Bootloader in Assembly

Pre-requisites

> Install NASM and Qemu

The code

```
bits 16
org 0x7c00

mov si , 0

print:
        mov ah , 0x0e
        mov al , [hello + si]
        int 0x10
        add si , 1
        cmp byte [hello + si] , 0
        jne print

jmp $

hello:
        db "Hello World !" , 0

times 510 - ($ - $$) db 0
dw 0xAA55
```

The explanation

`bits 16` :

- tells the assembler NASM to generate 16 bit instructions
- necessary for **real mode**
  - Initial Mode the CPU is in when the computer boots
  `org 0x7c00` :
- the BIOS loads the code at this location in memory
- specifies the origin or the base address of the code in memory

`mov si ,0`:

- putting value 0 in `si` register
- `si` register is used to iterate over the string characters

`print:` :

- label
  - like a function

`mov ah , 0x0e` :

- Prepares the **BIOS teletype interrupt** (INT 10h, AH=0Eh) to print a character
- This function displays a character in **TTY mode** (character and scroll)

`mov al, [hello + si]` :

- Loads a **byte (character)** from memory at `hello + si` into the `AL` register.
- `AL` is where the character to print must be placed according to the lookup table

`int 0x10` :

- BIOS interrupt to print the character in `AL` using **function 0x0E** (TTY mode).
- This displays one character on the screen.
- `int` hear means interrupt

`add si, 1` :

- adds one to the `si` register
- to move to the next character in the string.

`cmp byte [hello + si], 0` :

- Compares the next character to `0` (null terminator).
- If it is 0, the string has ended.
- we placed the 0 in the hello label

`jne print`

- **Jump if Not Equal** (i.e., the character is not zero).
- If the end of the string hasn't been reached, jump back to `print:`
- create a loop where all the characters are printed from the string in `hello` label

`jmp $`

- infinite loop
- `$` means the same line
- `jmp` stand for jump
- so the CPU keeps on executing the same line

`hello` :

- label

`db "Hello World !" , 0`

- `db` = define bytes.
- Store the string "Hello World !" in memory and followed by a 0 which is called a `null terminator`
- used as input for the print loop

`times 510 - ($ - $$) db 0`:

- BIOS needs the boot sector to be of 512 bytes exactly
- the next line takes up 2 bytes so we run a loop using times to fill all the remaining bytes with 0
- `$` means current line , `$$` means start of section that is 0x7c00
- all the memory except the lines between start and current and except 2 bytes will be filled with 0

`dw 0xAA55` :

- Boot sector **signature** required by the BIOS.
- Must be **last two bytes** of the 512-byte boot sector
- If missing or incorrect, BIOS will not consider this a valid bootloader

Executing the code

- save the code as `boot.asm`
- `nasm boot.asm`
- `qemu-system-i386 boot` : in the same directory where the boot file generated by NASM is

Other terms used here :

1. **Boot Sector**

- The **boot sector** is the **first sector (512 bytes)** of a storage device (like a hard disk or USB stick) that contains code to **start the booting process**.
- When a computer is powered on, the **BIOS** (Basic Input/Output System) reads the boot sector into memory at address `0x7C00` and starts executing it.
- The boot sector must:
  - Be **exactly 512 bytes**
  - End with the **signature** `0xAA55` at the last two bytes
- A valid boot sector is the **first stage of a bootloader**.

2. **BIOS Teletype Interrupt**

- A **BIOS interrupt** is a service provided by the BIOS, accessed via the `int` instruction (e.g., `int 0x10`).
- The **teletype interrupt** (`int 0x10`, function `0x0E`) is used to **print a character to the screen** in **text mode**.
- Usage:
  - `AH = 0x0E` (select teletype function)
  - `AL = character to print`
  - Then call `int 0x10`

3. **TTY Mode (Teletype Mode)**

- TTY stands for **Teletypewriter**.
- In this context, **TTY mode** means:
  - Displaying **characters one at a time** on the screen
  - The BIOS scrolls the screen if needed
  - No graphics, just plain **text output**

4. **Label**

- A **label** in assembly is a **named location in memory or code**.
- It marks a position you can reference, like a **bookmark**.

```
hello:
    db "Hello", 0
```

- `hello` is a label.
- You can later reference it with `[hello]` or `hello + si` to read from that memory
- They are like functions in higher level languages

5. difference between `[hello + si]` and `hello + si`

- `[hello + si]` : Memory Access

    - The square brackets mean: Access the memory at the address `hello + si`.
    - This fetches the value stored at that memory location

```
mov al, [hello + si]
```

```
- take the value at memory address `hello + si` and put it in `AL`
```

- `hello + si` : Address
    - Without brackets, `hello + si` refers to the address itself, not the value at that address.
    - It's a calculated address, not dereferenced
        - Dereferencing means: go to that address and get the value stored there.
        - You dereference an address in assembly using square brackets `[]`