PC MBR analysis

Nasser M. Abbasi

sometime in 2000 Compiled on January 30, 2024 at 11:57pm

These are some notes while I was learning about MBR and how linux uses it on intel PC's

1 looking at MBR

```
$dd if=/dev/hda of=MBR bs=512 count=1

$od --address-radix=d -t x MBR
0000000 006cebfa 494c0000 00014f4c 00b50014
0000016 00000000 37d220a6 64802627 80262801
0000032 26260164 01016480 00000000 2a000000
0000048 01648026 7e80172e 80172f01 1730017e
0000064 31017e80 017e8017 7e801732 80173301
0000080 1734017e 35017e80 017e8017 00000000
0000096 00000000 00000000 00000000 b8000000
0000112 d88e07c0 006a068c 00683689 006c1e89
0000128 006e1688 8e9a00b8 0100b9c0 ff29f629
0000144 eaa5f3fc 9a000098 8ed88efa b000bcc0
0000160 8e9000b8 0db0fbd0 b00057e8 0052e80a
0000176 4de84cb0 0034be00 fc1000bb adc189ad
0000192 c809c289 e8462074 06720043 0200c381
0000208 b050eaeb 002ae820 e8e08858 c0310012
0000224 13cdc288 49b0cfeb ea0017e8 9b000000
0000240 04e8c050 580001e8 30040f24 02723a3c
0000256 ff300704 10cd0eb4 5b595ac3 40c2f6c3
0000272 e2805474 525153bf 13cd08b4 f088eb72
0000288 7316885a 30f28801 cd8651f6 c5d0c5d0
0000304 8903e580 5901710e f63fe183 93c801e1
0000320 92f3f758 c4fef1f6 01742688 8ad68892
0000336 3b017316 77017106 d0c48613 0ac8d0c8
0000352 89017406 01b85bc1 c313cd02 f9c0315b
```

```
0000368 000000c3 00000000 00000000 00000000
0000384 00000000 00000000 00000000 00000000
*
0000432 00000000 90900000 90909090 01809090
0000448 fe830001 003fbfff 53810000 000000eb
0000464 fe82c0c1 53c0fcff f3fd00eb 0000000e
0000480 00000000 00000000 00000000 00000000
0000496 00000000 00000000 00000000 aa550000
0000512
```

Let look at the partition table, bytes 446-510

```
0000432                            809090
0000448 fe830001 003fbfff 53810000 000000eb
0000464 fe82c0c1 53c0fcff f3fd00eb 0000000e
0000480 00000000 00000000 00000000 00000000
0000496 00000000 00000000 00000000
```

This is made up of 4 entries, each is 16 bytes. The structure of a partition table entry is

```
byte 1 : boot flag. 0 not active, 0x83 active
byte 2 : begin head number.
byte 3 : begin sector number of boot sector
byte 4 : cylinder number of boot sector
byte 5 : system id. 0x83 linux, 0x82 linux swap
byte 6 : end head number.
byte 7 : end sector number.
byte 8 : end cyl. number.
byte 9-12: relative sector number of start sector.
byte 13-16: number of sectors in partition.
```

So, given the above, we now look at each partition entry. The first is

```
0000432                            809090
0000448 fe830001 003fbfff 53810000 00
```

The above means that the partition is active (first byte is 0x80)

```
byte 2 : begin head number = 0x90 =  144.
byte 3 : begin sector number of boot sector = 0x90 = 144.
byte 4 : cylinder number of boot sector = 0xFE = 254
byte 5 : system id. 0x83 linux.
```

```
byte 6 : end head number = 0
byte 7 : end sector number = 1
byte 8 : end cyl. number = 0
byte 9-12: relative sector number of start sector 0x3FBFFF53
byte 13-16: number of sectors in partition 0x81000000
```

The second partition entry is

```
0000448                                0000eb
0000464 fe82c0c1 53c0fcff f3fd00eb 00
```

The above means that the partition is not active (first byte is 0x00)

```
byte 2 : begin head number = 0
byte 3 : begin sector number of boot sector = 0xEB = 235
byte 4 : cylinder number of boot sector = 0xFE = 254
byte 5 : system id. 0x82 linux swap
byte 6 : end head number = 0xc0 = 192
byte 7 : end sector number = 0xc1 = 193
byte 8 : end cyl. number = 0x53 = 83
byte 9-12: relative sector number of start sector 0xc0fcfff3
byte 13-16: number of sectors in partition 0xfd00eb00
```

We see that those are the only 2 paritions on that disk (/dev/hda). lets see what fdisk says

```
>fdisk /dev/hda

Command (m for help): p

Disk /dev/hda: 255 heads, 63 sectors, 1021 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1   *         1       960   7711168+  83  Linux
/dev/hda2           961      1021    489982+  82  Linux swap
```

we see that the first partition entry in the partition table says it is active, boot flag is '*'. (ie. code 0x83), and the system ID is 0x83, and the second entry is not actie, and system ID is 0x82. which matches the dump of the MBR.

```
for /dev/hda1:
end cyl. number is 960 = 0x3c0

for /dev/hda2:
start cyl number is : 961 = 0x3c1
end cyl number = 1021 = 0x3fd
```

The above start/end are the cyl numbers. To look at the start/end of partition in sector numbers

```
>fdisk -u /dev/hda

Command (m for help): p

Disk /dev/hda: 255 heads, 63 sectors, 1021 cylinders
Units = sectors of 1 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1   *        63  15422399   7711168+  83  Linux
/dev/hda2      15422400  16402364    489982+  82  Linux swap

for /dev/hda1:
start = 63 = 0x3f
end = 15422399 = 0xEB53BF

for /dev/hda2
start = 15422400 = 0xeb53c0
end = 16402364 = 0xfa47bc
```

2 booting Linux

1. power PC, PC runs in real mode, and jumps to 0XFFFF0. ROM-BIOS address.
2. BIOS initalizes interrupt vector at address 0.
3. BIOS loads boot sector (MBR) to 0x7C00, and jumps to it.
4. Linux bootsector is `usr/src/linux/arch/i386/boot/bootsect.S`, bootsect start at label `_start:`

   First thing it does is move the boot sector (itself) (512 bytes) to 0x9000 using this code below. we are still in real 16bit(?) mode.

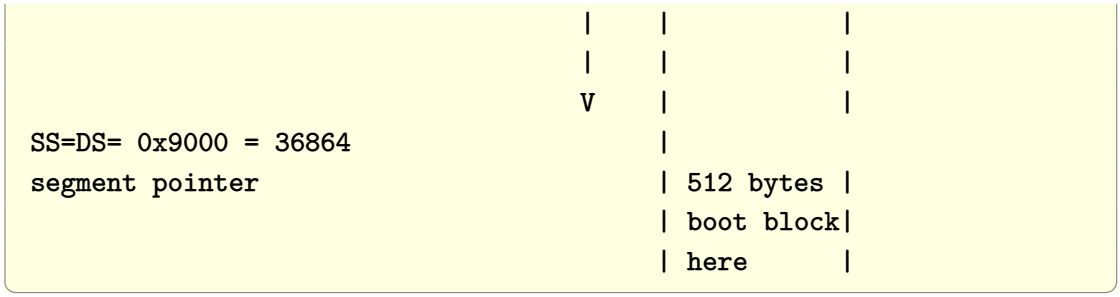
```
_start:
        movw    $BOOTSEG, %ax  ; accumlator now contains BOOTSEG = 0x07C0
        movw    %ax, %ds       ; ds, data segment register now contains 0x07C0
        movw    $INITSEG, %ax  ; INITSEG = 0x9000 from src/linux/include/asm/boot.h
        movw    %ax, %es       ; es, extra segment register now contains 0x9000
        movw    $256, %cx      ; counting register now contains  256
        subw    %si, %si       ; source index register  = 0
        subw    %di, %di       ; destiny index register = 0
        cld                    ; clear direction flag, so that after each
                               ; rep the si and di registers are automatically
                               ; incremented instead of decremented.
        rep                    ; repeat the following instruction while
                               ; counting register cx not zero.
        movsw                  ; move word string from location addressed by
                               ; source index register (si) to location addressed by
                               ; data segment register (di).
                               ; Notice: SI and DI are offset values into segments.
                               ; SI is an offset into segment identified by DS register
                               ; and DI offset into segment identified by ES register.
                               ; i.e we move bytes from ds:si to es:di .
                               ; After each movsw instruction
                               ; the source index register (SI) and destination
                               ; index register (DI) are incremented by number of
                               ; bytes moved, and the counting register is decremented
                               ; by one.
                               ; This will cause 256 words to be moved. each word
                               ; is 16 bits, or 2 bytes, so this results in moving
                               ; 512 bytes, which is the MBR, to start at address
                               ; 0x9000
        ljmp    $INITSEG, $go  ; transfer far (since to a different segment)
```

5. Now, we set stack as such

```
sp ----> 0x4000-12
ds = 0x9000
ss = 0x9000 (segment register).
```

So, stack is at 0x9000:0x4000-12.

```
SP= 0x4000-12 = 16384-12 = 16372  ------> |           |
                                     ^    |           |
```



To be continued....