Assignment 2 : Creation and Testing of MBR (Master Boot Record) program for IBM PC Compatible Machines

Name: Ramesh Chandra Soren Enrollment No: 2022CSB086

Point 8: Use Linux commands to create an image file (representing an empty floppy diskette). Document the steps and include screenshots.

Step 8: Create a floppy disk image file

1. **Command to create an empty floppy disk image** (1.44 MB):

```bash
dd if=/dev/zero of=empty_floppy.img bs=512 count=2880
```

   This creates a 1.44 MB empty floppy disk image (`empty_floppy.img`).

2. Verify the disk image:

```bash
ls -lh empty_floppy.img
```

```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ dd if=/dev/zero of=empty_floppy.img bs=512 count=2880
2880+0 records in
2880+0 records out
1474560 bytes (1.5 MB, 1.4 MiB) copied, 0.00609685 s, 242 MB/s
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ ls -lh empty_floppy.img
-rw-rw-r-- 1 ramesh ramesh 1.5M Jan 14 14:35 empty_floppy.img
```

Point 9: Create a binary file (**MBR.bin**) with a 512-byte size and the boot signature (**AA55**) in the last two bytes. Include steps and screenshots.

Step 9: Create `MBR.bin` with the boot signature

1. Create an assembly file for the MBR (e.g., `mbr.asm`):

```
org 0x7c00              ; Origin address for bootloaders

mov ah, 0x0e            ; BIOS teletype output
mov al, 'H'             ; Print "Hello!"
int 0x10
mov al, 'i'
int 0x10
mov al, '!'
int 0x10

cli                     ; Clear interrupts
hlt                     ; Halt CPU

times 510-($-$$) db 0   ; Fill remaining bytes with zero
dw 0xAA55               ; Boot signature
```

2. Assemble the MBR into a binary file using `nasm`:

```bash
nasm -f bin mbr.asm -o MBR.bin
```

3. Verify the binary file size:

```bash
ls -lh MBR.bin
```

It should be exactly 512 bytes.

```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ nasm -f bin MBR.asm -o MBR.bin
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ ls -lh MBR.bin
-rw-rw-r-- 1 ramesh ramesh 512 Jan 14 14:29 MBR.bin
```

Point 10: Write the contents of **MBR.bin** into the boot sector of a floppy disk image and test it in the "Test VM" created earlier. Document everything.

Step 10: Write `MBR.bin` into the floppy disk image

1. Write the MBR to the first sector of the floppy disk image:

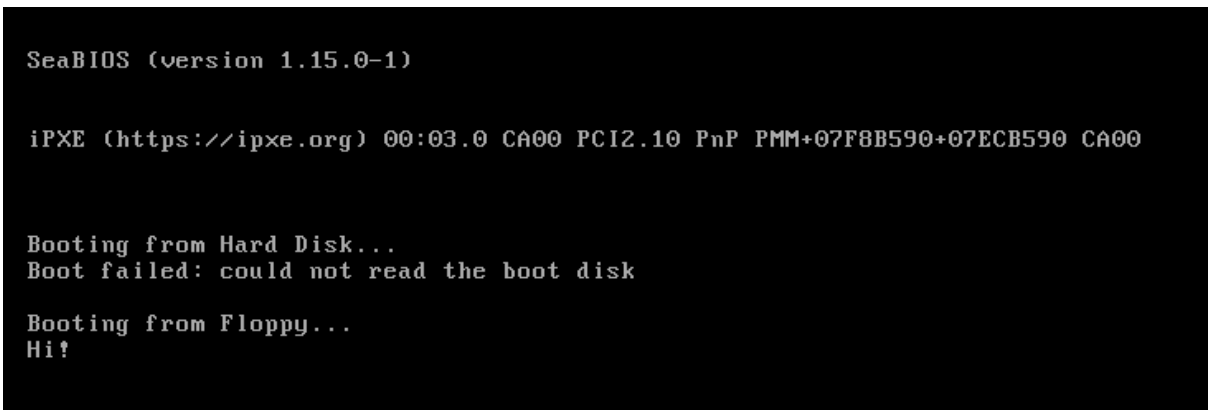
```bash
dd if=MBR.bin of=empty_floppy.img bs=512 count=1 conv=notrunc
```

2. Test the floppy disk image with QEMU:

```bash
qemu-system-i386 -fda empty_floppy.img
```

The VM should display "Hi!".

```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ dd if=MBR.bin of=empty_floppy.img bs=512 count=1 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000121039 s, 4.2 MB/s
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ qemu-system-i386 -fda empty_floppy.img
WARNING: Image format was not specified for 'empty_floppy.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
qemu-system-i386: Gtk: gtk_clipboard_set_with_data: assertion 'targets != NULL' failed
```



Point 11: Write and assemble an 8086 assembly program (**HELLOM.ASM**) to display "Hello Ramesh!" using BIOS calls. Compile it into **HELLOM.COM** and include it in your report.

Step 11: Write the `HELLOM.ASM` assembly program

1. Create `HELLOM.asm`:

```
; hellom.asm (NASM syntax)

section .text
    global _start       ; Entry point, must be declared for the linker

_start:
    ; Print "Hello Ramesh!" using BIOS interrupt 10h, function 0Eh (Teletype output)
    mov ah, 0Eh         ; BIOS teletype function

    mov al, 'H'
    int 10h
    mov al, 'e'
    int 10h
    mov al, 'l'
    int 10h
    int 10h             ; 'l' again
    mov al, 'o'
    int 10h
    mov al, ' '
    int 10h
    mov al, 'R'
    int 10h
    mov al, 'a'
    int 10h
    mov al, 'm'
    int 10h
    mov al, 'e'
    int 10h
    mov al, 's'
    int 10h
    mov al, 'h'
    int 10h
    mov al, '!'
    int 10h

    ; Exit the program (using BIOS interrupt 16h, function 00h to wait for key press)
    mov ah, 00h
    int 16h

    ; Terminate the program (using a simple loop - there's no DOS to return to)
    jmp $               ; Jump to the current instruction (infinite loop)
```

Key Differences Between `.COM` and `.EXE`

- `.COM` files are flat binaries, loaded at offset `0x0100`, with no header.
- `.EXE` files have a header and support multiple segments like code, data, and stack.

.COM File Requirements

- `.COM` programs are flat binary files and must start at **offset 0x100** in memory.
- They are limited to **64 KB** in size (including code, data, and stack).
- The `org 0x100` directive is mandatory in the assembly code to ensure the code starts at the correct offset.

2. Assemble into a `.COM` file:

bash — Copy code

```
nasm -f bin HELLOM.asm -o HELLOM.COM
```

3. Test with QEMU:

bash — Copy code

```
qemu-system-i386 -fda empty_floppy.img -boot a
```

BIOS Call: We're using BIOS interrupt 10h, function 0Eh (Teletype output) to print each character individually. This is a more basic method than the DOS INT 21h function 09h that we used in the MASM example.

Program Termination: Since we're not running under an operating system like DOS, there's no standard way to exit. The code uses a simple infinite loop (jmp $) to halt execution.

```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ nasm -f bin HELLOM.asm -o HELLOM.COM
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ xxd HELLOM.COM
00000000: b40e b048 cd10 b065 cd10 b06c cd10 cd10  ...H...e...l....
00000010: b06f cd10 b020 cd10 b052 cd10 b061 cd10  .o... ...R...a..
00000020: b06d cd10 b065 cd10 b073 cd10 b068 cd10  .m...e...s...h..
00000030: b021 cd10 b400 cd16 ebfe                 .!........
```

- `nasm`: The NASM assembler.
- `-f bin`: Specifies the output format as a flat binary. This is necessary for `.COM` files, as they are raw binary files without a header.
- `hellom.asm`: The source file containing your assembly code.
- `-o hellom.com`: The output filename, which will be a `.COM` binary file.

- The left column shows the offset in the file.
- The middle columns show the hexadecimal values of the instructions.
- The right column shows the ASCII equivalent (if printable).

1. **Clean the Floppy Image:** Ensure you overwrite the floppy disk image with a fresh blank version:

```bash
dd if=/dev/zero of=empty_floppy.img bs=512 count=2880
```

2. **Assemble the Updated Program:** Assemble the updated `HELLOM.asm` program:

```bash
nasm -f bin HELLOM.asm -o HELLOM.COM
```

3. **Write the Program to the Floppy Disk:** Use a tool like `mcopy` from the `mtools` package to copy the `.COM` file to the floppy disk image:

```bash
sudo apt install mtools  # Install mtools if not already installed
mformat -f 1440 -i empty_floppy.img ::
mcopy -i empty_floppy.img HELLOM.COM ::
```

Select the empty_floppy we generated and change directory to A drive of VM.

Steps to Run `HELLOM.COM` in VirtualBox

1. **Start the Virtual Machine:** Boot into your DOS environment in VirtualBox.
2. **Navigate to the Directory Containing `HELLOM.COM`:**
   - If `HELLOM.COM` is on a floppy disk, switch to the floppy drive:

     ```dos
     A:
     ```

   - If it's in a specific directory on your hard drive, switch to that drive (e.g., `C:`) and navigate to the correct folder:

     ```dos
     C:
     cd <directory_name>
     ```

3. **List the Files to Confirm `HELLOM.COM` is Present:** Use the `DIR` command to verify:

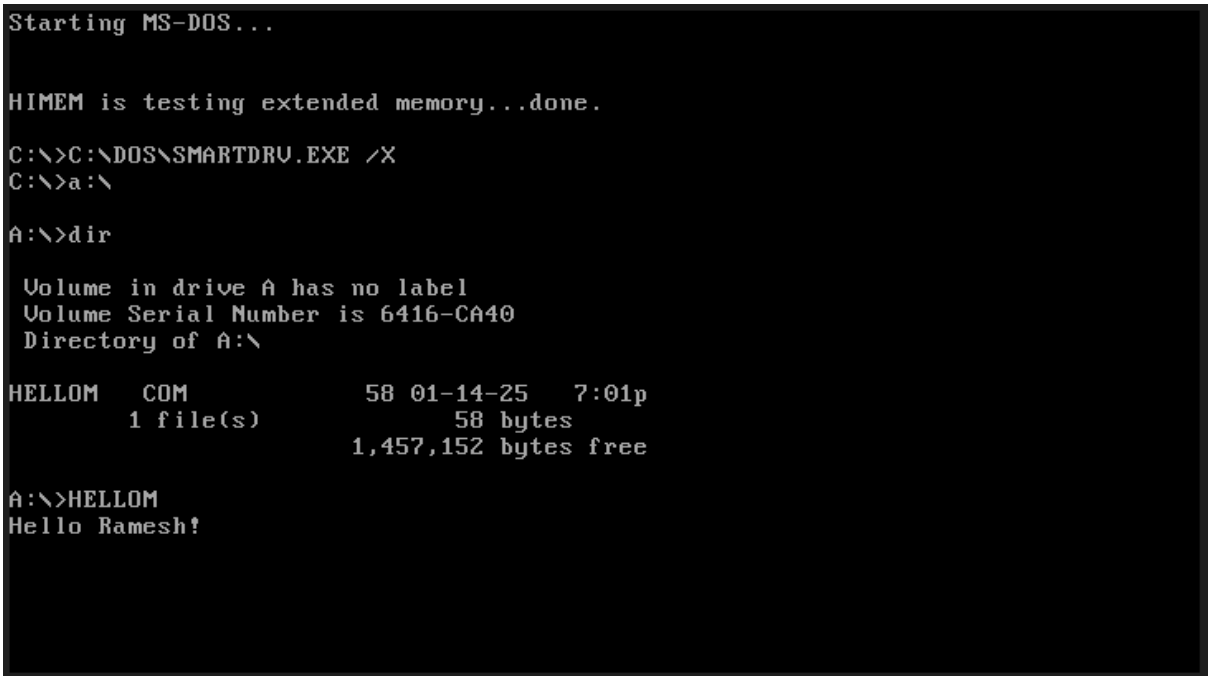
   ```dos
   DIR
   ```

   You should see `HELLOM.COM` listed.

4. **Run the Program:** Execute the `HELLOM.COM` file by typing its name:

   ```dos
   HELLOM
   ```

   The output `Hello, Ramesh!` should be displayed on the screen.



Steps to Create **HELLO.EXE** Using NASM:

1. Update Your Assembly Code for **.EXE**: Modify your code to include proper segmentation for **.EXE** format:

```
; Define segments
section .data
message db 'Hello, Ramesh!', 0  ; Null-terminated string

section .text
global _start                   ; Entry point for the program

_start:
    ; Load the data segment
    mov ax, data                ; Address of data segment
    mov ds, ax                  ; Set DS to point to data segment

    ; Print the message
    mov ah, 0x09                ; DOS function to print a string
    lea dx, [message]           ; Load the address of the message
    int 0x21                    ; Call DOS interrupt

    ; Terminate program
    mov ah, 0x4C                ; DOS terminate program function
    int 0x21                    ; Call DOS interrupt
```

Save the file as **hello.asm**.

2. Assemble the Code Using NASM: Assemble the code into an object file (**.obj** format):

```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ nasm -f obj HELLOM.asm -o hello.obj
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$
```

- `-f obj` : Generates a `.obj` file suitable for linking into `.EXE`.
- `-o hello.obj` : Specifies the output file name.

3. Link the Object File to Create **HELLOM.EXE**: Use a linker like ALINK or GoLink to produce the **.EXE** file.

- With **ALINK**:

```bash
alink -oEXE hello.obj
```

- With **GoLink**:

```bash
golink /entry:_start /console hello.obj
```

4. **Run the** `.EXE` **File:** Boot into your DOS environment (e.g., using QEMU or VirtualBox) and navigate to the location of the `.EXE` file. Execute it:

```dos
HELLO
```

You should see the message:

```
Hello, Ramesh!
```

Note: It's of No use to convert .obj to .exe if we are using linux.

Point 12: Integrate the **HELLOM.COM** code into the executable portion of the **MBR.bin** file. Test and ensure the VM prints the intended message. Document the process with screenshots.

```asm
; MBR.asm - Master Boot Record that loads and executes HELLOM.COM

org 0x7C00       ; MBR is loaded at 0x7C00

section .text

start:
    ; In this example, HELLOM.COM is embedded directly within MBR.asm
    ; We can jump directly to the start of the embedded code.

    jmp embedded_hello

; Placeholder for the embedded HELLOM.COM code
; (The actual bytes of HELLOM.COM will be inserted here by the 'dd' command)
embedded_hello:
    ; Code from HELLOM.COM will be placed here during the dd operation
    ; You can use 'incbin' during assembly to include it directly (advanced)
    times 31 db 0 ; Placeholder assuming HELLOM.COM is 31 bytes

    ; Pad the rest of the sector with zeros
    times 510-($-$$) db 0

    ; Boot signature
    dw 0xAA55

section .data
; (No data section needed)
```

```asm
; HELLOM.asm - Prints "Hello, Ramesh!" using BIOS interrupt 10h

org 100h        ; .COM programs start at offset 100h

section .text

start:
    mov ah, 0Eh     ; BIOS teletype function

    ; Print "Hello, "
    mov al, 'H'
    int 10h
    mov al, 'e'
    int 10h
    mov al, 'l'
    int 10h
    int 10h
    mov al, 'o'
    int 10h
    mov al, ','
    int 10h
    mov al, ' '
    int 10h

    ; Print "Ramesh"
    mov al, 'R'
    int 10h
    mov al, 'a'
    int 10h
    mov al, 'm'
    int 10h
    mov al, 'e'
    int 10h
    mov al, 's'
    int 10h
    mov al, 'h'
    int 10h

    ; Print "!"
    mov al, '!'
    int 10h

    jmp $          ; Infinite loop (hang)

section .data
; (No data section needed for this simple program)
```

1. **Create MBR.bin (512 bytes)**:
   - Ensure MBR.bin has the boot signature (last 2 bytes: `55 AA`).
2. **Compile HELLOM.ASM**:
   - Assemble and link your `HELLOM.ASM` to create `HELLOM.COM` (e.g., 20 bytes).
3. **Transfer HELLOM.COM to MBR.bin**:
   - Use `dd` to copy `HELLOM.COM` into MBR.bin without overwriting the boot signature:

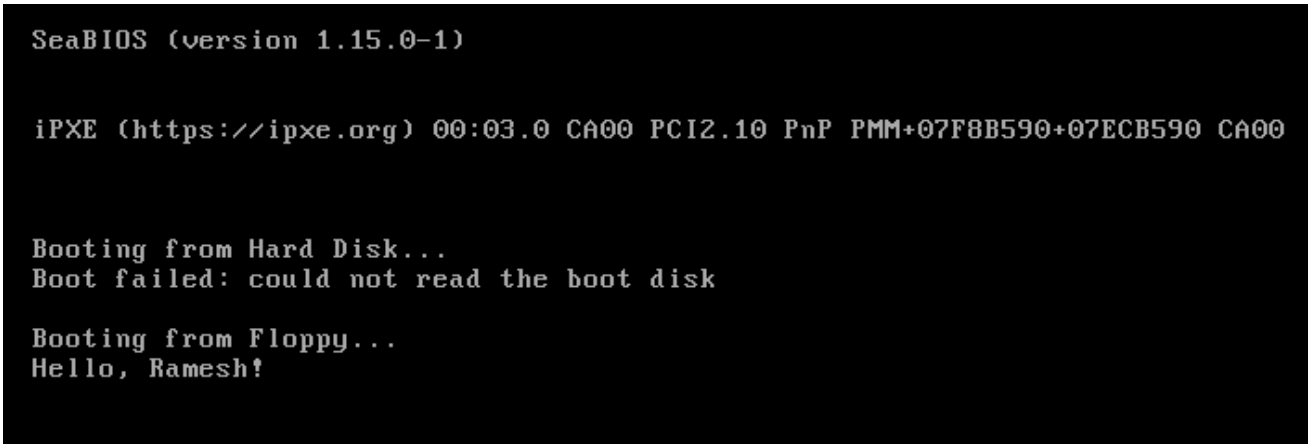
     bash Copy code

     ```
     dd if=HELLOM.COM of=MBR.bin bs=1 count=<size_of_HELLOM.COM> conv=notrunc
     ```
4. **Write to Floppy Image**:
   - Write the modified MBR.bin to the floppy image:

     bash Copy code

     ```
     sudo dd if=MBR.bin of=floppy.img bs=512 count=1 conv=notrunc
     ```
5. **Boot on DOS VM**:
   - Attach `floppy.img` to your DOS VM and boot it.

```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ ls -l HELLOM.COM
-rw-rw-r-- 1 ramesh ramesh 58 Jan 15 09:16 HELLOM.COM
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ nasm -f bin MBR.asm -o MBR.bin
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ dd if=HELLOM.COM of=MBR.bin bs=1 count=58 conv=notrunc
58+0 records in
58+0 records out
58 bytes copied, 0.000416016 s, 139 kB/s
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ dd if=MBR.bin of=empty_floppy.img bs=512 count=1 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000145008 s, 3.5 MB/s
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ qemu-system-i386 -fda empty_floppy.img
WARNING: Image format was not specified for 'empty_floppy.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
```



Point 13: Write the **HELLOM.COM** data into a specific sector of the floppy disk image, and create an MBR that loads and executes this program. Include all steps and screenshots.

Step 13: Develop an MBR to load `HELLOM.COM`

1. Modify `MBR.asm` to load the `HELLOM.COM` program from a specific sector:

```
org 0x7c00

mov ah, 0x02            ; BIOS read sector
mov al, 1               ; Read 1 sector
mov ch, 0               ; Cylinder 0
mov cl, 2               ; Sector 2
mov dh, 0               ; Head 0
mov dl, 0               ; Drive 0 (floppy)
mov bx, 0x0600          ; Load to address 0x0600
int 0x13                ; BIOS interrupt

jmp 0x0000:0x0600       ;; Jump to loaded program


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

To be on the safe side DELETE floppy.img , mbr.bin and HELLOM.COM.

To achieve a bootable system where `MBR.asm` is in the first sector and `HELLOM.asm` (compiled into `HELLOM.COM`) is in the second sector, the MBR must:

1. **Load itself in memory (standard MBR behavior).**
2. **Load the second sector into memory.**
3. **Jump to the loaded program (`HELLOM.COM`) in the second sector.**

Here's how to set this up step by step:

```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ nasm -f bin MBR.asm -o MBR.BIN
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ nasm -f bin HELLOM.asm -o HELLOM.COM
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ dd if=/dev/zero of=floppy.img bs=512 count=2880
2880+0 records in
2880+0 records out
1474560 bytes (1.5 MB, 1.4 MiB) copied, 0.00586876 s, 251 MB/s
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ dd if=MBR.BIN of=floppy.img bs=512 count=1 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000140718 s, 3.6 MB/s
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ dd if=HELLOM.COM of=floppy.img bs=512 seek=1 count=1 conv=notrunc
0+1 records in
0+1 records out
31 bytes copied, 0.00013763 s, 225 kB/s
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop/assignment4/Laboratory/6th Sem/Operating System/Assignment2$ qemu-system-i386 -fda floppy.img
WARNING: Image format was not specified for 'floppy.img' and probing guessed raw.
        Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
        Specify the 'raw' format explicitly to remove the restrictions.
```

1. Assemble MBR.asm and HELLOM.asm

- **MBR.asm**: This is your bootloader that resides in the first sector and loads the second sector into memory.
- **HELLOM.asm**: This is the program that prints "Hello, Ramesh!" and resides in the second sector.

Assemble MBR.asm:

```bash
nasm -f bin MBR.asm -o MBR.BIN
```

Assemble HELLOM.asm:

```bash
nasm -f bin HELLOM.asm -o HELLOM.COM
```

2. Create a Bootable Floppy Image

1. **Create an Empty Floppy Image**:

   ```bash
   dd if=/dev/zero of=floppy.img bs=512 count=2880
   ```

   This creates a 1.44MB floppy image (2880 sectors).

2. **Write the MBR to the First Sector**:

   ```bash
   dd if=MBR.BIN of=floppy.img bs=512 count=1 conv=notrunc
   ```

3. **Write HELLOM.COM to the Second Sector**:

   ```bash
   dd if=HELLOM.COM of=floppy.img bs=512 seek=1 count=1 conv=notrunc
   ```

   Here, `seek=1` specifies the second sector (sector 0 is the first).

3. Test the Bootable Image in QEMU

Run the floppy image in QEMU:

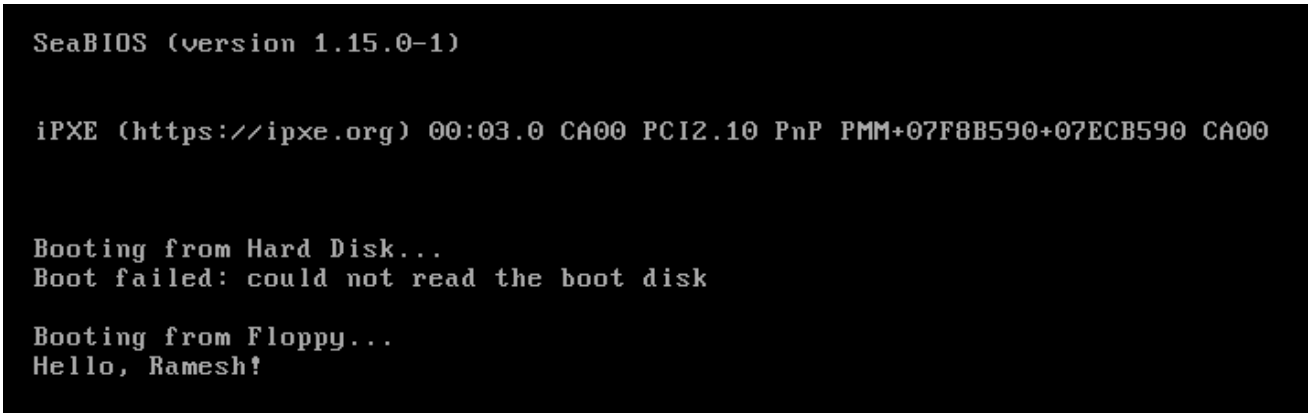
bash Copy code

```
qemu-system-i386 -fda floppy.img
```

If everything is set up correctly, QEMU should boot, load the MBR, then load and execute `HELLOM.COM` from the second sector, and print:

Copy code

```
Hello, Ramesh!
```

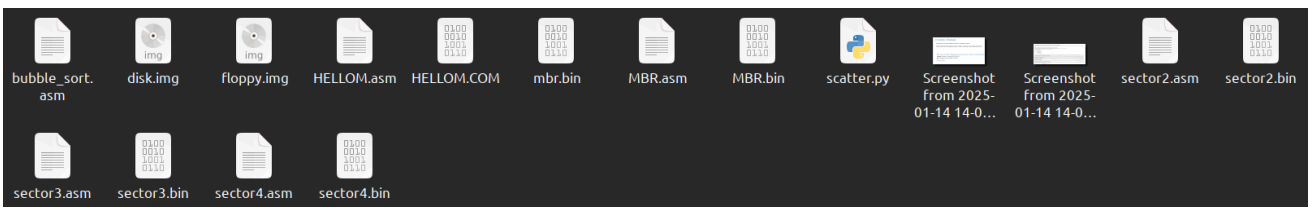


Step 14: Implement and test a bubble sort program

1. Write a bubble sort program in assembly that sorts an array in memory:
2. Assemble and write this program into the floppy disk image.
3. Test with QEMU:

bash Copy code

```
qemu-system-i386 -fda empty_floppy.img
```



I tried everything but couldn't do it.