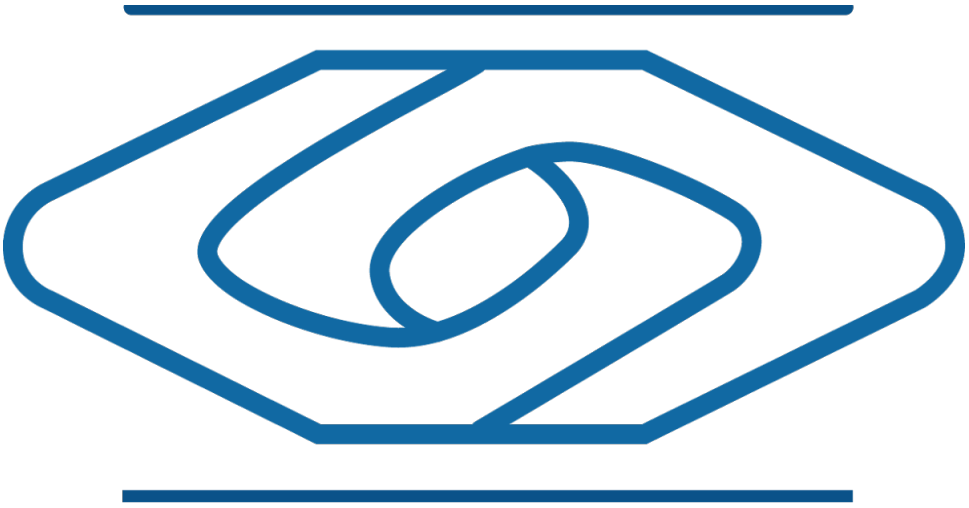


Instituto Politécnico Nacional

Escuela Superior de Computo

Práctica 2: Bootloader que escribe en la pantalla

Reporte

Profesor: Ulises Velez Saldaña

Alumno: Meza Madrid Raúl Damián

Clase: Sistemas operativos

Grupo: 2CM7

Contents

Introducción

Como se vio en la practica pasada, el boot loader aquel encargado de buscar y cargar el sistema operativo. Durante el proceso de booteo, el control lo tiene primero el BIos, después se lo pasa al boot loader. Una vez en el boot loader es posible llamar interrupciones del BIOs
Una *interrupcion* es un mecanismo que tienen las computadoras que detiene el ciclo de fetch para atender un evento solicitado o no solicitado. Se usan para:

- Seguridad
- Operaciones I/O
- Funciones del BIOS y O.S.

Esta práctica tiene como propósito utilizar las llamadas a interrupciones del BIOS de diferentes maneras para escribir un mensaje desde el boot loader. Así como utilizar los principios vistos en clase para las direcciones en la que el boot loader debe cargar.

1.1 Programas y herramientas utilizados

Esta práctica fue desarrollada en el sistema operativo Ubuntu 18.04.1 LTS. Los programas y herramientas utilizados, junto con el comando de instalación, en caso de que no estuvieran instalados ya.

- NASM : $ apt-get install nasm
- QEMU : $ apt-get install qemu
- dd
- genisoimage

Objetivo

Que el alumno aplique la teoría vista en clase, conozca el proceso y las herramientas necesarias para construir el cargador.

Desarrollo

Para desarrollar y ejecutar la práctica, se debe seguir el procedimiento de la práctica anterior para crear las imágenes tanto de Floppy Disk como CD-ROM
Se utilizo el sigueinte script de bash compilar el programa, crear sus respectivas imagenes y montarlo en la máquina virtual.

```
#!/usr/bin/env bash
nasm src/bootloader.asm -f bin -o bin/bootloader
dd if=/dev/zero of=images/escomOS.flp bs=1024 count=1440
dd if=bin/bootloader of=images/escomOS.flp seek=0 count=1 conv=notrunc
genisoimage -quiet -V 'escomOS' -input-charset iso8859-1 -o ./images/escomOS.iso -b
qemu-system-i386 -fda images/escomOS.flp
qemu-system-i386 -cdrom images/escomOS.iso

nasm src/bootloaderVidMem.asm -f bin -o bin/bootloadervidMem
dd if=/dev/zero of=images/escomOSVidMem.flp bs=1024 count=1440
dd if=bin/bootloaderVidMem of=images/escomOSVidMem.flp seek=0 count=1 conv=notrunc
genisoimage -quiet -V 'escomOSVidMem' -input-charset iso8859-1 -o ./images/escomOSV
qemu-system-i386 -fda images/escomOSVidMem.flp
qemu-system-i386 -cdrom images/escomOSVidMem.iso
```

3.1 Primera parte

El primero debe utilizar la interrupción 10H del BIOs. Debe considerar que el bootloader es cargado en la dirección 0x07c00(ya sea en el segmento y offset 0000:07C00 o 07C0:0000). Sigue el siguiente proceso:

- Mueve DS a 0x07C0 antes de invocar la interrupción del BIOS [2]
- Asegúrate de que la bandera de dirección indique de menor a mayor (DF=0) [5] [8] [7]
- Debes investigar como se escribe un carácter con la interrupción 10H [1] [6] [4]
- Declara una variable con el mensaje que se desea escribir y colocar un cero al final para marcar el fin del mensaje
- En un ciclo escribe cada carácter verificando que no haya llegado a cero

3.1.1 Interrupciones tipo 10h a 1Fh

Las rutinas de interrupciones 10h a 1Fh pueden ser llamadas por programas para realizar varias operaciones de entrada salida (I/O) o para revisar el estado de la maquina.

3.1.2 Interrupción 10h – Vídeo

La rutina de interrupción 10h es el driver de vídeo. Esta asociada con cada dispositivo de I/O. Existe un dispositivo o controlador I/O que actúa como interfaz

tipo hardware entre el procesador y el dispositivo mismo de I/O. El dispositivo controlador ejecuta varias tareas de bajo nivel especificas al dispositivo de I/O. Esto permite que la CPU interactúe con el dispositivo a un nivel mas alto [1]

3.1.3 INT 10h / AH = 0Eh - Salida a teletipos.

Entradas:
AL = Carácter a escribir.

Esta función muestra un carácter en la pantalla, avanzando el cursor y moviendo la pantalla como sea necesario. La escritura siempre es a la pagina activa. [6]

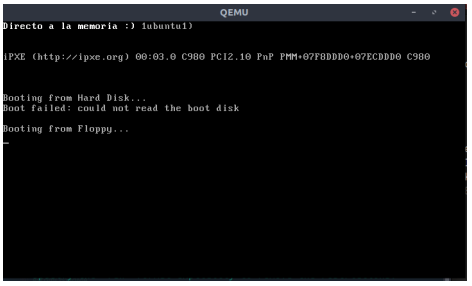
3.1.4 Programa

```
; Trabajando en 16 bits
bits 16
start:
; Mover el DS (Direction Segment) a la direccion 17c0
  mov ax, 07c0h
  mov ds, ax
; cld limpia la vandera de direccion (DF=0)
        cld
; Entra en modo TYY
  mov ah, 0Eh
  ;Se declara el mensaje y se guarda la direccion del mismo en SI
        mov si, msg
; Comienza el ciclo, lodsb es para sacar uno por uno los
↪  caracteres de SI  a AL
.loop       lodsb
; Compara el contenido en al con un 0
        cmp al, 0
; Si el resultado es verdadero, va a done:
        je done
; Interrupcion para escribir el caracter
        int 10h
; Regresa a loop
        jmp .loop
; Se declara la cadena con el 0 al final
msg:        db 'Este no es mi primer ni ultimo hola mundo', 0
; Funcion cuando encuentra el fin de la cadena (linea 18)
done:
```

```
; Magic Numbers
times 510 - ($ - $$) db 0
dw 0xAA55
```

3.1.5 Resultados

El bootloader ahora muestra nuestro mensaje después del mensaje de 'Booting from ...'



3.2 Segunda parte

El segundo bootloader debe escribir directamente en la memoria de video. Usa como ejemplo el programa anterior y mueve carácter por carácter a la memoria de video. Considera lo siguiente:

- Investiga y documenta todo lo que puedas sobre la memoria de video.
- Recuerda que cara caracter tiene un atributo en la memoria de video y que la dirección por defecto es B8000.

3.2.1 Memoria de video

A la hora de escribir en la memoria de video, la cual comienza en 0xb8000, existen dos bytes por cada cuadro o celda en la pantalla. El carácter a mostrar es el primer byte, mientras que el segundo es el atributo del mismo. Para escribir un carácter en la esquina superior izquierda, es necesario mover la palabra formada por [3]

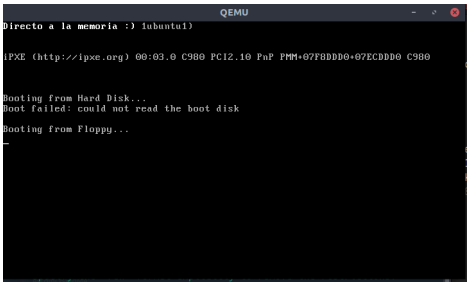
3.2.2 Programa

```
; Trabajando en 16 bits
bits 16
start:
; Mover el DS (Direction Segment) a la direccion 17c0
  mov ax, 07c0h
  mov ds, ax
```

```
7  ; Se usa el ES (Extra Segment) a la direccion de video
8    mov ax,0xb800
9    mov es,ax
10 ;Se declara el mensaje y se guarda la direccion del mismo en SI
11   mov si, msg
12 ;Movemos el DI a la direccion para la esquina superior izquierda
13   mov di,0x0
14 ; Comienza el ciclo
15 .loop
16   mov al,[si]     ; se carga el byte del caracter
17   mov ah,0x0f     ; se carga el byte de la configuracion, fondo
       ↪ negro letra blanca
18   mov [es:di],ax ; se carga el registro de al y ah (ax)a la
       ↪ direccion correspondiente
19   inc si          ; siguiente caracter
20   add di,2        ; se le suma 2 la direccion (1 por el caracter y
       ↪ otro por la configuracion)
21   cmp al,0        ; Compara el contenido con un 0
22   je done         ; Si es verdadero, se va a done:
23   jmp .loop       ; Repite el ciclo
24
25   ; Se declara la cadena con el 0 al final
26 msg:         db 'Directo a la memoria :)', 0
27   ; Funcion cuando encuentra el fin de la cadena (linea 18)
28 done:
29   ; Magic Numbers
30   times 510 - ($ - $$) db 0
31   dw 0xAA55
```

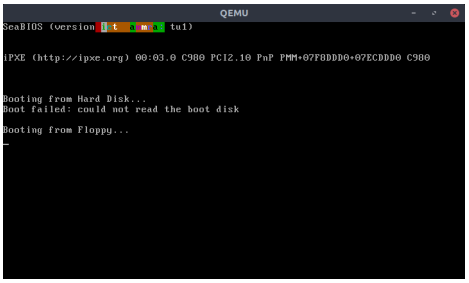
3.2.3 Resultados

El programa sobre-escribió el texto de la maquina virtual y escribió nuestro mensaje en colores mas brillantes



3.3 Errores y problemas

El primer error fue que cuando quería dejar el numero 0xb800 de la misma manera en la que escribí 07c0h, , la escribí como b800h pero el programa dejaba de funcionar. Honestamente no me moleste en investigar la razón del problema, así que solo lo cambie de vuelta.
El segundo error fue que escribí di cuando iba si era di en la instruccion mov [es:di],ax ; lo cual resulto en muchos caracteres de colores.



Codigo (Github)

Todo el codigo de esta practica se puede encontrar en :`https://github.com/asdf1234Damian/Operating-Systems/tree/master/Practica02/Proyecto`

References

[1] King Fahd University of Petroleum Minerals. BIOS Interrupts . `http://faculty.kfupm.edu.sa/COE/aimane/assembly/pagegen-141.aspx.htm`. [Online; consultado en 3-Marzo-2019].

[2] anonymous. Basic NASM bootstrap. `https://stackoverflow.com/questions/10853425/basic-nasm-bootstrap`. [Online; consultado en 25-Febrero-2019].

[3] Dirk Wolfgang Glomp. RE: bootloader printing on video memory 0xb8000. `https://stackoverflow.com/questions/22961979/bootloader-printing-on-video-memory-0xb8000`. [Online; consultado en 25-Febrero-2019].

[4] Eugene Obrezkov. How to implement your own “Hello, World!” boot loader. `https://blog.ghaiklor.com/how-to-implement-your-own-hello-world-boot-loader-c0210ef5e74b`. [Online; consultado en 25-Febrero-2019].

[5] Fahad Uddin. Direction Flag in x86. https://stackoverflow.com/questions/10380076/direction-flag-in-x86. [Online; consultado en 25-Febrero-2019].

[6] Gabriele Cecchetti. The list of all interrupts that are currently supported by the emulator. . http://www.gabrielececchetti.it/Teaching/CalcolatoriElettronici/Docs/i8086_and_DOS_interrupts.pdf. [Online; consultado en 3-Marzo-2019].

[7] linuxassembly. The Netwide Assembler: NASM. http://www.posix.nl/linuxassembly/nasmdochtml/nasmdoca.html. [Online; consultado en 25-Febrero-2019].

[8] Melika. What are cld and std for in x86 assembly... https://stackoverflow.com/questions/9636691/what-are-cld-and-std-for-in-x86-assembly-language-what-does-df-do/9636772. [Online; consultado en 25-Febrero-2019].

[9] Paul de Weerd, UNIX and network engineer. Re: fdisk / signature: 0xAA55. https://www.mail-archive.com/misc@openbsd.org/msg18029.html. [Online; consultado en 25-Febrero-2019].