# EXPERIMENT 2

EX NO: 2 write a boot loader - to load a particular OS. OS image – code to access from BIOS to load the OS.

**Pre requisite: install NASM, run the following in terminal**

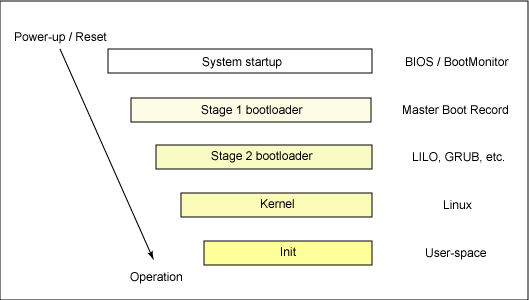
sudo apt-get install nasm

**Aim:**

To Study about boot loader function

**Description**

# Boot Sequence



## 

## POST

When a computer is switched on or reset, it runs through a series of diagnostics called POST - **P**ower-**O**n **S**elf-**T**est. This sequence culminates in locating a bootable device, such as a floppy, cdrom or a harddisk in the order that the firmware is configured to.

## The Bootsector

The first 512 bytes of a disk are known as the **bootsector** or **Master Boot Record**. The boot sector is an area of the disk reserved for booting purposes. If the bootsector of a disk contains a valid boot sector (the last word of the sector must contain the signature 0xAA55), then the disk is treated by the BIOS as bootable.

To boot an operating system, the BIOS runtime searches for devices that are both active and bootable in the order of preference defined by the complementary metal oxide semiconductor (CMOS) settings. A boot device can be a floppy disk, a CD-ROM, a partition on a hard disk, a device on the network, or even a USB flash memory stick.

Commonly, Linux is booted from a hard disk, where the Master Boot Record (MBR) contains the primary boot loader. The MBR is a 512-byte sector, located in the first sector on the disk (sector 1 of cylinder 0, head 0). After the MBR is loaded into RAM, the BIOS yields control to it.

Reference URLs:

<http://www.ibm.com/developerworks/library/l-linuxboot/index.html#resources>

<http://www.brokenthorn.com/Resources/OSDev3.html>

<https://en.wikibooks.org/wiki/X86_Assembly/Bootloaders>

<http://viralpatel.net/taj/tutorial/hello_world_bootloader.php>

<http://wiki.osdev.org/Tutorials#Booting>

<http://mikeos.sourceforge.net/write-your-own-os.html>

<https://sourceforge.net/p/squeaknos/git/ci/bc73aeae09ef3ac8b1a9e708ba07589236b4bb15/tree/boot/iso.template/boot/grub/stage2_eltorito>

<https://www.gnu.org/software/grub/manual/legacy/Making-a-GRUB-bootable-CD_002dROM.html>

<http://wiki.osdev.org/Bootable_El-Torito_CD_with_GRUB_Legacy>

**Steps for the lab**

Download and install NASM from

<http://www.osdever.net/downloads/assemblers/nasm-0.98.35-1.i386.rpm>

sudo apt-get install nasm

1. Create a first Bootloader that does nothing

Create a file firstBootLoader.asm

[BITS 16] ;tell the assembler that its a 16 bit code

[ORG 0x7C00] ;Origin, tell the assembler that where the code will

;be in memory after it is been loaded

JMP $ ;infinite loop

TIMES 510 - ($ - $$) db 0 ;fill the rest of sector with 0

DW 0xAA55 ; add boot signature at the end of bootloader

Explanation:

[BITS 16]: Our code starts with [BITS 16] which is an assembler directive. This will tell assembler that our code is a 16 bit code.

[ORG 0x7C00]: Then we have used [ORG 0x7C00] which tell assembler to assemble the instructions from Origin 0x7C00. BIOS loads bootloader at physical address 0x7C00 hence we have assemble our bootloader starting from that location.

JMP $: JMP at location $ means jumping to the same location. Thus this nothing but an infinite loop. We just want to hang our code here.

TIMES 510 - ($ - $$) db 0: As bootloader is always of length 512 bytes, our code does not fit in this size as its small. We need to use rest of memory and hence we clear it out using TIMES directive. $ stands for start of instruction and $$ stands for start of program. Thus ($ - $$) means length of our code.

DW 0xAA55: This is boot signature. This tells the BIOS that this is a valid bootloader. If bios does not get 0x55 and 0xAA at the end of the bootloader than it will treat bootloader as invalid. Thus we provide this two bytes at the end of our bootloader.

Compile the program

nasm firstBootLoader.asm -f bin -o boot.bin

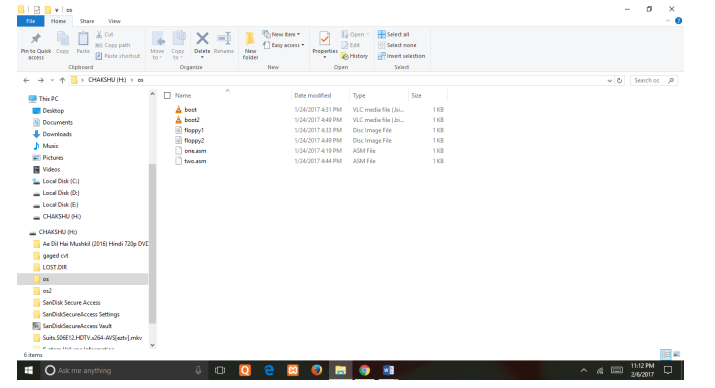
Create a floppy image using

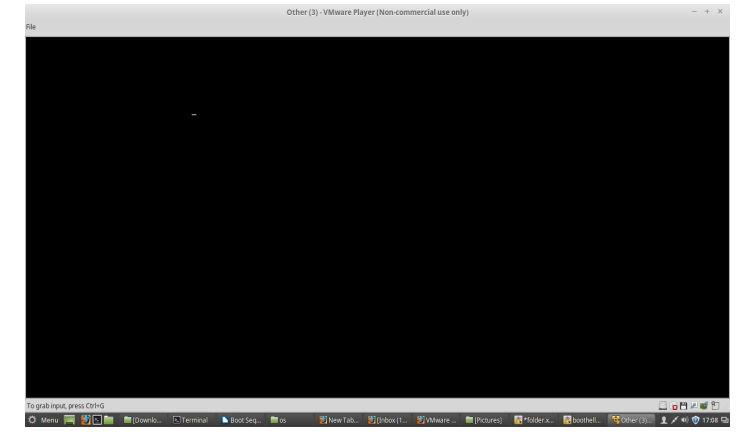
dd if=boot.bin bs=512 of=floppy1.img

Attach the floppy image to the Virtual machine and boot. You should see a blank screen

Output:

Sample Output:





(ii) Create 2nd Bootloader that prints ‘A’ on the screen

we will use BIOS video interrupt int 0x10.

INT 0x10 is a BIOS video interrupt. All the video related calls are made through this interrupt.

To use this interrupt we need to set the values of some register.

AL = ASCII value of character to display

AH = 0x0E ;Teletype mode (This will tell bios that we want to print one character on screen)

BL = Text Attribute (This will be the fore ground and background color

of character to be displayed. 0x07 in our case.)

BH = Page Number (0x00 for most of the cases)

Once all the registers all filled with appropriate value, we can call interrupt.

Code saved in secBootLoader.asm

[BITS 16] ;Tells the assembler that its a 16 bit code

[ORG 0x7C00] ;Origin, tell the assembler that where the code will

;be in memory after it is been loaded

MOV AL, 65

CALL PrintCharacter

JMP $ ;Infinite loop, hang it here.

PrintCharacter: ;Procedure to print character on screen

;Assume that ASCII value is in register AL

MOV AH, 0x0E ;Tell BIOS that we need to print one charater on screen.

MOV BH, 0x00 ;Page no.

MOV BL, 0x07 ;Text attribute 0x07 is light-grey font on black background

INT 0x10 ;Call video interrupt

RET ;Return to calling procedure

TIMES 510 - ($ - $$) db 0 ;Fill the rest of sector with 0

DW 0xAA55 ;Add boot signature at the end of bootloader

Repeat previous two compile and image creation steps

nasm SecBootLoader.asm -f bin -o boot2.bin

dd if=boot2.bin bs=512 of=floppy2.img

Output:

We should see a ‘A’ on the screen

(iii) Third try: Hello World Bootloader

Its time to create our final Hello World bootloader. We have enough experience now and can code it without wasting a second. So once again start your favourite text editor and start writing following cod

[BITS 16] ;Tells the assembler that its a 16 bit code

[ORG 0x7C00] ;Origin, tell the assembler that where the code will

;be in memory after it is been loaded

MOV SI, HelloString ;Store string pointer to SI

CALL PrintString ;Call print string procedure

JMP $ ;Infinite loop, hang it here.

PrintCharacter: ;Procedure to print character on screen

;Assume that ASCII value is in register AL

MOV AH, 0x0E ;Tell BIOS that we need to print one character on screen.

MOV BH, 0x00 ;Page no.

MOV BL, 0x07 ;Text attribute 0x07 is light grey font on black background

INT 0x10 ;Call video interrupt

RET ;Return to calling procedure

PrintString: ;Procedure to print string on screen

;Assume that string starting pointer is in register SI

next\_character: ;Lable to fetch next character from string

MOV AL, [SI] ;Get a byte from string and store in AL register

INC SI ;Increment SI pointer

OR AL, AL ;Check if value in AL is zero (end of string)

JZ exit\_function ;If end then return

CALL PrintCharacter ;Else print the character which is in AL register

JMP next\_character ;Fetch next character from string

exit\_function: ;End label

RET ;Return from procedure

;Data

HelloString db 'Hello World', 0 ;HelloWorld string ending with 0

TIMES 510 - ($ - $$) db 0 ;Fill the rest of sector with 0

DW 0xAA55 ;Add boot signature at the end of bootloader

**Compile and run like last time**

nasm thirdBootLoader.asm -f bin -o boot3.bin

dd if=boot3.bin bs=512 of=floppy3.img

load in Virtual machine and see Hello World printed