



A T M E
College of Engineering



LECTURE NOTES

**COURSE: DIGITAL DESIGN AND COMPUTER
ORGANISATION**

COURSE CODE: 1BCS303

SEMESTER: III

COURSE COORDINATOR

Mrs. Razikha Amreen M I

(ACADEMIC YEAR 2026-27)

INSTITUTIONAL VISION AND MISSION

Vision

Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

DEPARTMENT VISION AND MISSION

Vision

To nurture competent and ethical cyber security professionals capable of contributing to secure and resilient digital ecosystems.

Mission

M1: To impart strong foundational knowledge in cyber security through effective teaching–learning practices and a continuously evolving curriculum that addresses emerging cyber threats and technologies.

M2: To promote research, innovation, and industry collaboration in emerging areas of cyber security to address real-world digital security challenges.

M3: To instill professional ethics, teamwork, leadership qualities, and a sense of social responsibility in addressing real-world cyber challenges.

M4: To foster a culture of lifelong learning among students by encouraging higher studies, professional development, and active engagement with the global cyber security community.

Program Educational Objectives

PEO1 –Excel as cyber security professionals or entrepreneurs by applying strong fundamentals of cyber security engineering.

PEO2 – Pursue higher studies, research, and professional certifications while engaging in lifelong learning to keep pace with evolving cyber technologies and threats.

PEO3 – Exhibit leadership, teamwork, professional ethics, and social responsibility while addressing cyber security challenges in multidisciplinary environments.

Program Specific Outcomes

PSO1 – Apply principles of computer science, programming, algorithms, operating systems, and computer networks to design and develop efficient and reliable computing solutions.

PSO2 – Identify, analyze, and mitigate cyber threats and vulnerabilities using techniques related to network security, ethical hacking, penetration testing, and digital forensics.

PSO3 – Apply cyber security policies, cyber laws, risk management practices, and incident response strategies to ensure the protection and resilience of digital systems.

Basic Structure of Computers, Instructions&Programs

Computer Organization explains the function and design of the various units of digital computers that store and process information. It also deals with the input units of the computer which receive information from external sources and the output units which send computed results to external destinations. The input, storage, processing, and output operations are governed by a list of instructions that constitute a program. It deals about computer hardware and computer architecture.

Computer hardware consists of electronic circuits, magnetic and optical storage devices, displays, electromechanical devices, and communication facilities.

Computer architecture encompasses the specification of an instruction set and the functional behavior of the hardware units that implement the instructions

Functional Units

A computer consists of five functionally independent main parts: input, memory, arithmetic and logic, output, and control units, as shown in Figure 1.1.

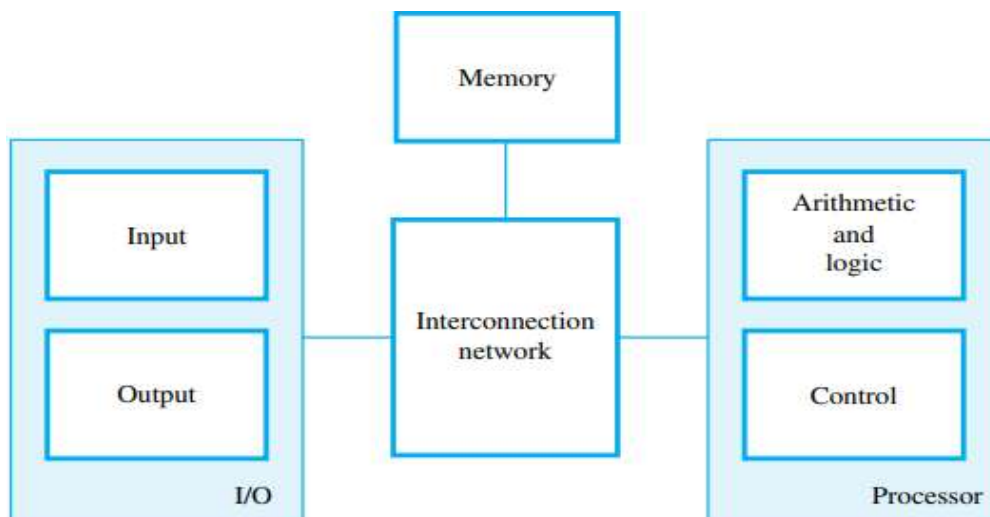


Figure 1.1 Basic functional units of a computer.

- the **input unit** accepts coded information from human operators using devices such as keyboards, or from other computers over digital communication lines.
- The information received is stored in the computer's memory, either for later use or to be processed immediately by the **arithmetic and logic unit**.
- *The processing steps are specified by a program that is also stored in the memory.*
- *Finally, the results are sent back to the outside world through the output unit.*
- All of these actions are coordinated by the **control unit**.
- An **interconnection network** provides the means for the functional units to exchange information and coordinate their actions.

Input Units

- Computers accept coded information through input units.
- The most common input device is the keyboard.
- Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted to the processor.
- Many other kinds of input devices for human-computer interaction are available, including the touchpad, mouse, joystick, and trackball.
- These are often used as graphic input devices in conjunction with displays.
- Microphones can be used to capture audio input which is then sampled and converted into digital codes for storage and processing.
- Similarly, cameras can be used to capture video input.
- Digital communication facilities, such as the Internet, can also provide input to a computer from other computers and database servers.

The function of the memory unit is to **store programs and data**.

There are two classes of storage, called **primary and secondary**.

Primary Memory :

Primary memory, also called **main memory**, is a fast memory that operates at electronic speeds.

Programs must be stored in this memory while they are being executed.

The memory consists of a large number of semiconductor storage cells, each capable of storing one bit of information. These cells are rarely read or written individually.

Instead, they are handled in groups of fixed size called **words**. The memory is organized so that one word can be stored or retrieved in one basic operation. The number of bits in each word is referred to as the word length of the computer, typically **16, 32, or 64 bits**.

- To provide easy access to any word in the memory, a distinct address is associated with each word location.
- Addresses are consecutive numbers, starting from 0, that identify successive locations.
- A particular word is accessed by specifying its address and issuing a control command to the memory that starts the storage or retrieval process.
- Instructions and data can be written into or read from the memory under the control of the processor.
- It is essential to be able to access any word location in the memory as quickly as possible.
- A memory in which any location can be accessed in a short and fixed amount of time after specifying its address is called a random-access memory (RAM). The time required to access one word is called the memory access time. This time is independent of the location of the word being accessed. It typically ranges from a few nanoseconds (ns) to about 100 ns for current RAM units.

Cache Memory

- As an adjunct to the main memory, a smaller, faster RAM unit, called a **cache**, is used to hold sections of a program that are currently being executed, along with any associated data.
- The cache is tightly coupled with the processor and is usually contained on the same

- The purpose of the cache is to facilitate high instruction execution rates.
- At the start of program execution, the cache is empty.
- All program instructions and any required data are stored in the main memory.
- As execution proceeds, instructions are fetched into the processor chip, and a copy of each is placed in the cache.
- When the execution of an instruction requires data located in the main memory, the data are fetched and copies are also placed in the **cache**.

Secondary Storage

- Primary memory is essential, it tends to be expensive and does not retain information when power is turned off.
- Thus additional, less expensive, permanent secondary storage is used when large amounts of data and many programs have to be stored, particularly for information that is accessed infrequently.
- Access times for secondary storage are longer than for primary memory.
- A wide selection of secondary storage devices is available, including magnetic disks, optical disks (DVD and CD), and flash memory devices.

Arithmetic and Logic Unit

- Most computer operations are executed in the **arithmetic and logic unit (ALU)** of the processor.
- Any arithmetic or logic operation, such as addition, subtraction, multiplication, division, or comparison of numbers, is initiated by bringing the required operands into the processor, where the operation is performed by the ALU.
- **For example**, if two numbers located in the memory are to be added, they are brought into the processor, and the addition is carried out by the ALU.
- The sum may then be stored in the memory or retained in the processor for immediate use.
- When operands are brought into the processor, they are stored in high-speed storage elements called **registers**.
- Each register can store one word of data.
- Access times to registers are even shorter than access times to the cache unit on the processor chip.

Output Unit

- The output unit is the counterpart of the input unit.
- Its function is to send processed results to the outside world.
- A familiar example of such a device is a **printer**.
- Most printers employ either photocopying techniques, as in laser printers, or ink jet streams.
- Such printers may generate output at speeds of 20 or more pages per minute.
- However, printers are mechanical devices, and as such are quite slow compared to the electronic speed of a processor.
- Some units, such as graphic displays, provide both an output function, showing text

- and graphics, and an input function, through touchscreen capability.

Control Unit

- The memory, arithmetic and logic, and I/O units store and process information and perform input and output operations.
- The operation of these units must be coordinated in some way.
- This is the responsibility of the **control unit**.
- The control unit is effectively the nerve center that sends control signals to other units and senses their states.
- I/O transfers, consisting of input and output operations, are controlled by program instructions that identify the devices involved and the information to be transferred.
- Control circuits are responsible for generating the timing signals that govern the transfers and determine when a given action is to take place.
- Data transfers between the processor and the memory are also managed by the control unit through timing signals.
- A large set of control lines (wires) carries the signals used for timing and synchronization of events in all units.
 - The operation of a computer can be summarized as follows:
 - The computer accepts information in the form of programs and data through an input unit and stores it in the memory.
 - Information stored in the memory is fetched under program control into an arithmetic and logic unit, where it is processed
 - Processed information leaves the computer through an output unit
 - All activities in the computer are directed by the control unit

BASIC OPERATIONAL CONCEPTS:

The program to be executed is stored in memory. Instructions are accessed from memory to the processor one by one and executed.

STEPS FOR INSTRUCTION EXECUTION

Consider the following instruction

Ex: 1 Add LOCA, R₀

This instruction is in the form of the following instruction format

Opcode Source, Destination

Where Add is the *operation code*, LOCA is the Memory operand and R₀ is Register operand. This instruction adds the contents of memory location LOCA with the contents of Register R₀ and the result is stored in R₀ Register.

The symbolic representation of this instruction is

R₀ [LOCA] + [R₀]

The contents of memory location LOCA and Register R₀ before and after the execution of this instruction is as follows

Before instruction execution

LOCA = 23H

R₀ = 22H

After instruction execution

LOCA = 23H

R₀ = 45H

The steps for instruction execution are as follows

1. Fetch the instruction from memory into the IR (instruction register in CPU).
2. Decode the instruction 1111000000 10011010
3. Access the Memory Operand
4. Access the Register Operand
5. Perform the operation according to the Operation Code.
6. Store the result into the Destination Memory location or Destination Register.

Ex:2 Add R₁, R₂, R₃

This instruction is in the form of the following instruction format
Opcode, Source-1, Source-2, Destination

Where R₁ is Source Operand-1, R₂ is the Source Operand-2 and R₃ is the Destination. This instruction adds the contents of Register R₁ with the contents of R₂ and the result is placed in R₃ Register.

The symbolic representation of this instruction is

R₃ [R₁] + [R₂]

The contents of Registers R₁, R₂, R₃ before and after the execution of this instruction is as follows.

Before instruction execution

R₁ = 24H

R₂ = 34H

R₃ = 38H

After instruction execution

R₁ = 24H

R₂ = 34H

R₃ = 58H

The steps for instruction execution is as follows

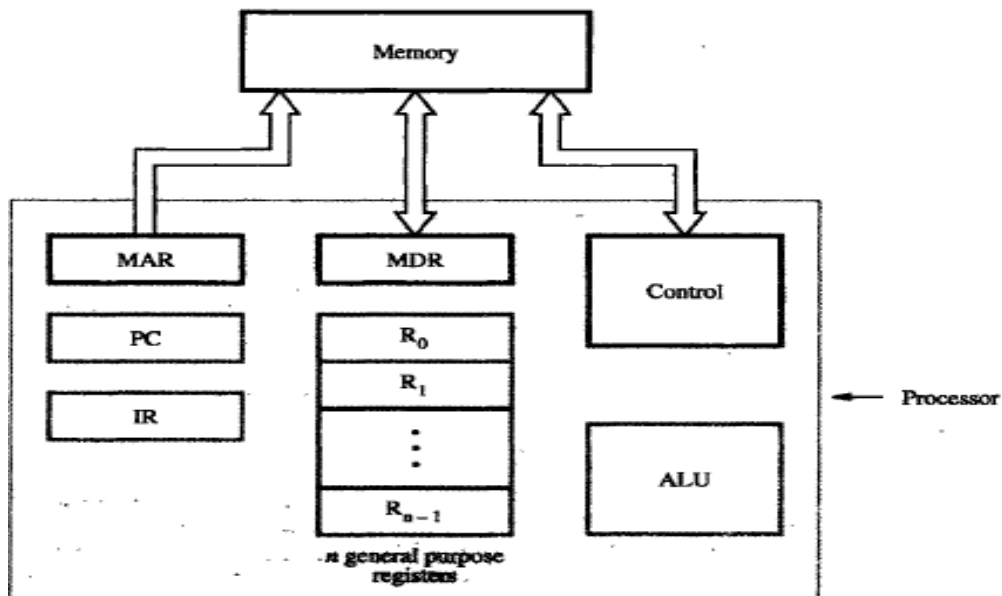
1. Fetch the instruction from memory into the IR.
2. Decode the instruction
3. Access the First Register Operand R₁
4. Access the Second Register Operand R₂
5. Perform the operation according to the Operation Code.
6. Store the result into the Destination Register R₃.

CONNECTION BETWEEN MEMORY AND PROCESSOR

The connection between Memory and Processor is as shown in the figure.

The Processor consists of different types of registers.

1. MAR (Memory Address Register)
2. MDR (Memory Data Register)
3. Control Unit
4. PC (Program Counter)
5. General Purpose Registers
6. IR (Instruction Register)
7. ALU (Arithmetic and Logic Unit)



The functions of these registers are as follows

MAR

- It establishes communication between Memory and Processor
- It stores the address of the Memory Location as shown in the figure.

MAR

5000h

Memory

5000	23h
5001	43h
5002	78h
5003	65h

MDR

- It also establishes communication between Memory and the Processor.
- It stores the **contents** of the memory location (data or operand), written into or read from memory as shown in the figure.

MDR

23h

Memory

23h	5000
43h	5001
78h	5002
65h	5003

2. CONTROL UNIT

- It controls the data transfer operations between memory and the processor.
- It controls the data transfer operations between I/O and processor.
- It generates control signals for Memory and I/O devices.

3. PC (PROGRAM COUNTER)

- It is a special purpose register used to hold the address of the next instruction to be executed.
- The contents of PC are incremented by 1 or 2 or 4, during the execution of current instruction.

- ## GENERAL PURPOSE REGISTER / REGISTER ARRAY

$\mathbf{R}_0$
$\mathbf{R}_1$
$\mathbf{R}_2$
$\cdot$
$\mathbf{R}_{n-1}$

- ## 5. IR (INSTRUCTION REGISTER)

6. ALU (ARITHMETIC and LOGIC UNIT)

- The content of MDR is moved to IR.

BUS STRUCTURE

1. Address bus :

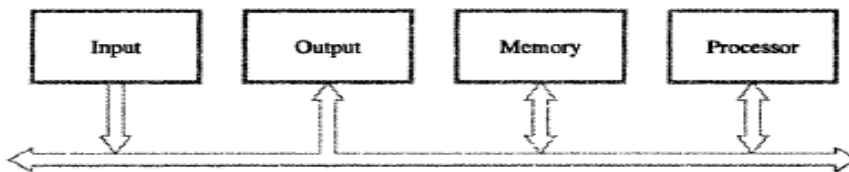
- ## 2. Data bus

- It is a bidirectional bus.
- The CPU sends data from Memory to CPU and vice versa as well as from I/O to CPU and vice versa by means of this bus.

3. **Control bus:**

- This bus carries control signals for Memory and I/O devices. It generates control signals for Memory namely MEMRD and MEMWR and control signals for I/O devices namely IORD and IOWR.

The structure of single bus organization is as shown in the figure



- The I/O devices, Memory and CPU are connected to this bus is as shown in the figure.
- It establishes communication between two devices, at a time.

Features of Single bus organization are

- Less Expensive
- Flexible to connect I/O devices.
- Poor performance due to single bus.

There is a variation in the devices connected to this bus in terms of speed of operation. Few devices like keyboard, are very slow. Devices like optical disk are faster. Memory and processor are faster, but all these devices uses the same bus. Hence to provide the synchronization between two devices, a buffer register is attached to each device. It holds the data temporarily during the data transfer between two devices.

2. **PERFORMANCE**

- The performance of a Computer System is based on hardware design of the processor and the instruction set of the processors.
- To obtain high performance of computer system it is necessary to reduce the execution time of the processor.
- Execution time: It is defined as total time required executing one complete program.
- The processing time of a program includes time taken to read inputs, display outputs, system services, execution time etc.
- The performance of the processor is inversely proportional to execution time of the processor.

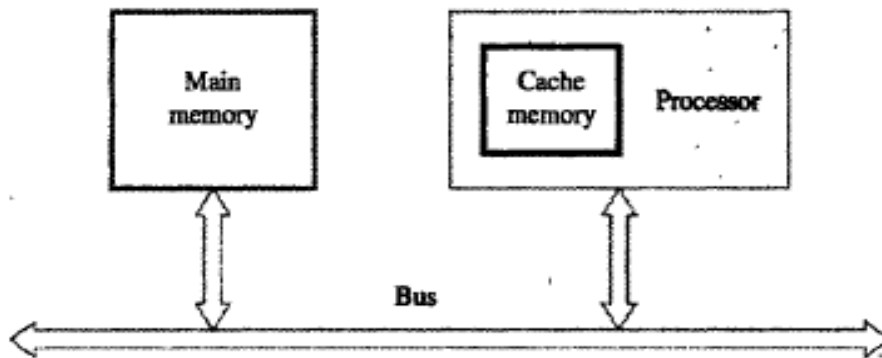
More performance = Less Execution time.

Less Performance = More Execution time.

The Performance of the Computer System is based on the following factors

1. *Cache Memory*
2. *Processor clock*
3. *Basic Performance Equation*
4. *Instructions*
5. *Compiler*

CACHE MEMORY: It is defined as a *fast access memory* located in between CPU and Memory. It is part of the processor as shown in the fig



The processor needs more time to read the data and instructions from main memory because main memory is away from the processor as shown in the figure. Hence it slowdown the performance of the system.

The processor needs less time to read the data and instructions from Cache Memory because it is part of the processor. Hence it improves the performance of the system.

PROCESSOR CLOCK: The processor circuits are controlled by timing signals called as Clock. It defines constant time intervals and are called as Clock Cycles. To execute one instruction there are 3 basic steps namely

1. Fetch
2. Decode
3. Execute.

The processor uses one clock cycle to perform one operation as shown in the figure

Clock Cycle	→	T1	T2	T3
Instruction	→	Fetch	Decode	Execute

The performance of the processor depends on the length of the clock cycle. To obtain high performance reduce the length of the clock cycle. Let „P“ be the number of clock cycles generated by the Processor and „R „ be the Clock rate .

The Clock rate is inversely proportional to the number of clock cycles.

i.e $R = 1/P$.

Cycles/second is measured in Hertz (Hz). Eg: 500MHz, 1.25GHz.

Two ways to increase the clock rate –

- Improve the IC technology by making the logical circuit work faster, so that the time taken for the basic steps reduces.
- Reduce the clock period, P.

BASIC PERFORMANCE EQUATION

Let „T „be *total time* required to execute the program.

Let „N „be the *number of instructions* contained in the program.

Let „S „be the *average number of steps* required to one instruction.

Let „R“ be number of clock cycles per second generated by the processor to execute one program.

Processor Execution Time is given by

$$T = N * S / R$$

This equation is called as Basic Performance Equation.

For the programmer the value of T is important. To obtain high performance it is necessary to reduce the values of N & S and increase the value of R

Performance of a computer can also be measured by using **benchmark** programs.

SPEC (System Performance Evaluation Corporation) is a non-profitable organization, that measures performance of computer using SPEC rating. The organization publishes the application programs and also time taken to execute these programs in standard systems.

$$SPEC = \frac{\text{Running time of reference Computer}}{\text{Running time of computer under test}}$$

DIFFERENCES MULTIPROCESSOR AND MULTICOMPUTER

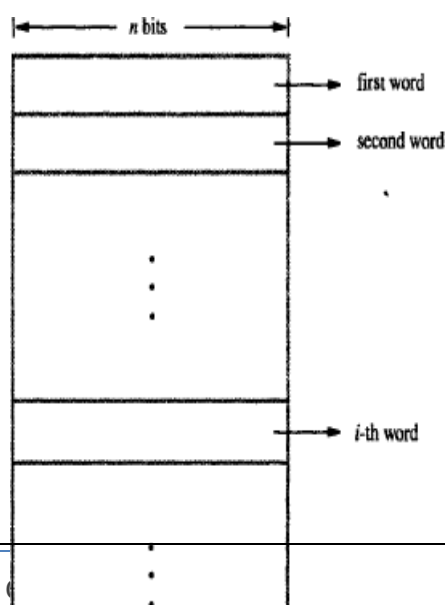
MULTIPROCESSOR	MULTICOMPUTER
1. It is a process of interconnection of two or more processors by means of system bus.	It is a process of interconnection of two or more computers by means of system bus.
2. It uses common memory to hold the data and instructions.	It has its own memory to store data and instructions.
3. Complexity in hardware design.	Not much complexity in hardware design.
4. Difficult to program for multiprocessor system.	Easy to program for multiprocessor system

MEMORY LOCATIONS AND ADDRESSES

1. Memory is a storage device. It is used to store character operands, data operands and instructions.
2. It consists of number of semiconductor cells and each cell holds 1 bit of information. A group of 8 bits is called as byte and a group of 16 or 32 or 64 bits is called as word.

Word length = 16 for 16 bit CPU and Word length = 32 for 32 bit CPU. Word length is defined as number of bits in a word.

- Memory is organized in terms of bytes or words.
- The organization of memory for 32 bit processor is as shown in the fig.



The contents of memory location can be accessed for read and write operation. The memory is accessed either by specifying address of the memory location or by name of the memory location.

- **Address space :** It is defined as number of bytes accessible to CPU and it depends on the number of address lines.

BYTE ADDRESSABILITY

Each byte of the memory are addressed, this addressing used in most computers are called byte addressability. Hence Byte Addressability is the process of assignment of address to successive bytes of the memory. The successive bytes have the addresses 1, 2, 3, 4,..... 2^n-1 . The memory is accessed in words.

In a 32 bit machine, each word is 32 bit and the successive addresses are 0,4,8,12,... and so on.

Address	32 – bit word			
0000	0 th byte	1 st byte	2 nd byte	3 rd byte
0004	4 th byte	5 th byte	6 th byte	7 th byte
0008	8 th byte	9 th byte	10 th byte	11 th byte
0012	12 th byte	13 th byte	14 th byte	15 th byte
.....				
n-3	n-3 th byte	n-2 th byte	n-1 th byte	n th byte

BIG ENDIAN and LITTLE ENDIAN ASSIGNMENT

Two ways in which byte addresses can be assigned in a word.

Or

Two ways in which a word is stored in memory.

1. Big endian
2. Little endian

BIG ENDIAN ASSIGNMENT

Word address	Byte address			
0	0	1	2	3
4	4	5	6	7
⋮				
$2^k - 4$	$2^k - 4$	$2^k - 3$	$2^k - 2$	$2^k - 1$

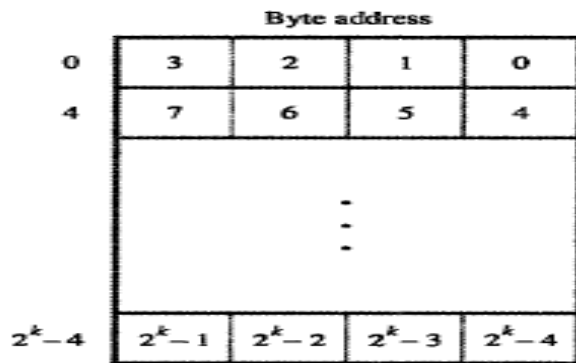
In this technique lower byte of data is assigned to higher address of the memory and higher byte of data is assigned to lower address of the memory.

The structure of memory to represent 32 bit number for big endian assignment is as shown in the above figure.

LITTLE ENDIAN ASSIGNMENT

In this technique lower byte of data is assigned to lower address of the memory and higher byte of data is assigned to higher address of the memory.

The structure of memory to represent 32 bit number for little endian assignment is as shown in the fig.



Eg – store a word “JOHNSMAN” in memory starting from word 1000, using Big Endian and Little endian.

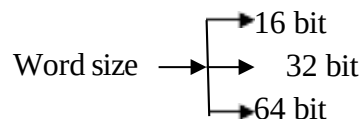
Bigendian -

1000	J 1000	O 1001	H 1002	N 1003
1004	S 1004	M 1005	A 1006	N 1007

Little endian -

1000	J 1003	O 1002	H 1001	N 1000
1004	S 1007	M 1006	A 1005	N 1004

WORD ALIGNMENT



The structure of memory for 16 bit CPU, 32 bit CPU and 64 bit CPU are as shown in the figures 1,2 and 3 respectively

For 16 bit CPU

5000	34H
5002	65H
5004	86H
5006	93H
5008	45H

For 32 bit CPU

5000	34H
5004	65H
5008	86H
5012	93H
5016	45H

For 64 bit CPU

5000	34H
5008	65H
5016	86H
5024	93H
5032	45H

It is process of assignment of addresses of two successive words and this address is the number of bytes in the word is called as Word alignment.

ACCESSING CHARACTERS AND NUMBERS

The character occupies 1 byte of memory and hence byte address for memory.
The numbers occupies 2 bytes of memory and hence word address for numbers.

MEMORY OPERATION

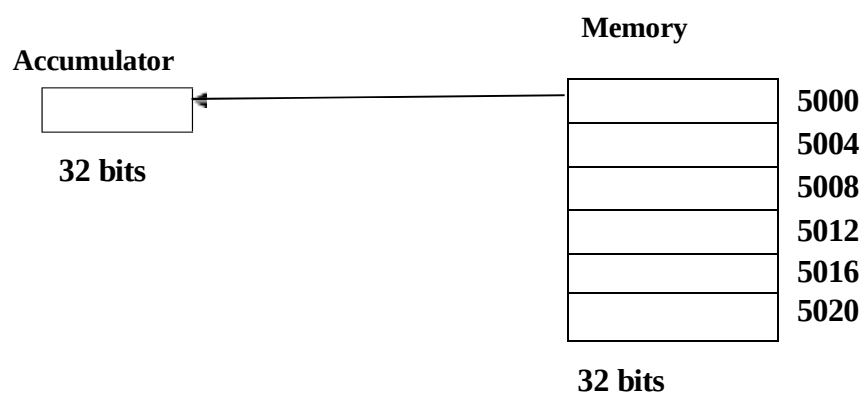
Both program instructions and operands are in memory. To execute each instruction has to be read from memory and after execution the results must be written to memory. There are two types of memory operations namely 1. Memory read and 2. Memory write

Memory read operation [Load/ Read / Fetch]

Memory write operation [Store/ write]

1. MEMORY READ OPERATION:

- ✓ It is the process of transferring of 1 word of data from memory into Accumulator (GPR).
- ✓ It is also called as Memory fetch operation.
- ✓ The Memory read operation can be implemented by means of LOAD instruction.
- ✓ The LOAD instruction transfers 1 word of data (1 word = 32 bits) from Memory into the Accumulator as shown in the fig.



Steps for Memory Read Operation

- (1) The processor loads MAR (Memory Address Register) with the address of the memory location.
- (2) The Control unit of processor issues memory read control signal to enable the memory component for read operation.
- (3) The processor reads the data from memory into the Accumulator by means of bi-directional data bus.

[MAR] Memory Accumulator

MEMORY WRITE OPERATION

- It is the process of transferring the 1 word of data from Accumulator into the Memory.
- The Memory write operation can be implemented by means of STORE instruction. The STORE instruction transfers 1 word of data from Accumulator into the Memory location as shown in the fig

Accumulator

32 bits

**5000
5004
5008
5012
5016
5020**

Steps for Memory Write Operation

- The processor loads MAR with the address of the Memory location.
- The Control Unit issues the Memory Write control signal.
- The processor transfers 1 word of data from the Accumulator into the Memory location by means of bi-directional data bus.

COMPUTER OPERATIONS (OR) INSTRUCTIONS AND INSTRUCTION EXECUTION

The Computer is designed to perform 4 types of operations, namely

- Data transfer operations
- ALU Operations
- Program sequencing and control.
- I/O Operations.

Data Transfer Operations

a) Data transfer between two registers.

Format: Opcode Source1 , Destination

The processor uses MOV instruction to perform data transfer operation between two registers. The mathematical representation of this instruction is $R1 \rightarrow R2$.

Ex : MOV R₁, R₂ : R₁ and R₂ are the registers.

Where MOV is the operation code, R₁ is the source operand and R₂ is the destination operand. This instruction transfers the contents of R₁ to R₂.

EX: Before the execution of MOV R₁, R₂, the contents of R₁ and R₂ are as follows

R₁ = 34h and R₂ = 65h

After the execution of MOV R₁, R₂, the contents of R₁ and R₂ are as follows

R₁ = 34H and R₂ = 34H

b) Data transfer from memory to register

The processor uses **LOAD** instruction to perform data transfer operation from memory to register. The mathematical representation of this instruction is

[LOCA] → ACC. Where ACC is the Accumulator.

Format : opcode operand

Ex: LOAD LOCA

For this instruction Memory Location is the source and Accumulator is the destination.

c) Data transfer from Accumulator register to memory

The processor uses **STORE** instruction to perform data transfer operation from Accumulator register to memory location. The mathematical representation of this instruction is

[ACC] → LOCA. Where, ACC is the Accumulator.

Format: opcode operand

Ex: STORE LOCA

For this instruction accumulator is the source and memory location is the destination.

2. ALU Operations

The instructions are designed to perform arithmetic operations such as Addition, Subtraction, Multiplication and Division as well as logical operations such as AND, OR and NOT operations.

Ex1: ADD R₀, R₁

The mathematical representation of this instruction is as follows:

$R_1 \leftarrow [R_0] + [R_1]$; Adds the content of R_0 with the content of R_1 and result is placed in R_1 .

Ex2: SUB R_0, R_1

The mathematical representation of this instruction is as follows:

$R_1 \leftarrow [R_0] - [R_1]$; Subtracts the content of R_0 from the content of R_1 and result is placed in R_1 .

EX3: AND R_0, R_1 ; It Logically multiplies the content of R_0 with the content of R_1 and result is stored in R_1 . ($R_1 = R_0 \text{ AND } R_1$)

Ex4: NOT R_0 ; It performs the function of complementation.

I/O Operations: The instructions are designed to perform INPUT and OUTPUT operations. The processor uses MOV instruction to perform I/O operations. The input Device consists of one temporary register called as DATAIN register and output register consists of one temporary register called as DATAOUT register.

a) Input Operation: It is a process of transferring one WORD of data from DATA IN register to processor register.

Ex: MOV DATAIN, R_0

The mathematical representation of this instruction is as follows,

$R_0 \leftarrow [\text{DATAIN}]$

b) Output Operation: It is a process of transferring one WORD of data from processor register to DATAOUT register.

Ex: MOV R_0 , DATAOUT

The mathematical representation of this instruction is as follows,

$[R_0] \rightarrow \text{DATAOUT}$

REGISTER TRANSFER NOTATION

- There are 3 locations to store the operands during the execution of the program namely 1. Register 2. Memory location 3. I/O Port. Location is the storage space used to store the data.
- The instructions are designed to transfer data from one location to another location. Consider the first statement to transfer data from one location to another location
- “Transfer the contents of Memory location whose symbolic name is given by AMOUNT into processor register R_0 .”
- The mathematical representation of this statement is given by

$R_0 \leftarrow [\text{AMOUNT}]$

Consider the second statement to add data between two registers

- “Add the contents of R_0 with the contents of R_1 and result is stored in R_2 ”
- The mathematical representation of this statement is given by

$R_2 \leftarrow [R_0] + [R_1]$.

Such a notation is called as “Register Transfer Notation”.

It uses two symbols

1. A pair of square brackets $[]$ to indicate the contents of Memory location and
2. $\leftarrow$ to indicate the data transfer operation.

ASSEMBLY LANGUAGE NOTATION

Consider the first statement to transfer data from one location to another location

- “Transfer the contents of Memory location whose symbolic name is given by AMOUNT into processor register R_0 .”
 - The assembly language notation of this statement is given by
- | | | |
|--------|---------|-------------|
| MOV | AMOUNT, | R_0 |
| Opcode | Source | Destination |

This instruction transfers 1 word of data from Memory location whose symbolic name is given by AMOUNT into the processor register R₀.

- “Add the contents of R₀ with the contents of R₁ and result is stored in R₂”
- The assembly language notation of this statement is given by

ADD R₀ , R₁, R₂
Opcode source1, Source2, Destination

This instruction adds the contents of R₀ with the contents of R₁ and result is stored in R₂.

- The mathematical representation of this statement is given by
 $R_2 \leftarrow [R_0] + [R_1]$.

Such a notations are called as “Assembly Language Notations”

BASIC INSTRUCTION TYPES

There are 3 types basic instructions namely

1. Three address instruction format
2. Two address instruction format
3. One address instruction format

Consider the arithmetic expression $Z = A + B$, Where A,B,Z are the Memory locations.

Steps for evaluation

1. Access the first memory operand whose symbolic name is given by A.
2. Access the second memory operand whose symbolic name is given by B.
3. Perform the addition operation between two memory operands.
4. Store the result into the 3rd memory location Z.
5. The mathematical representation is $Z \leftarrow [A] + [B]$.

a) Three address instruction format : Its format is as follows

opcode	Source-1	Source-2	destination
--------	----------	----------	-------------

Destination $\leftarrow$ [source-1] + [source-2]

Ex: ADD A, B, Z

$Z \leftarrow [A] + [B]$

a) Two address instruction format : Its format is as follows

opcode	Source	Source/destination
--------	--------	--------------------

Destination $\leftarrow$ [source] + [destination]

The sequence of two address m/c instructions to evaluate the arithmetic expression

$Z \leftarrow A + B$ are as follows

MOV A, R₀
MOV B, R₁
ADD R₀, R₁
MOV R₁, Z

b) One address instruction format : Its format is as follows

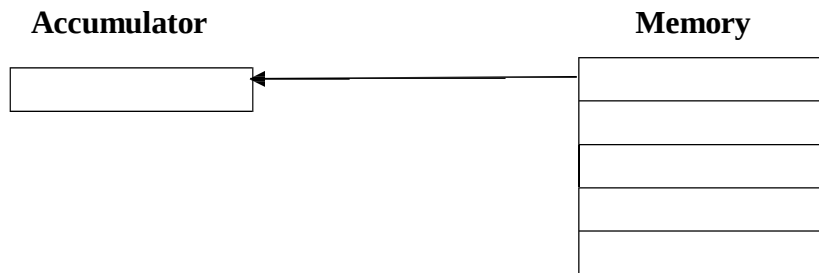
opcode	operand
---------------	----------------

Ex1: LOAD B

This instruction copies the contents of memory location whose symbolic name is given by „B“ into the Accumulator as shown in the figure.

The mathematical representation of this instruction is as follows

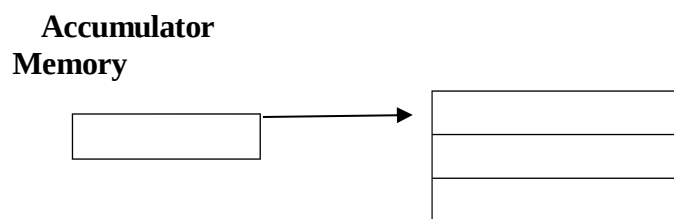
$ACC \leftarrow [B]$



Ex2: STORE B

This instruction copies the contents of Accumulator into memory location whose symbolic name is given by „B“ as shown in the figure. The mathematical representation is as follows

$B \leftarrow [ACC]$.



Ex3: ADD B

- This instruction adds the contents of Accumulator with the contents of Memory location „B“ and result is stored in Accumulator.
- The mathematical representation of this instruction is as follows
 $ACC \leftarrow [ACC] + [B]$

STRAIGHT LINE SEQUENCING AND INSTRUCTION EXECUTION

Consider the arithmetic expression

$$C = A + B, \text{ Where } A, B, C \text{ are the memory operands.}$$

The mathematical representation of this instruction is

$$C = [A] + [B].$$

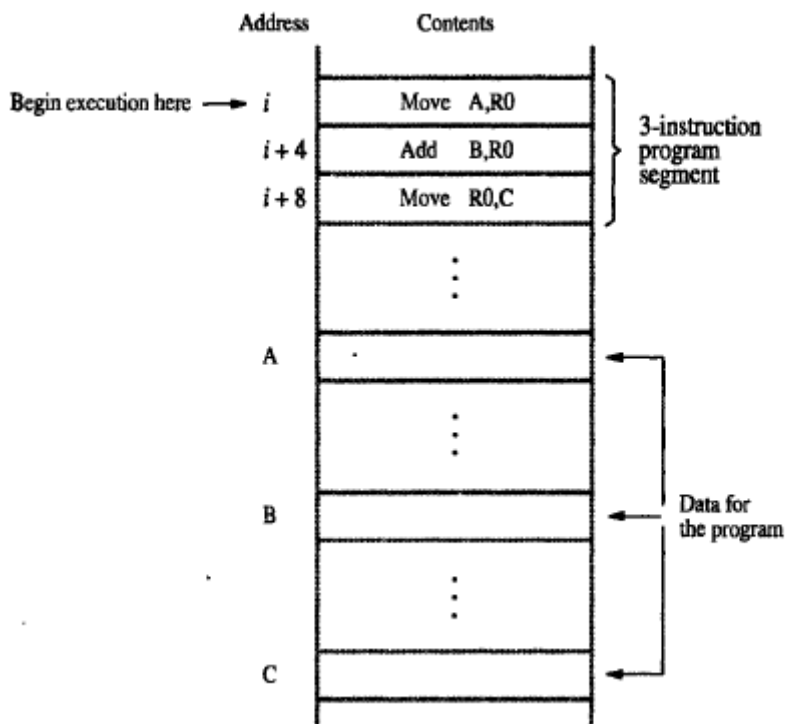
The sequence of instructions using two address instruction format are as follows

```
MOV  A,  R0
ADD  B,  R0
MOV  R0, C
```

Such a program is called as 3 instruction program.

NOTE: The size of each instruction is 32 bits.

- The 3 instruction program is stored in the successive memory locations of the processor is as shown in the fig.



- The system bus consists of uni-directional address bus, bi-directional data bus and control bus
 “It is the process of accessing the 1st instruction from memory whose address is stored in program counter into Instruction Register (IR) by means of bi-directional data bus and at the same time

after instruction access the contents of PC are incremented by 4 in order to access the next instruction. Such a process is called as “Straight Line Sequencing”.

INSTRUCTION EXECUTION

There are 4 steps for instruction execution

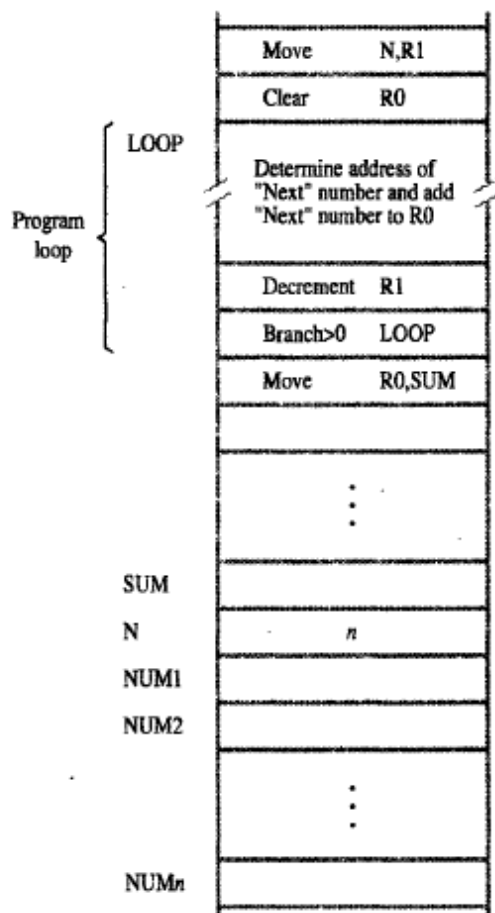
- 1 Fetch the instruction from memory into the Instruction Register (IR) whose address is stored in PC.

$$IR \leftarrow [PC]$$
- 2 Decode the instruction.
- 3 Perform the operation according to the opcode of an instruction
- 4 Load the result into the destination.
- 5 During this process, Increment the contents of PC to point to next instruction (In 32 bit machine increment by 4 address)

$$PC \leftarrow [PC] + 4.$$
- 6 The next instruction is fetched, from the address pointed by PC.

BRANCHING

Suppose a list of „N“ numbers have to be added. Instead of adding one after the other, the add statement can be put in a loop. The loop is a straight-line of instructions executed as many times as needed.



The „N“ value is copied to R1 and R1 is decremented by 1 each time in loop. In the loop find the value of next element and add it with R0.

In conditional branch instruction, the loop continues by coming out of sequence only if the condition is true. Here the PC value is set to „LOOP“ if the condition is true.

Branch > 0 LOOP // if >0 go to LOOP

The PC value is set to LOOP, if the previous statement value is >0 i.e. after decrementing R1 value is greater than 0.

If R1 value is not greater than 0, the PC value is incremented in a normal sequential way and the next instruction is executed.

CONDITION CODE BITS

- The processor consists of series of flip-flops to store the status information after ALU operation.
- It keeps track of the results of various operations, for subsequent usage.
- The series of flip-flop-flops used to store the status and control information of the processor is called as “Condition Code Register”. It defines 4 flags. The format of condition code register is as follows.

C	V	Z	N
---	---	---	---

1 (NEGATIVE) Flag:

It is designed to differentiate between positive and negative result.
It is set 1 if the result is negative, and set to 0 if result is positive.

2 Z (ZERO) Flag:

It is set to 1 when the result of an ALU operation is found to zero, otherwise it is cleared.

3 V (OVER FLOW) Flag:

In case of 2^s Complement number system n-bit number is capable of representing a range of numbers and is given by -2^{n-1} to $+2^{n-1}$. The Over-Flow flag is set to 1 if the result is found to be out of this range.

4 C (CARRY) Flag:

This flag is set to 1 if there is a carry from addition or borrow from subtraction, otherwise it is cleared.

Addressing Modes

The various formats of representing operand in an instruction or location of an operand is called as “Addressing Mode”. The different types of Addressing Modes are

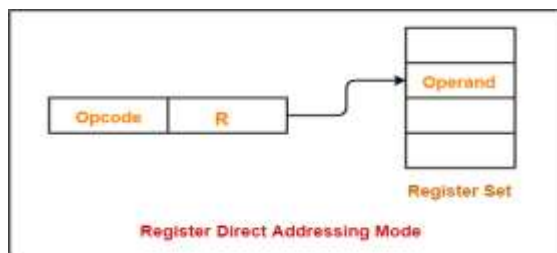
- Register Addressing
- Direct Addressing
- Immediate Addressing
- Indirect Addressing
- Index Addressing

- f) Relative Addressing
- g) Auto Increment Addressing
- h) Auto Decrement Addressing

a. REGISTER ADDRESSING:

In this mode operands are stored in the registers of CPU. The name of the register is directly specified in the instruction.

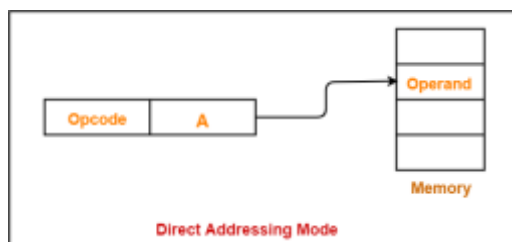
Ex: MOVE R₁,R₂ Where R₁ and R₂ are the Source and Destination registers respectively. This



instruction transfers 32 bits of data from R₁ register into R₂ register. This instruction does not refer memory for operands. The operands are directly available in the registers.

Ex: MOVE LOCA, R₁ : Where LOCA is the memory location and R₁ is the Register.

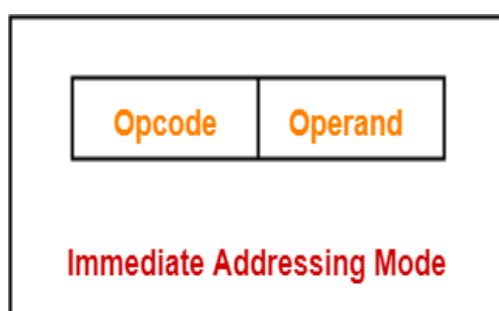
This instruction transfers 32 bits of data from memory location X into the General Purpose Register R₁.



c. IMMEDIATE ADDRESSING

In this Addressing Mode operands are directly specified in the instruction. The source field is used to represent the operands. The operands are represented by # (hash) sign.

Ex: MOVE #23, R0



d. INDIRECT ADDRESSING

In this Addressing Mode effective address of an operand is stored in the memory location or General Purpose Register.

The memory locations or GPRs are used as the memory pointers.

Memory pointer: It stores the address of the memory location.

There are two types Indirect Addressing

- i) Indirect through GPRs
- ii) Indirect through memory location

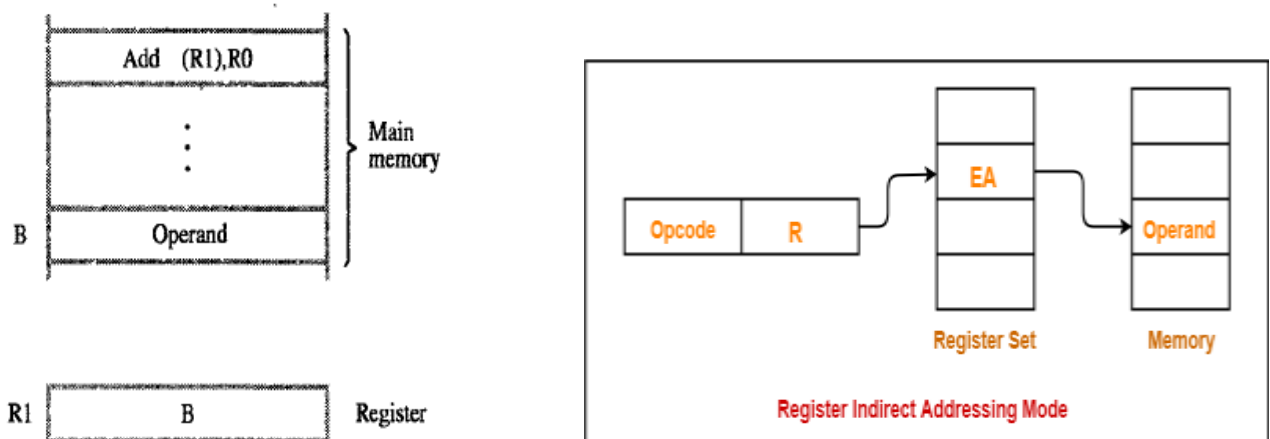
i) Indirect Addressing Mode through GPRs

In this Addressing Mode the effective address of an operand is stored in the one of the General Purpose Register of the CPU.

Ex: ADD (R₁), R₀ ; Where R₁ and R₀ are GPRs

This instruction adds the data from the memory location whose address is stored in R₁ with the contents of R₀ Register and the result is stored in R₀ register as shown in the fig.

The diagrammatic representation of this addressing mode is as shown in the fig.



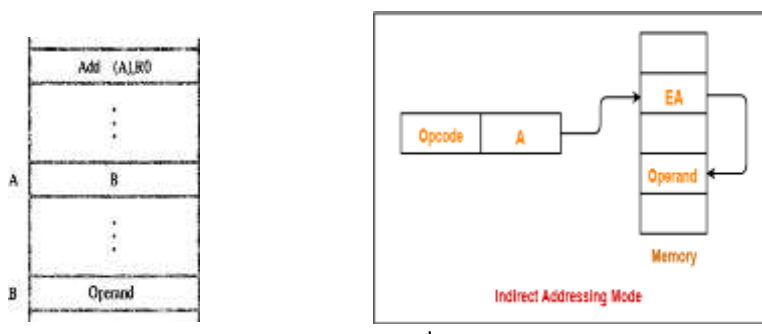
ii) Indirect Addressing Mode through Memory Location.

In this Addressing Mode, effective address of an operand is stored in the memory location.

Ex: ADD (X), R₀

This instruction adds the data from the memory location whose address is stored in „X memory location with the contents of R₀ and result is stored in R₀ register.

The diagrammatic representation of this addressing mode is as shown in the fig.



e. INDEX ADDRESSING MODE

In this addressing mode, the effective address of an operand is computed by adding constant value with the contents of Index Register and any one of the General Purpose Register namely R_0 to R_{n-1} can be used as the Index Register. The constant value is directly specified in the instruction.

The symbolic representations of this mode are as follows

1. $X(R_i)$ where X is the Constant value and R_j is the GPR.

It can be represented as

$$EA \text{ of an operand} = X + (R_i)$$

2. (R_i, R_j) Where R_i and R_j are the General Purpose Registers used to store addresses of an operand and constant value respectively. It can be represented as
The EA of an operand is given by

$$EA = (R_i) + (R_j)$$

3. $X(R_i, R_j)$ Where X is the constant value and R_i and R_j are the General Purpose Registers used to store the addresses of the operands. It can be represented as

The EA of an operand is given by

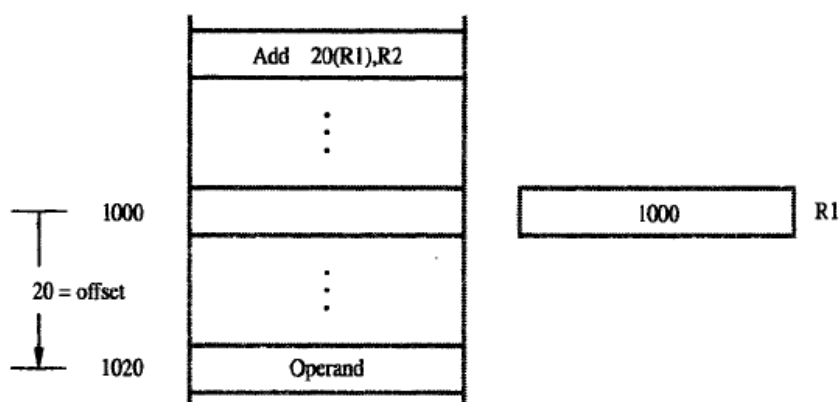
$$EA = (R_i) + (R_j) + X$$

There are two types of Index Addressing Modes

i) Offset is given as constant.

ii) Offset is in Index Register.

Note : Offset : It is the difference between the starting effective address of the memory location and the This instruction adds the contents of memory location whose EA is the sum of contents of R_1 with 20 and with the contents of R_2 and result is placed in R_2 register. The diagrammatic representation of this mode is as shown in the fig.



ii) Offset is in Index Register

Ex: ADD 1000(R_1), R_2 R_1 holds the offset address of an operand.

The EA of an operand is given by

$$EA = 1000 + [R_1]$$

effective address of the operand fetched from memory.

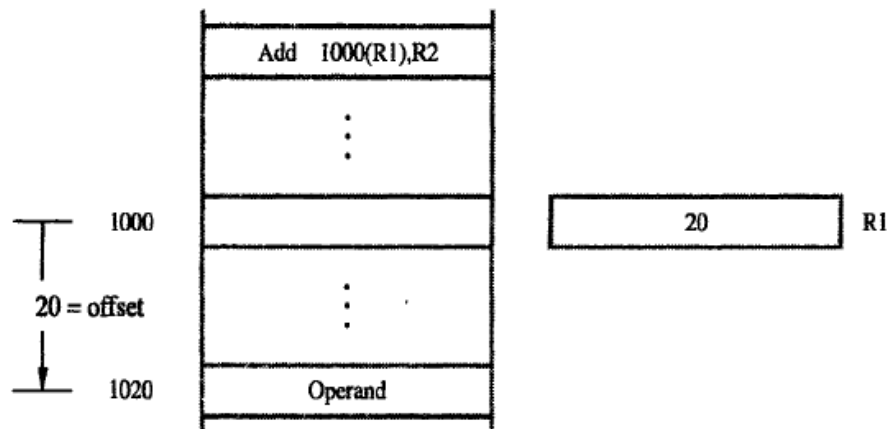
i) Offset is given as constant

Ex: ADD 20(R₁), R₂

The EA of an operand is given by $EA = 20 + [R_1]$

This instruction adds the data from the memory location whose address is given by $[1000 + [R_1]]$ with the contents of R₂ and result is placed in R₂ register.

The diagrammatic representation of this mode is as shown in the fig.



- f. **RELATIVE ADDRESSING MODE:** In this Addressing Mode EA of an operand is computed by the Index Addressing Mode. This Addressing Mode uses PC (Program Counter) to store the EA of the next instruction instead of GPR

The symbolic representation of this mode is $X(PC)$. Where X is the offset value and PC is the Program Counter to store the address of the next instruction to be executed.

It can be represented as

EA of an operand = $X + (PC)$.

This Addressing Mode is useful to calculate the EA of the target memory location.

g. **AUTO INCREMENT ADDRESSING MODE**

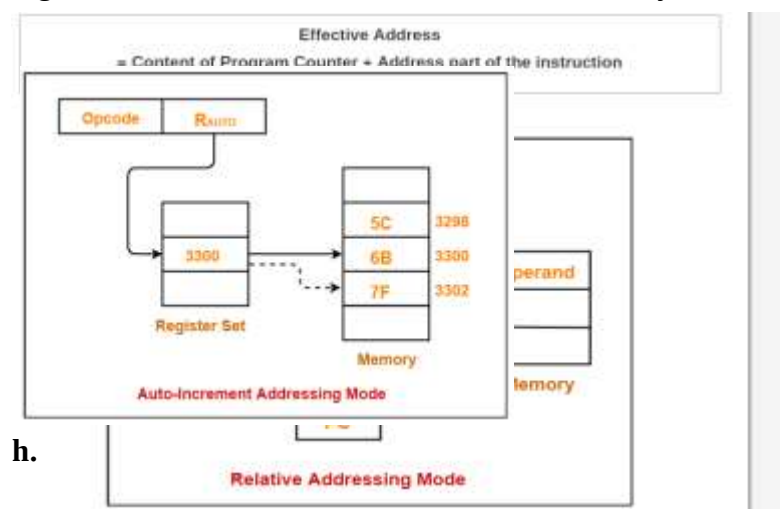
In this Addressing Mode, EA of an operand is stored in the one of the GPR^s of the CPU. This Addressing Mode increments the contents of memory register by 4 memory locations after operand access.

The symbolic representation is

$(R_i) +$ Where R_i is the one of the GPR.

Ex: MOVE $(R_1) +$, R₂

This instruction transfers data from the memory location whose address is stored in R₁ into R₃ register and then it increments the contents of R₁ by 4 memory locations.



AUTO DECREMENT ADDRESSING MODE

In this Addressing Mode , EA of an operand is stored in the one of the GPR^s of the CPU. This Addressing Mode decrements the contents of memory register by 4 memory locations and then transfers the data to destination.

The symbolic representation is

-(R_i) Where R_i is the one of the GPR.

Ex: MOVE -(R₁) , R₂

This instruction first decrements the contents of R₁ by 4 memory locations and then transfer's data of that location to destination register.