Introduction

Package content is intended to get an USB Pendrive/Flash Stick bootable either as ZIP/LS120 (Super-Floppy) Drive or as Harddisk. GRLDR (Grub4Dos) is used as Bootmanager. Unfortunately it is not foreseeable, how BIOS categorizes the USB stick or better, what criteria it uses to fix its categorie. So the package additionally contains some customizable tools to find out, what BIOS thinks. You further need an appropriate HexEditor, which may directly read (and write to) storage media. It would be great, if you were familiar with "debug" (or a better Assembler program) and X86-Assembler-Language. You find appropriate information and download links in Appendix.

Content of ZIP-Package

MBR-SPY.REC, FLOPPY.PBR , MINI.MBR , G4D_FAT32.PBR, MENU.LST and this Manual

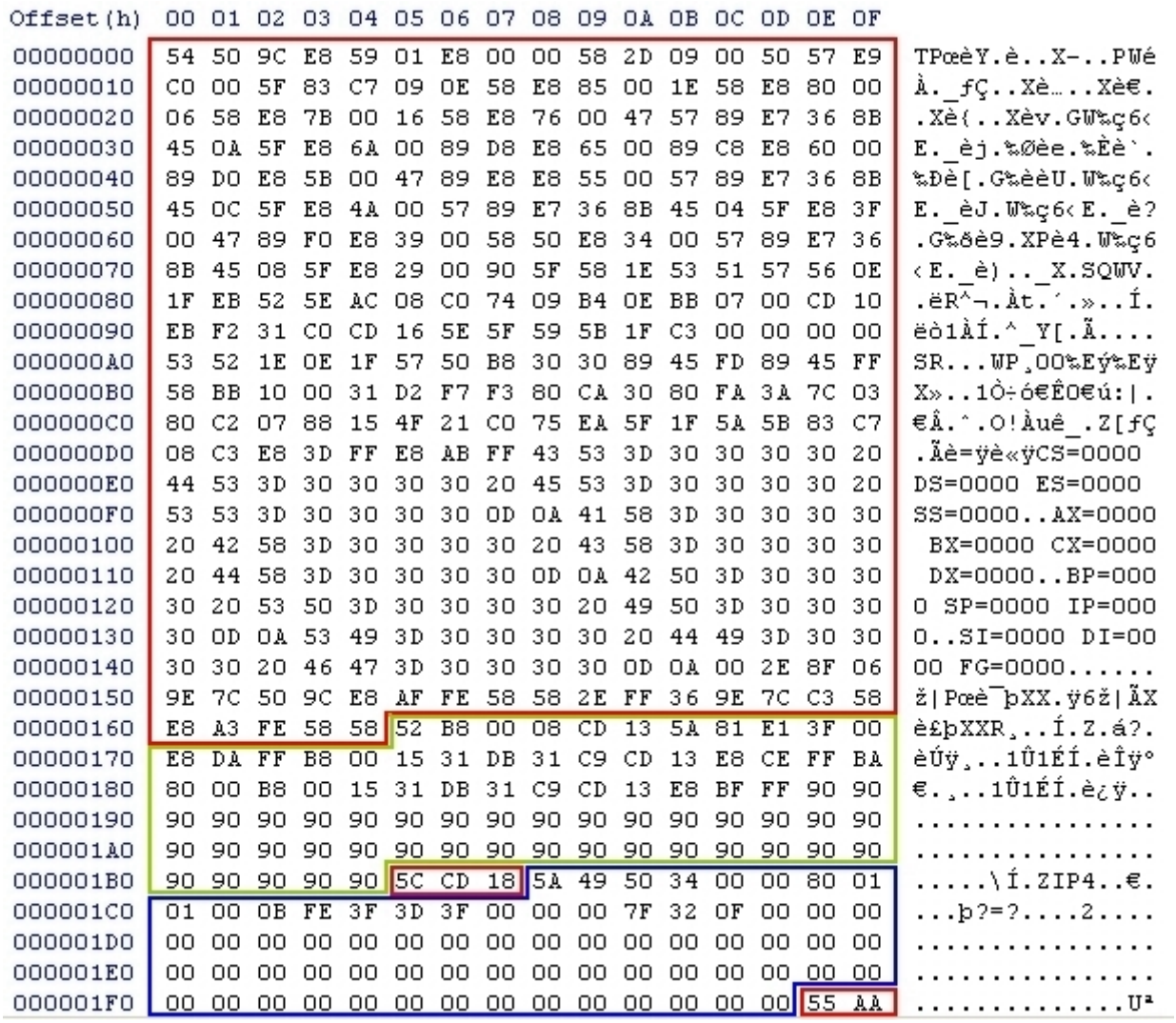
Table of Content

1. MBR-SPY.REC

You need a HexEditor (see Appendix) to handle this file. It contains X86 Machine code, which shows the values of (real mode 16 bit) Processor Registers and has a free area, where you may place your own test code. I strongly suggest, that you backup the existing MBR just by copying it to the next sector before you change anything at the first sector. With a simple relative "call 024D" (includes "debug" offset 100h; with offset 0 it is 014D) you launch the screen output within your code. You need to be familiar with X86 Assembler for coding . MBR-SPY.REC has a size of 440 Bytes, i.e. it doesn't overwrite Disk Signature and Partition table, if copied to the first Byte of Sector 1 (LBA 0; HxD counts from 0). If your aim is to reveal ZIP/LS120 booting, it could make sense to erase Disk signature and Partition table, but not "Boot magic" (55AA, last two Bytes of sector, which is number AA55h /Little Endian).

It is strongly recommended, that you shutdown your system (power off NOT hibernation) and look into BIOS (boot sequence) at startup, before you try a changed MBR. A "warm" reboot may NOT recognize the changes you made or your USB Stick may get ignored.

Display routine waits for a key after each call. It returns to BIOS at last, and BIOS boots the next device in Boot sequence. So you should not have a crash, unless you coded nonsense. Keep in mind, that this sector works at RAM address 0000:7C00, while booting... and it's NOT moved. You should prefer relative addressing, whenever it's possible. Further note, that "debug" starts at byte 0100h, NOT 0000h (0100h -> 7C00h). If you want to load a disk sector to RAM, use 0000:7E00 as target buffer - the next free RAM space in same segment. "debug" doesn't know all 16 bit real mode Assembler mnemonics; a short table of X86 mnemonics is downloadable, see Appendix. So disassembled code might look "queer" sometimes - as do "disassembled" strings (have a look at the dump screen !) or truncated Multi-Byte-Commands.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	54	50	9C	E8	59	01	E8	00	00	58	2D	09	00	50	57	E9	TPœèY.è..X-..PWé
00000010	C0	00	5F	83	C7	09	0E	58	E8	85	00	1E	58	E8	80	00	À._ƒÇ..Xè…...Xè€.
00000020	06	58	E8	7B	00	16	58	E8	76	00	47	57	89	E7	36	8B	.Xè{..Xèv.GW‰ç6‹
00000030	45	0A	5F	E8	6A	00	89	D8	E8	65	00	89	C8	E8	60	00	E._èj.‰Øèe.‰Èè`.
00000040	89	D0	E8	5B	00	47	89	E8	E8	55	00	57	89	E7	36	8B	‰Ðè[.G‰èèU.W‰ç6‹
00000050	45	0C	5F	E8	4A	00	57	89	E7	36	8B	45	04	5F	E8	3F	E._èJ.W‰ç6‹E._è?
00000060	00	47	89	F0	E8	39	00	58	50	E8	34	00	57	89	E7	36	.G‰ðè9.XPè4.W‰ç6
00000070	8B	45	08	5F	E8	29	00	90	5F	58	1E	53	51	57	56	0E	‹E._è).._X.SQWV.
00000080	1F	EB	52	5E	AC	08	C0	74	09	B4	0E	BB	07	00	CD	10	.ëR^¬.Àt.´.»..Í.
00000090	EB	F2	31	C0	CD	16	5E	5F	59	5B	1F	C3	00	00	00	00	ëò1ÀÍ.^_Y[.Ã....
000000A0	53	52	1E	0E	1F	57	50	B8	30	30	89	45	FD	89	45	FF	SR...WP¸00‰Eý‰Eÿ
000000B0	58	BB	10	00	31	D2	F7	F3	80	CA	30	80	FA	3A	7C	03	X»..1Ò÷ó€Ê0€ú:|.
000000C0	80	C2	07	88	15	4F	21	C0	75	EA	5F	1F	5A	5B	83	C7	€Â.ˆ.O!Àuê_.Z[ƒÇ
000000D0	08	C3	E8	3D	FF	E8	AB	FF	43	53	3D	30	30	30	30	20	.Ãè=ÿè«ÿCS=0000
000000E0	44	53	3D	30	30	30	30	20	45	53	3D	30	30	30	30	20	DS=0000 ES=0000
000000F0	53	53	3D	30	30	30	30	0D	0A	41	58	3D	30	30	30	30	SS=0000..AX=0000
00000100	20	42	58	3D	30	30	30	30	20	43	58	3D	30	30	30	30	BX=0000 CX=0000
00000110	20	44	58	3D	30	30	30	30	0D	0A	42	50	3D	30	30	30	DX=0000..BP=000
00000120	30	20	53	50	3D	30	30	30	30	20	49	50	3D	30	30	30	0 SP=0000 IP=000
00000130	30	0D	0A	53	49	3D	30	30	30	30	20	44	49	3D	30	30	0..SI=0000 DI=00
00000140	30	30	20	46	47	3D	30	30	30	30	0D	0A	00	2E	8F	06	00 FG=0000.....
00000150	9E	7C	50	9C	E8	AF	FE	58	58	2E	FF	36	9E	7C	C3	58	ž|Pœè¯þXX.ÿ6ž|ÃX
00000160	E8	A3	FE	58	58	52	B8	00	08	CD	13	5A	81	E1	3F	00	è£þXXR¸..Í.Z.á?.
00000170	E8	DA	FF	B8	00	15	31	DB	31	C9	CD	13	E8	CE	FF	BA	èÚÿ¸..1Û1ÉÍ.èÎÿº
00000180	80	00	B8	00	15	31	DB	31	C9	CD	13	E8	BF	FF	90	90	€.¸..1Û1ÉÍ.è¿ÿ..
00000190	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	.....
000001A0	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	.....
000001B0	90	90	90	90	90	5C	CD	18	5A	49	50	34	00	00	80	01	.....\Í.ZIP4..€.
000001C0	01	00	0B	FE	3F	3D	3F	00	00	00	7F	32	0F	00	00	00	...þ?=?....2....
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA	.....Uª

Image 1: MBR_SPY.REC

Custom code area reaches from Byte 165h to Byte 1B4h (80 Bytes; pretty much for Machine code). The present code calls three times Interrupt 13h and gives three (even four with the initial) screen outputs ! Unused Bytes should be set to 90h (NOP), since last three Bytes (5Ch CDh 18h) are essential code, too. Area within bluish margin may be erased (set to 00h; Disk Signature and Partition table) or overwritten by code. Then you have to put the above mentioned three last Bytes at the end of your code (POP SP, INT 18). But this might not be a good idea, because BIOS might take values other than 00h as Partition data. "Boot magic"(55AA) should be left untouched.

What does the sample code do ?

```
0000:7C65 52            PUSH    DX          Put DX value on Stack; DL=00h→Floppy
                                            or 80h→Harddisk; given by BIOS)
0000:7C66 B80008        MOV     AX,0800     Function AH=08h (Query geometry of
                                            Boot medium from BIOS)
0000:7C69 CD13          INT     13          Call Interrupt 13h (Disk services)
0000:7C6B 5A            POP     DX          Restore DX from Stack
0000:7C6C 81E13F00      AND     CX,003F     Filter lower 6 bits of CX: Sect./track
0000:7C70 E8DAFF        CALL    024D        Show Register content (Look at CX !)
```

```
0000:7C73 B80015        MOV     AX,1500   AH=15h (Query drive type)
0000:7C76 31DB          XOR     BX,BX     BX → 0000h
0000:7C78 31C9          XOR     CX,CX     CX → 0000h
0000:7C7A CD13          INT     13        result in AH:02h→removable;03h→fixed
0000:7C7C E8CEFF        CALL    024D      Show Register content (Look at AX,left Byte)
0000:7C7F BA8000        MOV     DX,0080   DL=80h (fixed disk)
0000:7C82 B80015        MOV     AX,1500   same procedure as above; might be
                                          identical, if DL had this value before
0000:7C85 31DB          XOR     BX,BX
0000:7C87 31C9          XOR     CX,CX
0000:7C89 CD13          INT     13
0000:7C8B E8BFFF        CALL    024D      Show Register content (Look at AX)
0000:7C8E 90            NOP               Do nothing, but go on
```

MBR-SPY first shows Register content before any code is executed, so that you instantly see, what value DX got by BIOS (the lower Byte DL is significant). This matches (or should, due to the specifications) of course the BIOS assignment, which you see, when you change Boot sequence in BIOS. Keep in mind, that you may find your USB pendrive as removable device or as harddisk. Sometimes you see a little +sign at the appropriate menu item as indicator, that there is more than one choice. Sometimes you have to enable USB at boot time by a separate menu item („USB legacy support enabled“), or your BIOS offers USB 2.0 support at boot (enable it !). Unfortunately most BIOSes use USB 1.1, which is sssllloooowww(12 Mbit/s→1.5MB/s).

If you intend to get your USB stick booting as a ZIP/LS120 drive, although it is able to boot as HDD, you have to play around with criteria, that your BIOS might use to determine its type.

1. Use a (nowadays) small stick (512 MB→1GB)
2. Play with geometry (LS 120 : C/H/S 963/8/32 120MB; ZIP : C/H/S 96/64/32 100MB or 239/64/32 250MB)
3. Use fourth primary partition and PBR at sector 20h(32); do not use more than one partition.
4. Use a PBR at sector 0 (CHS: 1) : sticks are often “shipped” this style, but NOT bootable.
5. Use one or more (up to three; at sector 0, 32 and 63) combined MBR/PBR(Grub4 Dos)
6. Do NOT use Windows MBRs. They are NOT bootable as Floppy.

I see one advantage of a bootable Superfloppy style pendrive : harddisk drive index gets NOT shifted. Your built-in bootable harddisk stays drive 80h (in real mode).

2. FLOPPY.PBR

FLOPPY.PBR is a simple non-bootable 1,44MB-Floppy Boot Record, which throws a message, when booted, waits for a pressed key and lets BIOS boot the next device in Boot sequence. Of course, you can use it as such. In this context, it serves as „verbose“ PBR. You may append the sector number, where you gonna put it, to the text string (NULL-Byte 00h means „end of string“,so don't use it inside) and get the information, which one is really used. Put one to sector 32 (20h) and the other to 63 (3Fh). Vista uses 2048 (800h), but you don't consider to launch Vista on a ZIP style USB Pendrive, do you (Vista PE works !)?

3. MINI.MBR

What does it do (not very much !) ?

1. It gets loaded and started by the BIOS at RAM location 0000:7C00h.
2. It moves the code (except for the moving routine) to 0000:061Bh.
3. It shows a busy message : „Booting USB Pendrive . . .“
4. It looks, whether it was called as Floppy or as Harddisk by the BIOS.
5. As a Floppy, it loads sectors from C/H/S 0/1/1 (hopefully PBR) to 0000:7C00h and jumps to its code.
6. When called as Harddisk, it queries the disk geometry, given by the BIOS and calculates the Sector per Track value, since PBR is regularly located at the start of the second track. Finally PBR gets loaded to RAM and takes over.
7. If there is any (disk read) fault, BIOS tries to boot next device in boot sequence.

What are the prerequisites for this minimalist design ?

1. BIOS "tells" MBR, how it boots the involved media: as floppy (DL=00h) or as harddisk (DL=80h).
2. (Super-)Floppy disk read access works through Function AH=02h of Interrupt 13h (CHS based).
3. (Super-)Floppy PBR is at C/H/S = 1/1/0 (to test before with my MBR-SPY and FLOPPY.PBR, see above)
4. HDD read access works through Function AH=42h of (extended) Interrupt 13h (LBA based).
5. Successful Reading does NOT need more than a single access attempt (not to confuse with multiple sector reading or multiple access) or a disk reset.
6. BIOS tells correctly, how it sees Disk geometry (CHS based); Sector/track value is used for (LBA) PBR location of HDD.
7. Partition table is totally ignored by MBR code (Bootflag doesn't matter; any of the primary partitions is possible).
8. File system of Boot partition is FAT32 (FS-ID 0Bh; for best compatibility and to avoid rapid flash memory wear out).
9. Modified G4D-FAT32-PBR is used to launch a slightly adapted GRLDR Boot manager

4. Modified G4D_FAT32.PBR

My modifications overwrite the following initial commands with NOPs (90h):

```
0000:7C5C B6FF          MOV     DH,FF     These commands only make sense, if code is
0000:7C5E 80FEFF        CMP     DH,FF     modified by an external program (MBR?).
0000:7C61 7502          JNZ     0165      Otherwise PBR doesn't boot Harddisks.
0000:7C63 B200          MOV     DL,00
```

PBR holds the Boot Parameter Block (BPB), which is of course dependent on your partitioning data, i.e. you have to adapt it. I marked the locations for lower (LLLL) partition border and size (SSSS) values in BPB of my PBR. My own way was simpler. I made a total backup (image) of my 512MB stick, cleared the first 100 sectors by filling them with 00h and let Windows Disk manager create a new first primary partition (size : whole stick). Then I (quick-)formatted it to FAT32. As expected, PBR was at sector 63 (3Fh). I exchanged MBR code with the help of my HexEditor as well as PBR code conserving BPB (copying in two parts). The first 11 Bytes are a jump command, a NOP and the string „GRLDR“ filled up to eight characters by three spaces (20h). Do not accidentally overwrite more Bytes at this location. The second part starts from Byte 5A (after string „FAT 32“ and three further spaces. Put the whole rest of the code

there. Look, that it ends with 55AA at the sector border.

If you need a PBR at sector 32 (20h) it's NOT more complicated. Just do the same as described above and create all at sector 63 (3Fh). Label your new created volume (8 characters,upper case), when formatting. Let your HexEditor search the disk for that label. Copy all from sector 63 to the end of the hexadecimal line, following the label string. Paste all to the beginning of sector 32. Correct the „Hidden sector“ value at Byte ..1C from 3Fh to 20h, that's all. Run a „chkdsk“ to be sure it works. Don't forget to open your USB Stick for Writing, otherwise writing item is grayed out.

5. Modified (GRLDR) internal MENU.LST

You only have to modify GRLDR's internal MENU.LST, if your stick boots as SuperFloppy, because floppies are ignored when searching for the menu file. Of course, it finds MENU.LST on your HDD (partitions), but this is an unwanted effect here. We need to have it looking with priority to the SuperFloppy content. So I added one single line to the internal list (marked fat) :

```
pxedetect
configfile
default 0
timeout 1

title find /menu.lst, /boot/grub/menu.lst, /grub/menu.lst
        errorcheck off
        root (fd0,0) || root (hd0,0)
        configfile /menu.lst
        configfile /boot/grub/menu.lst
        configfile /grub/menu.lst
        find --set-root --ignore-floppies --ignore-cd /menu.lst && configfile /menu.lst
        find --set-root --ignore-floppies --ignore-cd /boot/grub/menu.lst && configfile /boot/grub/menu.lst
        find --set-root --ignore-floppies --ignore-cd /grub/menu.lst && configfile /grub/menu.lst
        errorcheck on
        commandline

title commandline
        commandline

title reboot
        reboot

title halt
        halt
```

It's easy to exchange with the help of a HexEditor. Do NOT edit my file, because it should have UNIX line endings (0Ah, instead of 0Dh 0Ah). The internal MENU.LST is at the end of GRLDR file. So load my MENU.LST to the HexEditor and copy it totally to Clipboard, then load GRLDR, go to the end of the file and (over)write (not paste) my list exactly where the original list is located. Note, that it has a NULL-Byte (00h) at its end. This is very important as end marker. Store the changed GRLDR in Root folder of your USB stick (this location is mandatory). And don't forget to create an individual MENU.LST there as well; not to mention the programs, that have to be put on the stick.

Here is a(n external) MENU.LST sample:

---------- snip ----------

```
background 3030ff
foreground e0e090
splashimage
clear
timeout 30
default 1

title   Start OS on 1st built-in Harddisk\n * Boot, whatever boots on built-in first Harddisk *
root (fd0,0) && chainloader (hd0)+1
root (fd0,0) && boot
map --unhook
map (hd0) (hd1)
map (hd1) (hd0)
map --hook
chainloader (hd0)+1

title   W98 EBD Floppy Image\n * Unrestricted Access to MS-DOS interrupts*
map --unhook
root (fd0,0) || root (hd0,0)
map --mem /W98EBD.ima (fd1)
map --hook
map (fd0) (fd1)
map (fd1) (fd0)
map --hook
root (fd0)
chainloader +1

title   Puppy Lupu 501\n * All you need in case of Emergency *
root (fd0,0) || root (hd0,0)
kernel /lupu501/vmlinuz live-media-path=lupu501
initrd /lupu501/initrd.gz

title   GParted 0.4.5-3test\n * Boot "ISO-Hybrided" ISO-Image as HDD *
map --unhook
root (fd0,0) || find --set-root --ignore-floppies /gparted_iso/gparted-live-0.4.5-3test.iso
map --heads=0 --sectors-per-track=0 /gparted_iso/gparted-live-0.4.5-3test.iso (hd16)
map --hook
root (hd16)
chainloader +1

title   Windows Vista Repair
map --unhook
root (fd0,0) || find --set-root --ignore-floppies /VistaRvy.iso
map --heads=0 --sectors-per-track=0 /VistaRvy.iso (hd64)
map --hook
root (hd64)
chainloader /BOOTMGR
```

```
title   Start GRUB console (type "help"; return with [Esc])
commandline

title   Restart\n * Use this to change boot sequence in BIOS *
reboot

title   Shutdown\n * Power off system *
halt
```

---------- snap ----------

6. Appendix

Hints

- Be aware, that you might ruin your system, if you accidentally overwrite sectors on your harddisk. In no way shall I be made liable for any loss or damage, resulting from using my programs or given advices.
- To be clear : I do NOT expect, that my solution works on ANY USB Stick and on ANY BIOS. I further do NOT expect that you may configure ANY USB Stick arbitrarily as bootable LS120/ZIP (SuperFloppy) or Harddisk on the same system (BIOS). My aim was to get as much types of USB Sticks boot at all with Grub4Dos either way... and to be independent from M$ Boot Records !
- It is possible to partition your HDD-style SuperFloppy, but newer Windows versions show only first partition. There is a filter driver for Hitachi Microdrives, which enables access to partitions on removable drives.
- MBR-SPY does NOT actualize SP Register value as well as IP value, when called a second and any further times. Why ? PUSHs and POPs should match, before display routine is called or values might be „mixed up“. The whole program might crash even. So be careful. SP has to return to its previous value. To get the actual IP value needs so much code (there is no MOV AX,IP or PUSH IP) and the information is really not needed : when your code runs, you know what instructions are executed. FG is the Flag Register. Each bit has a special meaning. The Carry Flag is important for Interrupt 13h. It's the lowest bit, so odd values mean, that Carry Flag is set (=fault).
- The BIOS suggested logical drive parameters may NOT (and need NOT) correspond to the geometry given by your partition table as long as the Sector / Track value is identical, or you filled in the appropriate start sector value manually into the correct location within the Disk Access Packet (DAP) area of my MBR (Byte 68h to 6Fh, Little Endian style). In most cases you only need to change Byte 68h. Be aware, that this measure has only influence on a HDD booting USB pendrive. Your PBR has to cope with MBR's partition geometry! It makes no sense to waste space for a big „Hidden Sector“ area, unless you want to store something at that location. Partition managers sometimes „align borders to cylinder boundaries“. In worst case, this means a loss of 8MB disk space at the beginning and maybe at the end of the disk as well. Between different partitions cylinder alignment makes sense.
- WIN/MSDOS PBRs are NOT bootable with this kind of minimized MBR, because they use drive index from MBR (Bootflag), exchanged with other stuff via a common stack. My MBR does NOT present these informations. Win/MSDOS MBR doesn't accept floppy drive indices (the Bootflag is used as drive index to the boot medium, so always 80h; a floppy has to have no MBR, due to M$).
- You may alter the Boot message of my MBR as long as you leave ONE NULL-Byte at the end (00h), and you don't overwrite Disk Signature or -- even worse -- the Partition table. There is no code adaptation necessary. Perhaps you write a Limerick or a love poem for it? Use 0Dh 0Ah for forced Linefeed. Tab (09h) doesn't work.
- I intentionally did NOT put a screenshot of my MBR-SPY here: it's spartan; see it with your own eyes.
- If your system has an internal Floppy drive, don't be upset, if it sounds like a machine gun at (HDD) boot.

MINI.MBR explained

```
7C00 33C0          XOR     AX,AX      AX=0000h (One Byte less than mov ax,0000)
7C02 8ED0          MOV     SS,AX      SS=0000h (Stack Segment)
7C04 BC007C        MOV     SP,7C00    SP=7C00h; so Stack is at 0000:7C00h
7C07 FB            STI                Set Interrupts (enabled)
7C08 50            PUSH    AX         AX=0000h on Stack (SP=7BFE)
7C09 07            POP     ES         ES=0000h (Extra Segment gets AX value from Stack; SP=7C00h)
7C0A 50            PUSH    AX
7C0B 1F            POP     DS         DS=0000h (Data Segment ; Code Segment CS is automatically set
                                      to 0000h). So all segment registers are aligned to 0000h.
7C0C FC            CLD                Clear Direction Flag (to upwards/increment for MOVSB command)
7C0D BE1B7C        MOV     SI,7C1B    Set register SI=7C1Bh (Source Index; Segment is DS)
7C10 BF1B06        MOV     DI,061B    Set register DI=061Bh (Destination Index; Segment is ES)+
7C13 50            PUSH    AX         AX=0000h on Stack
7C14 57            PUSH    DI         DI=061Bh on Stack (prepares Far Return)
7C15 B9E501        MOV     CX,01E5    CX=01E5h (Counter Register; 200h - 1Bh = 1e5h)
7C18 F3            REPZ               Repeat following command until CX=0000h (zero), decrement CX
7C19 A4            MOVSB              Move Byte from DS:SI to ES:DI, increment both pointers)
7C1A CB            RETF               Return far to 0000:061Bh (the next command at new location!)
061B E85700        CALL    0674       Call Display Subroutine for a simple Screen Message
061E 08D2          OR      DL,DL      DL is not changed, but Flags are set : DL=00h Zero-Flag;
                                      DL=01h - 7Fh Sign-Flag +; DL=80h - FFh; Sign-Flag -
0620 7C12          JL      0634       Jump, if DL>7fh->Harddisk
0622 B82002        MOV     AX,0220    AH=02h (Read disk function); AL=20h(Read 32 sectors at once)++
0625 BB007C        MOV     BX,7C00    ES:BX=0000:7C00h (Set Read Buffer Address Pointer)
0628 B601          MOV     DH,01      DH=01h (Head number); DL=xxh (Drive index, provided by BIOS)
062A B90100        MOV     CX,0001    CX=0001h (Combined Cylinder and Sector Number)
062D CD13          INT     13         Call BIOS Interrupt 13h (Services for Storage Media)
062F 720C          JB      063D       Jump on below = Jump on carry (Carry-Flag set indicates fault)
0631 0E            PUSH    CS         CS=0000h on Stack
0632 54            PUSH    SP         SP=7C00h on Stack (Note, that DL still contains Drive index)
0633 CB            RETF               Return Far (Uses last two Stack Words for jumping to loaded
                                      PBR -->0000:7C00h)
0634 52            PUSH    DX         DX(DL=80h !) on Stack
0635 B80008        MOV     AX,0800    AH=08h (Read Drive Parameters; we only need CX as result)
0638 CD13          INT     13         Call Interrupt 13h
063A 5A            POP     DX         Restore DX from Stack
063B 7302          JNB     063F       Jump on no fault over next command
063D CD18          INT     18         Call BIOS Interrupt 18h (= Boot Next Device in Boot sequence)
063F 81E13F00      AND     CX,003F    Bit manipulation: extract lower 6 Bit of Register CX
                                      = Sect./track value as BIOS sees it; standard location of PBR)
0643 31C0          XOR     AX,AX      AX=0000h
0645 3B066806      CMP     AX,[0668]  Compare Content of AX to Word at RAM address DS:0668h (see DAP
                                      below)+++
0649 7504          JNZ     064F       Jump on Non Zero directly to Function Extended Read (AH=42h)
                                      This offers to manually set sector value, if BIOS tells
                                      nonsense.
064B 890E6806      MOV     [0668],CX  Load sect./track value to DAP (Disk Acces Packet Buffer)
064F B442          MOV     AH,42      AH=42h (Extended Read Function)
0651 BE6006        MOV     SI,0660    Set Pointer to DAP address(DS:SI=0000:0660h)
0654 CD13          INT     13         Call Interrupt 13h (Note, that DL is NOT altered.)
0656 EBD7          JMP     062F       Jump back to Carry Test and either proceed to INT 18h (fault)
                                      or jump to just loaded PBR (success)

0658                       56 30 2E 31 30                V0.10              Version
```

```
065D                                        2A 44 50                *DP           Separator
0660  10 00 20 00 00 7C 00 00-00 00 00 00 00 00 00 00   .. ..|..........           DAP
0670  45 4E 44 2A                                      END*                       Separator

0674 60            PUSHA             Put AX CX DX BX (SP) BP SI DI  on Stack
0675 EB11          JMP     0688      Common trick to get the address of Message text (Call back to 0677h)
0677 5E            POP     SI        SI points to first character of text
0678 AC            LODSB             Load Byte at DS:SI to Register AL, increment SI
0679 08C0          OR      AL,AL     Sets Zero-Flag, if AL=00h(Null-Byte; end of text)
067B 7409          JZ      0686      Jump, if end of text is reached
067D B40E          MOV     AH,0E     AH=0eh (Display one character and move cursor)
067F BB0700        MOV     BX,0007   Set Display attributes
0682 CD10          INT     10        Call Interrupt 10h ( BIOS Basic Display functions )
0684 EBF2          JMP     0678      Jump to read next character
0686 61            POPA              Get DI SI BP BX DX CX AX from Stack (SP restores
                                     automatically)
0687 C3            RET               Return to main program; see above
0688 E8ECFF        CALL    0677      "Call" puts the IP value of next command on Stack for a return
                                     jump - or better : to get its address in this special case

0680                                  20 20 42 6F 6F               Boo           Busy message
0690  74 69 6E 67 20 55 53 42-20 70 65 6E 64 72 69 76   ting USB pendriv
06A0  65 20 2E 20 2E 20 2E 00                          e . . ..
```

+) Relocation of code omits of course Relocation Routine itself (Bytes 0000h to 001Ah), which is no more needed.

++) BIOS puts MBR to RAM address 0000:7C00h, and MBR sets Stack Pointer (SP) to 0000:7C00h (decrements with every "push"). If you load multiple sectors with ONE BIOS Interrupt 13h call, you must NOT exceed segment border. So there is a maximum sector number to load at this location : (10000h - 7c00h) Bytes / 200h Bytes/Sector = 42h (= 66 Sectors or 33,792 Bytes). I used 20h (= 32), since some PBRs use more than 1 sector (not mine). Keep in mind, that this is a physical access to the storage medium, no logical. Structures have to be contiguous (what they generally are, since they are NOT Files).

+++)Disk Access Packet structure :

```
10 00          Number of Bytes, which build the packet (all Little Endian/Intel style ! 0010h=16,
               fixed)
20 00          Number of Sectors to read( 0020h=32; Maximum 7Fh=127, but only 42h=66 at this
               location, see above)
007C
0000           Transfer Buffer address in RAM (0000:7C00h)

0000           You may put an individual start sector number other than 0000h(-> infinite loop)
0000           here. It does NOT get overwritten by the BIOS value!
0000
0000           32-bit address of Start Sector number for read access
```

Links

BIOS Interrupt 13h

HxD HexEditor Download

Grub4DOS 0.4.4 2009-10-16 ZIP Download

BIOS Boot Specif. V 1.01

BIOS EDD Specif. V 3.0 R 0.8

X86AssemblerCodeTable.pdf

Assembly Tutorial