Project 1: Bootloader

COS 318

Fall 2016

Project 1: Schedule

- Design Review
  - Monday, 9/26
  - 10-min time slots from 1:30pm-9:20pm
  - Write functions `print_char` and `print_string`
  - Answer the questions:
    - ✓ How to move the kernel from disk to memory?
    - ✓ How to create the disk image?
- Due date: Sunday, 10/2, 11:55pm

General Suggestions

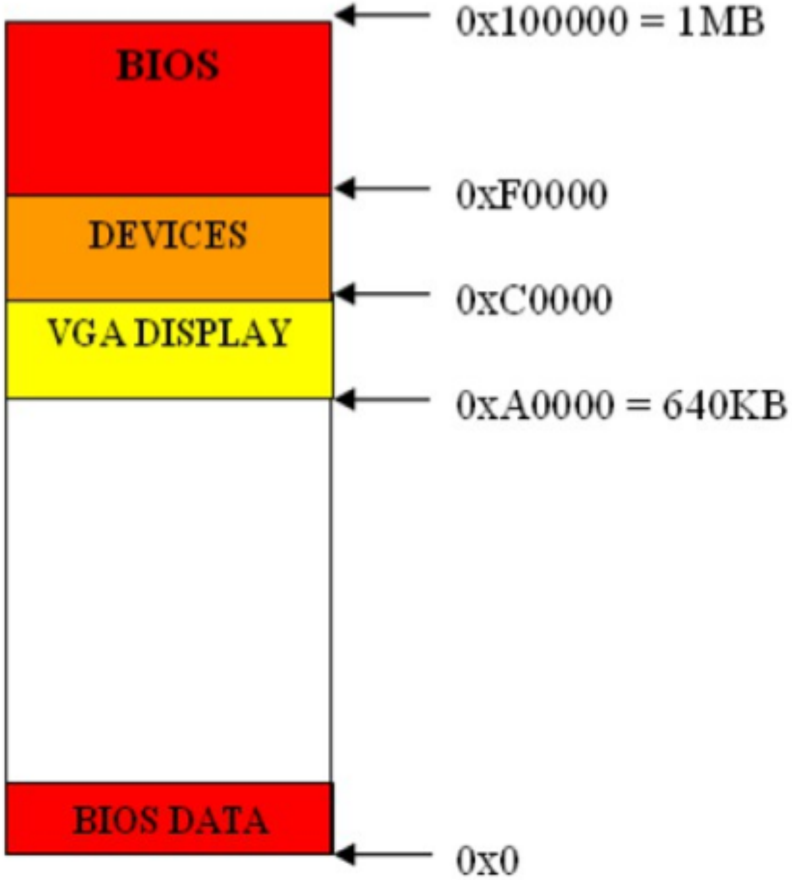
- Read ***assembly_example.s*** in start code pkg
  - /u/318 (subdirs: bin code share)
- Get ***bootblock.s*** working before starting on ***createimage.c***
- Read documentation on AT&T syntax x86 Assembly language
- Read provided documentation on ELF format
- Start as early as you can and get as much done as possible by the design review
- SSH instructions included in the assignment specification. Copy the start code from /u/318 into your personal folder (/u/netid)

Project 1 Overview

- Write a bootloader: ***bootblock.s***
  - How to set up and start running the OS
  - Written in X86 Assembly language (AT&T syntax)
- Implement a tool to create a bootable OS image: ***createimage.c***
  - Bootable image contains bootloader and kernel
  - How are executable files structured?
  - Become familiar with ELF format

Boot Process

- When powered up, nothing in RAM, so how do we get started?
  - Resort to hardware
  - Load BIOS from ROM
- BIOS:
  - Minimal functionality
  - Initialization of I/O devic
  - Search for bootable devic

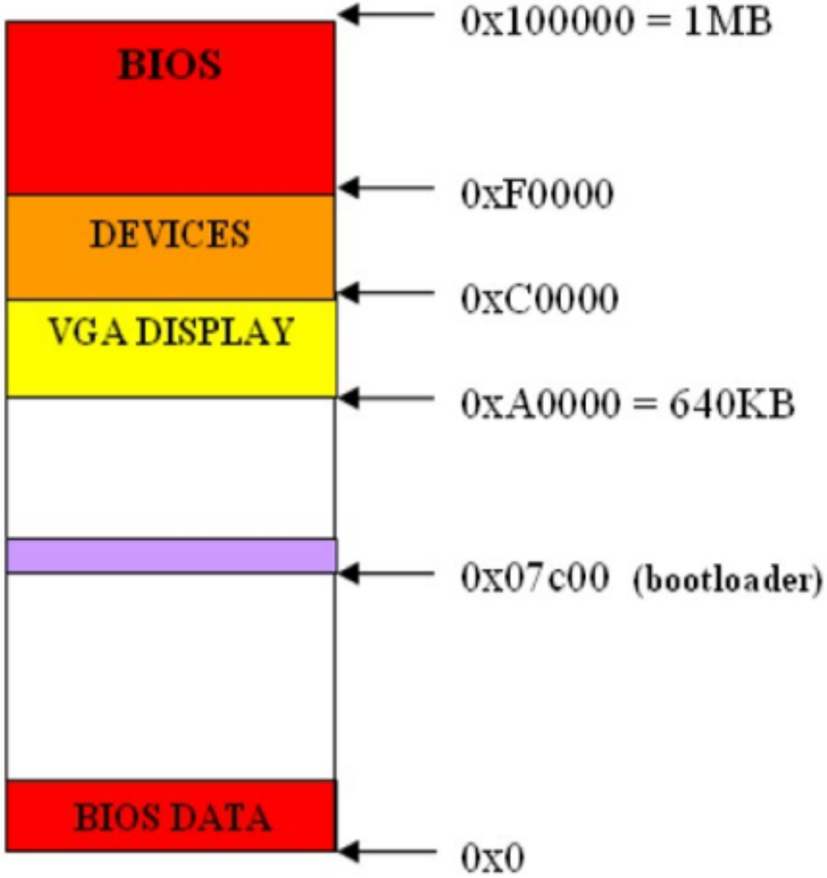


Loading the Bootloader

- Found bootable storage volume:
  - HDD, USB, Floppy
  - Load bootloader

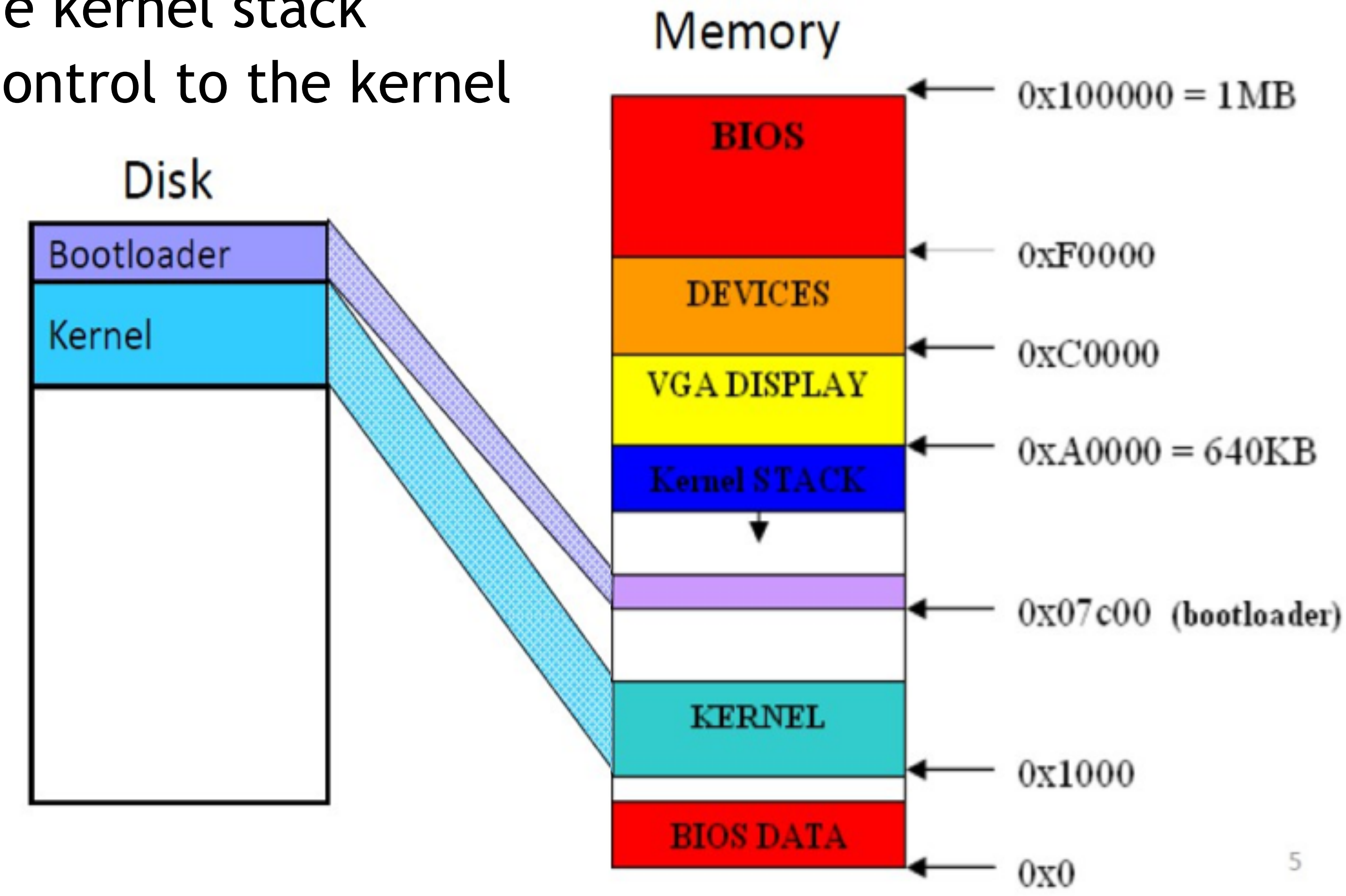
- How is this done?
  - Load first sector (512 bytes)
  - Memory location: 0x7c00
  - Switch control to this location to launch the bootloader

Memory



The Bootloader

- Three tasks:
  - Load the kernel into memory
  - Setup the kernel stack
  - Switch control to the kernel



The Master Boot Record (MBR)

- The MBR is loaded by BIOS at physical address 0x7c00, with %dl set to the drive number that the MBR was loaded from.
- For more information:
  - http://wiki.osdev.org/MBR_(x86)
  - http://wiki.osdev.org/Partition_Table

X86 Assembly – Quick Tutorial

- About numbers, need good bookkeeping
- Move data, perform simple arithmetic
- Need a lot of steps to do useful things
- KEY:
  - Understand memory addresses
  - Know where things are in memory

X86 Assembly – Quick Tutorial

- CPU State: Register Set

General-purpose registers: 8, 16, and 32 bits

31 16	15 8	7 0	16-bit	32-bit
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

Segment registers (16 bits)

CS
DS
SS
ES
FS
GS

Instruction Pointer (32 bits): EIP

Flags (32 bit): EFLAGS

X86 Assembly – Quick Tutorial

- Function of flags:
  - Control the behavior of CPU
  - Save the status of last instruction
- Important flags:
  - CF: carry flag
  - ZF: zero flag
  - SF: sign flag
  - IF: interrupt (sti, cli)
  - DF: direction (std, cld)

Memory Addressing

- 1MB of memory:
  - Valid address range: 0x00000 – 0xFFFFF
- Real-mode segmented model:
  - See full 1MB with 20-bit addresses
  - 16-bit segments and 16-bit offsets

Memory Addressing

- Format (AT&T syntax):
  segment:displacement(base,index)
- Offset = Base + Index + Displacement
- Address = (Segment * 16) + Offset
- Displacement: Constant
- Base: %bx, %bp
- Index: %si, %di
- Segment: %cs, %ds, %ss, %es, %fs, %gs

Memory Addressing (data)

segment:displacement(base,index)

- Components are optional
- Default segment:
  - %bp: %ss
  - %bx, %si, %di: %ds
  - You can override: %es:(%bx)
- Examples:
  - (%si) = %ds:(%si)
  - (%bp) = %ss:(%bp)
  - (%bs,%si) = %ds:(%bx,%si)
  - +4(%bp) = %ss:+4(%bp)
  - 100 = %ds:100
  - %ds:-10(%bx,%si)

AT&T Syntax

- Prefix register names with % (e.g. %ax)
- Instruction format: instr src,dest
  - movw %ax,%bx
- Prefix constants (immediate values) with $
  - movw $1,%ax
- Suffix instructions with size of data
  - b for byte, w for word (16 bits), l for long (32 bits)

Instructions: arithmetic & logic

- add/sub{l,w,b} source,dest
- inc/dec/neg{l,w,b} dest
- cmp{l,w,b} source,dest
- and/or/xor{l,w,b} source,dest
- …
- Restrictions
  - No more than one memory operand

Instructions: Data Transfer

- **mov{l,w,b} source,dest**
- **xchg{l,w,b} dest**
- **movsb/movsw**
  - %es:(%di) ← %ds:(%si)
  - Often used with %cx to move a number of bytes
    - movw $0x10,%cx
    - rep movsw
- Segment registers can only appear with registers

Instructions: stack access

- **pushw source**
  - %sp ← %sp - 2
  - %ss:(%sp) ← source
- **popw dest**
  - dest ← %ss:(%sp)
  - %sp ← %sp + 2
- Set up the stack before you actually use it

Instructions: Control Flow

- **jmp label**
  - %ip ← label
- **ljmp NEW_CS,offset**
  - %cs ← NEW_CS
  - %ip ← offset
- **call label**
  - push %ip + ?
  - %ip ← label
- **ret**
  - pop %ip
- **lcall** and **lret**

Instructions: Conditional Jump

- **j*   label**
  - jump to label if flag * is 1
- **jn*   label**
  - jump to label if flag * is 0
- *: bits of %eflags
  - Examples: js, jz, jc, jns, jnz, jnc

Assembly Program Structure

- Assembler directives:
  - Not instructions
  - Segment the program
- .text begins code segment
- .globl defines a list of symbols as global
- .data begins data segment
- .equ defines a constant (like #define)
  - e.g. .equ ZERO,$0x00
- .byte, .word, .asciz reserve space in memory

BIOS Services

- Use BIOS services through INT instruction:
  - Store the parameters in the registers
  - Trigger a software interrupt
- **int INT_NUM**
  - int $0x10 # video services
  - int $0x13 # disk services
  - int $0x16 # keyboard services

BIOS INT 0x13

- Function 2 reads from disk
  - %ah: 2
  - %al: number of sectors to read
  - %ch: cylinder number bits 0-7
  - %cl: sector number bits 0-5; bits 6-7 are bits 8-9 of the cylinder number
  - %dh: starting head number
  - %dl: drive number
  - %es:%bx: pointer to memory region to place data read from disk
- Returns:
  - %ah: return status (0 if successful)
  - Carry flag = 0 successful, = 1 if error occurred
- For more information, visit http://en.wikipedia.org/wiki/Cylinder-head-sector

Kernel Debugging

- Use *bochsdbg* provided in the bin directory of the start code
- Use the help command to learn about the other commands and parameters

Kernel Debugging

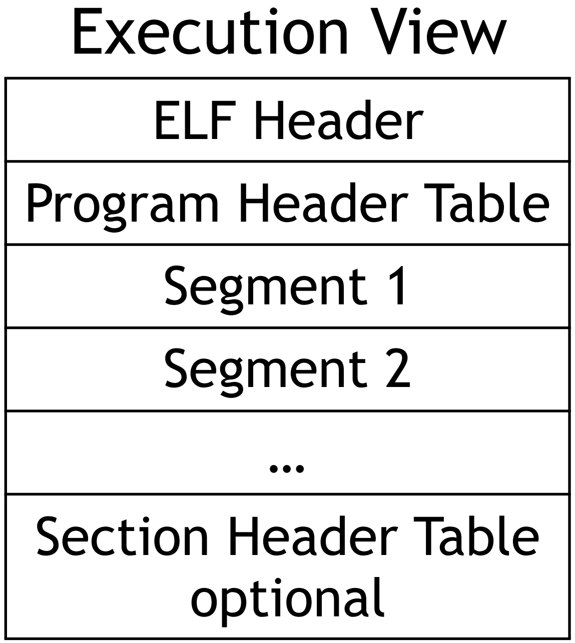
- Useful commands:
  - r | reg | regs | registers – show the registers
  - sreg – shows the segment registers
  - b – set a breakpoint
  - s – step
  - n – next
  - c – continue
  - d | del | delete <n> – delete a breakpoint
  - bpd <n> – disable a breakpoint
  - bpe <n> – enable a breakpoint
  - xp /n <addr> – exame memory at physical address <addr>
  - u | disasm | dissassemble /count <start> <end>

ELF Format

- Executable and linking format
- Created by assembler and link editor
- Object file: binary representation of programs intended to execute directly on a processor
- Support various processors/architectures:
  - represent control data in a machine-independent format

ELF Object File Format

- Header (pp. 1-3 – 1-5):
  - Beginning of file
  - Roadmap, file organization
- Program header table (p. 2-2):
  - Array, each element describes a segment
  - Tells system how to create the process image
  - Files used to create an executable program must have a program header.



Execution View

ELF Header

Program Header Table

Segment 1

Segment 2

…

Section Header Table
optional

p. 1-1 in the ELF manual