Simple Bootstrap implementation

ABOUT THIS DOCUMENT

/**
* Cihan KOMECOGLU *
* *
* cihan [at] enderunix.org *
* *
*http://www.enderunix.org/cihan *
* *
* EnderUNIX Software Development Team *
* *
* http://www.enderunix.org *
* *
* Version : 1.0 *
* *
* Date : 24.01.2008 *
* *
* Tags : operating system , boot , kernel programming , assembly *
* *
* Level : Beginning - medium *
* *
* Latest version of paper: http://www.enderunix.org/docs/helloboot-eng.pdf *
* *
**/

Thanks to Murat Balaban and Metin Kaya ...

Simple boot process basically requires somethings. These are:

- BIOS will place your boot code at 07C00h address of memory and the process takes into consideration of this address.

- Boot code which you compile is need to be formatted as a plain binary file. This compilation type will be shown afterwards.

-Size of boot code has to be 512 bytes.

- Last two bytes of boot code must be AA55h. These bytes -like EOF- indicate it is the end of the boot code. These bytes are named as "Boot Record Signature". If you don't use these two bytes at the end of your boot code, then you get "**Boot Record Signature AA55 not found**" error during booting.

- After the boot code is compiled, we must write this boot code at the first sector of disk (sector zero) because after computer is started, bios will look for the boot code at first sector of bootable disk in order of priority specified in bios settings.

***Note:It is better to test the boot code via a floppy disc. Otherwise, you have to install operating system's bootstrap to rescue your system on a disaster. This is a difficult job!**

After this information, it is time to write a simple boot code.

```
;Mini bootstrap
    ;mini.asm

        loop:
                jmp loop

        times 510-($-$$) db 0
        dw 0AA55h
```

We use nasm compiler to compile this assembly code.
To compile with nasm issue this command:

```
$ nasm -f bin mini.asm -o miniboot
```

To write floppy disk, this command should be given:

```
$ dd if=miniboot of=/dev/fd0
```

The boot code denoted above is loaded at 07C00h adress of the memory by the bios. While booting, you don't get any output messages printed onto screen. The boot code executes only a forever loop via 'jmp loop' line. I explain at the end of the document how to boot 'hello world' code.

```
times 510-($-$$) db 0
        dw 0AA55h
```

We are increasing the size of the boot code to 512 bytes by filling it with zeros. Boot code must 512 bytes long as I previously said. 'times' is defined only in nasm. First line of the code writes 510 bytes that consists of zeros. AA55H covers two bytes and makes the sum of bytes 512 in total.

We will have problems when we want to use variables. Because we could not know the values of the segments in this implementation of the boot code.

To solve this problem, we have to add these lines to the boot code:

```
mov ax, cs
mov ds, ax
mov es, ax
```

In the boot code, we set values of data segment and extra data segment as value of code segment. Hence, our boot code looks like variables at the same time. When you compile and run this code, you will not see differences between the previos boot code and the new one. However, this implementation is necesaary to access variables.

Now, let me show how we can boot 'hello world' program.

- How to boot Hello World?

Before I explained the codes, I will give information shortly about how to boot program. As it was showed before, bios only copied the boot code to the memory at 07C00h address and this code entered into a forever loop.
Now, we need to copy the boot code to the first sector of the floppy disc (sector zero) and the hello world code to the second sector of floppy disc (sector one).
The other thing we need to do is to place hello world program (or any program you want) into specific memory adress by using bios interrupts.
As I refered before , the boot code must be in first sector of floppy, although other programs can placed anywhere. Indeed, we do not have to copy the program to the first sector of floppy. The important thing is that we must know which sectors the program was written and how many sectors will be read by the program.

By the way, bios interrupts provide accessing and using devices. The interrupt 10h was used to print messages to the screen in the hello world program. Registers of the processor must be assigned as specific values before using an interrupt. After that the interrupt can be called to do something.

You can see the list of interrupts and how the interrupts can be use though this web site:

http://en.wikipedia.org/wiki/BIOS_call

Now, we can deal with our hello boot code.

```
;*****************boot.asm ***************

[ORG 0]

        jmp reset

reset:
        mov ax, 0      ;ax and floppy device resets
        mov dl, 0
        int 13h
        jc reset

floppy:
        mov ax, 1000h        ;memory adress which hello boot program will be placed
        mov es, ax
        mov bx, 0

        mov ah, 2      ;copy program to specified memory address
        mov al, 1      ;read one sector from floppy disk
        mov ch, 0      ;cylinder = 0
        mov cl, 2      ;sector = 2
        mov dh, 0      ;head = 0
        mov dl, 0      ; show us this is floppy driver
        int 13h

        jc floppy      ;try read again if any error

        jmp 1000h:0000      ;to jump memory address where hello world is

times 510-($-$$) db 0
dw 0AA55h

;*****************************************
```

To compile:

```
        $ nasm -f bin boot.asm -o progboot
```

To write floppy disk

```
        # dd  if=progboot of=/dev/fd0
```

```
        mov ax, 1000h
        mov es, ax
        mov bx, 0
```

Memory addres is specified in the es register to indicate where hello world program will be installed. Hello boot program can be installed at any addresses except A000:0000 – FFFF:000F addresses where the bios is installed. The bx register includes the ofset value of this address. Our program will be placed at 1000:0000 memory address.

```
mov ah, 2
mov al, 1
mov ch, 0
mov cl, 2
mov dh, 0
mov dl, 0
int 13h
```

The first line of the code (move ah,2) indicates that data will be read into specified memory area. Furthermore, the al register shows the amount of the sector which will be read. Reading one sector from floppy disk is enough, since hello boot program is less than 512 byte. An outstanding point is that the cl register is assigned as 2 to imply the sector number. However we will write hello word program to the sector which is numbered as 1 of the floppy. First sector number is zero in floppy disk, but sector numbers start from one not zero in bios routine. For this reason, we assigned the cl as 2. Finally, we performed reading the sector from disk by calling bios interrupt (int 13h).

```
jmp 1000h:0000
```

Via this command, boot process will jump to the memory that hello.asm was installed. We had clarified the memory location which hello.asm will be installed in "floppy" tags with es and bx registers.
"jmp 1000h:0000" will jump to "jmp main" line in hello.asm.

```
;******************** hello.asm *********
[ORG 0]

jmp main

msg db 'Hello World'

main:
        mov si, msg   ;for writing a message
        mov ax, cs
        mov ds, ax
        mov es, ax
write:
        lodsb         ;copy character to AL register
        cmp al,0      ;check whether it is the end of the message
        je loops

        mov ah, 0Eh ;write character to screen
        mov bx, 7
        int 10h

        jmp write
```

loops:
jmp loops

;**

To compile :
$ nasm –f bin hello.asm –o hello

To copy to floppy disk:
dd seek=1 conv=sync if=hello of=/dev/fd0

mov si, msg

Si register is index register that used to show content of a message in data segment.
In this way, the content of the msg variable can be displayed.

mov ax, cs
mov ds, ax
mov es, ax

Here, as we explained before, registers are being updated to access the defined variables. If you look at boot.asm, this was not done because of any variables' not being defined. However, there is the "msg" variable, thus registers in hello.asm must be updated.

Write:
.
.
.
mov ah, 0Eh
mov bx, 7
int 10h

jmp write
.
.

On the upper code, the string is being written character by character onto screen via bios interrupt. We decide whether it is the end of the string or not by using the al register. 0Eh value which is assigned to the ah register clarifies that it will write something onto the screen.