8086 assembler tutorial for beginners (part 11)

making your own operating system

Usually, when a computer starts it will try to load the first 512-byte sector (that's Cylinder **0**, Head **0**, Sector **1**) from any diskette in your **A:** drive to memory location **0000h:7C00h** and give it control. If this fails, the BIOS tries to use the MBR of the first hard drive instead.

This tutorial covers booting up from a floppy drive, the same principles are used to boot from a hard drive. But using a floppy drive has several advantages:

- you can keep your existing operating system intact (windows, dos, linux, unix, be-os...).
- it is easy and safe to modify the boot record of a floppy disk.

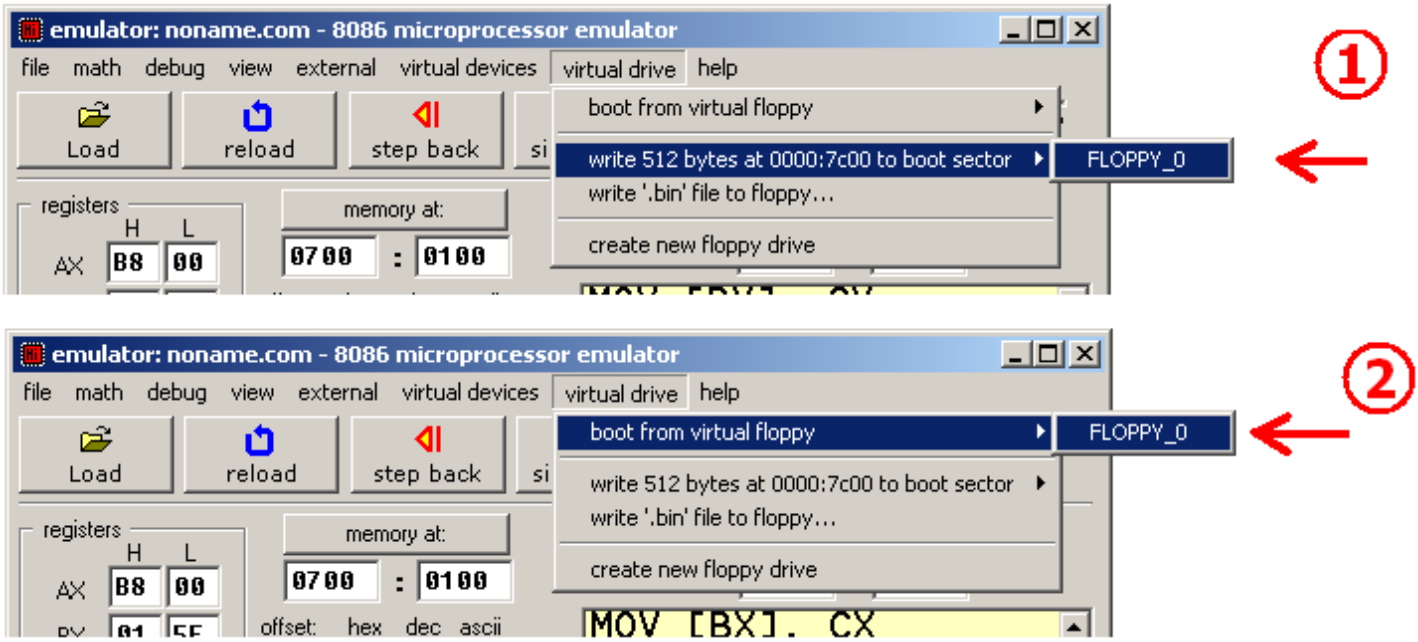
example of a simple floppy disk boot program:

```
; directive to create BOOT file:
#make_boot#

; Boot record is loaded at 0000:7C00,
; so inform compiler to make required
; corrections:
ORG 7C00h

PUSH    CS   ; make sure DS=CS
POP     DS

; load message address into SI register:
LEA SI, msg

; teletype function id:
MOV AH, 0Eh

print:   MOV AL, [SI]
        CMP AL, 0
        JZ done
        INT 10h   ; print using teletype.
        INC SI
        JMP print

; wait for 'any key':
done:     MOV AH, 0
          INT 16h
```

```
; store magic value at 0040h:0072h:
;   0000h - cold boot.
;   1234h - warm boot.
MOV     AX, 0040h
MOV     DS, AX
MOV     w.[0072h], 0000h ; cold boot.

JMP 0FFFFh:0000h        ; reboot!


new_line EQU 13, 10

msg DB  'Hello This is My First Boot Program!'
    DB  new_line, 'Press any key to reboot', 0
```

copy the above example to the source editor and press **emulate**. the emulator automatically loads **.bin** file to **0000h:7C00h** (it uses supplementary .binf file to know where to load).

you can run it just like a regular program, or you can use the **virtual drive** menu to **write 512 bytes at 7c00h to boot sector** of a virtual floppy drive (it's "**FLOPPY_0**" file in Emulator's folder). after your program is written to the virtual floppy drive, you can select **boot from floppy** from **virtual drive** menu.

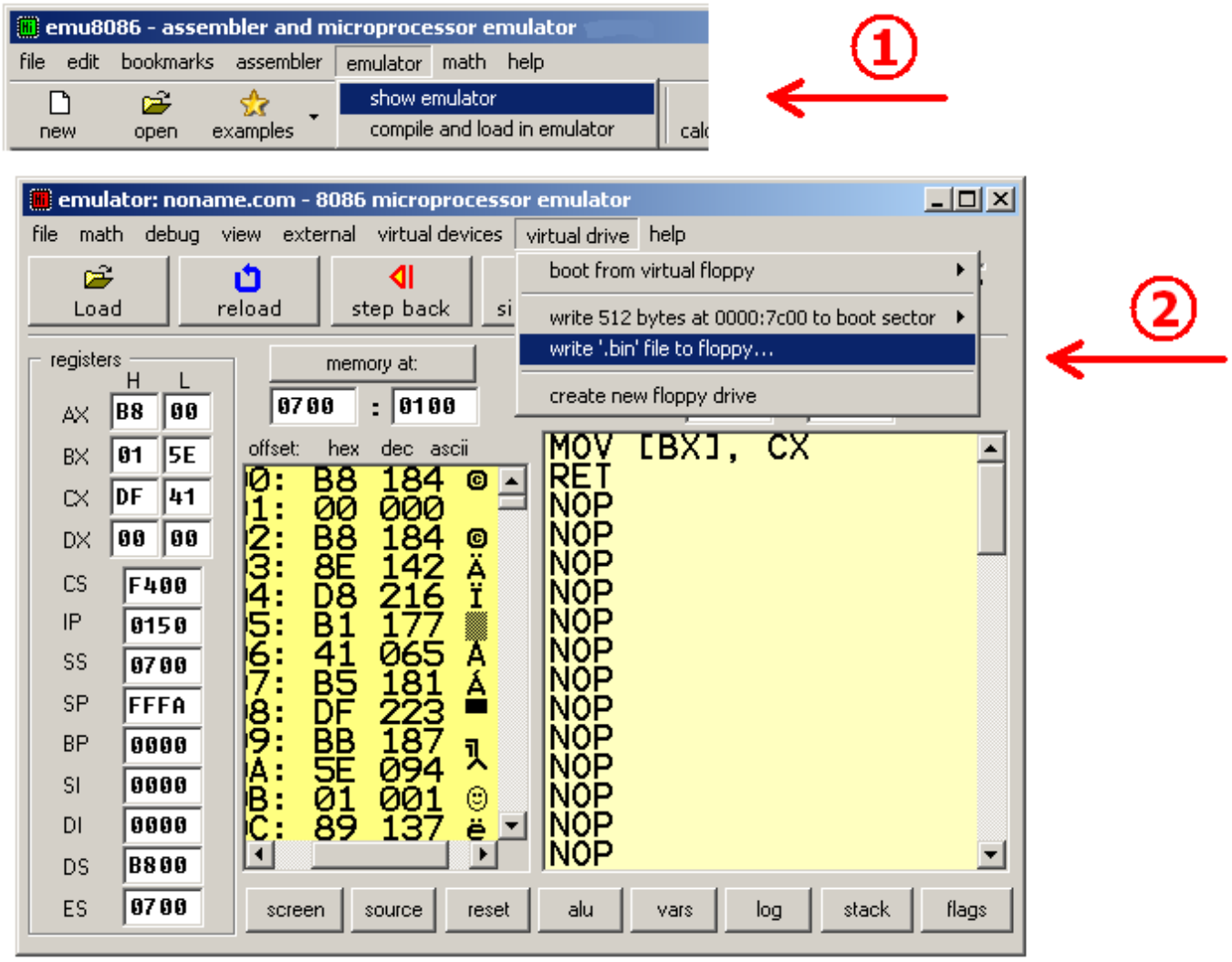


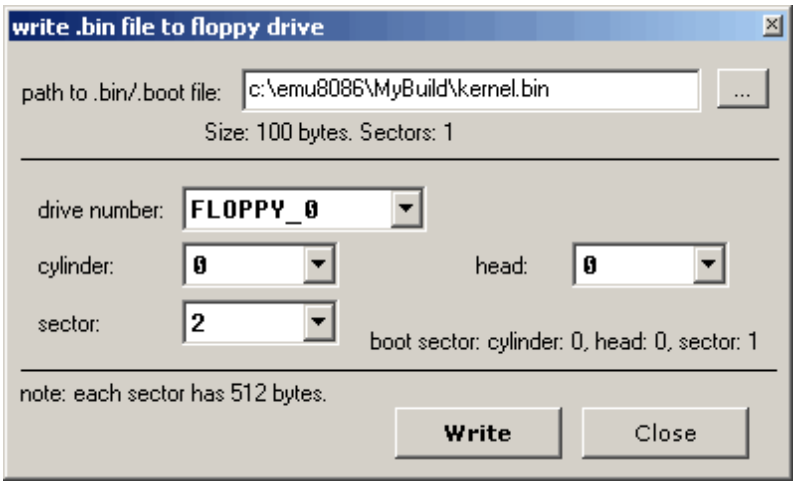
.bin files for boot records are limited to 512 bytes (sector size). if your new operating system is going to grow over this size, you will need to use a boot program to load data from other sectors (just like *micro-os_loader.asm* does). an example of a tiny operating system can be found in c:\emu8086\examples:

micro-os_loader.asm
micro-os_kernel.asm

To create extensions for your operating system (over 512 bytes), you can use additional sectors of a floppy disk. It's recommended to use "**.bin**" files for this purpose (to create "**.bin**" file select "**BIN Template**" from "**File**" -> "**New**" menu).

To write "**.bin**" file to virtual floppy, select **"Write .bin file to floppy..."** from **"Virtual drive"** menu of emulator, you should write it anywhere but the boot sector (which is Cylinder: **0**, Head: **0**, Sector: **1**).





you can use this utility to write **.bin** files to virtual floppy disk ("**FLOPPY_0**" file), instead of "**write 512 bytes at 7c00h to boot sector**" menu. however, you should remember that **.bin** file that is designed to be a boot record should always be written to cylinder: **0**, head: **0**, sector: **1**

Boot Sector Location:

Cylinder: 0
Head: 0
Sector: 1

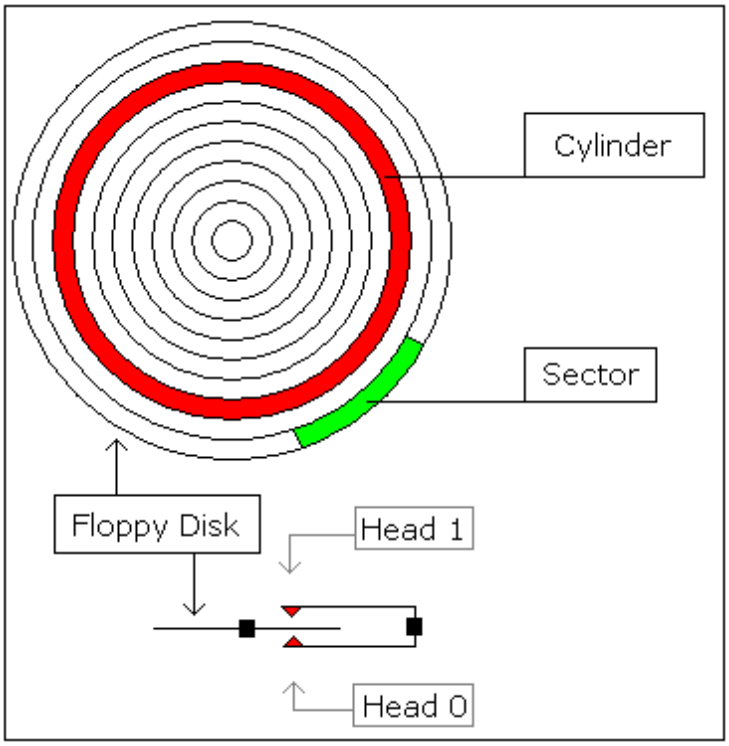
to write .bin files to real floppy disk use writebin.asm, just compile it to com file and run it from command prompt. to write a boot record type: **writebin loader.bin** ; to write kernel module type: **writebin kernel.bin /k**
/k - parameter tells the program to write the file at sector 2 instead of sector 1. it does not matter in what order you write the files onto floppy drive, but it does matter where you write them.

---

mote: this boot record is not MS-DOS/Windows compatible boot sector, it's not even Linux or Unix compatible, operating system may not allow you to read or write files on this diskette until you re-format it, therefore make sure the diskette you use doesn't contain any important information. however you can write and read anything to and from this disk using low level disk access interrupts, it's even possible to protect valuable information from the others this way; even if someone gets the disk he will probably think that it's empty and will reformat it because it's the default option in windows operating system... such a good type of self destructing data carrier :)

---

idealized floppy drive and diskette structure:



for a **1440 kb** diskette:

- floppy disk has 2 sides, and there are 2 heads; one for each side (**0..1**), the drive heads move above the surface of the disk on each side.
- each side has 80 cylinders (numbered **0..79**).
- each cylinder has 18 sectors (**1..18**).
- each sector has **512** bytes.
- total size of floppy disk is: 2 x 80 x 18 x 512 = **1,474,560** bytes.

note: the MS-DOS (windows) formatted floppy disk has slightly less free space on it (by about 16,896 bytes) because the operating system needs place to store file names and directory structure (often called FAT or file system allocation table). more file names - less disk space. the most efficient way to store files is to write them directly to sectors instead of using file system, and in some cases it is also the most reliable way, if you know how to use it.

to read sectors from floppy drive use **INT 13h / AH = 02h**.

<<< previous part <<< **>>> Next Part >>>**