

Bootsector Authoring

Gareth Owen

Requirements

During this article I assume that you have good knowledge of the assembly language and intel architecture. If not, read an assembly tutorial, they aren't hard to find...

Start

Creating your own bootsector is simpler than you may think, the only requirement is that the bootsector is 512 bytes long, and at offset 0x1FE (decimal=510), the word 0xAA55 is placed. This is the first thing the BIOS does when the PC boots up, it first looks on the first floppy drive at the first sector for 0xAA55 at the end, and if it finds it then it loads it into memory, and starts executing it, otherwise it tries the primary harddisk, and if that isn't found it just bombs out with an error.

You should place your boot sector at:

```
Sector 1
Cylinder 0
Head 0
```

I recommend you start playing about with floppys first instead of your hard disk because the hard disk bootsector stores information about the file system if you are running DOS/Windows, if you overwrite that, then you have just lost your hard disk contents :-)

The BIOS loads the bootsector at linear offset 0x7C00, the state of the registers are:

```
DL = Boot drive, 1h = floppy1, 80h = primary harddisk, etc
CS = 0
IP = 0x7c00
```

So instead of adding [ORG 7C00h] to the top of your file, you can add:

```
mov ax, 0x7C0
mov ds, ax
mov es, ax
mov fs, ax
mov gs, ax
```

And that will set-up the segment registers so they point to the start of your bootsector..

Most boot sectors usually just store the boot drive, load the kernel from disk, and jump to it.. Some will also load protected mode.

Since most people find it easier looking at source code and figuring it out than reading documentation i have included sources for a boot sector and a boot sector writer.

Here is the bootsector...

```
;***** START *****
; Boot sector authoring example by Gareth Owen (gaz@athene.co.uk)
; This should be accompanied with an article explaining bootsectors

[BITS 16] ; the bios starts out in 16-bit real mode
[ORG 0] ; Data offset = 0
```

Bootsector Authoring

Gareth Owen

```
jmp start      ; skip over our data and functions, we cannot execute data :-),
               ; well, you can, but i am not held responsible for the results :)

; Boot sector authoring example by Gareth Owen (gaz@athene.co.uk)
; This should be accompanied with an article explaining bootsectors

[BITS 16]      ; the bios starts out in 16-bit real mode
[ORG 0]        ; Data offset = 0

jmp start      ; skip over our data and functions, we cannot execute data :-),
               ; well, you can, but i am not held responsible for the results :)

; -----
; Data used in the boot-loading process
; -----
bootdrv        db 0
bootmsg        db 'Gareth Owen',39,'s Boot Sector Example',13,10,0

rebootmsg      db 'Press any key to reboot',13,10,0

; these are used in the processor identification
processormsg   db 'Checking for 386+ processor: ',0
need386        db 'Sorry... 386+ required!',13,10,0
found386       db 'Found!',13,10,0

whatever       db 'Insert your code to do something here',13,10,0

;*****
; Functions we are going to use ...
;*****
detect_cpu:
    mov si, processormsg      ; tell the user what we're doing
    call message

    ; test if 8088/8086 is present (flag bits 12-15 will be set)
    pushf                    ; save the flags original value

    xor ah,ah                ; ah = 0
    push ax                   ; copy ax into the flags
    popf                      ; with bits 12-15 clear

    pushf                     ; Read flags back into ax
    pop ax
    and ah,0f0h               ; check if bits 12-15 are set
    cmp ah,0f0h
    je no386                  ; no 386 detected (8088/8086 present)

    ; check for a 286 (bits 12-15 are clear)
    mov ah,0f0h               ; set bits 12-15
    push ax                   ; copy ax onto the flags
    popf

    pushf                     ; copy the flags into ax
    pop ax
    and ah,0f0h               ; check if bits 12-15 are clear
    jz no386                  ; no 386 detected (80286 present)
    popf                      ; pop the original flags back
```

Bootsector Authoring

Gareth Owen

```
        mov si, found386
        call message

no386:   ret                ; no 8088/8086 or 286, so atleast 386
        mov si, need386   ; tell the user the problem
        call message
        jmp reboot       ; and reboot when key pressed

; *****
message:                ; Dump ds:si to screen.
        lodsb            ; load byte at ds:si into al
        or al, al       ; test if character is 0 (end)
        jz done
        mov ah, 0eh     ; put character
        mov bx, 0007    ; attribute
        int 0x10       ; call BIOS
        jmp message
done:
        ret
; *****
getkey:
        mov ah, 0      ; wait for key
        int 016h
        ret
; *****
reboot:
        mov si, rebootmsg ; be polite, and say we're rebooting
        call message
        call getkey      ; and even wait for a key :)

        db 0EAh         ; machine language to jump to FFFF:0000
(reboot)

        dw 0000h
        dw 0FFFFh
        ; no ret required; we're rebooting! (Hey, I just saved a byte :)

; *****
; The actual code of our boot loading process
; *****
start:
        mov ax, 0x7c0    ; BIOS puts us at 0:07C00h, so set DS accordingly
        mov ds, ax      ; Therefore, we don't have to add 07C00h to all our
data

        mov [bootdrv], dl ; quickly save what drive we booted from

        cli            ; clear interrupts while we setup a stack
        mov ax, 0x9000 ; this seems to be the typical place for a stack
        mov ss, ax
        mov sp, 0xffff ; let's use the whole segment. Why not? We can :)
        sti            ; put our interrupts back on

        ; Interestingly enough, apparently the processor will disable
```

Bootsector Authoring

Gareth Owen

```
; interrupts itself when you directly access the stack segment!
; Atleast it does in protected mode, I'm not sure about real mode.

mov si,bootmsg ; display our startup message
call message

call detect_cpu ; check if we've got a 386

.386 ; use 386 instructions from now on (I don't want to manually include
; operand-size(66h) or address-size(67h) prefixes... it's annoying :)

mov si,whatever ; tell the user we're not doing anything interesting here
call message
call getkey

call reboot

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

;***** GBOOTSECT END *****
```

Here is the code for writing the bootsector to a floppy disk. It has been compiled with DJGPP for DOS. It writes the file 'bootsect', onto Sector 1, Cylinder 0, Head 0 of the floppy drive.

```
/**START***/
#include <bios.h>
#include <stdio.h>

void main()
{
    FILE *in;
    unsigned char buffer[520];

    if((in = fopen("bootsect", "rb"))==NULL)
    {
        printf("Error loading file\n");
        exit(0);
    }

    fread(&buffer, 512, 1, in);

    while(biosdisk(3, 0, 0, 0, 1, 1, buffer));

    fclose(in);
}
/**END***/
```

Well, if you still don't understand something, then mail me at gaz@athene.co.uk and i'll help you out.