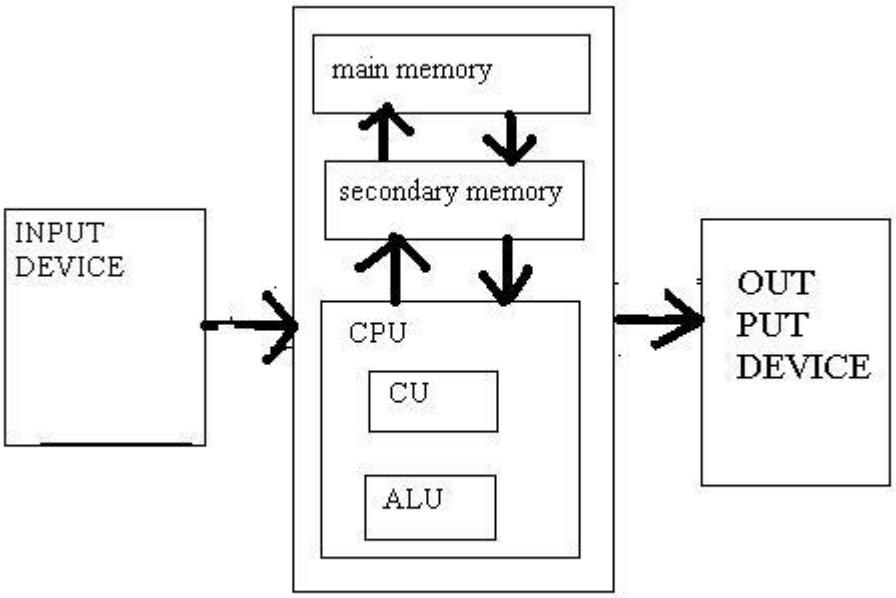
UNIT III

COMPUTER FUNDAMENTALS

Functional Units of a Digital Computer: Von Neumann Architecture – Operation and Operands of Computer Hardware Instruction – Instruction Set Architecture (ISA): Memory Location, Address and Operation – Instruction and Instruction Sequencing – Addressing Modes, Encoding of Machine Instruction – Interaction between Assembly and High Level Language.

3.1 Functional Units of Digital Computer

- A computer organization describes the functions and design of the various units of a digital system.
- A general-purpose computer system is the best-known example of a digital system. Other examples include telephone switching exchanges, digital voltmeters, digital counters, electronic calculators and digital displays.
- Computer architecture deals with the specification of the instruction set and the hardware units that implement the instructions.
- Computer hardware consists of electronic circuits, displays, magnetic and optic storage media and also the communication facilities.
- Functional units are a part of a CPU that performs the operations and calculations called for by the computer program.
- Functional units of a computer system are parts of the CPU (Central Processing Unit) that performs the operations and calculations called for by the computer program. A computer consists of five main components namely, Input unit, Central Processing Unit, Memory unit Arithmetic & logical unit, Control unit and an Output unit.



BLOCK DIAGRAM OF A DIGITAL COMPUTER

Input unit

- Input units are used by the computer to read the data. The most commonly used input devices are keyboards, mouse, joysticks, trackballs, microphones, etc.
- However, the most well-known input device is a keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Central processing unit

- Central processing unit commonly known as CPU can be referred as an electronic circuitry within a computer that carries out the instructions given by a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

Memory unit

- The Memory unit can be referred to as the storage area in which programs are kept which are running, and that contains data needed by the running programs.
- The Memory unit can be categorized in two ways namely, primary memory and secondary memory.
- It enables a processor to access running execution applications and services that are temporarily stored in a specific memory location.

- Primary storage is the fastest memory that operates at electronic speeds. Primary memory contains a large number of semiconductor storage cells, capable of storing a bit of information. The word length of a computer is between 16-64 bits.
- It is also known as the volatile form of memory, means when the computer is shut down, anything contained in RAM is lost.
- Cache memory is also a kind of memory which is used to fetch the data very soon. They are highly coupled with the processor.
- The most common examples of primary memory are RAM and ROM.
- Secondary memory is used when a large amount of data and programs have to be stored for a long-term basis.
- It is also known as the Non-volatile memory form of memory, means the data is stored permanently irrespective of shut down.
- The most common examples of secondary memory are magnetic disks, magnetic tapes, and optical disks.

Arithmetic & logical unit

- Most of all the arithmetic and logical operations of a computer are executed in the ALU (Arithmetic and Logical Unit) of the processor. It performs arithmetic operations like addition, subtraction, multiplication, division and also the logical operations like AND, OR, NOT operations.

Control unit

- The control unit is a component of a computer's central processing unit that coordinates the operation of the processor. It tells the computer's memory, arithmetic/logic unit and input and output devices how to respond to a program's instructions.
- The control unit is also known as the nerve center of a computer system.
- Let's us consider an example of addition of two operands by the instruction given as Add LOCA, RO. This instruction adds the memory location LOCA to the operand in the register RO and places the sum in the register RO. This instruction internally performs several steps.

Output Unit

- The primary function of the output unit is to send the processed results to the user. Output devices display information in a way that the user can understand.
- Output devices are pieces of equipment that are used to generate information or any other response processed by the computer. These devices display information that has been held or generated within a computer.

- The most common example of an output device is a monitor.

3.2 Von-Neumann Model

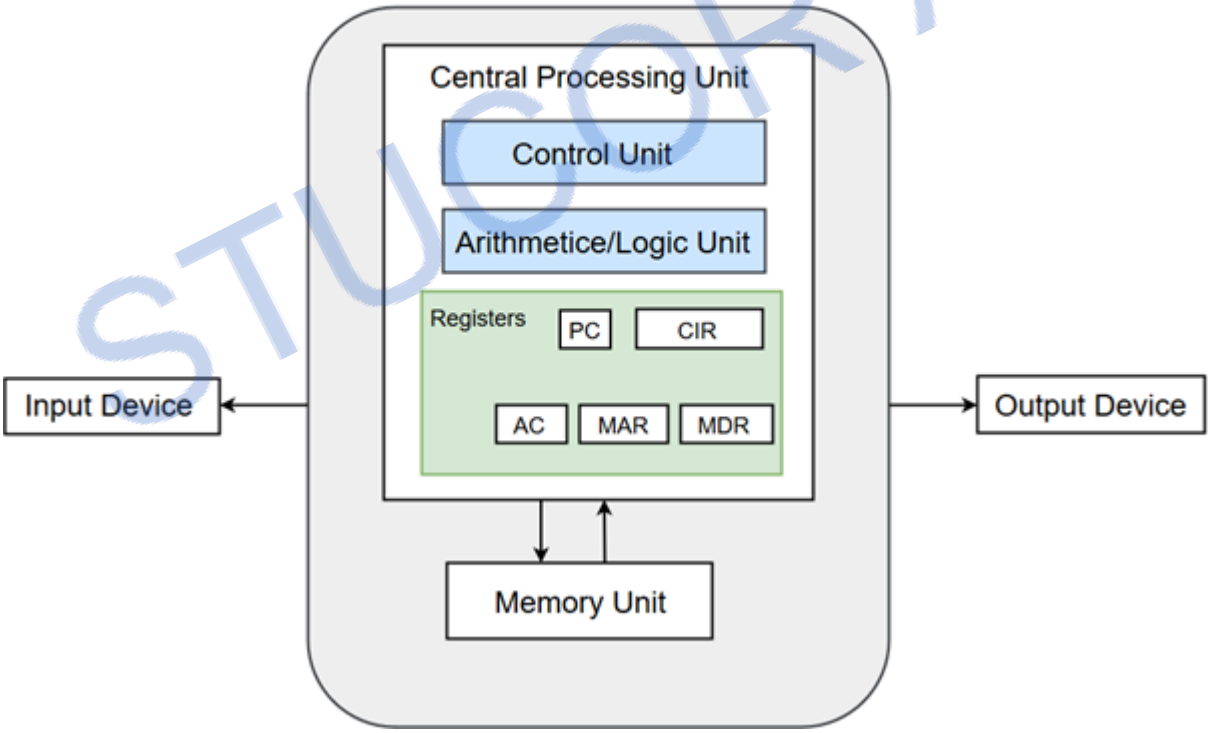
Von-Neumann proposed his computer architecture design in 1945 which was later known as Von-Neumann Architecture. It consisted of a Control Unit, Arithmetic, and Logical Memory Unit (ALU), Registers and Inputs/Outputs.

Von Neumann architecture is based on the stored-program computer concept, where instruction data and program data are stored in the same memory. This design is still used in most computers produced today.

A Von Neumann-based computer:

- Uses a single processor
- Uses one memory for both instructions and data.
- Executes programs following the fetch-decode-execute cycle

Von-Neumann Basic Structure:



Components of Von-Neumann Model:

- Central Processing Unit
- Buses
- Memory Unit

Central Processing Unit

The part of the Computer that performs the bulk of data processing operations is called the Central Processing Unit and is referred to as the CPU.

The Central Processing Unit can also be defined as an electric circuit responsible for executing the instructions of a computer program.

The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer.

The major components of CPU are Arithmetic and Logic Unit (ALU), Control Unit (CU) and a variety of registers.

Arithmetic and Logic Unit (ALU)

The Arithmetic and Logic Unit (ALU) performs the required micro-operations for executing the instructions. In simple words, ALU allows arithmetic (add, subtract, etc.) and logic (AND, OR, NOT, etc.) operations to be carried out.

Control Unit

The Control Unit of a computer system controls the operations of components like ALU, memory and input/output devices.

The Control Unit consists of a program counter that contains the address of the instructions to be fetched and an instruction register into which instructions are fetched from memory for execution.

Registers

Registers refer to high-speed storage areas in the CPU. The data processed by the CPU are fetched from the registers.

Following is the list of registers that plays a crucial role in data processing.

Registers	Description
MAR (Memory Address Register)	This register holds the memory location of the data that needs to be accessed.
MDR (Memory Data Register)	This register holds the data that is being transferred to or from memory.
AC (Accumulator)	This register holds the intermediate arithmetic and logic results.
PC (Program Counter)	This register contains the address of the next instruction to be executed.
CIR (Current Instruction Register)	This register contains the current instruction during processing.

Buses

Buses are the means by which information is shared between the registers in a multiple-register configuration system.

A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during each particular register transfer.

Von-Neumann Architecture comprised of three major bus systems for data transfer.

Bus	Description
Address Bus	Address Bus carries the address of data (but not the data) between the processor and the memory.
Data Bus	Data Bus carries data between the processor, the memory unit and the input/output devices.
Control Bus	Control Bus carries signals/commands from the CPU.

Memory Unit

A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of the storage. The memory stores binary information in groups of bits called words. The internal structure of a memory unit is specified by the number of words it contains and the number of bits in each word.

Two major types of memories are used in computer systems:

1. RAM (Random Access Memory)
2. ROM (Read-Only Memory)

3.3 Operation and Operands of Computer Hardware Instruction

Computer instruction is a binary code that determines the micro-operations in a sequence for a computer. They are saved in the memory along with the information. Each computer has its specific group of instructions. They can be categorized into two elements as ***Operation codes*** (Opcodes) and ***Address***. Opcodes specify the operation for specific instructions, and an address determines the registers or the areas used for that operation.

Operands are definite elements of computer instruction that show what information is to be operated on. The most important general categories of data are

1. Addresses
2. Numbers
3. Characters
4. Logical data

In many cases, some calculation must be performed on the operand reference to determine the main or virtual memory address.

In this context, addresses can be considered to be unsigned integers. Other common data types are numbers, characters, and logical data, and each of these is briefly described below. Some machines define specialized data types or data structures. For example, machine operations may operate directly on a list or a string of characters.

Addresses

Addresses are nothing but a form of data. Here some calculations must be performed on the operand reference in an instruction, which is to determine the physical address of an instruction.

Numbers

All machine languages include numeric data types. Even in non-numeric data processing, numbers are needed to act as counters, field widths, etc. An important difference between numbers used in ordinary mathematics and numbers stored in a computer is that the latter is limited. Thus, the programmer is faced with understanding the consequences of rounding, overflow and underflow.

Here are the three types of numerical data in computers, such as:

1. Integer or fixed point: Fixed point representation is used to store integers, the positive and negative whole numbers (… -3, -2, -1, 0, 1, 2, 3, …). However, the programmer assigns a radix point location to each number and tracks the radix point through every operation. High-level programs, such as C and BASIC usually allocate 16 bits to store each integer. Each fixed point binary number has three important parameters that describe it:

- Whether the number is signed or unsigned,
- The position of the radix point to the right side of the sign bit (for signed numbers), or the position of the radix point to the most significant bit (for unsigned numbers).
- And the number of fractional bits stored.

2. Floating point: A Floating Point number usually has a decimal point, which means **0, 3.14, 6.5,** and**-125.5** are Floating Point

The term *floating point* is derived from the fact that there is no fixed number of digits before and after the decimal point, which means the decimal point can float. There are also representations in which the number of digits before and after the decimal point is set, called ***fixed-point*** representations. In general, floating-point representations are slower and less accurate than fixed-point representations, but they can handle a larger range of numbers.

3. Decimal number: The decimals are an extension of our number system. We also know that decimals can be considered fractions with 10, 100, 1000, etc. The numbers expressed in the decimal form are called decimal numbersor decimals. For example:1, 4.09, 13.83, etc. A decimal number has two parts, and a dot separates these parts (.) called the ***decimal point***.

- **Whole number part:** The digits lying to the left of the decimal point form the whole number part. The places begin with ones, tens, hundreds, thousands and so on.
- **Decimal part:** The decimal point and the digits laying on the right of the decimal point form the decimal part. The places begin with tenths, hundredths, thousandths and so on.

Characters

A common form of data is text or character strings. While textual data are most convenient for humans. But computers work in binary. So, all characters whether letters, punctuation or digits, are

stored as binary numbers. All of the characters that a computer can use are called ***character set***s. Here are the two common standards, such as:

1. American Standard Code for Information Interchange (ASCII)
2. Unicode

ASCII uses seven bits, giving a character set of 128 characters. The characters are represented in a table called the ASCII table. The 128 characters include:

- 32 control codes (mainly to do with printing)
- 32 punctuation codes, symbols, and space
- 26 upper-case letters
- 26 lower-case letters
- numeric digits 0-9

We can say that the letter 'A' is the first letter of the alphabet; 'B' is the second, and so on, all the way up to 'Z', which is the 26th letter. In ASCII, each character has its own assigned number. Denary, binary and hexadecimal representations of ASCII characters are shown in the below table.

Character	Denary	Binary	Hexadecimal
A	65	1000001	41
Z	90	1011010	5A
a	97	1100001	61
z	122	1111010	7A
0	48	0110000	30
9	57	0111001	39
Space	32	0100000	20
!	33	0100001	21

A is represented by the denary number 65 (binary 1000001, hexadecimal 41), B by 66 (binary 1000010, hexadecimal 42) and so on up to Z, which is represented by the denary number 90 (binary 1011010, hexadecimal 5A).

Similarly, lower-case letters start at denary 97 (binary 1100001, hexadecimal 61) and end at denary 122 (binary 1111010, hexadecimal 7A). When data is stored or transmitted, its ASCII or Unicode number is used, not the character itself.

For example, the word "Computer" would be represented as:

1000011 1101111 1101101 1110000 1110101 1110100 1100101 1110010

On the other hand, ***IRA*** is also widely used outside the United States. A unique 7-bit pattern represents each character in this code. Thus, 128 different characters can be represented, and more than necessary to represent printable characters, and some of the patterns represent control

characters. Some control characters control the printing of characters on a page, and others are concerned with communications procedures.

IRA-encoded characters are always stored and transmitted using 8 bits per character. The 8 bit may be set to 0 or used as a parity bit for error detection. In the latter case, the bit is set such that the total number of binary 1s in each octet is always odd (odd parity) or always even (even parity).

Logical data

Normally, each word or other addressable unit (byte, half-word, and so on) is treated as a single unit of data. Sometimes, it is useful to consider an n-bit unit consisting of 1-bit items of data, each item having the value 0 or 1. When data are viewed this way, they are considered to be logical data.

The ***Boolean*** data can only represent two values: true or false. Although only two values are possible, they are rarely implemented as a single binary digit for efficiency reasons. Many programming languages do not have an explicit Boolean type, instead of interpreting 0 as false and other values as true. Boolean data refers to the logical structure of how the language is interpreted to the machine language. In this case, a Boolean 0 refers to the logic False, and true is always a non zero, especially one known as Boolean 1.

There are two advantages to the bit-oriented view:

- We may want to store an array of Boolean or binary data items, in which each item can take on only the values 0 and 1. With logical data, memory can be used most efficiently for this storage.
- There are occasions when we want to manipulate the bits of a data item.

3.4 Instruction set Architecture(ISA):

An Instruction Set Architecture (ISA) is **part of the abstract model of a computer that defines how the CPU is controlled by the software**. The ISA acts as an interface between the hardware and the software, specifying both what the processor is capable of doing as well as how it gets done

Two types of instruction set architectures are

The two main categories of instruction set architectures, **CISC (such as Intel's x86 series) and RISC (such as ARM and MIPS)**,

The ISA of a processor can be described using 5 catagories:

Operand Storage in the CPU
Number of explicit named operands
Operand location
Operations
Type and size of operands

The 3 most common types of ISAs are:

1. ***Stack*** - The operands are implicitly on top of the stack.
2. ***Accumulator*** - One operand is implicitly the accumulator.
3. *General Purpose Register (GPR)* - All operands are explicitely mentioned, they are either registers or memory locations

Stack	Accumulator	GPR
PUSH A	LOAD A	LOAD R1,A
PUSH B	ADD B	ADD R1,B
ADD	STORE C	STORE R1,C
POP C	-	-

The i8086 has many instructions that use implicit operands although it has a general register set. The i8051 is another example, it has 4 banks of GPRs but most instructions must have the A register as one of its operands.

Stack

Advantages: Simple Model of expression evaluation (reverse polish). Short instructions.
Disadvantages: A stack can't be randomly accessed This makes it hard to generate eficient code. The stack itself is accessed every operation and becomes a bottleneck.

Accumulator

Advantages: Short instructions.
Disadvantages: The accumulator is only temporary storage so memory traffic is the highest for this approach.

GPR

Advantages: Makes code generation easy. Data can be stored for long periods in registers.
Disadvantages: All operands must be named leading to longer instructions.

Reduced Instruction Set Computer (RISC):

RISC stands for Reduced Instruction Set Computer. The ISA is composed of instructions that all have exactly the same size, usualy 32 bits. Thus they can be pre-fetched and pipelined succesfuly. All ALU instructions have 3 operands which are only registers. The only memory access is through explicit LOAD/STORE instructions.
Thus C = A + B will be assembled as:

```
LOAD  R1,A
LOAD  R2,B
ADD   R3,R1,R2
STORE C,R3
```

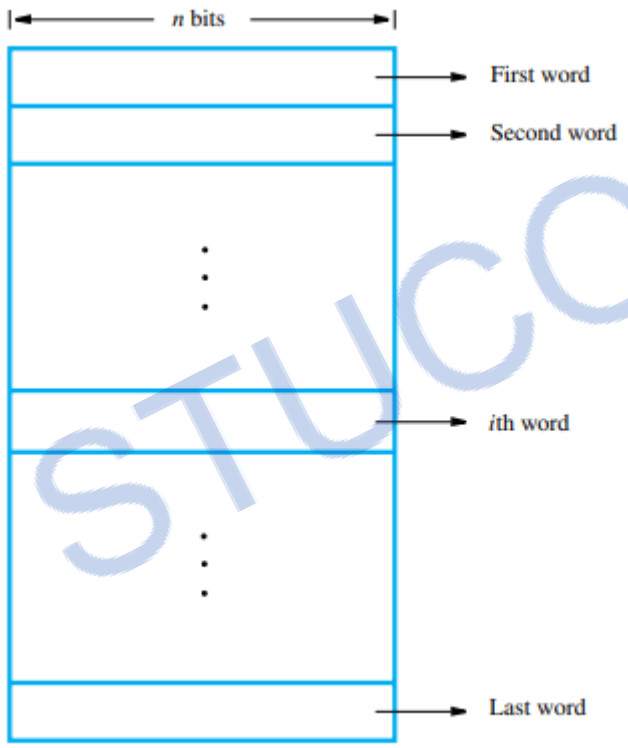
Although it takes 4 instructions we can reuse the values in the registers.

Complex Instruction Set Architecture (CISC) :
The main idea is that a single instruction will do all loading, evaluating, and storing operations just like a multiplication command will do stuff like loading data, evaluating, and storing it, hence it's complex

Memory Locations and Addresses

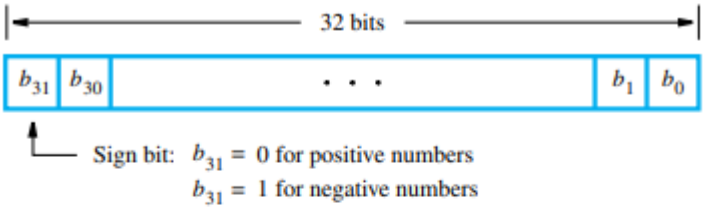
The memory consists of many millions of storage cells, each of which can store a bit of information having the value 0 or 1. Because a single bit represents a very small amount of information, bits are seldom handled individually.

The usual approach is to deal with them in groups of fixed size. For this purpose, the memory is organized so that a group of n bits can be stored or retrieved in a single, basic operation. Each group of n bits is referred to as a word of information, and n is called the word length. The memory of a computer can be schematically represented as a collection of words.

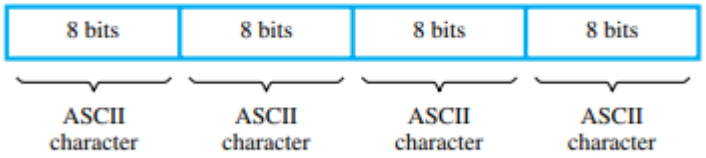


Memory words.

Modern computers have word lengths that typically range from 16 to 64 bits. If the word length of a computer is 32 bits, a single word can store a 32-bit signed number or four ASCII-encoded characters, each occupying 8 bits, as shown in Figure



(a) A signed integer



(b) Four characters

Examples of encoded information in a 32-bit word.

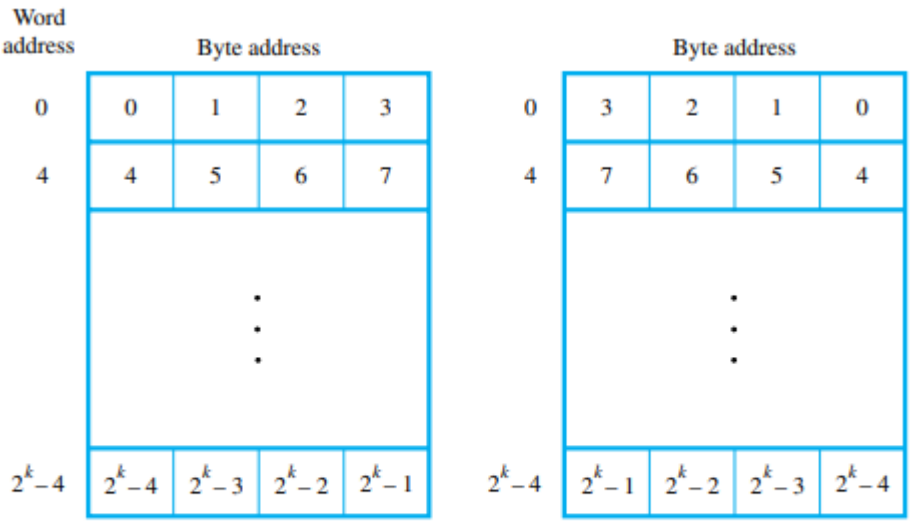
A unit of 8 bits is called a byte. Machine instructions may require one or more words for their representation.

After we have described instructions at the assembly-language level. Accessing the memory to store or retrieve a single item of information, either a word or a byte, requires distinct names or addresses for each location. It is customary to use numbers from 0 to 2k − 1, for some suitable value of k, as the addresses of successive locations in the memory. Thus, the memory can have up to 2k addressable locations. The 2k addresses constitute the address space of the computer. For example, a 24-bit address generates an address space of 224 (16,777,216) locations. This number is usually written as 16M (16 mega), where 1M is the number 220 (1,048,576). A 32-bit address creates an address space of 232 or 4G (4 giga) locations, where 1G is 230. Other notational conventions that are commonly used are K (kilo) for the number 210 (1,024), and T (tera) for the number 240

Byte Addressability :

A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits. It is impractical to assign distinct addresses to individual bit locations in the memory. The most practical assignment is to have successive addresses refer to successive byte locations in the memory. This is the assignment used in most modern computers. The term byte-addressable memory is used for this assignment. Byte locations have addresses 0, 1, 2,.... Thus, if the word length of the machine is 32 bits, successive words are located at addresses 0, 4, 8,..., with each word consisting of four bytes.

There are two ways that byte addresses can be assigned across words **big-endian** and **Little endian**



(a) Big-endian assignment (b) Little-endian assignment

Byte and word addressing.

The name big-endian is used when lower byte addresses are used for the more significant bytes (the leftmost bytes) of the word.

The name little-endian is used for the opposite ordering, where the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word. The words "more significant" and "less significant" are used in relation to the weights (powers of 2) assigned to bits when the word represents a number. Both little-endian and big-endian assignments are used in commercial machines. In both cases, byte addresses 0, 4, 8,..., are taken as the addresses of successive words in the memory of a computer with a 32-bit word length. These are the addresses used when accessing the memory to store or retrieve a word.

Memory Operations

Both program instructions and data operands are stored in the memory. To execute an instruction, the processor control circuits must cause the word (or words) containing the instruction to be transferred from the memory to the processor. Operands and results must also be moved between the memory and the processor.

Thus, two basic operations involving the memory are needed, , namely Read and Write.

Read Operation:

The Read operation transfers a copy of the contents of a specific memory location to the processor. The memory contents remain unchanged. To start a Read operation, the processor sends the address of the desired location to the memory and requests that its contents be read. The memory reads the data stored at that address and sends them to the processor.

Write Operation:

The Write operation transfers an item of information from the processor to a specific memory location, overwriting the former contents of that location. To initiate a Write operation, the processor sends the address of the desired location to the memory, together with the data to be written into that location. The memory then uses the address and data to perform the write.

3.5 Instructions and Instruction Sequencing

The tasks carried out by a computer program consist of a sequence of small steps, such as adding two numbers, testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen.

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

We begin by discussing instructions for the first two types of operations. To facilitate the discussion, we first need some notation

Register Transfer Notation

We need to describe the transfer of information from one location in a computer to another. Possible locations that may be involved in such transfers are memory locations, processor registers, or registers in the I/O subsystem.

Example 1:

R2 ← [LOC]

This expressionmeans that the contents of memory location LOC are transferred into processor register R2

Example 2:

As another example, consider the operation that adds the contents of registers R2 and R3, and places their sum into register R4. This action is indicated as

R4 ← [R2]+[R3]

This type of notation is known as Register Transfer Notation (RTN). Note that the righthand side of an RTN expression always denotes a value, and the left-hand side is the name of a location where the value is to be placed, overwriting the old contents of that location

Assembly-Language Notation:

to represent machine instructions and programs we use assembly –Language notation

Example 1:

Load R2, LOC

a generic instruction that causes the transfer described above, from memory location LOC to processor register R2, 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

Example 2:

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, registers R2 and R3 hold the source operands, while R4 is the destination

RISC and CISC Instruction Sets

One of the most important characteristics that distinguish different computers is the nature of their instructions. There are two fundamentally different approaches in the design of instruction sets for modern computers. One popular approach is based on the premise that higher performance can be achieved if each instruction occupies exactly one word in memory, and all operands needed to execute a given arithmetic or logic operation specified by an instruction are already in processor registers. This approach is conducive to an implementation of the processing unit in which the various operations needed to process a sequence of instructions are performed in "pipelined" fashion to overlap activity and reduce total execution time of a program. The restriction that each instruction must fit into a single word reduces the complexity and the number of different types of instructions that may be included in the instruction set of a computer. Such computers are called Reduced Instruction Set Computers (RISC).

An alternative to the RISC approach is to make use of more complex instructions which may span more than one word of memory, and which may specify more complicated operations. This approach was prevalent prior to the introduction of the RISC approach in the 1970s. Although the use of complex instructions was not originally identified by any particular label, computers based on this idea have been subsequently called Complex Instruction Set Computers (CISC).

Introduction to RISC Instruction Sets:

Two key characteristics of RISC instruction sets are: • Each instruction fits in a single word. • A load/store architecture is used, in which – Memory operands are accessed only using Load and Store instructions. – All operands involved in an arithmetic or logic operation must either be in processor registers, or one of the operands may be given explicitly within the instruction word.

At the start of execution of a program, all instructions and data used in the program are stored in the memory of a computer. Processor registers do not contain valid operands at that time . If operands are expected to be in processor registers before they can be used by an instruction, then it is necessary to first bring these operands into the registers. This task is done by Load instructions which copy the contents of a memory location into a processor register. Load instructions are of the form

Load destination, source

Or more specifically

Load processor_register, memory_location

Example:

The operation of adding two numbers is a fundamental capability in any computer.

The statement C = A + B

The required action can be accomplished by a sequence of simple machine instructions. We choose to use registers R2, R3, and R4 to perform the task with four instructions:

Load R2, A

Load R3, B

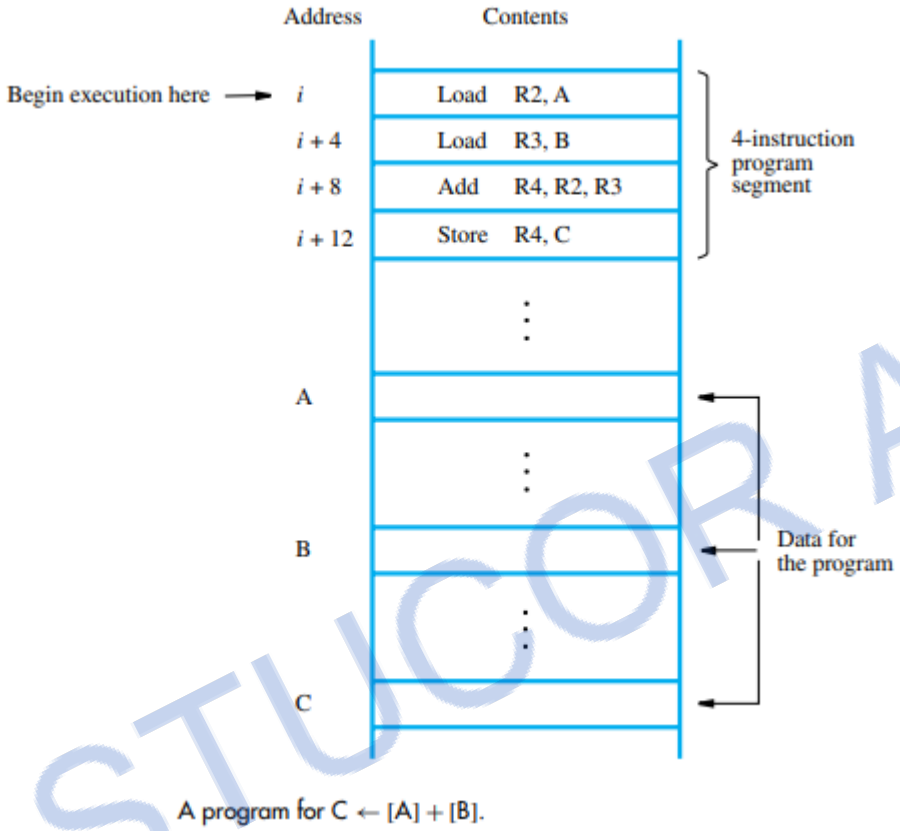
Add R4, R2, R3

Store R4, C

Instruction Execution and Straight-Line Sequencing:

we used the task C = A + B

implemented as C ← [A] + [B]



A program for C ← [A] + [B].

We assume that the word length is 32 bits and the memory is byte-addressable. 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. For simplicity, we assume that a desired memory address can be directly specified in Load and Store instructions, although this is not possible if a full 32-bit address is involved.

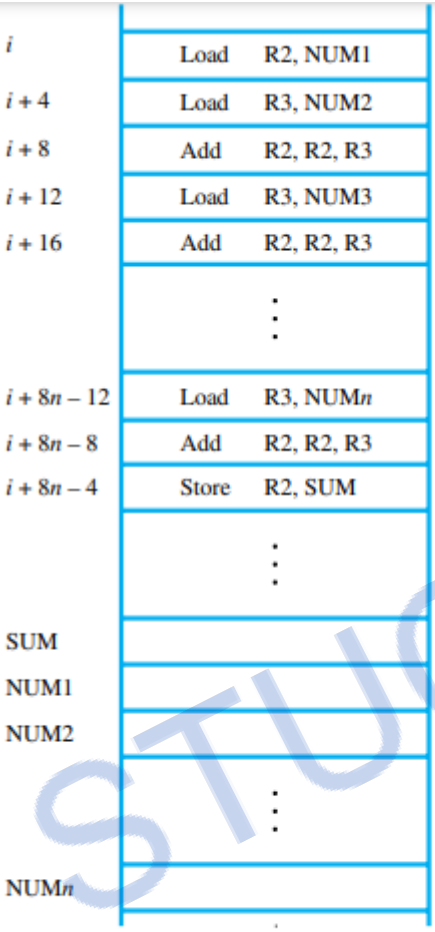
Let us consider how this program is executed. The processor contains a register called the program counter (PC), which holds the address of the next instruction to be executed. To begin executing a program, the address of its first instruction (i in our example) must be placed into the PC. Then, the 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:

Consider the task of adding a list of n number

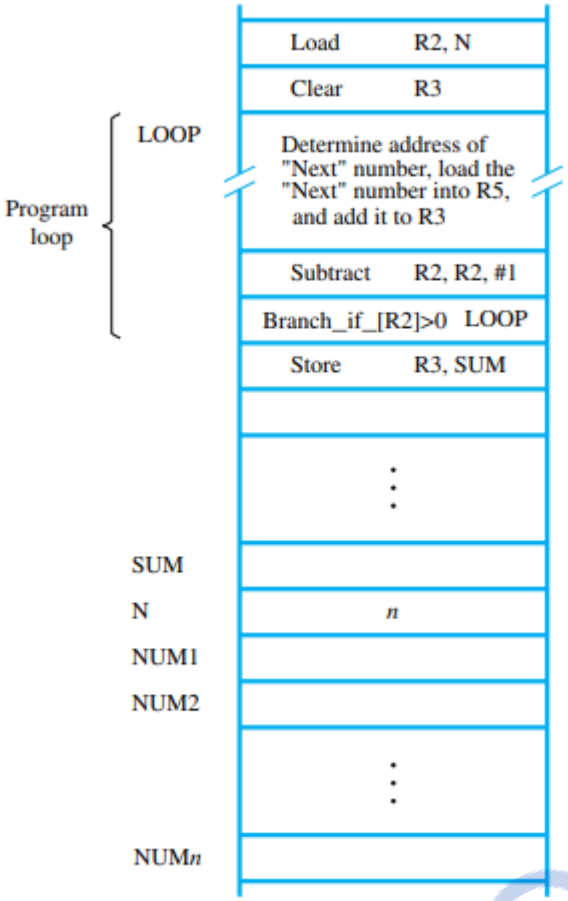
The addresses of the memory locations containing the n numbers are symbolically given as NUM1, NUM2,..., NUMn, and separate Load and Add instructions are used to add each number to the contents of register R2. After all the numbers have been added, the result is placed in memory location SUM.



A program for adding *n* numbers.

Instead of using a long list of Load and 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 body of the loop is a straight-line sequence of instructions executed repeatedly. It starts at location LOOP and ends at the instruction Branch_if_[R2]>0. During each pass through this loop, the address of the next list entry is determined, and that entry is loaded into R5 and added to R3. The address of an operand can be specified in various ways, as will be described in Section 2.4. For now, we concentrate on how to create and control a program loop. Assume that the number of entries in the list, n, is stored in memory location N, as shown. Register R2 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 R2 at the beginning of the program. Then, within the body of the loop, the instruction.



Using a loop to add *n* numbers.

Subtract R2, R2, #1

reduces the contents of R2 by 1 each time through the loop. Execution of the loop is repeated as long as the contents of R2 are greater than zero. We now introduce branch instructions. 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.

Branch_if_[R2]>0 LOOP

is a conditional branch instruction that causes a branch to location LOOP if the contents of register R2 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 R3. At the end of the nth 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 R3 into memory location SUM.

Encoding of Machine Instructions:

we have introduced a variety of useful instructions and addressing modes. We have used a generic form of assembly language to emphasize basic concepts without relying on processor-specific acronyms or mnemonics. Assembly-language instructions symbolically express the actions that must be performed by the processor circuitry.

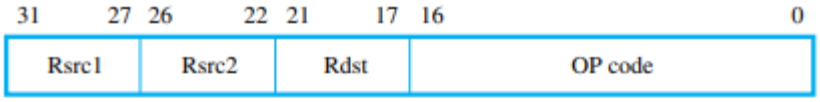
To be executed in a processor, assembly-language instructions must be converted by the assembler program, into machine instructions that are encoded in a compact binary pattern.

Let us now examine how machine instructions may be formed.

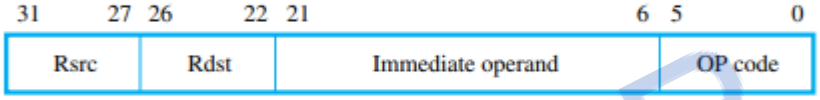
The Add instruction

Add Rdst, Rsrc1, Rsrc2

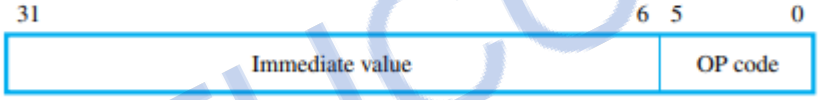
The above instruction representative of a class of three-operand instructions that use operands in processor registers. Registers Rdst, Rsrc1, and Rsrc2 hold the destination and two source operands. If a processor has 32 registers, then it is necessary to use five bits to specify each of the three registers in such instructions. If each instruction is implemented in a 32-bit word, the remaining 17 bits can be used to specify the OP code that indicates the operation to be performed



(a) Register-operand format



(b) Immediate-operand format



(c) Call format

Possible instruction formats.

Now consider instructions in which one operand is given using the Immediate addressing mode, such as Add Rdst, Rsrc, #Value

Of the 32 bits available, ten bits are needed to specify the two registers. The remaining 22 bits must give the OP code and the value of the immediate operand. The most useful sizes of immediate operands are 32, 16, and 8 bits. Since 32 bits are not available, a good choice is to allocate 16 bits for the immediate operand. This leaves six bits for specifying the OP code.

This format can also be used for Load and Store instructions, where the Index addressing mode uses the 16-bit field to specify the offset that is added to the contents of the index register.

The format in Figure b can also be used to encode the Branch instructions. The Branch-greater-than instruction at memory address 128.

BGT R2, R0, LOOP

if the contents of register R0 are zero. The registers R2 and R0 can be specified in the two register fields in Figure b. The six-bit OP code has to identify the BGT operation. The 16-bit immediate field can be used to provide the information needed to determine the branch target address, which is the location of the instruction with the label LOOP. The target address generally comprises 32 bits. Since there is no space for 32 bits, the BGT instruction makes use of the immediate field to give an offset from the location of this instruction in the

program to the required branch target. At the time the BGT instruction is being executed, the program counter, PC, has been incremented to point to the next instruction, which is the Store instruction at address 132. Therefore, the branch offset is 132 − 112 = 20. Since the processor computes the target address by adding the current contents of the PC and the branch offset, the required offset in this example is negative, namely −20. Finally, we should consider the Call instruction, which is used to call a subroutine. It only needs to specify the OP code and an immediate value that is used to determine the address of the first instruction in the subroutine. If six bits are used for the OP code, then the remaining 26 bits can be used to denote the immediate value. This gives the format shown in c.

Addressing Modes:

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:

First, the way the operand data are chosen during program execution is dependent on theaddressing mode of the instruction.

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).some notations 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.

Types Of Addressing Modes

Various types of addressing modes are:

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:

Implied Addressing Mode:

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

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 on the 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

- One of the operand is mentioned directly.
- Data is available as a part of instruction.
- Data is 8 0r 16 bit long.
- No memory reference is needed to fetch data

Immediate Mode :Eg.

Example 1 :

MOV CL, 03H

03 – 8 bit immediate source operand

CL – 8 bit register destination operand

Example 2:

ADD AX, 0525H

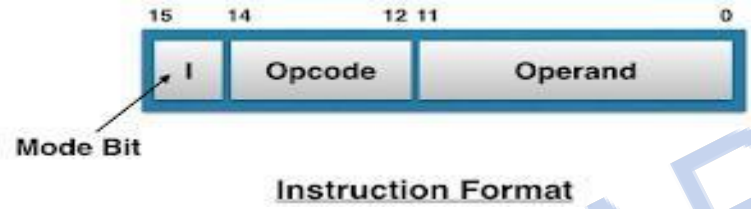
0525 – 16 bit immediate source operand

AX – 16 bit register destination operand.

2. Direct and Indirect Addressing Modes

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

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**. Now we will discuss about each in detail one by one.



Instruction Format

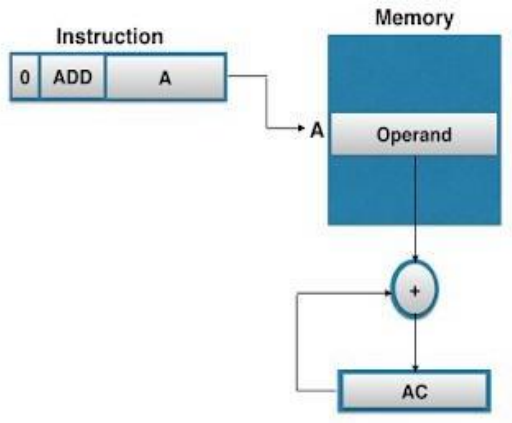
Direct Addressing Mode

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.

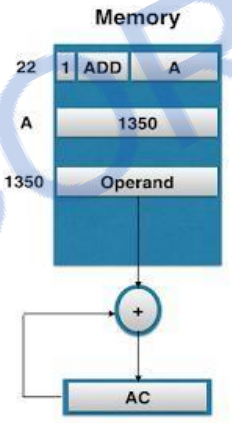
Indirect Addressing Mode:

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 in memory. 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

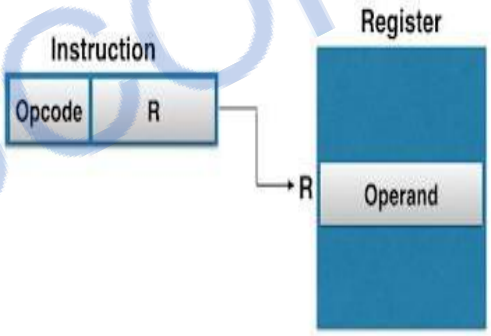
It look like as:

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.

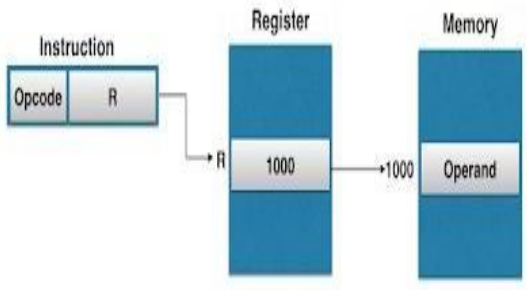
Example of such instructions are:

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)

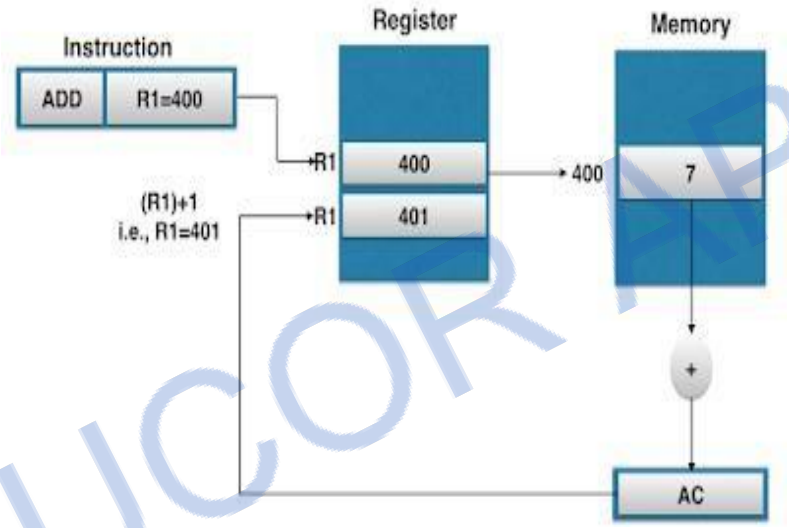
But, after accessing operand, register is incremented by 1.

As an example:

It look like as shown below:

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.



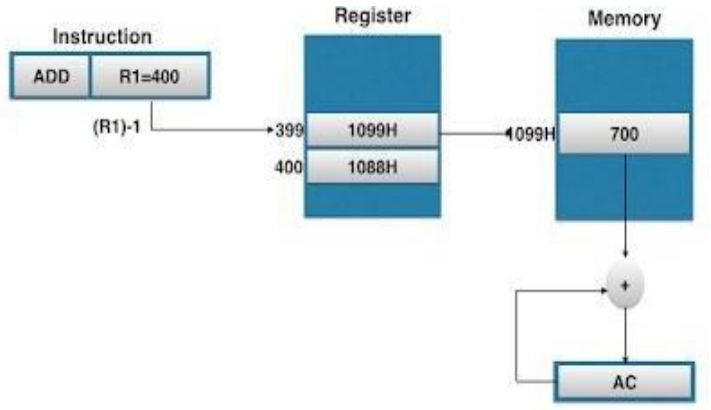
Auto-decrement Addressing Mode:

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

As an example:

It look like as shown below:



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)

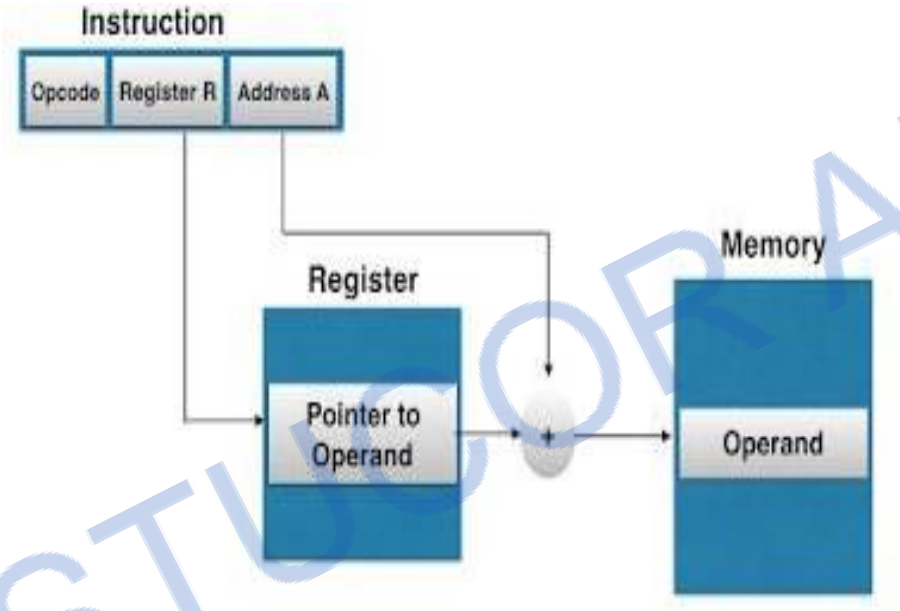
Means, Displacement Addressing Modes requires that the instruction have two address fields, at least one of which is explicit means, one is address field indicate direct address and other indicate indirect address.

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.

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. Base Register Addressing Mode
3. Indexing 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.

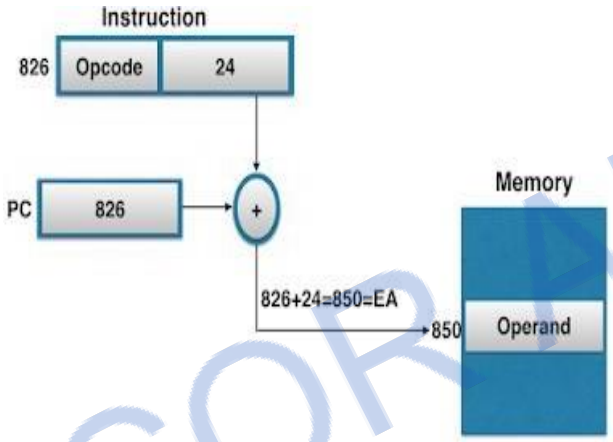
i.e., EA=A+PC

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 26+24=850

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

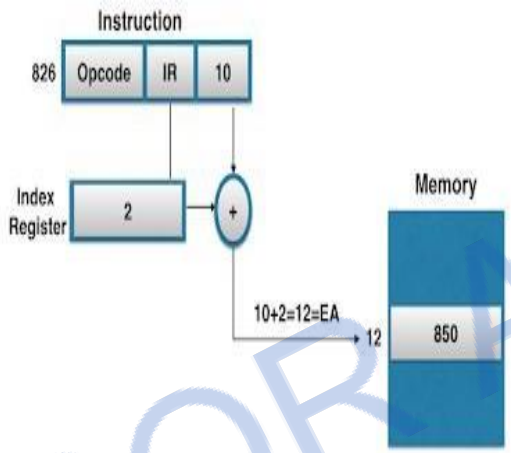
In indexed addressing mode, the content of Index Register is added to direct address part(or field) of instruction to obtain the effective address. Means, in it, the register indirect addressing field of instruction point to Index Register, which is a special CPU register that contain an Indexed value,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



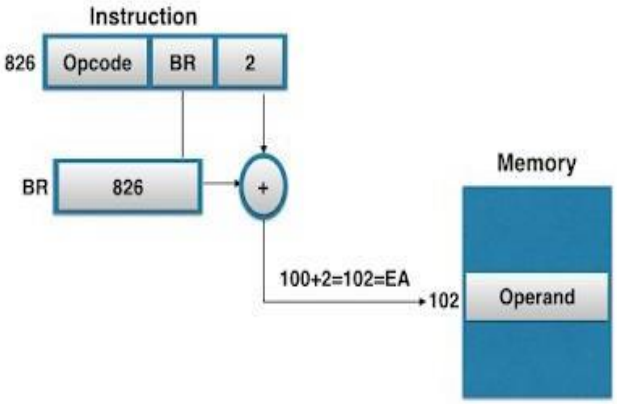
3. Base Register Addressing Mode:

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



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 thedirect 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.