QUESTIONS:

1. Flag registers kn kn s hty emu8086 mein

ANSWER:

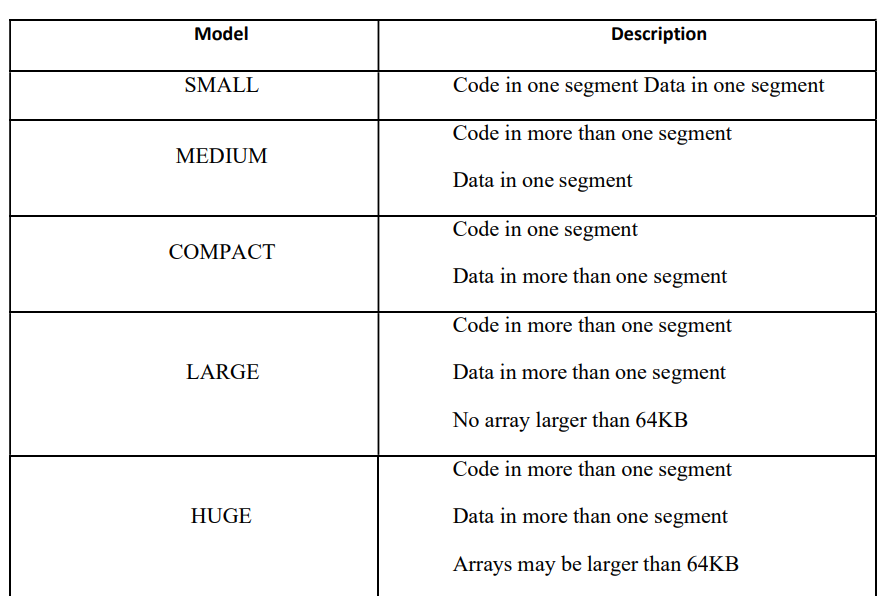
1. **Status Flags**
   1. Sign Flag (S)
   2. Zero Flag (Z)
   3. Auxiliary Cary Flag (AC)
   4. Parity Flag (P)
   5. Carry Flag (CY)
   6. Overflow Flag(0)
2. Control Flags
   1. Directional Flag
   2. Interrupt Flag
   3. Trap Flag
3. Emu8086 m hm kya dekh skty hain

ANSWER: Registers, flags and memory

1. Segment Registers kya hty .. kn kn s hty

ANSWER: A register that points to the base of the current segment being addressed.

* + - .CODE (The code segment contains a program‟s instructions)
    - .DATA (contains all the variable definitions)
    - .MODEL (The size of code and data in a program can have determined by specifying a memory model using the .MODEL directive.)
    - .STACK (The purpose of the stack segment declaration is to set aside a block of memory (the stack area) to store the stack.)



1. Architecture kya hta h

ANSWER:

1. Interpreter aur Assembler m farq

ANSWER:

1. Assembler kya krta h

ANSWER: The Assembler is a Software that converts an assembly language code to machine code. It takes basic Computer commands and converts them into Binary Code that Computer's Processor can use to perform its Basic Operations. These instructions are assembler language or assembly language.

1. data segment

ANSWER: In computing, a data segment (often denoted . data) is a portion of an object file or the corresponding address space of a program that contains initialized static variables, that is, global variables and static local variables.

1. word kitny bits ka hai

ANSWER: 16 bits

1. Dw kya hai

ANSWER: define word , define byte(DB),define double word(DD),define ten bytes(DT)

1. emulator per screen per kya atay hain

ANSWER:

1. .Data kya hai

ANSWER: To declare a data segment, we use the directive .DATA

1. stack 100ah mai 100ah kya hai

ANSWER: address

1. 8086 emulator

ANSWER:

1. Segments

ANSWER:

1. define each segment

ANSWER:

1. what does mov keyword do

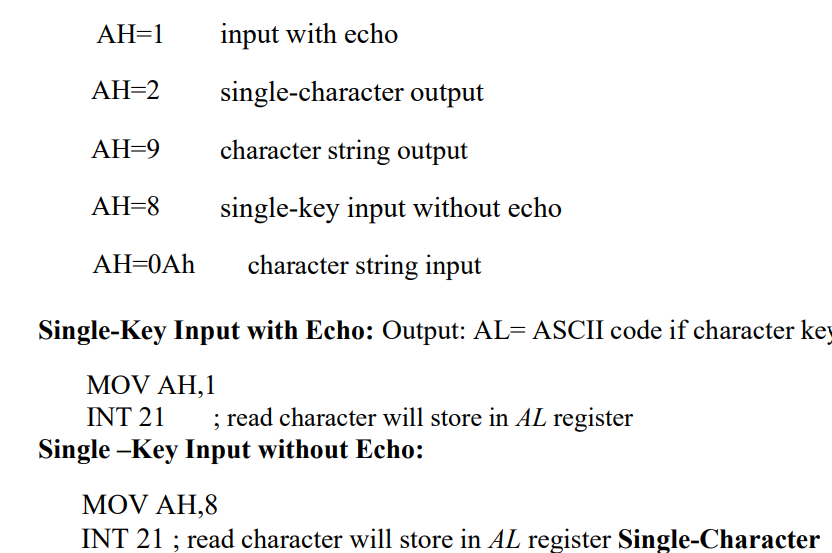
ANSWER: MOV copies the data in the source to the destination.

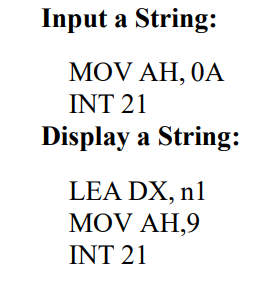
Syntax: Des,source .

this is mnemonics.

1. aur input lene ka syntax

ANSWER:





1. What are directives?

ANSWER: Directives are instructions to the assembler, specifying an action to be taken during the assembly process. Directives can define variables, macros, and procedures.

1. What is INT?

ANSWER: INT is an assembly language instruction for x86 processors that **generates a software interrupt**.

**Instruction Set Architecture (ISA) :** The ISA of a machine is the set of its attributes a system programmer needs to know in order to develop system software or an assembler requires for translation of an Assembly Language (HLL) code into machine language.

* Instruction Set
* Programmer Accessible Registers
* Memory-Processor Interaction
* Addressing modes
* Instruction Formats

1 Nibble = 4 bits

1 Word = 16 bits/2 bytes

1 Byte= 8 bits

**LOOP:** The CX register is used as a counter for loop instructions

**Syntax: loop destination\_label**

**Destination\_Label**: It is the address (or Label ) of the code that we want to execute again and again

**1h**= 1 in hexadecimal

**Int 21h** = This simply means that you are using function 01h of the Interrupt type 21.

**Data types:** int, char , string

**MUL:**

mov ax, 5 ; ax = 5

mov cx, 10 ; cx = 10

mul cx ; ax = ax \* cx ; actually dx:ax = ax \* cx

The result will be stored in ax, which is the implicit destination register.

Well, technically, the result will be stored in dx:ax. This is a *register pair*, and means that the high portion of the result will be stored in dx, while the low portion of the result will be stored in ax.

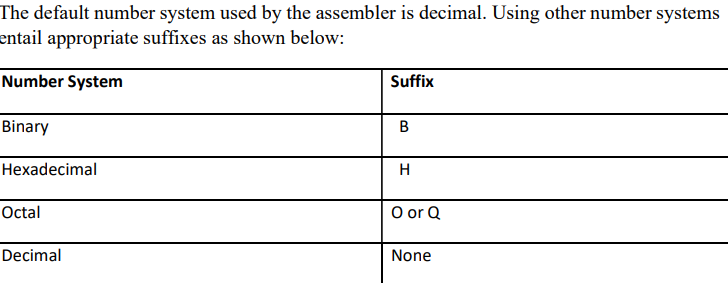
**DIV:**

mov ax, 5 ; ax = 5

mov bl, 10 ; cx = 10

div bl ; ax /bl

**SUFFIX**:



An **assembly language** is a type of **low-level** programming language that is intended to communicate directly with a computer's hardware.

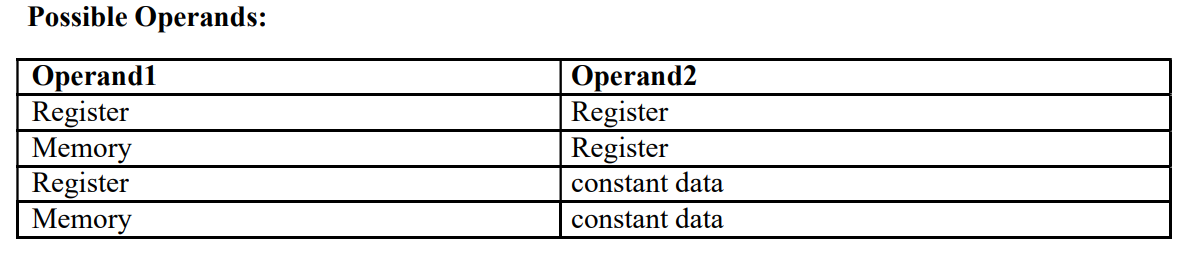
Assembly language consists of statements written with short **mnemonics** such as ADD, MOV, SUB, and CALL.

**Mnemonics**

**Assembly language** instructions use abbreviations called **mnemonics**. An example of a mnemonic assembly language instruction is **LDA 50** which stores the value **50** into a register of the CPU. Mnemonics are easier for humans to remember and understand than binary machine code instructions.

The following is a basic example of how an opcode and operand might look for one line in the machine code of a program:

| **Assembly** | **Mnemonic** | **Binary** | **Hex** |
| --- | --- | --- | --- |
| LOAD 0004 | LDA 0004 | 1010 0100 | A4 |



**Registers**:

* Store Data
* Transfer Data
* Increase the speed

**EMU8086** - MICROPROCESSOR EMULATOR is **a free emulator for multiple platforms**. It provides its user with the ability to emulate old 8086 processors, which were used in Macintosh and Windows computers from the 1980s and early 1990s.

A virtual env.