

## Boot code tutorial

- read the following code and write a simple explanation of what it is performing
  - the code is the first boot code (the role of the first boot code was described during the lecture)

boot.S

```
!
! Bootstrap loader which must be <= 512 bytes of code
! and have no data segment!
!
entry _start
!
!
! Note that the operating system must be <= 9*64k as we p
! and secondary boot code at 640k-64k..640k
!
```

## Boot code tutorial

boot.S

```
!
! The task of this piece of code is to load the secondary
! code written in Modula-2 for the 8088/8086. We also set
! ss, cs, ds registers. We set them to the same value 64
! and we know that the secondary boot code is small and w
! 64k including stack, data and code. This makes life alo
!
!
```

## Boot code tutorial

```
boot.S
TOPOFMEM      = 0xA0000      ! We assume every machin
TOPOFMEMSEG   = TOPOFMEM / 16
SIXTYFOURKSEG= 0x10000 / 16
BOTOFMEMSEG   = TOPOFMEMSEG - SIXTYFOURKSEG
STACKSIZE     = 0x1000      ! 4k of stack space

BOOTSEG       = 0x7C00 / 16 ! This is where the BIOS pu
! first sector (boot sector).

MAXSECONDSEG  = 0x8000 / 16 ! Max no of clicks of cod

STACKCLKS     = STACKSIZE / 16
STACKSEG      = TOPOFMEMSEG - STACKCLKS      ! Assign

SECONDSEG     = 0x90200 / 16 ! BOTOFMEMSEG
! Secondary boot code

SECTORSIZE    = 256 ! number of 2 byte words in a sect
SECONDSSIZE   = 14 ! max number of sectors which may
! secondary boot.

!BOOTDRIVE    = 0x00 ! floppy (/dev/fd0 or a:)
BOOTDRIVE     = 0x80 ! harddisk (USB-HDD in the BIOS)
```

## Boot code tutorial

boot.S

```
!
! we choose 14 since:
!
! (i) a 5 and 1/4 inch floppy drive has 15 sectors / t
! (ii) a 3 and 1/2 inch floppy drive has 18 sectors / t
! (iii) ROM BIOS insists that the bootsector is 1 sector
! (iv) therefore the minimum number of sectors availabl
! is 15-1 = 14
!
! (v) we could make boot more complicated (so it could
! from a range of tracks) but I really wanted to
! keep it as simple as possible and jump into Modu
! as possible. The whole intention of using a seco
! bootstage was to keep any complexity in a HLL
!
```

**Boot code tutorial**

boot.S

```
extern minbios_WriteChar

_start:
    jmp after_sig
    nop
OEM_ID: .ascii "luk-boot"
BytesPerSector: DW 0x0200
SectorsPerCluster: DB 0x01
ReservedSectors: DW 0x0001
TotalFATs: DB 0x02
MaxRootEntries: DW 0x00E0
TotalSectorsSmall: DW 0x0B40
MediaDescriptor: DB 0xF0
SectorsPerFAT: DW 0x0009
SectorsPerTrack: DW 0x0012
NumHeads: DW 0x0002
HiddenSectors: DD 0x00000000
TotalSectorsLarge: DD 0x00000000
DriveNumber: DB BOOTDRIVE
Flags: DB 0x00
Signature: DB 0x29
VolumeID: DD 0xFFFFFFFF
VolumeLabel: .ascii "luk-bootusb"
SystemID: .ascii "FAT12 "
```

**Boot code tutorial**

boot.S

```
after_sig:
    mov     DriveNumber, dl ! save the bios given bootdr
    !
    ! set up stack
    !
    mov ax, #BOTOFMEMSEG
    mov ss,ax
    mov ax, #0xffe0
    mov sp,ax

    mov ax, #'1
    pushax
    callminbios_WriteChar
    pop ax
```

**Boot code tutorial**

boot.S

```
!
! now jump to _load at a new code segment BOOTSEG:
!
! we need to do this so that the cs is initialised to
! The bios doesn't do this for us.
jmp_i_load,BOOTSEG ! jmp far _load:BOC
```

**Boot code tutorial**

boot.S

```
_load:
! excellent now we set our Data segment = Code segment
! we need to do this because there are some OS parameters
! right at the end of this 512 disk sector.
! We pick these up in Util.S just before we go into M

    mov ax,cs
    mov ds,ax ! set up Data Segment

    mov ax, #'2
    pushax
    callminbios_WriteChar
    pop ax

    call _SecondLoad
```

**Boot code tutorial**

boot.S

```

!
! now jump to _SecondLoad at a new code segment SECON
!
mov ax, #'4
pushax
callminbios_WriteChar
pop ax

xor dx, dx
mov dl, DriveNumber ! take DriveNumber value
jmp 0,SECONDSEG ! jmp far 0:SECONDSEG

```

**Boot code tutorial**

boot.S

```

_SecondLoad:
mov ax, #'3
pushax
callminbios_WriteChar
pop ax

mov ax, #SECONDSEG
mov es, ax ! ES = SECONDSEG
mov bx, #0x0 ! address = SECONDSEG:0
xor dx, dx ! dh (head no) = 0
mov dl, DriveNumber ! drive no
mov cx, #0x02 ! sector 2, track 0
mov ax, #0x0200+SECONDSIZE ! service 2, nr of sect
! (assume all on head 0, track 0)
int 0x13 ! read it
jnc ok_found ! ok - continue

mov ah, #'e
pushax ! display error message
callminbios_WriteChar
pop ax

```

**Boot code tutorial**

boot.S

```

mov dl, DriveNumber ! drive number
xor ah, ah
int 0x13
jmp _SecondLoad ! try to load Secondary boot ag:

ok_found:
ret

```