Unit-3

Topics to cover

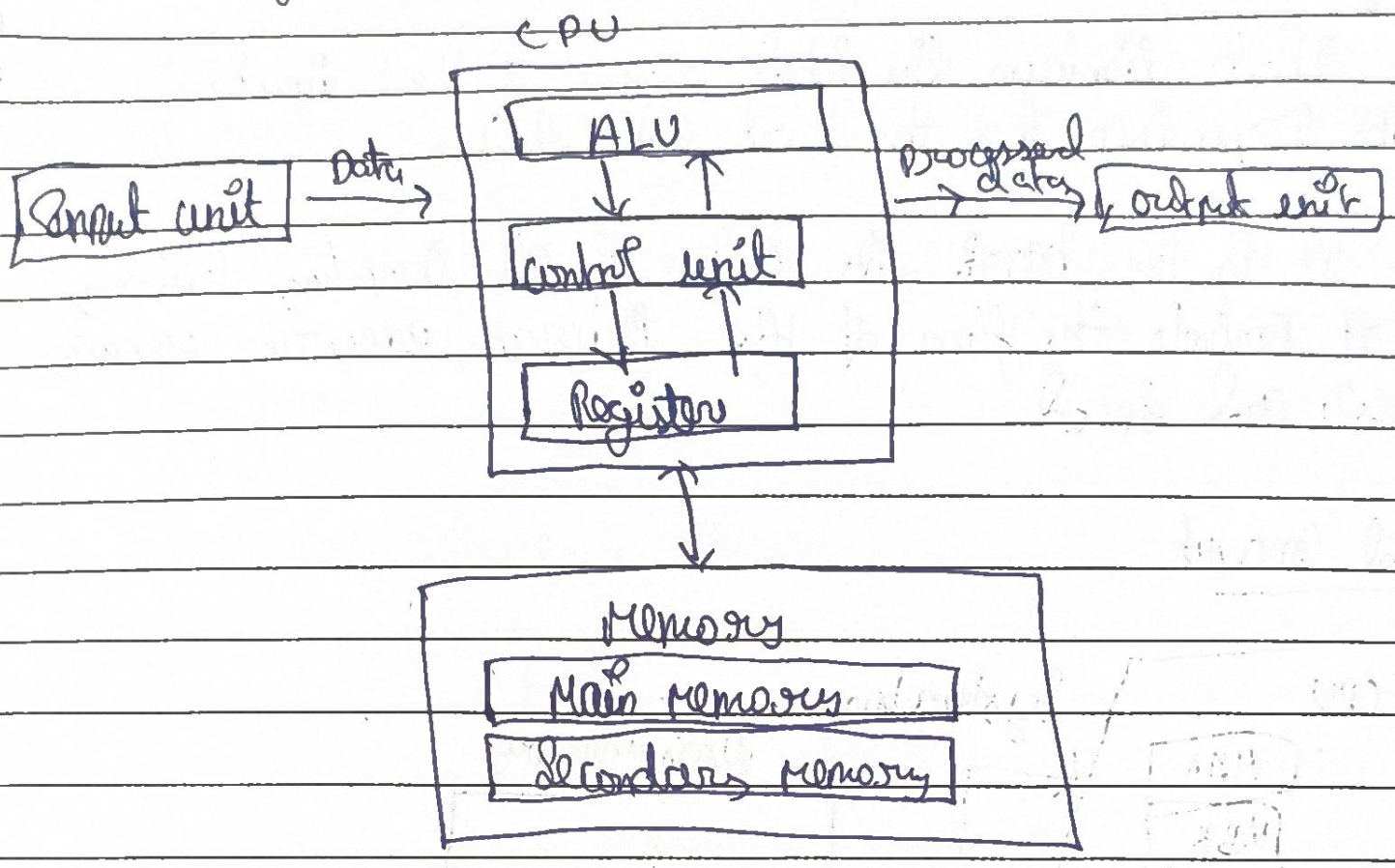
*) Von Neumann Architecture ✓
*) operations and operands ✓
*) ISA (Instruction set architecture : Memory location Address and operations ✓
*) Instruction set and Instruction sequencing ✓
*) Addressing modes, Encoding of machine instructions ✓
*) Assembly language and high level language ✓

Components of Computer System with the Block diagram in detail

Components of Computer System

1) Input (keyboard, mouse that takes info from the user)
2) Memory (Stores temporarily or permanently)
3) Arithmetic and logic circuit (Performs operations)
4) Output (display)
5) Control unit (Heart of all controls the flow)

Block diagram



Input unit : Receives informations from the user, and passes to the device

Eg: keyboard, mouse

Memory unit : Memory unit is used to store programs as well as data

Eg: SSD, HDD

*) primary storage (Main memory) : It is responsible for executing programs currently, they are temporary storage, they are costly and low capacity but fast

*) Secondary memory : It is permanent storage, cost effective more capacity but slow

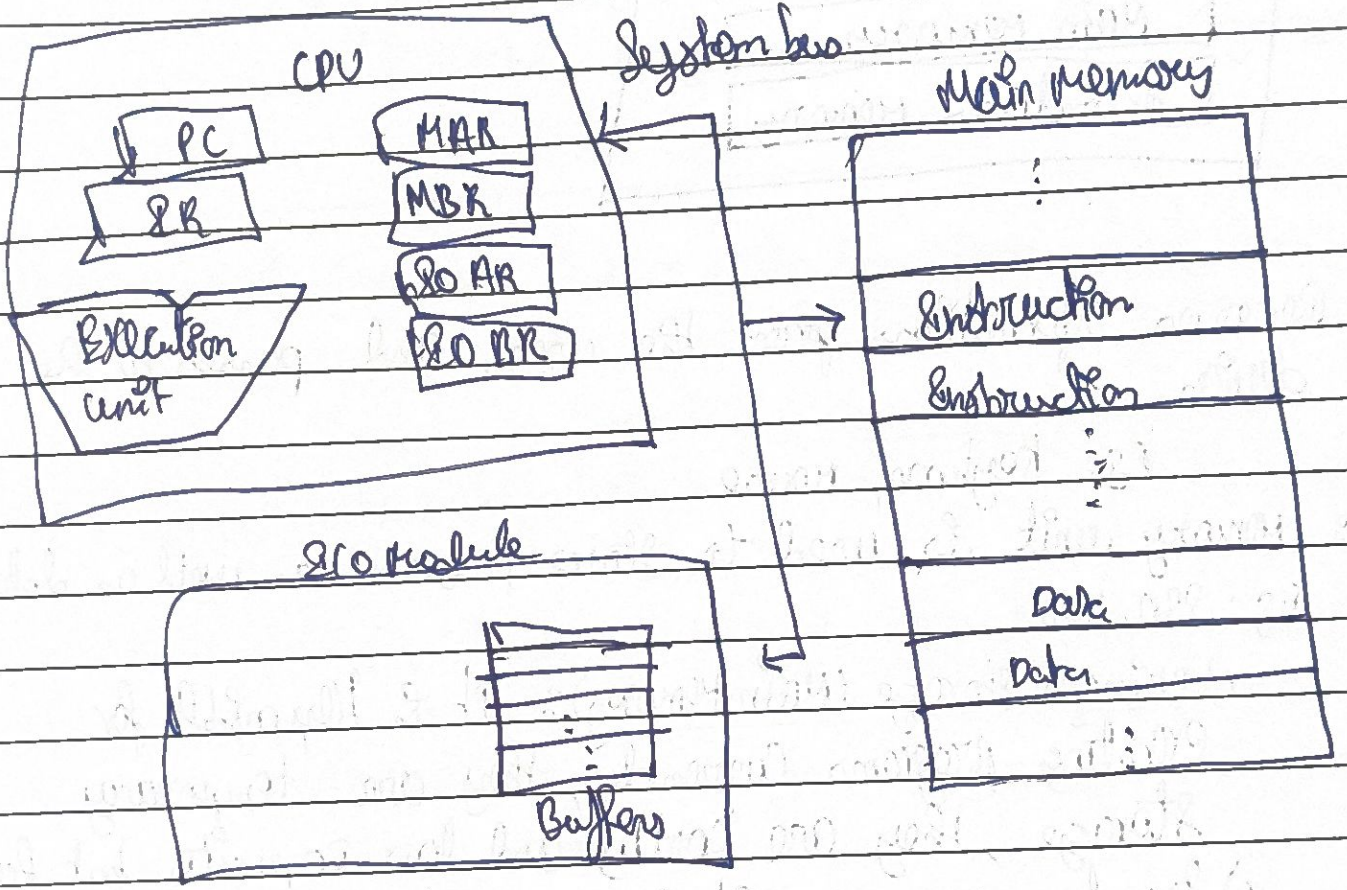
*) Arithmetic logic unit is next

ALU: Arithmetic Logic Unit : It is responsible for processing inputs and perform actions on them and get a final output it performs operations on operands

Output unit :
That displays the final output of the inputs and is responsible for the front end stuffs

Control unit : It is the heart, the Embrio of all computing devices, it controls the flow of the program, execution, memory I/O and speed.

Basic operational Concept



PC → Program Counter
IR → Information Register
MAR → Memory Address Register
MBR → Memory Buffer Register
I/O AR → Input / output Address register
I/O BR → Input / output Buffer register

* PC → Program Counter contains the address of instruction that is about to be executed next.
* IR → It has the instruction that is executed successfully
* MAR → Memory Address Register contains the address of the word in main memory that is being accessed
* MBR → Memory Buffer Register (also called as MDR Memory Data Register) It contains the data to be written on Memory or receives the data read from memory.

There are 3 phase in process

1) Fetch:
   * The CPU looks at the program counter (PC) to see which memory address holds the next instruction
   * It retrieves the instruction from memory
   * While waiting for the memory to respond, the PC increments. This ensures the CPU is ready to grab the next sequential instruction in the following cycle.

2) Decode:
   * The fetched data is stored in the Instruction Register (IR).
   * The IR now holds the binary machine code.
   * The processor interprets (decodes) this binary to figure out exactly exactly what tasks needs to be performed

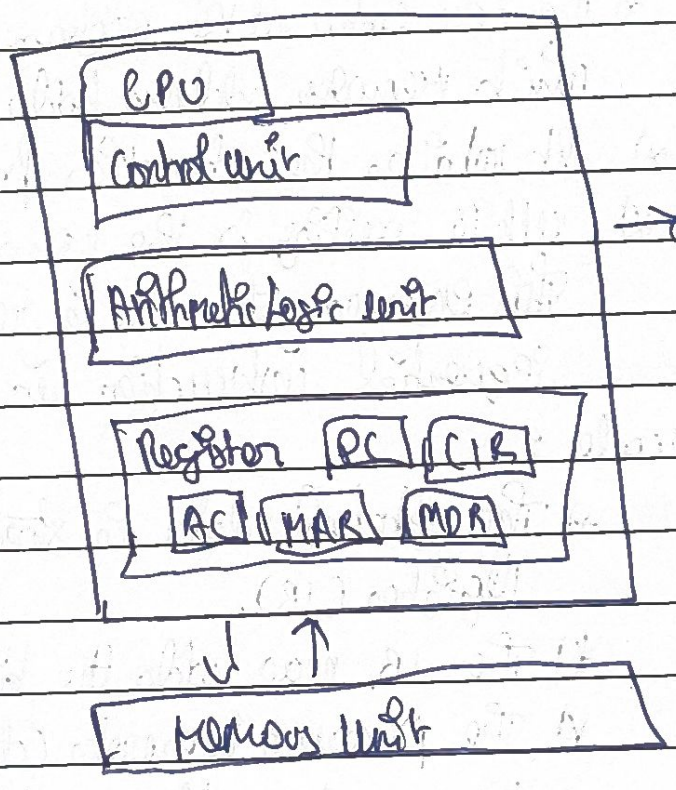
3) Execute:
   - The processor carries out the instruction
   - Complex operations: This often requires further memory access (eg. / fetching two numbers to add them)
   - Role of Registers:
     i) The MAR is used to find the address of the data
     ii) The MBR is used to temporarily holds that data during transfer.

M T W T F S S
Page No.: YOUVA
Date:

Von Neumann Architecture :

* Von Neumann architecture was first published by John Von Neumann in 1945. His computer Architecture design consists of a Control unit, Arithmetic and logic unit (ALU), memory unit, Registers and Inputs / outputs

* Von Neumann architecture is based on the Stored program Counter Concept, where instructions data and program data are stored in the same memory. This design is still used in most Computers today



* Just explain about input output and CPUs
* Registers

PC → Program Counter, Contains the address of the next instruction to be executed

CIR → Current instruction Register, Contains the current instructions during processing

AC → Accumulator, where intermediate arithmetic and logic results are stored

MAR → Memory address Registers, Holds the memory location of data that needs to be accessed

MDR → Memory data Register, holds data that is being transferred to or from Memory

Busses:

Address Bus → Address of the data is carried
Data Bus → Carries Data
Control Bus → Carries control signals

Bottleneck

* Only one instruction can be processed at a time
* This is called von neuman Bottleneck

Operations and operands

Operations → set of instructions that are ~~commonly~~ performed using MIPS assembly language
operands → are in which the operations are performed

MIPS Instructions

Arithmetic

* add → `add $S1, $S2 $S2` → $S1 + $S2 = $S3 are three operands
* Subtract → `Sub $S1, $S2, $S3` → $S1 = $S2 - $S3
* add immediate → `addi $S1, $S2, 100` → used to add constants

Data transfer :

- Load word lw $s1, 100($s2) → $s1 = Memory[$s2+100] word from memory to register
- Store word sw $s1, 100($s2) => Memory[$s2 +100] = $s1
- Load byte lb $s1, 100($s2) $s1 = Memory[$s2 +100]
- Store byte sb $s1, 100($s2) => Memory[$s2 +100] = $s1
- Load upper immediate lui $s1, 100

Conditional Branch: ✱

*) branch on equal beq $s1, $s2, $s2*25 if ($s1=2 $s2) goto PC+4+100

*) branch on not equal bne $s1, $s2, 25 if $s1 != $s2 go to PC+4+100

*) Set on Less than slt $s1, $s2 $s3 if ($s2 < $s3) $s1 = 1; else $s1 = 0

*) Set less than immediate slti $s1, $s2, 100 if ($s2 < 100) $s1 = 1 else $s1 = 0

Unconditional Jump:

*) Jump j 2500 go to 10000

*) Jump Register Jr $ra go to $ra

*) Jump and Link jal 2500 $ra = PC+4; go to 10000

Operands: operands are the data that operations are performed on

	Register Operand	Memory operands
Def:	These are fast dedicated storage locations	elements are stored in Main memory
Usage:	MIPS Arithmetic instruction	No Arithmetic, Only transfers to Main
Characteristics:	There are limited in number and fixed	There are more and Accessing is achieved by address

Instruction set architecture

① Instruction set Architecture usually defines a family of Microprocessor ex: Intel
② It is interface between a hardware and low level software
③ ISA includes the instruction set itself, rules for using the instructions, addressing modes, and instruction encoding. The MIPS Architecture is an example of a RISC (Reduced Instruction Set Computer) ISA
④ To achive the highest performance and conserve energy an instruction set architecture must have a sufficient number of registers and must use registers efficiently.
⑤ MIPS instruction set architecture has a small set of instructions and addressing modes. It is designed for use with high-level programming languages.
⑥ It allows parallel execution, provides large set of registers
⑦ It has fixed instruction size (32-bits)
⑧ MIPS can be categorized as

- Arithmetic / logical / Shift / Comparision
- Control Instructions
- Load / Store
- Others
- It has three instruction encoding formats ~~R-type 1, R-type 2, R-type 3~~

R type
I - Type
J - Type

Memory locations and Addresses

- Computer stores data in memory, it consists of millions of cells, one cell contains one bit, one bit is a very small amount of information
- For this reason, the bits are grouped as n bits adjacent to other, these set of bits are word and n is the length of the word, most of ~~these~~ memories word length is 8-bit or 1 byte
- Thus to access these words stored in a memory Address is used for locating, thus number of address lines provided decides the capability of Addressing
- Ex: if processor has 16 lines, it can access upto 2^{16} Memory locations

Memory capacity	Address line required
1k	10
2k	11
4k	12
8k	13
16k	14
32k	15
64k	16

Byte Addressability and Alignment

- The 8-bit length is known as a byte
- The most architectures, Individual bytes are addressable; this is called byte addressability. Each successive byte location is addressed with byte 0, byte 1, byte 2 and so on
- for a 32 bit word length machine, successive words are located at addresses 0, 4, 8 (since 32-bit word = 4bytes)

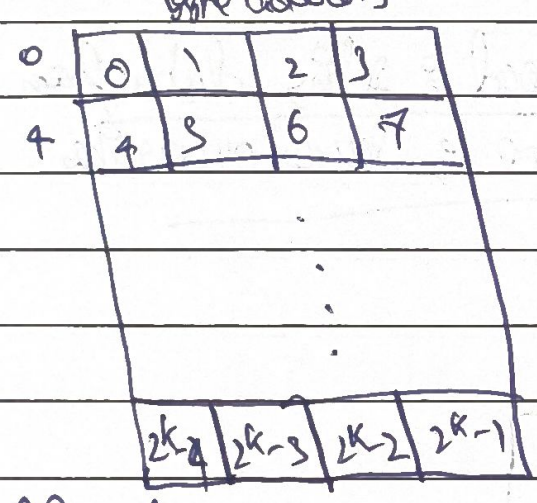
o word alignment: Words must start at natural word boundaries (addresses that are multiples of the word size in bytes) for a 32-bit (4-byte) system, aligned addresses are 0, 4, 8, 12 ...

Endianness:

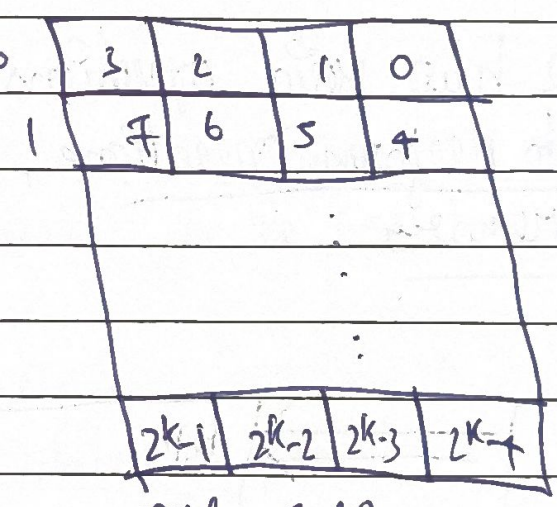
They are of referred to byte addresses are aligned across the bytes within a word.

- Big Endian: Lower byte addres are used for more significant bytes (leftmost bytes) of the word
- Little Endian: Lower Byte addresses are used for the less significant bytes (The Rightmost bytes) of the word.

word address



Big Endian



Little Endian

word alignment

- word alignment is requirement in computer architecture that specifies where data words must begin in memory.
- Natural word boundaries: words are considered aligned in memory if they begin at addresses corresponding to the natural size of the word. These addresses are called aligned addresses
- Alignment Rule: In general, for a word length of N bytes, aligned words must start at addresses that are multiples of N.

Ex:

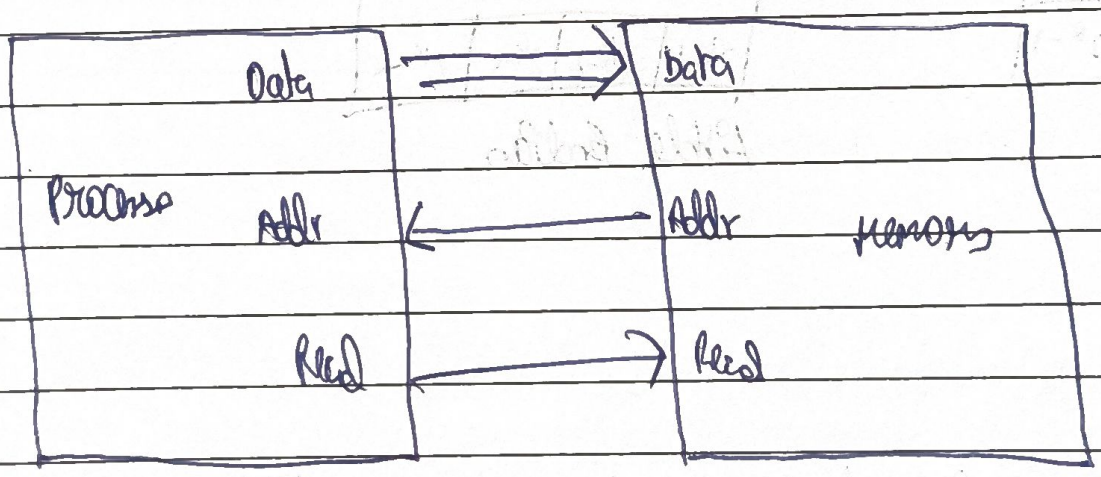
- 16-bit word length
- 32-bit word length
- 64-bit word length

a) MIPS Requirement: In the MIPS architecture, where the word size is 32-bits (4 bytes), words must start at a addresses that are multiple of 4

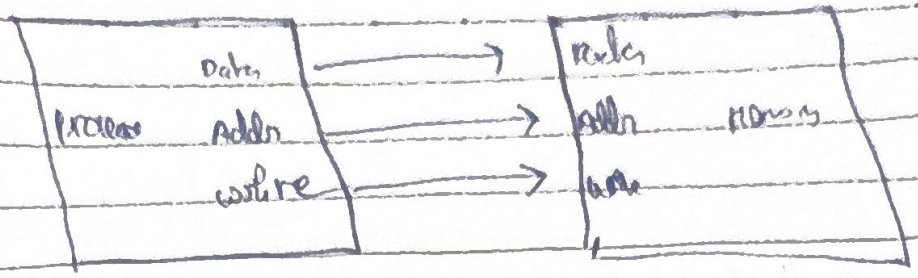
b) Unaligned Addresses: If words begin an arbitrary byte addresses that is not a multiple of the word length, the address is called an unaligned address

Memory operations

* There are two main operations, i) load : fetching data from memory to perform operations, ii) store : Saves or overrides data in memory



i) load



Instruction and Instruction Sequencing:

A Computer program is essentially a sequence of instruction designed to perform task. Instructions sequencing is the process of writing these instructions in a proper order to ensure correct execution

Instruction element and types

Each instruction contains specific information field, called elements of instruction, necessary for the CPU to execute it.

a) Operation Code (opCode): Specifies the operations to be performed by the binary code.
a) Source/Destination operand: Directly Specifies the data source or the destination for the result
a) Source operand Address: Specifies the address of the operands(s) which may be in CPU registers or memory
a) Destination operand Address: Specifies where the result of the operation is to be Stored
a) Next instruction Address: Tells the CPU where to find fetch the next instructions. This is explicitly specified for JUMP and BRANCH instructions, but for others, its the address immediately following the current instruction

The instructions are capable of

- Data transfer
- AL operations
- Program sequencing and control
- I/O Control

Instructions are also categorized by the number of memory addresses they refere

- Three - address
- Two - address
- One - address
- Zero - address

Instruction Sequencing

The processor uses the program counter (PC), which holds the address of the next instruction, to manage the execution sequence.

A) Straight line sequencing

- Start: The address of the first instructions is placed into the PC
- Execution: The processor fetches and executes instructions one at a time, in the order of increasing memory addresses
- PC update: During execution, the PC is incremented by the length of

M T W T F S S

Page No:

Date:

YOUVA

3) Branching (Program Control)

*) Straight line sequencing cannot handle decision-making or executing block of code repeatedly, repeatedly
*) These Transfer program control from one sequence to another
*) Conditional branch: Transfers if specified condition is true
*) Unconditional Jump: Transfers without any specified conditions
*) When Branch occurs, the target address is loaded in PC And the next instruction is fetched from this new address, breaking the sequential flow
*) Conditional Branches are used for looping, forcing a sequence of instructions repeat until a condition is met.

*) Initialize
*) Process
*) Loop control
*) Result.

4) Conditional codes

*) Decision making instruction rely on condition code flags, which are bits stored in a Status Register (or flag register)
*) ALU operations set or reset these flags to record the results conditions. The branch instructions then test these flags to make logical decisions
*) Common flags include

o) Carry/Borrow
o) Zero
o) Negative or Sign
o) Overflow

M T W T F S S

Page No.:

Date:

YOUVA

Addressing Modes (Giri's notes)

The different ways that a processor can access data are referred as addressing modes

There are twelve addressing modes:

1) Register Addressing Mode
2) Direct " "
3) Indirect " "
4) Register Indirect Addressing Mode
5) Relative addressing mode
6) Immediate " "
7) Displacement " "
8) Base Register " "
9) Index " "
10) Auto increment " "
11) Auto Decrement " "
12) Stack " "

① Register addressing modes

*) It is fast and efficient addressing modes because the data is accessed directly from the register, without needing to fetch it from memory

Ex: MOV $R_2 R_1$

This instruction copies the contents of Register R_2 to Register R_1

② Direct addressing mode:

It is a scheme in which the address specifies which memory used or register contains the operands.

Example [LOAD R1,1000] R1 = memory[1000]

③ Indirect Addressing mode

In this, the address field of the instruction contains the memory location or register where the effective address of the operand is present

Ex: LOAD R1,(1000)

④ Register Indirect Addressing mode

The address of the target memory location will be stored in the register and the register will be mentioned in the instruction

Ex: LOAD R1,(R2)

⑤ Relative Addressing mode:

⇒ Here the referenced register is program counter and hence this addressing mode is also known as pc relative addressing mode

Ex: JMP REL 8

Jumps to a memory address that is located at an offset of 8 byte from the current instruction pointer.

(6) Immediate Addressing mode:

It is a type of address field, where operand field is does not require any reference to memory,

Example: MOV R0, #25

(7) Displacement Addressing mode:

It instruction has two address field : Value and referred register. The effective address is computed by adding content of referred register to value

Ex: Ex: Value [R1]

(8) Base Register Addressing mode:

* It is used to implement inter segment transfer of control

Ex: MOV [R1+B], A

(9) Index Addressing mode:

* It is used to access elements in arrays which are stored in memory at consecutive locations

Ex: MOV [R1+15], R

The contents of the memory address generated by the addition of base memory address and displacement is copied to register R.

⑩ Auto Increment Addressing mode:

The content present in the register is initially incremented and then the content that is incremented in the register is used in the form of an effective address

Ex: MOV R0, (R2)+

⑪ Auto Decrement Addressing mode:

The contents of a register specified in the instruction are decremented and then they are used as an effective address to access a memory location

Ex:

MOV -(R0), R1

⑫ Stack Addressing mode:

Stack pointer always contains the address of (Top of stack) where the operand is to be stored, thus, the address of the operand is the contents of stack pointer

EX: PUSH R.

M T W T F S S

Page No.: YOUVA

Date:

Assembly language and high level language

The interaction between assembly and high level

High level Languages (python, C, C++)

*) These are user friendly
*) one-many, each statement of HLL represents many machine codes
*) HLL programs are faster than Assembly, but execution is slow, since all will be translated to machine codes
*) HLL statements are converted into machine codes in compiler or interpreter

Assembly language

*) Its a low-level language
*) Not user friendly and hard to code
*) It is faster in execution, slower in construction
*) Assembly languages are converted into machine codes by an assembler

Compiler:	Interpreter
*) It executes code overall	It executes line by line
*) Faster execution speed	Slower
*) Ex: C, C++	python