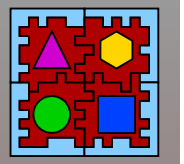


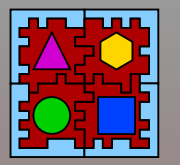
Dedicated Operating System

The Boot:
Getting ready for the OS

LISHA/UFSC

Prof. Dr. Antônio Augusto Fröhlich
guto@lisha.ufsc.br
http://www.lisha.ufsc.br/Guto

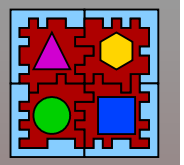
March 2011



Where are we now?

- BIOS brought the system on
  - BIOS initialized a complex architecture
    - BIST, POST, hooks
  - First instruction fetched
    - 0x7c00
  - Lots of “jmp” so far, no calls, why?
    - Where is the stack?
- BIOS got the system ready for

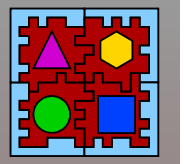
The Bootstrap => Today class



Dedicated Operating System

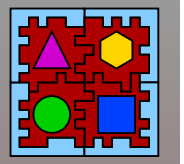
EPOS Bootstrap: src/boot/pc_boot.S

```
; CONSTANTS
;==========================================================
; PHYSICAL MEMORY MAP
; 0x0000 0000 -+------------------------+ BOOT_IDT
;              | IDT (4 K)              |
; 0x0000 1000 -+------------------------+ BOOT_GDT
;              | GDT (4 K)              |
; 0x0000 2000 -+------------------------+
;              :                        :
;              | BOOT STACK (23 K)      |
; 0x0000 7c00 -+------------------------+ BOOTSTRAP_STACK
;              | BOOT CODE (512 b)      | BOOTSTRAP_CODE
; 0x0000 7e00 -+------------------------+
;              | RESERVED (512 b)       |
; 0x0000 8000 -+------------------------+ DISK_IMAGE
;              | DISK IMAGE (608 K)     |
;              :                        :
;              |                        |
; 0x000a 0000 -+------------------------+
;              | UNUSED (384K)          |
;              :                        :
;              |                        |
; 0x000f f000 -+------------------------+
```



EPOS Bootstrap: Notes

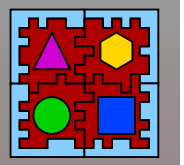
- Code to be ran at real mode (16 bits)
- Interrupts (IDT)
  - At real mode, always at 0x0000
  - At protected mode, anywhere (IDTR)
- Segmentation (GDT)
  - Always on at x86, in any mode
  - GDTR
- Bootstrap code
  - Code starts at 0x7c00
  - Stack stays bellow 0x7c00
  - It uses BIOS to access discs and load system



EPOS Bootstrap: main entry point

```
entry main
main:
      cli ; disable interrupts
      xor ax,ax ; data segment base = 0x00000
      mov ds,ax
      mov es,ax
      mov ss,ax
      mov sp,#BOOTSTRAP_STACK ; set stack pointer
```

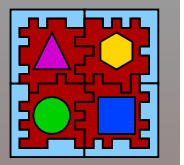
- Basic segments initialization



Dedicated Operating System

EPOS Bootstrap: disc image layout

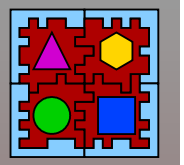
```
; DISK IMAGE LAYOUT
; -+-----------------------+ DISK_IMAGE_SYS_INFO
;  | SYS_INFO (512 bytes)  |
; -+-----------------------+ DISK_IMAGE_SETUP
;  | SETUP                 |
;  :                       :
; -+-----------------------+
;  | SYSTEM                |
;  :                       :
; -+-----------------------+
;  | INIT                  |
;  :                       :
; -+-----------------------+
;  | LOADER/APP1           |
;  :                       :
; -+-----------------------+
;  | APP1                  |
;  :                       :
; -+-----------------------+
;  :                       :
; -+-----------------------+
;  | APPn                  |
;  :                       :
; -+-----------------------+
```



EPOS Bootstrap: loading image from disc

```
; Load the boot image from the disk into "DISK_IMAGE"
      mov si,#msg_loading
      call print_msg
      push es
      mov ax,#IMAGE_SEG
      mov es,ax               ; don't try to load es directly
      mov bx,#0               ; set es:bx to DISK_IMAGE
      mov ax,[n_sec_image]
      mov cx,#0x0002          ; starts at track #0, sector #2,
      mov dx,#0x0000          ;  side #0, first drive
      call load_image
      pop es
      mov si,#msg_done
      call print_msg

; Stop the drive motor
      call stop_drive
```

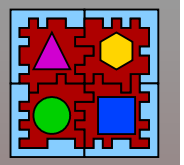
- Function calling (e.g., “print_msg”)
  - Parameter passing



EPOS Bootstrap: print_msg – BIOS access

```
;=============================================================
; PRINT_MSG                                                  =
;                                                           =
; Desc: Print a \0 terminated string on the screen using the BIOS   =
;       Message must end with 00h ;                         =
;                                                           =
; Parm: si  -> pointer to the string                        =
;=============================================================
print_msg:
      pushf
      push ax
      push bx
      push bp
      cld

print_char:
      lodsb
      cmp al,#0
      jz end_print
      mov ah,#0x0E
      mov bx,#0x0007
      int 0x10
      jmp print_char

end_print:
      pop bp
      pop bx
      pop ax
      popf
      ret
```

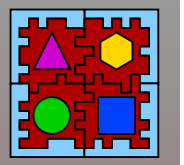
- BIOS access
  - “int 0x10”
- CISC magics
  - “cld”
  - “lodsb”



EPOS Bootstrap: load_sector – BIOS access

```
;=============================================================================
; LOAD_ONE_SECTOR                                                            =
;                                                                            =
; Desc: Load a single sector from disk using the BIOS.                       =
;                                                                            =
; Parm: es:bx  -> buffer                                                     =
;       cx     -> track (ch) and sector number (cl)                          =
;       dx     -> side (dh) and drive number (dl)                            =
;=============================================================================
load_sector:
      pushf
      push ax

      mov ax,#0x0201   ; function #2, load 1 sector
      int 0x13
      cli              ; int 0x13 sets IF
      jc ls_disk_error ; if CY=1, error on reading

      pop ax
      popf
      ret

ls_disk_error:
      mov si,#msg_disk_error
      call print_msg   ; print error msg if disk is bad
      call stop_drive

ls_disk_halt:
      jmp ls_disk_halt ; halt
```

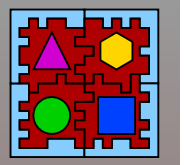
- “method signatures” depend on BIOS API



EPOS Bootstrap: loading image from disc

```
; Load the boot image from the disk into "DISK_IMAGE"
        mov si,#msg_loading
        call print_msg
        push es
        mov ax,#IMAGE_SEG
        mov es,ax               ; don't try to load es directly
        mov bx,#0               ; set es:bx to DISK_IMAGE
        mov ax,[n_sec_image]
        mov cx,#0x0002          ; starts at track #0, sector #2,
        mov dx,#0x0000          ;  side #0, first drive
        call load_image
        pop es
        mov si,#msg_done
        call print_msg

; Stop the drive motor
        call stop_drive
```

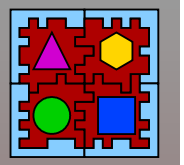
- Print => Load Image => Print => Stop Drive
- Why stop drive?



EPOS Bootstrap: memory initialization

```
; Get extended memory size (in K)
;       xor dx,dx
;       mov ah,#0x88
;       int 0x15        ; what if memory size > 64 Mb?
;       push    ds
;       push    #INFO_SEG
;       pop ds
;       mov [0],ax
;       mov [2],dx
;       pop ds

; Say hello;
mov si,#msg_hello
call    print_msg

; Enable A20 bus line
call    enable_a20
```

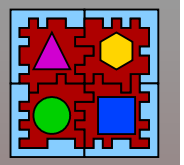
- BIOS info on memory is unreliable
- EPOS will test memory by itself
- A20 line
  - DOS access data and IO in one segment until 80186
  - 80286 needed this line to access 16MB memory, so we have this legacy



EPOS Bootstrap: entering protected mode

```
; Zero IDT and GDT
        cld
        xor ax,ax
        mov cx,#0x1000    ; IDT + GDT = 8K (4K WORDS)
        mov di,#BOOT_IDT  ; initial address (relative to ES)
        rep               ; zero IDT and GDT with AX
        stosw

; Set GDT
        mov si,#GDT_CODE  ; Set GDT[1]=GDT_CODE and
        mov di,#BOOT_GDT  ; GDT[2]=GDT_DATA
        add di,#8         ; offset GDT[1] = 8
        mov cx,#8         ; sizeof GDT[1] + GDT[2] = 8 WORDS
        rep               ; move WORDS
        movsw

; Set GDTR
        lgdt GDTR

; Enable Protected Mode
        mov eax,cr0
        or al,#0x01  ; set PE flag and MP flag
        mov cr0,eax
```

- Protected mode configured
- But not entered yet...



EPOS Bootstrap: breaking the pre-fetch queue

```
; Adjust selectors
        mov bx,#2 * 8    ; adjust data selectors to use
        mov ds,bx        ; GDT[2] (DATA) with RPL = 00
        mov es,bx
        mov fs,bx
        mov gs,bx
        mov ss,bx

; As Linux as86 can't generate 32 bit instructions, we have to code
; it by hand. The instruction below is a inter segment jump to
; GDT[GDT_CODE]:SETUP. Jump into "SETUP" (actually ix86 Protected
; Mode starts here)
;        jmp 0x0008:#SETUP_ENTRY
        .byte 0x66
        .byte 0xEA
        .long SETUP_ENTRY
        .word 0x0008
```

- Need to break the pre-fetch queue
  - Force CPU to read configuration registers
- LDT and IDT not assempled yet
  - Next class: pc_setup.cc