UNIT - III

COMPUTER FUNDAMENTALS

Introduction :-

Computer architecture acts as the interface between the hardware and the lowest level of software.

Computer Architecture refers to

* Attributes of a system visible to programmers like data types of variables.
* Attributes that have a direct impact on the execution of programs like clock cycle.

Computer Architecture is defined as study of the structure, behaviour and design of computers.

Computer Organisation :-

It refers to the operational units and their interconnections that realize the architecture specifications. It describes the function of and design of the various units of digital computers that store and process information.

The attributes in computer organisation refers to

→ control signals, memory technology & peripheral interface
→ Data representation
→ Ilo Mechanisms
→ Addressing Techniques.

Computer Architecture :-

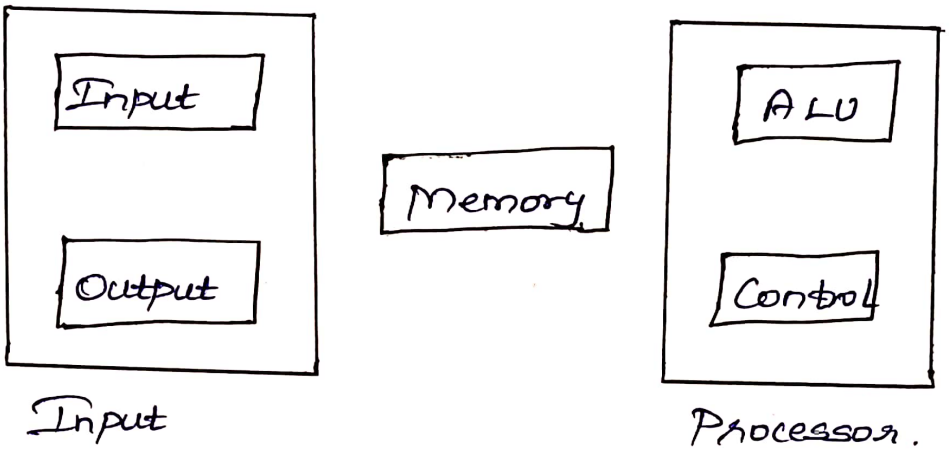
It is concerned with the structure and behaviour of the computer. It includes the information formats, the instruction set and techniques for addressing memory.

Architecture → Visible to programmers

Organization → Implemented features in the system.

Basics of a computer system :-

↳ Von Neumann Architecture :-



Input Unit :-

- Computer accepts the coded information through input unit.
- Input devices accept data and instructions from the user or from another computer system (such as a computer on the Internet).
- The most common input device is the keyboard, which accepts letters, numbers and commands from the user.
- Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over cable to the memory of the computer.
- Another important type of input device is the mouse which lets you select options from on-screen menus.
- A mouse by moving it across a flat surface and pressing its button.

Memory Unit :-

→ Memory unit is used to store programs and data.

→ Memory is classified into primary and secondary storage.

Primary Storage :-

→ It is also called main memory.

→ It operates at high speed and it is expensive.

→ It is made up of large number of semiconductor cells, each capable of storing one bit information.

→ These cells are grouped together in a fixed size called word. This facilitates reading and writing the content of one word.

→ Each word is associated with a distinct address that identifies word location. A given word is accessed by specifying its address.

Word length :-

→ The number of bits in each word is called word length of the computer.

→ The word lengths ranges from 16 to 64 bits. Programs must reside in the primary memory during execution.

RAM :- [Random Access Memory]

→ Memory in which any location can be reached in a short and fixed amount of time by specifying its address is called random-access memory.

Memory Access time :-

→ Time required to access one word is called memory access time.

→ This time is fixed and independent of the word being accessed.

→ It typically ranges from few nano seconds (ns) to about 100ns.

Caches :-

→ They are small and fast RAM units.
→ They are tightly coupled with the processor.
→ They are often contained on the same integrated circuits (IC) chip to achieve high performance

Secondary stage :-

→ Secondary memory is used 't' when large volumes of data programs have to be stored.
→ It is cheaper than primary memory and it's capacity is high.
→ Various secondary devices are magnetic tapes and disks, optical disks (CD-Roms) floppy etc....

Arithmetic and logic Unit :-

→ Most computer operations are executed in ALU.
→ The arithmetic logic section performs arithmetic operations such as addition, subtraction, multiplication and division.
→ A single processor to control a number of external devices such as keyboards, displays, magnetic and optical disks, sensors and mechanical controllers.

Control Unit :-

→ Control unit co-ordinates the operation of memory, arithmetic and logic unit, input unit and output unit in some proper way.
→ Control unit sends control signals to other units and senses their states.

→ The actual timing signals that govern the transfer are generated by the control circuits.

→ The data transfer between the processor and memory are also controlled by the control unit through timing signals.

UNIPROCESSORS TO MULTIPROCESSORS :-

The performance of the computer has drastically increased when the technology has drifted from uniprocessor systems to multiprocessor systems.

* As the core computing units were made more powerful, the performance of the processors also increased significantly.

↳ uniprocessor system is a type of architecture that is based on a single computing unit. All the operations were done sequentially on the same unit. Multiprocessor systems are based on executing

Instructions on multiple computing units.

* The multiprocessor architecture is based on Flynn taxonomy

"Flynn's taxonomy is a classification of parallel computer architecture that are based on the number of concurrent instruction and data streams available in the architecture".

SISD → Single Instruction, Single data
MISD → Multiple Instruction, Single data
SIMD → Single Instruction Multiple data
MIMD → Multiple Instruction Multiple data

Instruction Stream \ Data Stream	Single	Multiple
Single	Single Instruction Single data SISD	Single instruction Multiple data SIMD
Multiple	Multiple Instruction Single data MISD	Multiple Instruction Multiple data MIMD

SINGLE INSTRUCTION AND SINGLE DATA :-

* This is a uniprocessor machine which is capable of executing a single instruction operating on a single data stream.

* The machine instructions are processed in a sequential manner and computers adopting this model are popularly called Sequential computers.

* most conventional computers have SISD architecture

* All the instructions and data to be processed have to be stored in primary memory.

* The speed of the processing element in the SISD model is limited by the rate at which the computer can transfer information internally.

MULTIPLE INSTRUCTION, SINGLE DATA (MISD)

* An MISD computing system is a multiprocessor machine capable of executing different instruction on different processing elements but all of them operating on the same dataset.

SINGLE INSTRUCTION, MULTIPLE DATA (SIMD)

* This machine capable of executing the same instruction on all the CPUs but operating on different data streams.

* machines based on an SIMD model are well suited to scientific computing since they involve lots of vector and Matrix operations. So that the information can be passed to all the processing Elements (PEs) organized data elements of vectors can be divided into Multiple sets and each PE can process one data set.

MULTIPLE INSTRUCTION, MULTIPLE DATA (MIMD)

*This is capable of executing Multiple Instructions on multiple data sets.

* Each PE in the MIMD model has separate instructions and data streams; therefore machine built using this model are capable to any kind of application

* unlike SIMD and MISD Machine, PEs in MIMD Machines work asynchronously.

INSTRUCTIONS :-

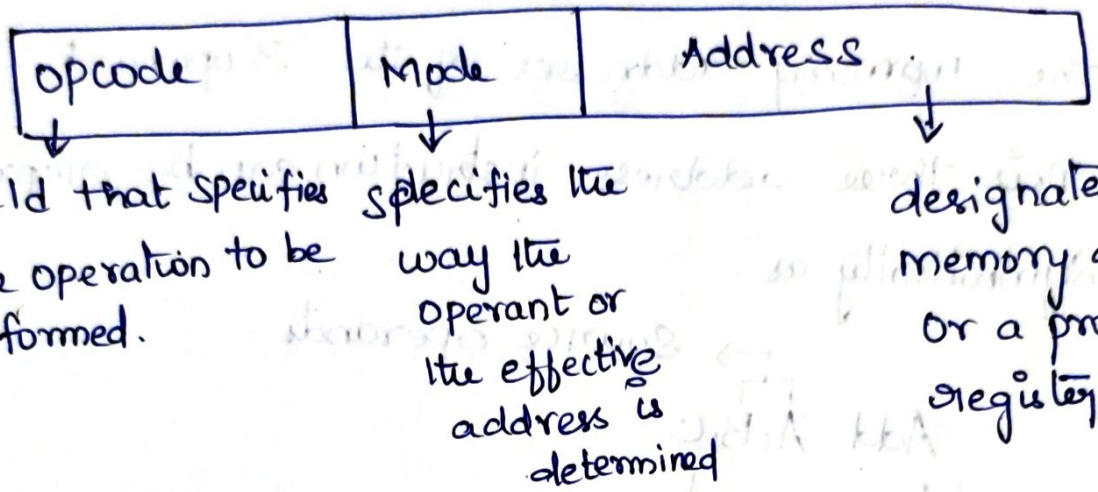
"An Instruction is a binary code, which Specifies a basic operation for the computer".

* A computer hardware must speak its language. The words of a computer language is called instructions. and its vocabulary is called an instruction set.

Operation code [op-code] defines the operation type. operands define the operation of source and destination.

Instruction Set Architecture [ISA] :- describes the processor interms of what the assembly

language programmer sees ; ie; the instructions and registers.



Types of operations :-

* Data transfer between the memory and the processor registers.
* Arithmetic and logic operations on data.
* program sequencing and control.
* I/o transfer.

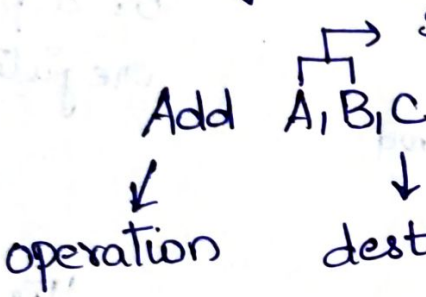
Performing a basic instruction is represented in many ways : They are

- 3-address instruction
- 2-address instruction
- 1- address instruction
- 0 - address instruction.

3- ADDRESS INSTRUCTION :-

Let us first assume that this action is to be

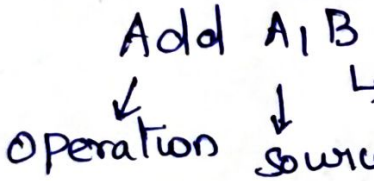
accomplished by a single machine instruction. Furthermore, assume that this instructions contains the memory addresses of the 3 operands - A, B and C. This three address instruction can be represented symbolically as.



The general instruction format will be of this type

operation Destination Source 1, Source 2.

2-address Instruction :-



which performs the operation $B \leftarrow [A] + [B]$. when the sum is calculated the result is sent to the memory and stored in location B, replacing the original content of this location. This means that operand B is both a source and a destination.

1 Address Instruction

Add A

* means → add the contents of memory location A to the contents of the accumulator register and place the sum back into the accumulator.

↳ Load A.
↳ and
↳ Store A.

Zero address Instruction :-

In these Instructions, the locations of all operands are defined implicitly. Such instruction are found in machines that store operands in a structure are called a pushdown stack.

Logical Instructions :-

AND, OR, XOR, Shift

ARITHMETIC INSTRUCTIONS :-

* Data types
* Integers : unsigned, Signed, Byte, short, long.
* Real numbers : Single precision (float), Double precision (double) operations
* Addition, subtraction, Multiplication, Devision.

DATA TRASFER INSTRUCTIONS :-

* Register transfer : MOVE
* memory transfer : Load, store
* I/o transfer : In, out
* control transfer instructions.
* unconditional branch.
* conditional branch
* procedure call
* Return.

OPERATIONS AND OPERANDS.

OPERATIONS OF THE COMPUTER HARDWARE :-

* Every computer must be able to perform arithmetic. The MIPS assessble to assembly language notation

```
add a, b, c
```

Instructs a computer to add the two variables b and c and to put their sum in 'a'.

Each MIPS arithmetic instruction performs only one operation and must always have exactly three variables.

* For example suppose want to place the sum of four variables b, c, d and e into variable 'a

The following sequence of instructions adds the four variables:

add a, b, c : The sum of 'b' and 'c' is placed in 'a'

add a, a, d : The sum of 'b', 'c' and 'd' is now in 'a'

add a, a, e : The sum of b, c, d and e is now in 'a'

Thus, it takes three instructions to sum the the four variables. The words to the right of the Sharp symbol (;) on each line above are comments for the human reader and the computer ignores them.

OPERANDS :-

↳ operand is a variable used to perform any kind of operations.

↳ operand must be from a limited number of Special locations, build directly in hardware called register.

↳ Registers are primitives used in a hardware design that also visible to the programmer when the computer is completed.

↳ MIPS architecture has 32 bits registers and group of 32 bits are called 'word'.

Word : The Natural unit to access in a computer, usually a group of 32 bits; corresponds to the size of a register in the MIPS architecture.

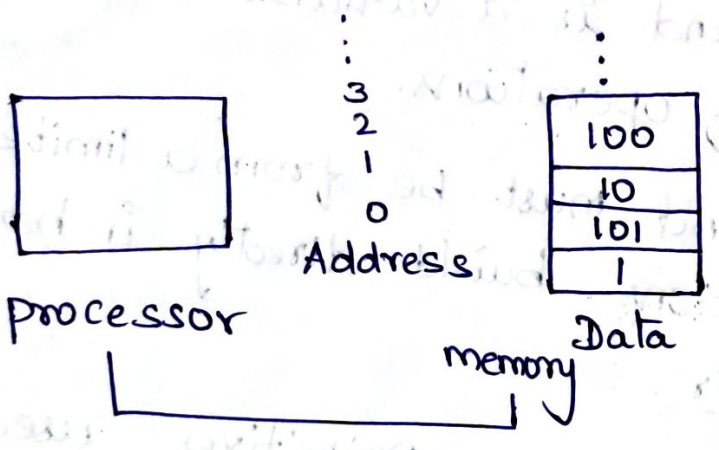
(i) MEMORY OPERANDS :-

programming languages have

* Simple variable that contain single data element.

* Complex data structures – array and structure

These complex data structures can contain many more data elements than there are registers in a computer.



Data transfer instructions :-

* Arithmetic operations occurs only on register in MIPS instructions.

* MIPS must include instructions that transfer data between memory and register. Such

instructions are called data transfer instructions.

* To access a word in memory, the instruction must supply the memory address.

memory :-

* memory is a large size and single dimensional array.

* The address acting as the index to that array starting at 0.

Load :-

* The data transfer instruction that copies data from memory to a register is traditionally called load.

* The format of the load instruction is the name of the operation followed by the register to be loaded, then a constant and register used to access memory.

MIPS Address (LOAD)

* The sum of the constant portion of the instruction and the content of the second register forms the memory address.

* The actual MIPS name for this instruction is lw, standing for load word.

Store :-

* The instruction complementary to load is called Store.

* It copies data from a register to memory.

* The format of a store is the name of the operation, followed by the register to be stored and then offset to select the array element and finally the base register.

* The MIPS address is specified in part by a constant and in part by the contents of a register. The actual MIPS name is SW standing for store word.

Name	Example	Comments
32 Register	$s0, $s7 $t0, $t9 $zero, $a0 - $a3 $v0 - $v1	They can be accessed quickly, In MIPS architecture, the data must be loaded into the register to perform arithmetic operation
2^{30} memory words	Memory[0] Memory[i]	The contents can be accessed only after data transfer instruction. MIPS use byte addressing.

Category	Instruction	operation
Arithmetic	Add $S1, $S2, $S3	$S_1 = S_2 + S_3$. There are three operands in this instruction. The data resides in the register.
	Sub $S1, $S2, $S3	$S_1 = S_2 - S_3$ There are 3 operands in this register.
	Add $S1, $S2, 50	$S_1 = S_2 + 50$ This is add immediate instruction. It has two operands and one constant value.
Data transfer	Lw $S1, 50($S2)	S_1 = memory [S_2 + 50] Data is transferred from memory to regis
	Sw $S1, 50($S2)	memory [S_2 + 50] = S_1 Data is transferred from register to memory.

CONSTANT OR IMMEDIATE OPERANDS :-

↳ Constant variables are used as one of the operand for many arithmetic operations in MIPS architecture.

↳ Many times a program will use a constant in an operation. For example incrementing an index to point to the next element of an array.

Eg : add I $S3, $S3, 10

This instruction is interpreted as addition of content of $S3 and the value 10. The sum is stored in $S3. Add I means add Immediate, since one of the operant is in Immediate addressing mode.

* As per the design principle "make common case faster" the constant operands must be loaded faster from the memory.

* Since constants occur more frequently in the instruction, they are mentioned in the instruction itself rather than to load from register.

REPRESENTATION OF INSTRUCTION

Signed and unsigned Numbers:-

* A single digit of a binary number is thus the 'atom' of computing, since all information is composed of binary digits or bits.

* This fundamental building block can be one or two values which can be thought of as several alternatives : high or low, on or OFF, true or false or '1' or '0'.

Generalizing the point, in any number base, the value of ith digit 'd' is

$$d \times Base^{i}$$

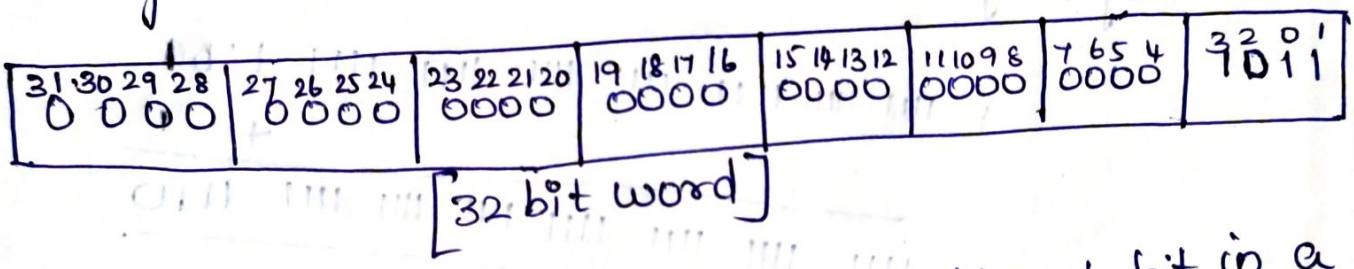
where i starts at 0 and increases from right to left.

For example 1011_2 represents

$= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$

$= 8 + 0 + 2 + 2$

$= (11)_{10}$ we number the bits 0, 1, 2, 3.... from right to left in a word.



[32 bit word]

LSB [Least Significant bit] : The Right most bit in a MIPS word [Bit '0']

MSB [Most Significant bit] : The Left most bit in a MIPS word [Bit 31].

The MIPS word is 32 bits long. So we can represent 2^{32} different 32 bit pattern. It is natural to let these combinations represent the numbers from

0 to 2^{32}

* lb [load byte] sign-extends to fill the 24 left most bits of the register.

* lh (load half) sign extends to fill the 16 left-most bits of the register.

* lbu (load byte unsigned) and

* lhu (load half-word unsigned) for unsigned integers.

Problem:-

Negate 2_{ten} and then check the result by negating -2_{ten}.

2_{ten} = 0000 0000 0000 0000 0000 0000 0000 0010_{ten}.

Negating this number by inverting the bits and adding one.

```
  1111 1111 1111 1111 1111 1111 1111 1101
+                                       1
-----------------------------------------
  1111 1111 1111 1111 1111 1111 1111 1110
```

$\Rightarrow -2_{ten}$.

Going the other direction

```
  1111 1111 1111 1111 1111 1111 1111 1110
```

First inversion

```
  0000 0000 0000 0000 0000 0000 0000 0001
+                                       1
-----------------------------------------
  0000 0000 0000 0000 0000 0000 0000 0010
```

$\Rightarrow 2_{ten}$

In MIPS language,

↳ Register $S0 to $S7 map onto register 16 to 23.

↳ Register $t0 to $t7 map onto register 8 to 15.

Hence $S0 means register 16

↳ $S1 means register 17.

↳ $S2 means register 18

↳ $t0 means register 8.

↳ $t1 means register 9 and so on.

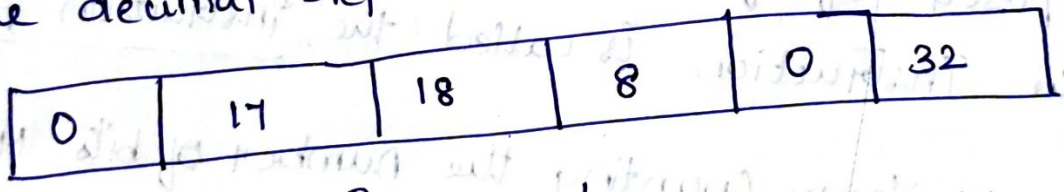
Examples: Translating a MIPS Assembly Instruction into a machine Instruction.

* The real MIPS language version of the instruction represented symbolically as

add $t0, $S1, $S2

First as a combination of decimal numbers and then a binary numbers.

The decimal representation is



from the about segment.

* Each of these segment of an instruction is called a Field.

* The First and last fields containing '0' and

* The second field gives the number of the register that is the first source operand of t

addition operation (17 = $S1)

* The third field gives the other source operand for the addition (18 = $S2)

* The fourth field contains the number of the register that is to receive the sum (8 = $t0)

* The fifth field is unused in this instruction, so it is set to '0'. Thus this instruction add register $S1 to register $S2 and places the sum in register $t0.

This instruction can also be represented as fields of binary numbers as opposed to decimal.

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits.

INSTRUCTION FORMAT :-

* A form of representation of an instruction composed of fields of binary number. This layout of the instruction is called the instruction format.

* From counting the number of bits, MIPS instruction takes exactly 32 bits - the same size as a data word.

* our design principle that simplicity favors regularity, all MIPS instructions are 32 bit long.

Machine language :-

Binary representation used for communication within a computer system.

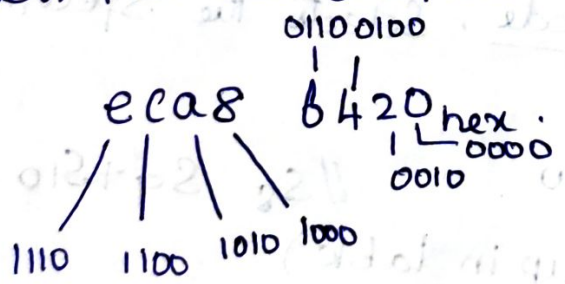
Hexadecimal :-

All computer data sizes are multiples of 4 hexadecimal (base 16) numbers are popular. Convert by replacing each group of four binary digits by a single hexadecimal digit and vice versa.

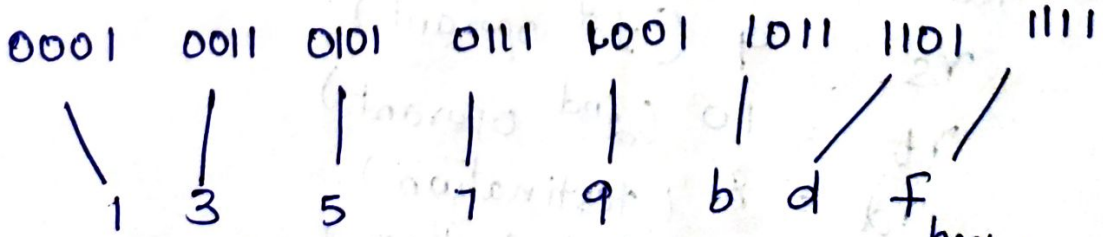
Hexadecimal	binary				
0 hex	0000	7 hex	0111	13	1101
1 hex	0001	8 hex	1000	14	1110
2 hex	0010	9 hex	1001	15	1111
3 hex	0011	10 hex	1010		
4 hex	0100	11 hex	1011		
5 hex	0101	12 hex	1100		
6 hex	0110				

Example : Binary to hexadecimal and Back.

Convert the following hexadecimal and binary numbers into the other base.



and then the other direction.



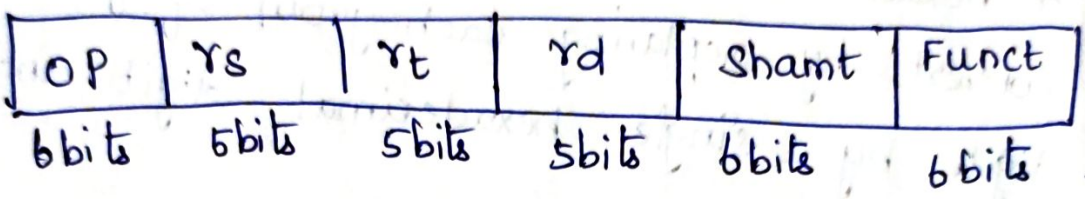
MIPS FORMAT :-

MIPS Fields has two kinds of format such as

(i) R-type (or) R Format (for register)

(ii) I-type (or) I-Format (for immediate).

(i) R-FORMAT



OP ⇒ Basic operation of the instruction, traditionally called op code.

rs ⇒ The first register source operand

rt ⇒ The second register source operand

rd ⇒ The register distination operand. It gets the result of the operation.

Shamt ⇒ Shift amount.

funct ⇒ function. This field, often called the function function code, selects the specific variant

Example

add $8, $9, $10 // $S_8 = S_9 + S_{10}$

opcode = 0 (look up in table)

funct = 32 (look up in table)

rs = 9 (1st operant)

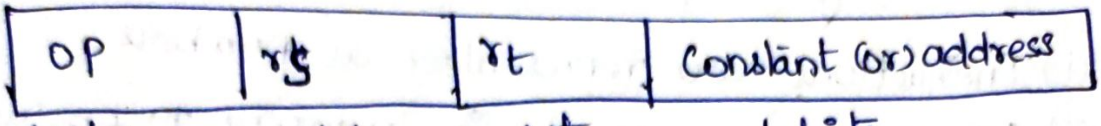
rt = 10 (2nd operant)

rd = 8 (destination)

Shamt = 0 (not a Shift)

I-Format :-

The fields of i-format are



* The 16 bit address means a load word instruction can load any word within region of $\pm 2^{15}$ or 32,768 bytes ($\pm 2^{13}$ or 8192 words) of the address in the base register rs.

* Similarly, add immediate is limited to constant no larger than $\pm 2^{15}$.

I-FORMAT Example :-

MIPS Instruction : lw $to, 1200($t1)

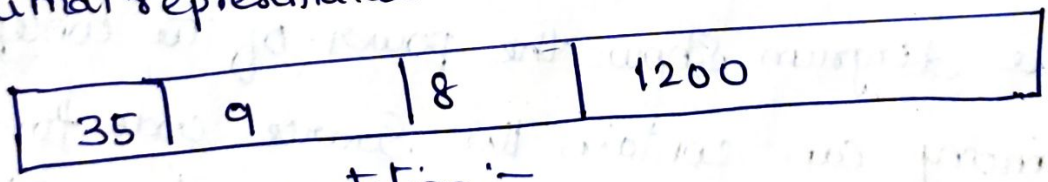
Opcode = 35 (look up in table)

rs = 9 (base register)

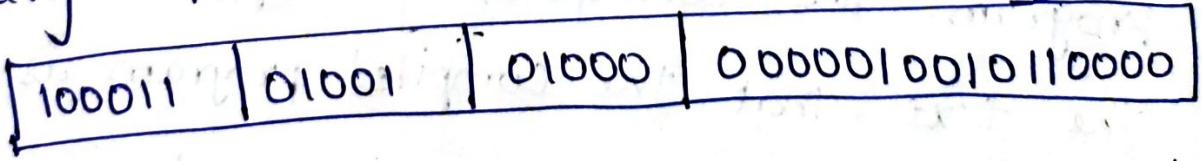
rt = 8 (destination register).

immediate = 1200 (offset).

decimal representation



binary representation :-



Note :- The meaning of the rt field has changed for this instruction : in a load coord instruction.

The rt field specifies the destination register which receives the result of load.

Stored-program Concept :-

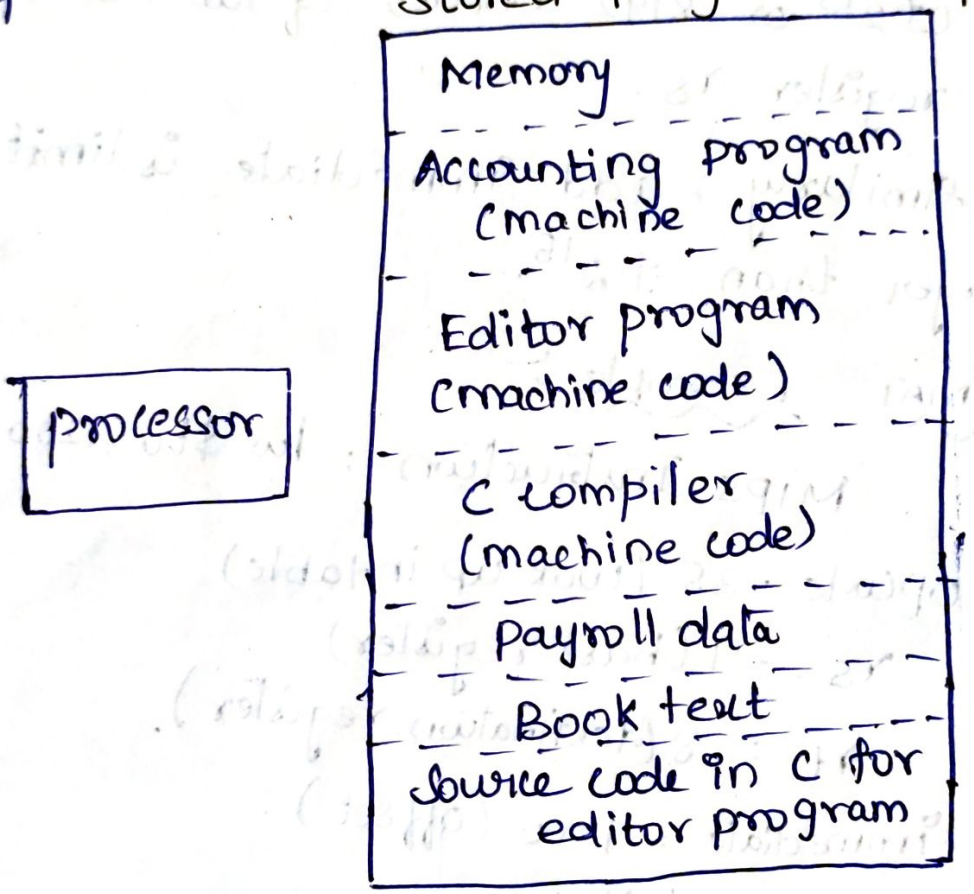
Today's computers are built on two key principles -

(i) Instructions are represented as numbers.

(ii) programs are stored in memory to be read or written, just like data.

↳ These principles lead to the stored-program concept.

Stored Program Concept.



* The diagram shows the power of the concept; Specifically memory can contain the source code for an editor program, the corresponding compiled machine code the text that the compiled program is using and even the compiler that generated the machine code.

* One consequence of instructions as number is that programs are often shipped as files of binary numbers.

* The commercial implications is that computers can inherit ready-made software, provided they are compatible with an existing instruction set.

* Such binary compatibility often leads industry to align around a small number of instructions set architecture.

LOGICAL OPERATION

* The first computer operated on full words. It was useful to operate on fields of bits within a word or even on individual bits.

* Examining characters within a word, each of which is stored as 8 bits.

* The programming languages and instruction set architecture to simplify, the packing and unpacking of bits into words. These instructions are called logical operations.

Logical operation	C operator	Java operator	MIPS instruction
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit by bit AND	&	&	and, andi

Logical operation	C-operator	Java operator	MIPS Instruction
Bit by bit OR	|	|	or, ori
Bit by bit NOT	~	~	nor

* The first class of such operations is called shifts
* They move all the bits in a word to the left or right, filling the emptied bits with '0's.

Shift

Left shift
Left shifting by i bit is equivalent to multiplying the number by 2^i

Right shift
Right shifting by i bit is equivalent to dividing the number by 2^i.

Category	Instruction	operation
AND	and $\$S_1, \$S_2, \$S_3$	$S_1 = S_2 \& S_3$
OR	or $\$S_1, \$S_2, \$S_3$	$S_1 = S_2 \mid S_3$
NOR	nor $\$S_1, \$S_2, \$S_3$	$S_1 = \sim(S_2 \mid S_3)$
NAND	nand $\$S_1, \$S_2, \$S_3$	$S_1 = \sim(S_2 \& S_3)$
ADD Immediate	andi $\$S_1, \$S_2, 100$	$S_1 = S_2 + 100$
OR immediate	ori $\$S_1, \$S_2, 100$	$S_1 = S_2 \mid 100$
Shift left logical	sll $\$S_1, \$S_2, 10$	$S_1 = S_2 << 10$
Shift right logical	srl $\$S_1, \$S_2, 10$	$S_1 = S_2 >> 10$

CONTROL OPERATION :-

Decision making and branching makes the computer more powerful.

Decision making :- Decision making in MIPS assembly language includes two decision-making instructions.

conditional branches

- Branch if Equal (BEQ)
  - Statement : Beq register 1, register 2, L1
  - In this instruction, go to the statement labeled L1 if the value in register 1 is equal to the value of register 2.
- Branch if not Equal (BNE).
  - Eg : bne register 1, register 2, L1
  - In this instruction, go to the statement labeled L1 if the value of register 1 does not equal to value of register 2.

Conditional branch is an instruction that requires the comparison of two values and that allows for a subsequent transfer of control to a new address in the program based on the outcome of the comparison".

Loops :- Looping statement is used to execute the same task more than one time until certain condition gets failed.

Ex. Compiling a while loop in C.

```
while (save[i] == k)
    i += [1;
```

BASIC BLOCK

* A basic block is a sequence of instruction without branches and without branch targes or branch labels.

* one of the first early phases of compilation is breaking the problem into basic blocks.

* It is useful to test for equality and inequality is probably the most popular test.

* It is useful to see if a variable is less than another variable.

For example, a for loop may want to test to see if the index variable is less than '0'.

ADDRESSING MODE :-

→ The different ways in which the operants of an instruction are specified are called as addressing modes.

→ multiple forms of addressing are generally called addressing modes.

The addressing modes are the following.

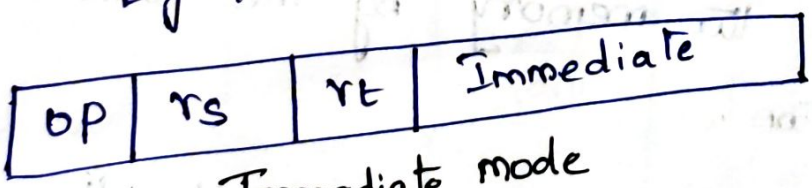
IMMEDIATE ADDRESSING :-

* where the operant is a constant within the instruction itself.

* has the advantage of not requiring an extra memory access to fetch the operant, hence will be executed faster, However the size of operand is limited to 16 bits.

* The Jump instruction format can also be considered as an example of immediate addressing

Eg : ADD 3 [Add 3 to contents of accumulator and 3 is operand]



Immediate mode

* DIRECT ADDRESSING :-

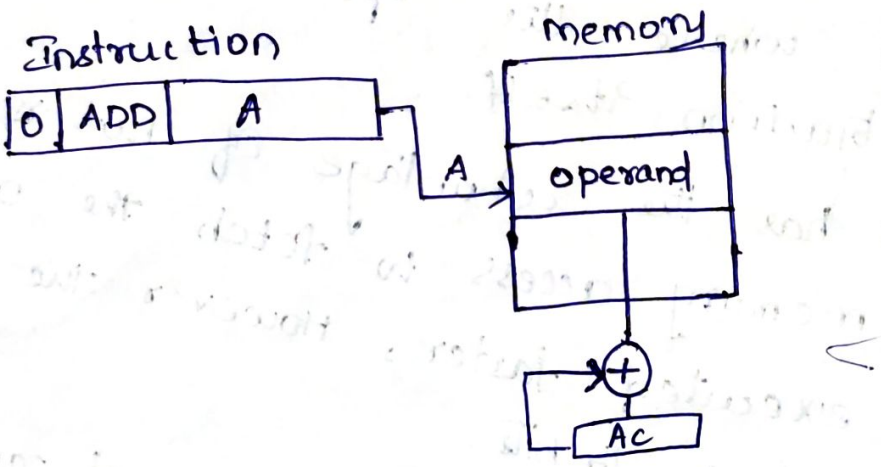
* In direct addressing mode, effective address

of the operant is given in the address field of the instruction.

* It requires one memory reference to read the operand from the given location and provides only a limited address space.

* Length of the address field is usually less than the word length.

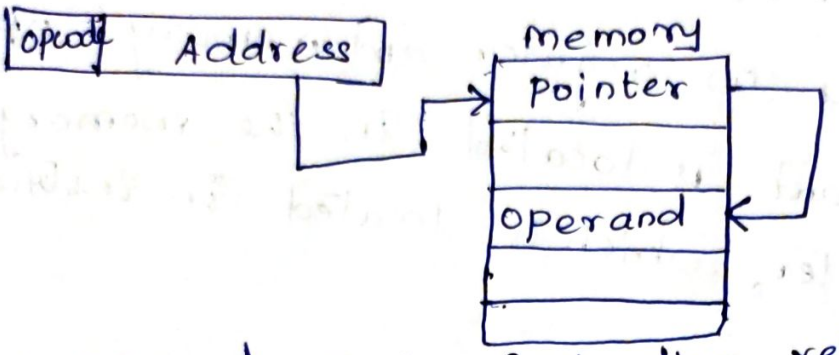
Example : move (P, Q) R0 → operants
Add (P, Q) R0 → Register.



* INDIRECT OR PSEUDO DIRECT ADDRESSING :-

* In the mode, the offset value is specified indirectly into the memory by the pointer available in the instruction.

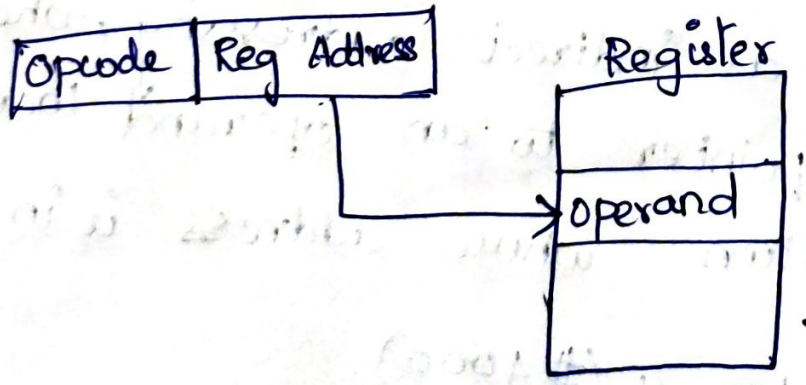
* The address field of the instruction contains the effective address.



* The operand of the instruction refers the control into the memory location indirectly by the pointed present inside the memory. Hence the pointer was direct to the operand.

REGISTER ADDRESSING :-

* It is the simplest addressing modes of all
* where the operand is a register.
* works must faster than other addressing modes because it does not involves with memory accesses.
* The register address is specified as a part of instruction.



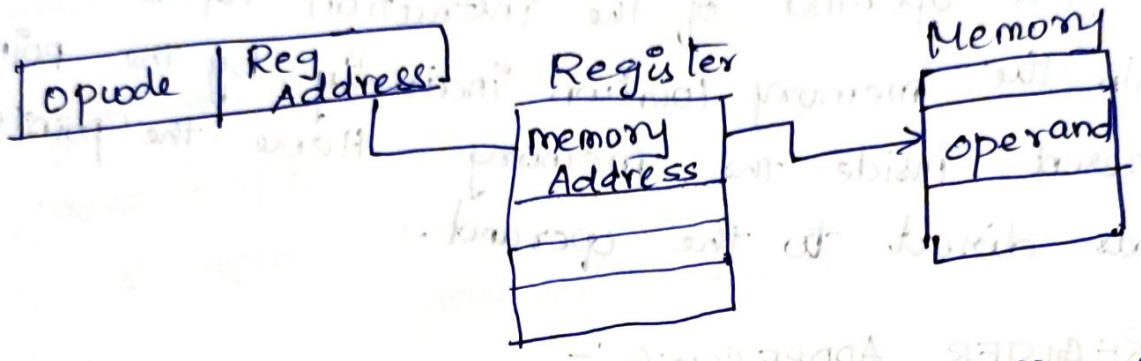
→ operand is directly obtained by the register which is present in the instruction.

E.g: ADD R_1, R_2, R_3.

(5) Register Indirect Addressing :-

↳ works on register and memory operands.

↳ operand is located in the memory pointed by the register, which is located in instruction



↳ The operand is obtained by the memory address which is located in register by searching that memory location into a memory.

(6) Base or Displacement Addressing :-

* The operand is at the memory location whose address is the sum of register and a constant in the instruction.

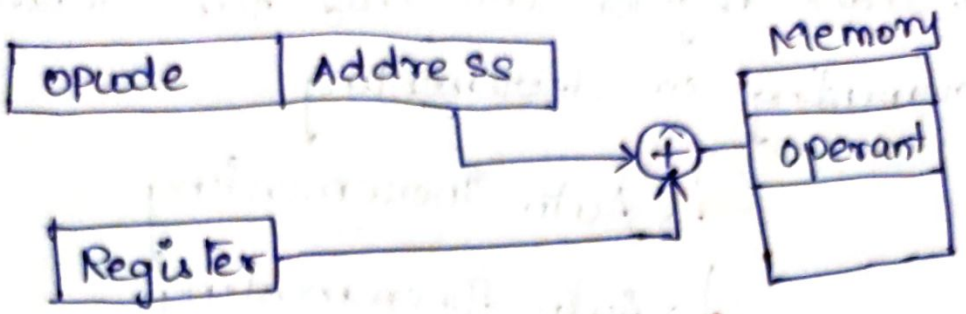
* Also known as indirect addressing, where register acts a pointer to an operand located at the memory location whose address is in the register

Ex:- LW R0, 4 (#4000)

R0 → Rs

4 → OFF set value

4000 → Memory address.

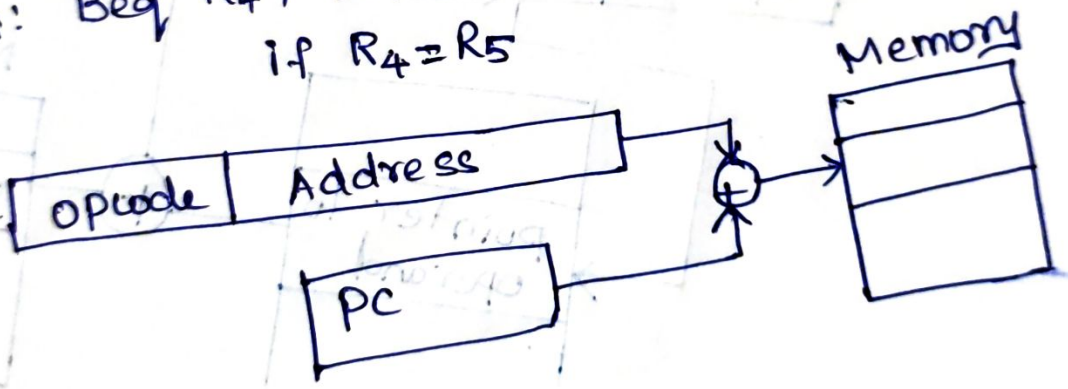


⑦ PC- Relative Addressing :-

* The branch address is the sum of the pc and a constant in the instruction.

* It is also known as program Counter addressing is a data or instruction memory location is specified as an offset value to the incremented.

Ex: Beq R_4, R_5, label.
if $R_4 = R_5$



⑧ INDEXing Addressing :-

* The address field refers to a main memory address and the oref reg contains a positive displacement from the address.

* The oreg refers sometimes explit and implict.

* Index register are used for iterative task for incrementing or decrementing.

↳ Auto Incrementing

↳ Auto Decrementing.

Ex:- Auto Incrementing :-

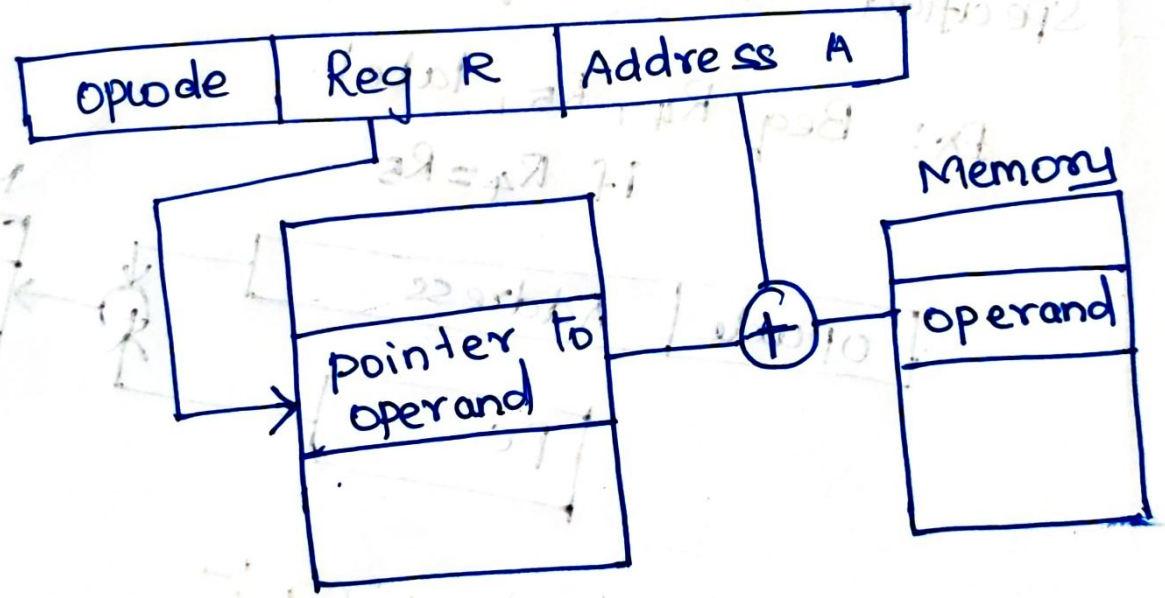
Effective Address = A+(R)

R = [R]+1

Auto decrementing :-

EA = A+[R]

R = [R]-1



Encoding of Machine Instructions :-

→ To be executed in a processor, an instruction must be encoded in a binary pattern. Such encoded instructions are referred to as Machine instructions.

→ The instructions that use symbolic-names and acronyms are called assembly language instructions.

→ The instructions that perform operations such as add, subtract, move, shift, rotate and branch may use operands of different sizes such as 32-bit and 8-bit numbers.

→ The instruction Add R1, R2 ; Has to specify the registers R1 and R2, in addition to the OP code. If the processor has 16 registers, then four bits are needed to identify each register. Additional bits are needed to indicate that the Register-addressing mode is used for each operand.

→ The instruction Move 24(R0), R5 ; Requires 16 bits to denote the OP code and the two registers, and some bits to express that the source operand uses the Index addressing mode and that the index value is 24.

In all these examples, the instructions can be encoded in a 32-bit word.

→ The OP code for given instruction refers to type of operation that is to be performed.

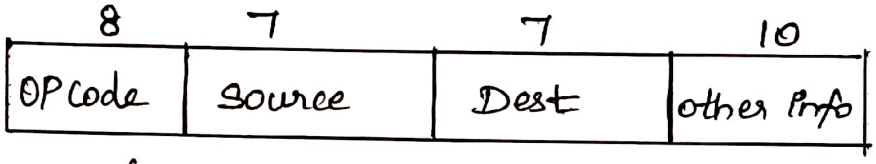
→ Source and destination field refers to source and destination operand respectively.

→ The "other info" field allows us to specify the additional information that may be needed such as an index value or an immediate operand.

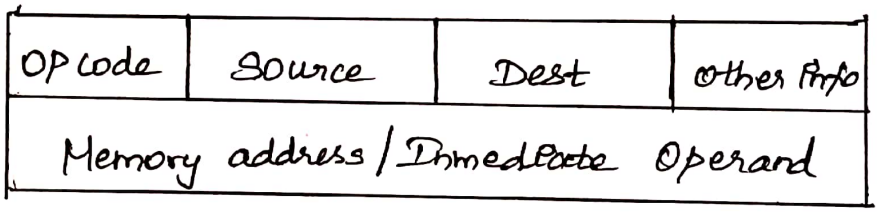
→ Using multiple words, we can implement complex instructions, closely resembling operations in high level programming languages. The term complex instruction set computers (CISC) refers to processors that use.

→ CISC approach results in instructions of variable length, dependent on the number of operands and the type of addressing modes used.

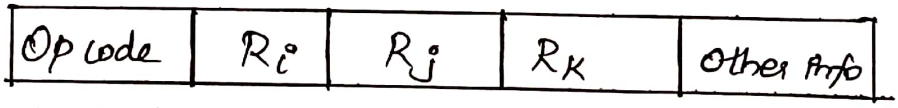
→ In RISC (Reduced instruction set computers), any instruction occupies only one word.



(a) one-word instruction



(b) Two-word Instruction



(c) Three-operand instruction.

(Fig) Encoding instructions into 32-bit words.

→ The RISC approach introduced other restrictions such as that all manipulation of data must be done on operands that are already in registers.

(Eg): Add R_1, R_2, R_3.

→ In RISC type machine, the memory references are limited to only Load/Store operations.

Assembly Level Language :-

It is a low-level language that allows users to write a program using alphanumeric mnemonic codes, instead of numeric code for a set of instructions. Examples of a large assembly language of the present time is IBM PC DOS.

High Level Language :-

It is a machine-independent type of language. It let users write various programs in such a language that resembles the English words (and alphabets) and all the familiar mathematical symbols. The very first high-level language was the COBOL language. C #, Python, etc -- are a few examples of high-level languages.

Difference between Assembly language and high-level language :-

Assembly Language	High Level Language.
1.) The assembley language requires an assembler for the process of conversion.	1.) High level language requires an interpreter/ compiler for the process of conversion.
2.) We perform the conversion of an assembly language into a machine language.	2.) We perform the conversion of a high-level language into an assembly language and then into a machine level languag
3.) It is a machine-dependent type of language.	3.) It is a machine-independen type of language.
4.) It makes use of the mnemonic codes for operation.	4.) It makes use of the English statements for operation.
5.) It provides support for various low-level operations.	5.) It doesnot provide any support for low-level languages.
6.) The code is more compact in this case.	6) No code compactness is present in this case.

7.) Accessing the hardware component is very easy in this case.	7.) Accessing the hardware component is very difficult in this case.
8.) It is Processor-dependent.	8.) It is processor-independent
9.) It has better accuracy.	9.) Accuracy is much lesser in this case.
10.) An assembly language performs better than any high-level language, in general.	10.) The performance is comparatively not so good.
11.) It is shorter in assembly language.	11.) It is larger in a high-level language.
12.) Execution of code takes less time in this case because the code is not very large.	12.) It takes up more time for execution because it needs to execute a large code.
13.) It is way more efficient because of the shorter executable codes.	13.) It is comparatively less efficient because the executable codes are comparatively longer in length.
14.) We can do that directly at a physical address in the case of an assembly language. [Reading of pointers]	14.) It is not possible to do so in the case of a high-level language.
15.) It is very difficult to debug and understand the code of an assembly language.	15.) It is very easy to debug and understand the code of an assembly language.