulator is 700.

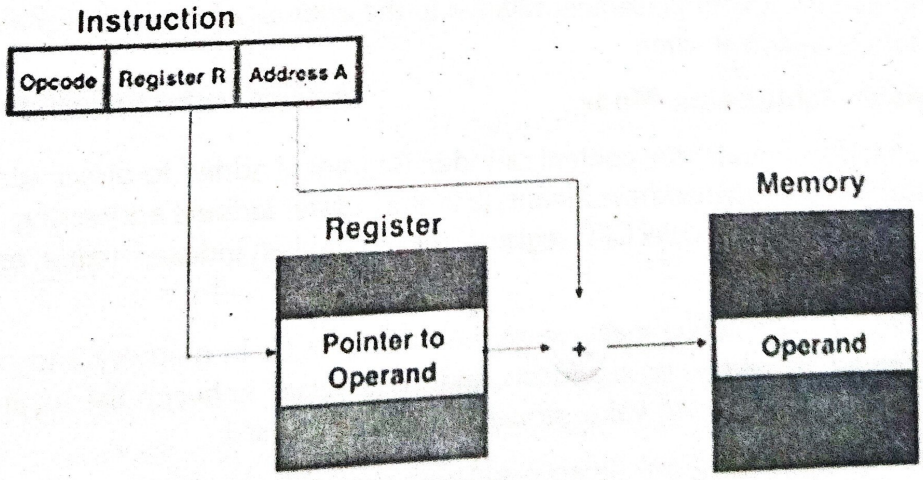
6. Displacement Based Addressing Modes:

Displacement Based Addressing Modes is a powerful addressing mode as it is a combination of direct addressing or register indirect addressing mode.

i.e., EA=A+(R)

Means, Displacement Addressing Modes requires that the instruction have two address fields, at least one of which is explicit means, one is address field indicate direct address and other indicate indirect address.That is, value contained in one addressing field is A, which is used directly and the value in other address field is R, which refers to a register whose contents are to be added to produce effective address.

It look like as shown in figure below:



There are three areas where Displacement Addressing modes are used.In other words, Displacement Based Addressing Modes are of three types. These are:

1. Relative Addressing Mode
2. Indexing Addressing Mode
3. Base Register Addressing Mode

Now we will explore to each one by one.

1. Relative Addressing Mode

In Relative Addressing Mode , the contents of program counter is added to the address part of instruction to obtain the Effective Address.

That is, in Relative Addressing Mode, the address field of the instruction is added to implicitly reference register Program Counter to obtain effective address.

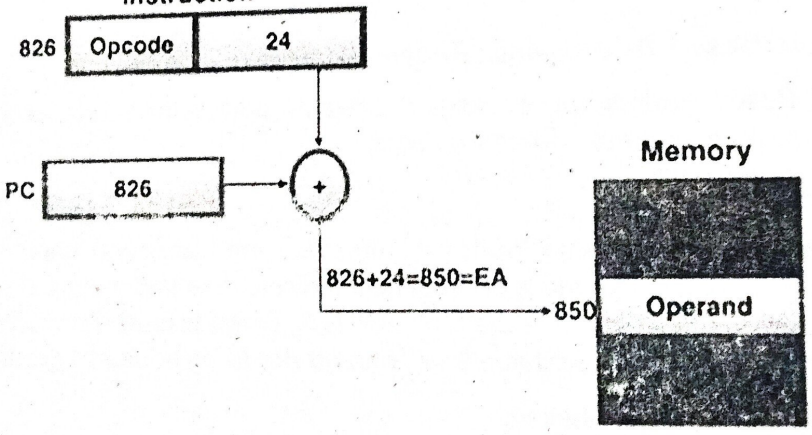
i.e., EA=A+PC

It becomes clear with an example:

Assume that PC contains the no.- 825 and the address part of instruction contain the no.- 24, then the instruction at location 825 is read from memory during fetch phase and the Program Counter is then incremented by one to 826.

The effective address computation for relative address mode is 826+24=850

It is depicted in fig. below:



Thus, Effective Address is displacement relative to the address of instruction.Relative Addressing is often used with branch type instruction.

2. Index Register Addressing Mode

In indexed addressing mode, the content of Index Register is added to direct address part(or field) of instruction to obtain the effective address.Means, in it, the register indirect addressing field of instruction point to Index Register, which is a special CPU register that contain an Indexed value, and direct addressing field contain base address.

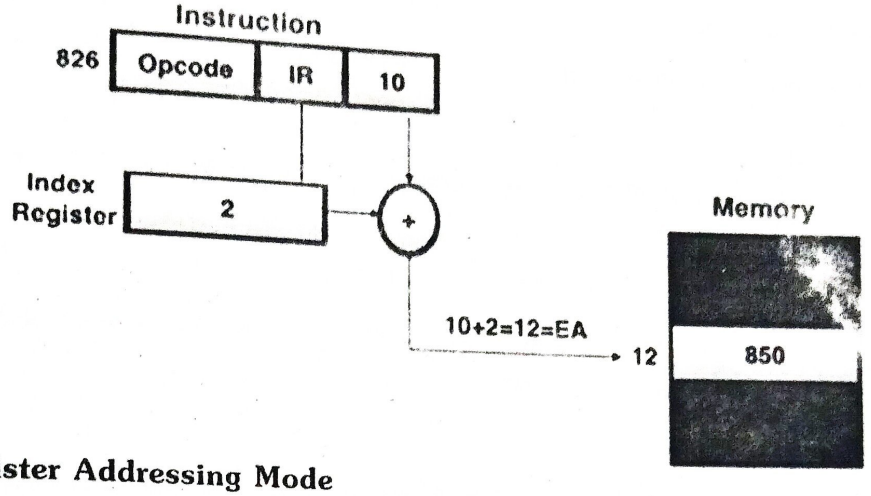
As, indexed type instruction make sense that data array is in memory and each operand in the array is stored in memory relative to base address.And the distance between the beginning address and the address of operand is the indexed value stored in indexed register.

Any operand in the array can be accessed with the same instruction, which provided that the index register contains the correct index value i.e., the index register can be incremented to facilitate access to consecutive operands.

Thus, in index addressing mode

EA=A+Index

It looks like as shown in fig. below:

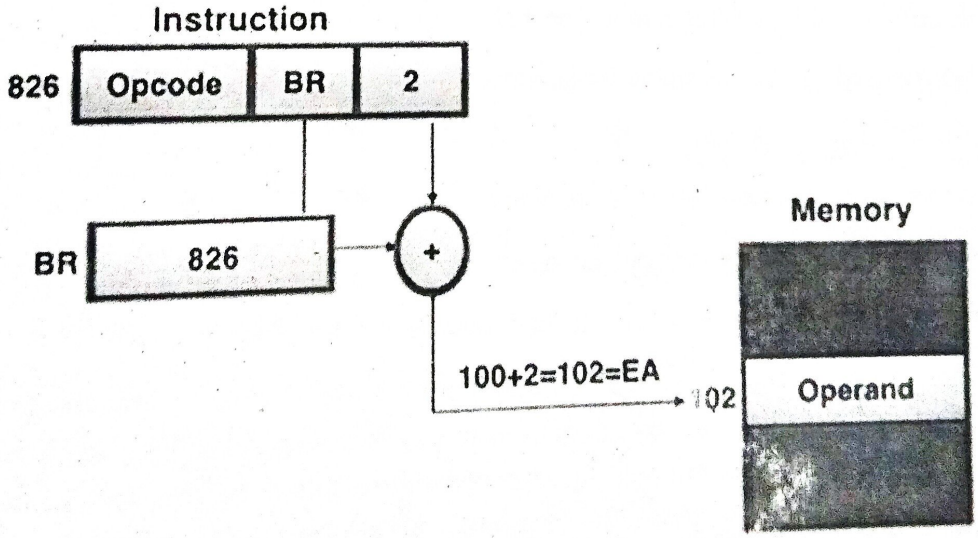


3. Base Register Addressing Mode

In this mode, the content of the Base Register is added to the direct address part of the instruction to obtain the effective address.Means, in it the register indirect address field point to the Base Register and to obtain EA, the contents of Instruction Register, is added to direct address part of the instruction.This is similar to indexed addressing mode except that the register is now called as Base Register instead of Index Register.

That is, the EA=A+Base

It looks like as shown in fig. below:



Thus, the difference between Base and Index mode is in the way they are used rather than the way they are computed.An Index Register is assumed to hold an index number that is relative to the address part of the instruction.And a Base Register is assumed to hold a base address and the direct address field of instruction gives a displacement relative to this base address.

Thus, the Base register addressing mode is used in computer to facilitate the relocation of programs in memory.

Means, when programs and data are moved from one segment of memory to another, then Base address is changed, the displacement value of instruction do not change.So, only the value of Base Register requires updation to reflect the beginning of new memory segment.

1.1.10 Assembly Language

Q18. What is assembly language and write the rules of it. **(Imp.)**

Ans :

- The symbolic program (contains letters, numerals, or special characters) is referred to as an assembly *language program.*
- The basic unit of an assembly language program is a line of code.
- The specific language is defined by a set of rules that specify the symbols that can be used and they may be combined to form a line of code.

Rules of the Language

Each line of an assembly language program is arranged in three columns called fields. The fields specify the following information.

The ***label*** field may be empty or it may specify a symbolic address.

A symbolic address consists of one, two, or three, but not more than three alphanumeric characters. The first character must be a letter; the next two may be letters or numerals. The symbol can be chosen arbitrarily by the programmer. A symbolic address in the label field is terminated by a comma so that it will be recognized as a label by the assembler.

The ***instruction*** field specifies a machine instruction or a pseudo instruction.

The instruction field in an assembly language program may specify one of the following items:

- A memory-reference instruction (MRI)
- A register-reference or input-output instruction (non-MRI)
- A pseudo instruction with or without an operand.

The ***comment*** field may be empty or it may include a comment.

A line of code may or may not have a comment, but if it has, it must be preceded by a slash for the assembler to recognize the beginning of a comment field. Comments are useful for explaining the program and are helpful in understanding the step-by-step procedure taken by the program. Comments are inserted for explanation purpose only and are neglected during the binary translation process.

Q19. Explain pseudo instruction.

Ans :

A pseudo instruction is not a machine instruction but rather an instruction to the assembler giving information about some phase of the translation. Four pseudo instructions that are recognized by the assembler are listed in Table.

Symbol	
ORG N	Hexac follow
END	Deno
DEC N	Signe
HEX N	Hexa

The ORG following line is t use ORG more t

The END terminated.

The other assembler how t

1.1.11 Basic

Q20. Write ab

Ans :

Input / O An I/O device i which provides to facilitate the that can be acc

Conside display screen. I/O. A solution waits for a sign and so on. Inp a character ke keyboard. The

The key

Symbol	Information for the Assembler
ORG N	Hexadecimal number N is the memory location for the instruction or operand listed in the following line
END	Denotes the end of symbolic program
DEC N	Signed decimal number N to be converted to binary
HEX N	Hexadecimal number N to be converted to binary

Table: Definition of Pseudo instructions

The ORG (origin) pseudo instruction informs the assembler that the instruction or operand in the following line is to be placed in a memory location specified by the number next to ORG. It is possible to use ORG more than once in a program to specify more than one segment of memory.

The END symbol is placed at the end of the program to inform the assembler that the program is terminated.

The other two pseudo instructions (DEC and HEX) specify the radix of the operand and tell the assembler how to convert the listed number to a binary number.

1.1.11 Basic I/O Operations

Q20. Write about basic I/O operations performed in basic computer.

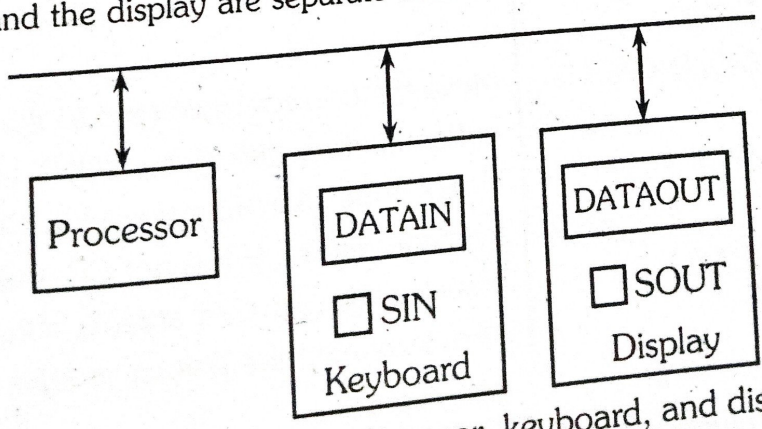
(Imp.)

Ans :

Input / Output operations are essential which has a significant effect on performance of a computer. An I/O device is connected to the interconnection network by using a circuit, called the device interface, which provides the means for data transfer and for the exchange of status and control information needed to facilitate the data transfers and govern the operation of the device. The interface includes some registers that can be accessed by the processor.

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. A solution to this problem is as follows: On output, the processor sends the first character and then waits for a signal from the display that the character has been received. It then sends the second character, and so on. Input is sent from the keyboard in a similar way; the processor waits for a signal indicating that a character key has been struck and that its code is available in some buffer register associated with the keyboard. Then the processor proceeds to read that code.

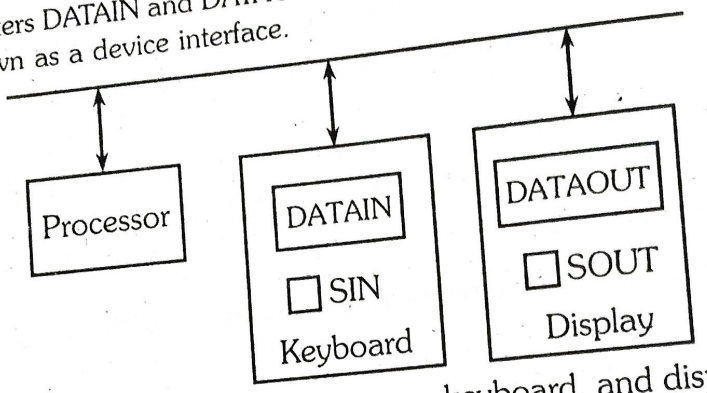
The keyboard and the display are separate devices as shown below.



Bus connection for processor, keyboard, and display

Consider the problem of moving a character-code from the keyboard to the processor (Figure). For this transfer, buffer-register DATAIN & a status control flags(SIN) are used.

- When a key is pressed, the corresponding ASCII code is stored in a DATAIN register associated with the keyboard. SIN=1 -> When a character is typed in the keyboard. This informs the processor that a valid character is in DATAIN. SIN=0 -> When the character is transferred to the processor
- An analogous process takes place when characters are transferred from the processor to the display. For this transfer, buffer-register DATAOUT & a status control flag SOUT are used. SOUT=1 -> When the display is ready to receive a character. SOUT=0 -> When the character is being transferred to DATAOUT.
- The buffer registers DATAIN and DATAOUT and the status flags SIN and SOUT are part of circuitry commonly known as a device interface.



Bus connection for processor, keyboard, and display

Memory - Mapped I/O

Program to read a line of characters and display it

	Move	#LOC, R0	Initialize pointer register R0 to point to the address of the first location in memory where the characters are to be stored
READ	TestBit	#3,INSTATUS	Wait for a character to be entered
	Branch=0	READ	in the keyboard buffer DATAIn.
	MoveByte	DATAIN,(R0)	Transfer the character from DATAIN into the memory (ths clears SIN to 0)
ECHO	TestBit	#3,OUTSTATUS	Wait for the display to become ready.
	Branch=0	ECHO	
	MoveByte	(R0),DATAOUT	Move the character just read to the display buffere register (this clears SOUT to 0).
	Compare	#CR,(R0)+	Check if the charcter just read in CR (carriage return), If it is not CR, then
	Branch ≠ 0	READ	branch back and read another character. Also, increment the pointer to store the next character.

Fig.: A program that reads a line of characters and displays it

Some address values are used to refer to peripheral device buffer-registers such as DATAIN & DATAOUT.

- No special instructions are needed to access the contents of the registers; data can be transferred between these registers and the processor using instructions such as Move, Load or Store.
- For example, contents of the keyboard character buffer DATAIN can be transferred to register R1in the processor by the instruction MoveByte DATAIN,R1
- The MoveByte operation code signifies that the operand size is a byte.
- The Testbit instruction tests the state of one bit in the destination, where the bit position to be tested is indicated by the first operand.

Short Question and Answers

1. Define Computer

Ans :

Computer

Computer is a fast electronic calculating machine that accepts digitized input information, processing it according to a list of internally stored instructions and produces the resulting output information. The list of instructions is called as a Computer program and the internal storage is called as Computer memory.

2. Low-Level Languages

Ans :

A language that corresponds directly to a specific machine. Low-level computer languages are either machine codes or are very close them. A computer cannot understand instructions given to it in high-level languages or in English. It can only understand and execute instructions given in the form of machine language i.e. binary. There are two types of low-level languages:

Machine Language

A language that is directly interpreted into the hardware. Machine language is the lowest and most elementary level of programming language and was the first type of programming language to be developed. Machine language is basically the only language that a computer can understand and it is usually written in hex. It is represented inside the computer by a string of binary digits (bits) 0 and 1. The symbol 0 stands for the absence of an electric pulse and the 1 stands for the presence of an electric pulse. Since a computer is capable of recognizing electric signals, it understands machine language.

Advantages

- Machine language makes fast and efficient use of the computer.
- It requires no translator to translate the code. It is directly understood by the computer.

Disadvantages

- All operation codes have to be remembered
- All memory addresses have to be remembered.
- It is hard to amend or find errors in a program written in the machine language.

3. What is digital computer?

Ans :

It is an electronic computer in which the input is discreterather than continuous, consisting of combinations of numbers, letters, and other characters written in an appropriate programming language and represented internally in binary notation, i.e., using only the two digits 0 and 1. By counting, comparing, and manipulating these digits or their combinations according to a set of instructions held in its memory, a digital computer can perform such tasks as to control industrial processes and regulate the operations of machines; analyze and organize vast amounts of business data; and simulate the behaviour of dynamic systems.

4. Types of busses

Ans :

- **System Bus:** A System Bus is usually a combination of address bus, data bus, and control bus respectively.
- **Internal Bus:** The bus that operates only with the internal circuitary of the CPU.
- **External Bus:** Buses which connects computer to external devices is nothing but external bus.
- **Back Plane:** A Back Plane bus includes a rowpf connectors into which system modules can be plugged in.
- **I/O Bus:** The bus used by I/O devices to communicate with the CPU is usually reffered as I/O bus.

- **Synchronou** units takes pl
- **Asynchrono** handshaking

5. What is sof

Ans :

A total com

1. Hardware c
2. Software re for a comp
3. The compu perform va

Types

1. Appli
2. Syste

6. What is

Ans :

The most programs. The and its machine only the period

At the st as shown belo and a copy is memory, the d is needed a se

7. Memor

Ans :

- Memor
- Each c
- Each g
- The wo
- A unit
- Access addres addres
- If 2^k thecor MB).

- **Synchronous Bus:** While using Synchro-nous bus, data transmission between source and destination units takes place in a **given timeslot** which is already known to these units.
- **Asynchronous Bus:** In this case the data transmission is governed by a special concept. That is handshaking control signals.

5. What is software?

Ans :

A total computer system includes both software and Hardware .

1. Hardware consists of physical components and all associated equipment.
2. Software refers to the collection programs that are written for the computer and writing a program for a computer consists of specifying, directly or indirectly a sequence of machine instructions.
3. The computer software consists of the instructions and data that the computer manipulates to perform various data processing tasks.

Types

1. Application software,
2. System software.

6. What is computer performance?

Ans :

The most important measure of the performance of a computer is how quickly it can compute 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 a processor, we should consider only the periods during which the processor is active.

At the start of execution, all program instructions and the required data are stored in the memory as shown below. 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 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.

7. Memory locations and addresses.

Ans :

- Memory consists of many millions of storage cells (flip-flops).
- Each cell can store a bit of information i.e. 0 or 1 (Figure).
- Each group of n bits is referred to as a word of information, and n is called the word length.
- The word length can vary from 8 to 64 bits.
- A unit of 8 bits is called a byte.
- Accessing the memory to store or retrieve a single item of information (word/byte) requires distinct addresses for each item location. (It is customary to use numbers from 0 through 2^k-1 as the addresses of successive-locations in the memory).
- If 2^k = no. of addressable locations; then 2^k addresses constitute the address-space of thecomputer. For example, a 24-bit address generates an address-space of 2^24 locations (16 MB).

8. List the type of operations that can perform on the computer.

Ans :

Computer programming consists of a sequence of small steps, such as adding two numbers, testing for particular condition ,reading the character from keyboard and sending a character to be displayed on screen.

A computer must have instructions capable of performing four types of operations:

1. Data transfers between the memory and the processor registers
2. Arithmetic and logic operations on data
3. Program sequencing and contr

Q9. What is Addressing Mode?

Ans :

The operation field of an instruction specifies the operation to be performed.And this operation must be performed on some data.So each instruction need to specify data on which the operation is to be performed.But the operand(data) may be in accumulator, general purpose register or at some specified memory location.So, appropriate location (address) of data is need to be specified, and in computer, there are various ways of specifying the address of data.These various ways of specifying the address of data are known as "Addressing Modes"

So Addressing Modes can be defined as-

"The technique for specifying the address of the operands "

And in computer the address of operand i.e., the address where operand is actually found is known as"Effective Address".

10. Assembly language.

Ans :

- The symbolic program (contains letters, numerals, or special characters) is referred to as an *assembly language program.*
- The basic unit of an assembly language program is a line of code.
- The specific language is defined by a set of rules that specify the symbols that can be used and they may be combined to form a line of code.

Choose the Correct Answers

1. Which of the following is not considered as a peripheral device? [a]
   (a) CPU (b) Keyboard
   (c) Monitor (d) All of the above

2. Which of the following computer memory is fastest? [a]
   (a) Register (b) Hard disk
   (c) RAM (d) None of the above

3. Which of these is NOT involved in the case of a memory write operation? [d]
   (a) Databus (b) MDR
   (c) MAR (d) PC

4. Which of these is required when we want to establish the communication links between a CPU and its peripherals? [a]
   (a) Memory data register (b) Memory address register
   (c) Instruction register (d) Index register

5. Which of these addressing modes is the most suitable for some high-level language statements? [b]
   (a) Indexed (b) Auto-decrement
   (c) Auto-increment (d) Displacement

6. _____ is the branch logic that provides decision-making capabilities in the control unit: [a]
   (a) Unconditional transfer (b) Controlled transfer
   (c) Conditional transfer (d) None of these

7. Which of the following bus structure is used to connect I/O devices? [c]
   (a) Star bus (b) Rambus
   (c) Single bus (d) Multiple bus

8. In the CPU, what is the functionality of the control unit? [a]
   (a) To decode program instruction (b) To perform logic operations
   (c) To store program instruction (d) To transfer data to primary storage

9. Computer address bus is - [c]
   (a) Multidirectional (b) Bidirectional
   (c) Unidirectional (d) None of the above

10. Which of the following circuit is used to store one bit of data? [a]
    (a) Flip Flop (b) Decoder
    (c) Encoder (d) Register

Fill in the Blanks

1. _____ register can interact with the secondary storage?
2. The Program Counter is also called as _____.
3. The two phases of executing an instruction are _____.
4. The condition flag Z is set to 1 to indicate _____.
5. During the execution of a program _____ register gets initialized first?
6. The instructions like MOV or ADD are called as _____.
7. _____ converts the programs written in assembly language into machine instructions.
8. The ALU makes use of _____ to store the intermediate results.
9. The control unit controls other units by generating _____.
10. A processor performing fetch or decoding of different instruction during the execution of another instruction is called _____.

ANSWERS

1. MAR
2. Instruction Pointer
3. Instruction fetch and instruction execution
4. The result is zero
5. PC
6. Opcode
7. Assembler
8. Accumulators
9. Timing signals
10. Pipe-lining