

Babystep2

From OSDev Wiki

Writing a message using the BIOS

Quick review:

1. Boot sector loaded by BIOS is 512 bytes
2. The code in the boot sector of the disk is loaded by the BIOS at 0000:7c00
3. Machine starts in Real Mode (http://www.osdev.org/wiki/Real_Mode)
4. Be aware that the CPU is being interrupted unless you issue the CLI assembly command

Babystep2: Writing a message using the BIOS

Tutorial

Previous
Babystep1

Next
Babystep3

Many (but not all) BIOS interrupts expect DS to be filled with a Real Mode segment value. This is why many BIOS interrupts won't work in protected mode. So if you want to use int 10h/ah=0eh to print to the screen, then you need to make sure that your seg:offset for the characters to print is correct.

In real mode, addresses are calculated as $\text{segment} * 16 + \text{offset}$. Since offset can be much larger than 16, there are many pairs of segment and offset that point to the same address. For instance, some say that the bootloader is loaded at 0000:7C00, while others say 07C0:0000. This is in fact the same address: $16 * 0x0000 + 0x7C00 = 16 * 0x07C0 + 0x0000 = 0x7C00$.

It doesn't matter if you use 0000:7c00 or 07c0:0000, but if you use ORG you need to be aware of what's happening. By default, the start of a raw binary is at offset 0, but if you need it you can change the offset to something different and make it work. For instance the following snippet accesses the variable msg with segment 0x7C0.

Asm Example:

```
; boot.asm
    mov ax, 0x07c0
    mov ds, ax

    mov si, msg
ch_loop: lodsb
    or al, al ; zero=end or str
    jz hang ; get out
    mov ah, 0x0E
    int 0x10
    jmp ch_loop
```

```

hang:
    jmp hang

msg    db 'Hello World', 13, 10, 0
    times 510-($-$$) db 0
    db 0x55
    db 0xAA

```

Here is the ORG version. This time, msg is accessed with segment 0. Note that you still need to tell DS what to be as it can hold any value.

```

[ORG 0x7c00]

    xor ax, ax ; make it zero
    mov ds, ax

    mov si, msg
ch_loop: lodsb
    or al, al ; zero=end of string
    jz hang ; get out
    mov ah, 0x0E
    int 0x10
    jmp ch_loop

hang:
    jmp hang

msg    db 'Hello World', 13, 10, 0

    times 510-($-$$) db 0
    db 0x55
    db 0xAA

```

Procedures

To save on writing space, the typical 'procedures' are often separated from the code using CALL/RET like the following:

```

[ORG 0x7c00]

    xor ax, ax ;make it zero
    mov ds, ax

    mov si, msg
    call bios_print

hang:

```

```

    jmp hang

msg    db 'Hello World', 13, 10, 0

bios_print:
    lodsb
    or al, al    ;zero=end of str
    jz done     ;get out
    mov ah, 0x0E
    int 0x10
    jmp bios_print
done:
    ret

times 510-($-$$) db 0
db 0x55
db 0xAA

```

For some inexplicable reason, loading SI **then** jumping to the procedure always bugged me. Fortunately for psychos like me NASM's macros let you pretend that you are passing a parameter (macro definitions has to go before it's being called).

```

%macro BiosPrint 1
    mov si, word %1
ch_loop:lodsb
    or al, al
    jz done
    mov ah, 0x0E
    int 0x10
    jmp ch_loop
done:
%endmacro

[ORG 0x7c00]
    xor ax, ax
    mov ds, ax

    BiosPrint msg

hang:
    jmp hang

msg    db 'Hello World', 13, 10, 0

times 510-($-$$) db 0
db 0x55
db 0xAA

```

And in case your code is becoming long and unreadable, you can break it up into different files, then include the files at the beginning of you main code. Like so:

```
jmp main

%include "othercode.inc"

main:
    ; ... rest of code here
```

Don't forget the jmp main at the start - otherwise some random other procedure will get called.

Retrieved from "<http://wiki.osdev.org/index.php?title=Babystep2&oldid=15862>"

Category: Babystep

-
- This page was last modified on 7 April 2014, at 08:43.
 - This page has been accessed 62,895 times.