Practical: 6

Aim: Write an Assembly language program to print the string in 8086.

Theory:

Assembly language is a low-level language that helps to communicate directly with computer hardware. It uses mnemonics to represent the operations that a processor has to do. Which is an intermediate language between high-level languages like C++ and the binary language. It uses hexadecimal and binary values, and it is readable by humans.

How Assembly Language Works?

Assembly languages contain mnemonic codes that specify what the processor should do. The mnemonic code that was written by the programmer was converted into machine language (binary language) for execution. An assembler is used to convert assembly code into machine language. That machine code is stored in an executable file for the sake of execution.

It enables the programmer to communicate directly with the hardware such as registers, memory locations, input/output devices or any other hardware components. Which could help the programmer to directly control hardware components and to manage the resources in an efficient manner.

Components of Assembly Language

- **Registers:** Registers are the fast memory locations situated inside the processor. Which helps ALU to perform arithmetic operations and temporary storing of data. Example: Ax (Accumulator), Bx, Cx.
- **Command:** An instruction in assembly code known as a command informs the assembler what to do. Assembly language instructions typically employ self-descriptive abbreviations to make the vocabulary simple, as “ADD” for addition and “MOV” for data movement.
- **Instructions:** Instructions are the mnemonic codes that we give to the processor to perform specific tasks like LOAD, ADDITION, MOVE. Example: ADD
- **Labels:** It is a symbolic name/identifier given to indicate a particular location or address in the assembly code. Example: FIRST to indicate starting of execution part of code.
- **Mnemonic:** A mnemonic is an acronym for an assembly language instruction or a name given to a machine function. Each mnemonic in assembly corresponds to a specific machine instruction. Add is an illustration of one of these machine commands. CMP, Mul, and Lea are among further instances.
- **Macro:** Macros are the program codes that can be used anywhere in the program through calling it once we define it. And it is often embedded with assemblers and compilers. We should define it using a directive %macro. Example: %macro ADD_TWO_NUMBERS 2
  add eax, %1

add eax, %2
%endmacro

- **Operands:** These are the data or values that we are given through instruction to perform some operation on it. Example: In ADD R1,R2 ; R1 and R2 are operands.
- **Opcode:** These are the mnemonic codes that specify to the processor which operation has to be done. Example: ADD means Addition.

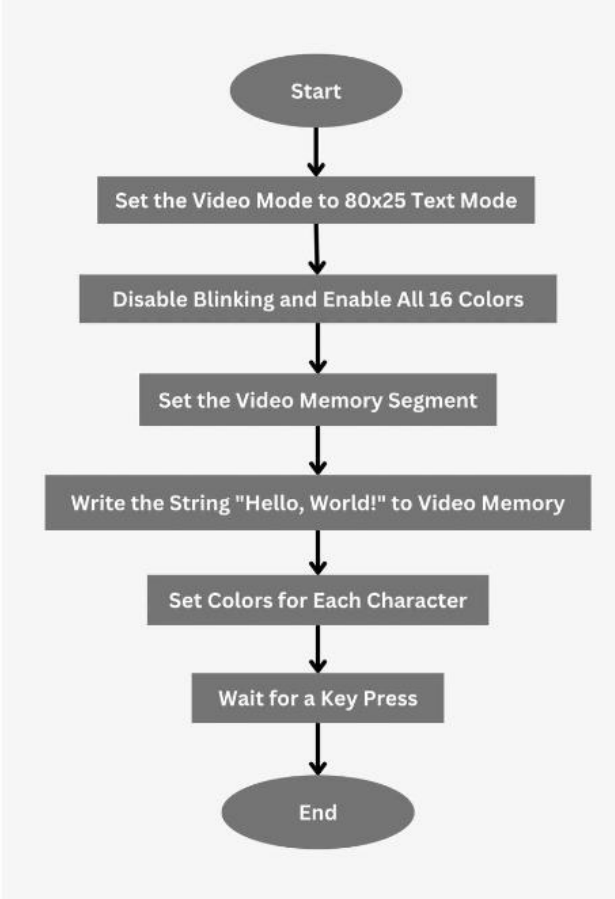
Algorithm Steps:

1. **Start the Program.**
2. **Set the Video Mode to 80x25 Text Mode**:
    - Load the value 3 into register AX to indicate **80x25 text mode**.
    - Use the **BIOS interrupt int 10h** to set the video mode.
3. **Disable Blinking and Enable All 16 Colors**:
    - Load 1003h into register AX and 0 into register BX to disable text blinking and enable full use of 16 colors.
    - Call **int 10h** to apply these video settings.
4. **Set the Video Memory Segment**:
    - Load the **video memory segment address 0B800h** into register AX.
    - Move the value in AX to **data segment register (DS)** to point the **data segment** to the video memory.
5. **Write the String "Hello, World!" to Video Memory**:
    - Write each character of the string "Hello, World!" directly to **even addresses** in video memory:
        - Write 'H' at address 02h.
        - Write 'e' at address 04h.
        - Write 'l' at address 06h, and so on until the last character, '!', at address 18h.
    - Ensure to leave odd addresses for **color attributes**, which will be set later.
6. **Set Colors for Each Character**:
    - Load 12 (the number of characters) into register CX.
    - Initialize DI to 03h (first color attribute address after 'H').
    - Start a loop to color each character:

- Set the **color attribute** byte at address DI to **light red on yellow** (11101100b).
- Increment DI by 2 to skip to the next color attribute byte (since each character occupies 2 bytes).
- Repeat the process until all 12 characters have been colored.

7. **Wait for a Key Press**:
    - Set the system to wait for a key press using **int 16h** (keyboard interrupt).
8. **End the Program**:
    - Return control to the operating system using the **ret** instruction.

Flowchart:



Conclusion: Hence, we have printed the string in 8086.