操作系统研讨课
Course: B0911011Y

蒋德钧

Fall Term 2018-2019

email: jiangdejun@ict.ac.cn
office phone: 62601007

中国科学院大学
University of Chinese Academy of Sciences

Lecture 0 Introduction

2018.09.12

中国科学院大学
University of Chinese Academy of Sciences

Lecture 0: Introduction

- Overview
  - Course introduction
  - Course administration

Lecture 0: Introduction

- Why?
- What?
- How?

Lecture 0: Introduction

- Why do I need to learn this course?
  - Getting credits
  - Capabilities of system developing
  - Basis of full stack view

Lecture 0: Introduction

- What do I learn in this course?
  - How to build a simple operating system
    - Bootloader
    - Kernel supporting multitasking
    - Process communication and management
    - Device driver
    - Virtual memory management
    - File system

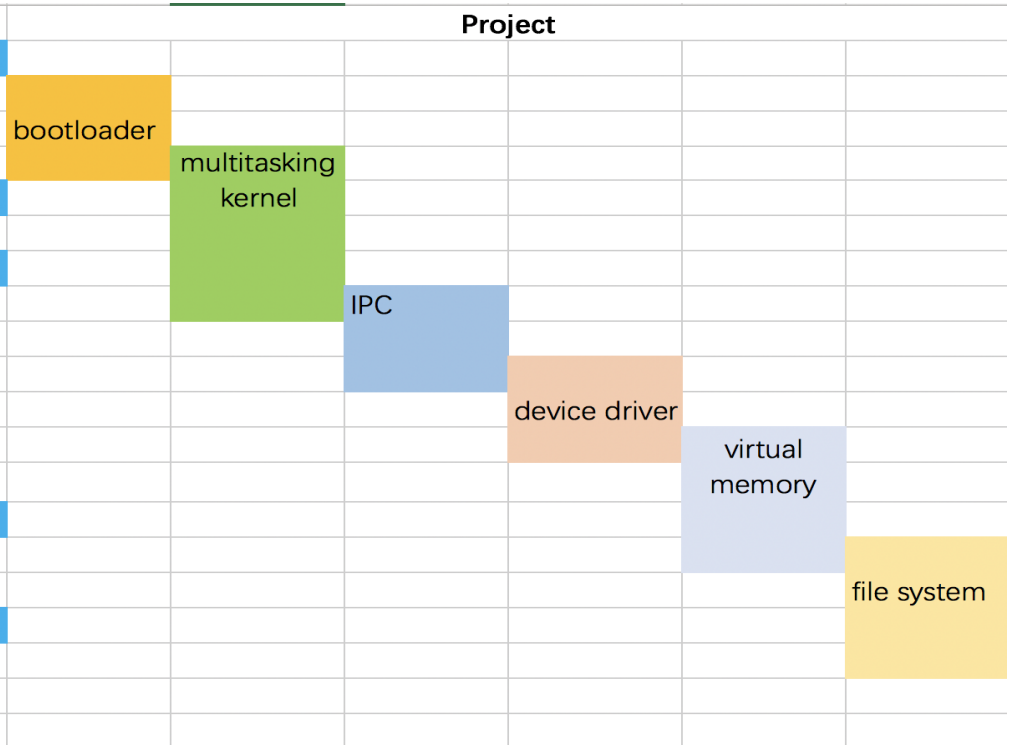
Lecture 0: Introduction

- How do I finish this course?
  - Think before practice
  - Discuss with TA at design review
  - Group working
  - XV6 may help

Lecture 0: Introduction

- Course administration
  - Classrooms：教 205（机房） & 221（机房）
  - Schedule

周次	课次	时间	内容
1		2018年9月5日	Noclass
2	1	2018年9月12日	P1 start
3	2	2018年9月19日	P1 design review
4	3	2018年9月26日	P1 due, P2 start
5		2018年10月3日	No class
6	4	2018年10月10日	P2 design review
7		2018年10月17日	No class
8	5	2018年10月24日	P2 due, P3 start
9	6	2018年10月31日	P3 design review
10	7	2018年11月7日	P3 due, P4 start
11	8	2018年11月14日	P4 design review
12	9	2018年11月21日	P4 due, P5 start
13	10	2018年11月28日	P5 design review
14		2018年12月5日	No class
15	11	2018年12月12日	P5 due, P6 start
16	12	2018年12月19日	P6 design review
17		2018年12月26日	No class
18	13	2019年1月2日	P6 due
19	14	2019年1月9日	Final due
20		2019年1月16日	



Lecture 0: Introduction

- Course administration
  - Lecturer
    - 蒋德钧：jiangdejun@ict.ac.cn
  - Teaching assistant
    - 卢天越：lutianyue@ict.ac.cn
    - 王盈： wangying01@ict.ac.cn
    - 韩书楷：hanshukai@ict.ac.cn
    - 覃晓婉：qinxiaowan@ict.ac.cn
  - Office hour
    - Make appointment

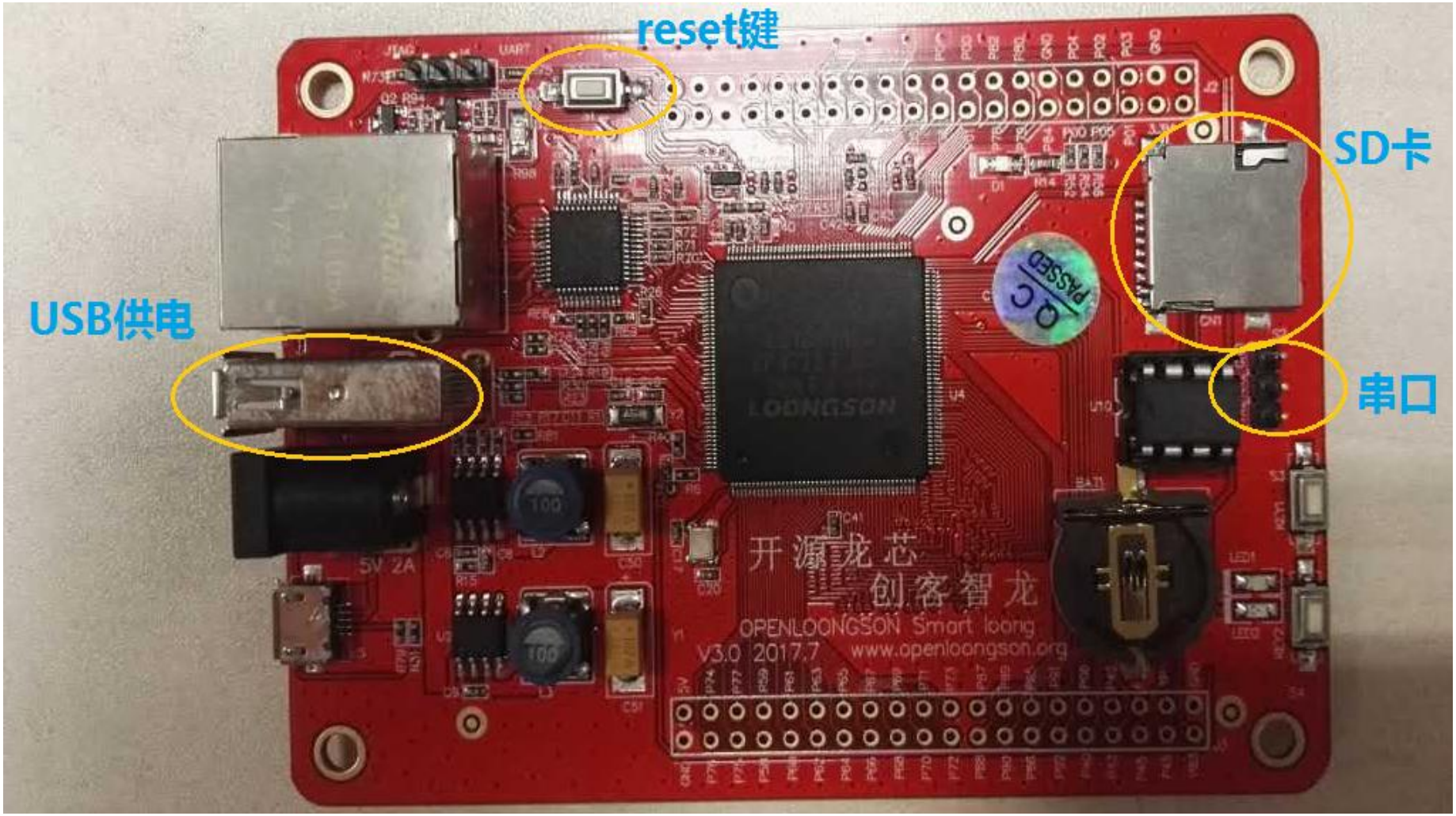
Lecture 0: Introduction

- Development environment
  - Software
    - Linux operating system with your own laptop
    - Virtual machine with VirtualBox or Physical machine
    - Ubuntu 12.04, kernel 3.11

Lecture 0: Introduction

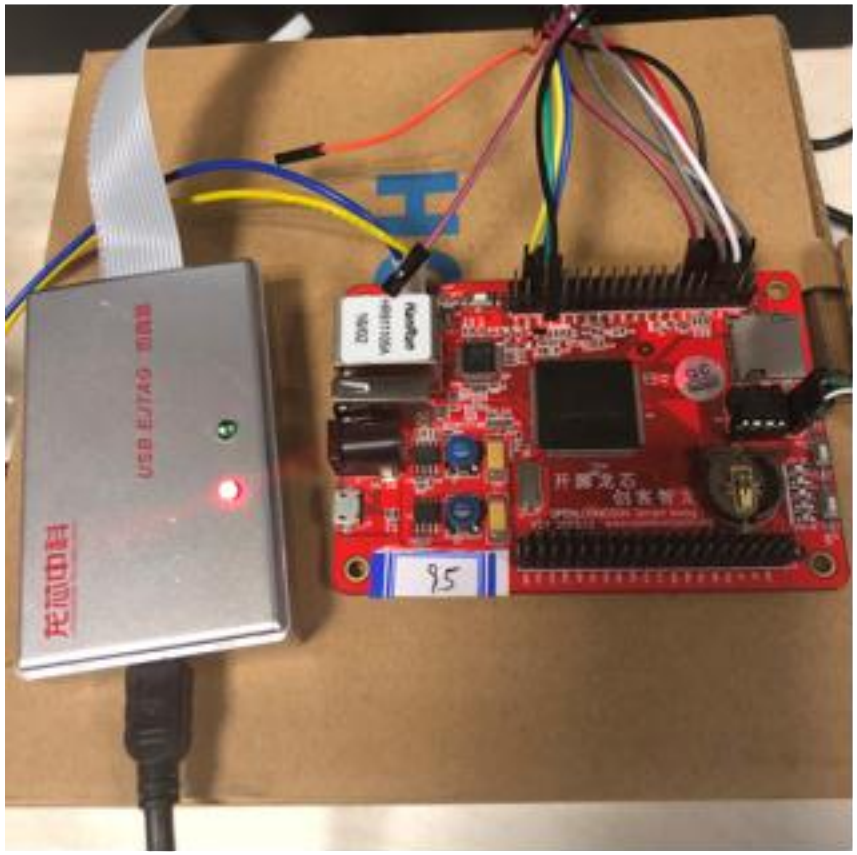
- Development environment
  - Hardware
    - One piece of Openloongson SoC board
    - One USB cable
    - One serial port cable
    - One SSD card and one card reader
    - Protection package(Optinal)

Lecture 0: Introduction



Lecture 0: Introduction

- Additional debug tools - ejtag



hb:设置断点

启动ejtag

disas:查看地址源码

cont:继续执行

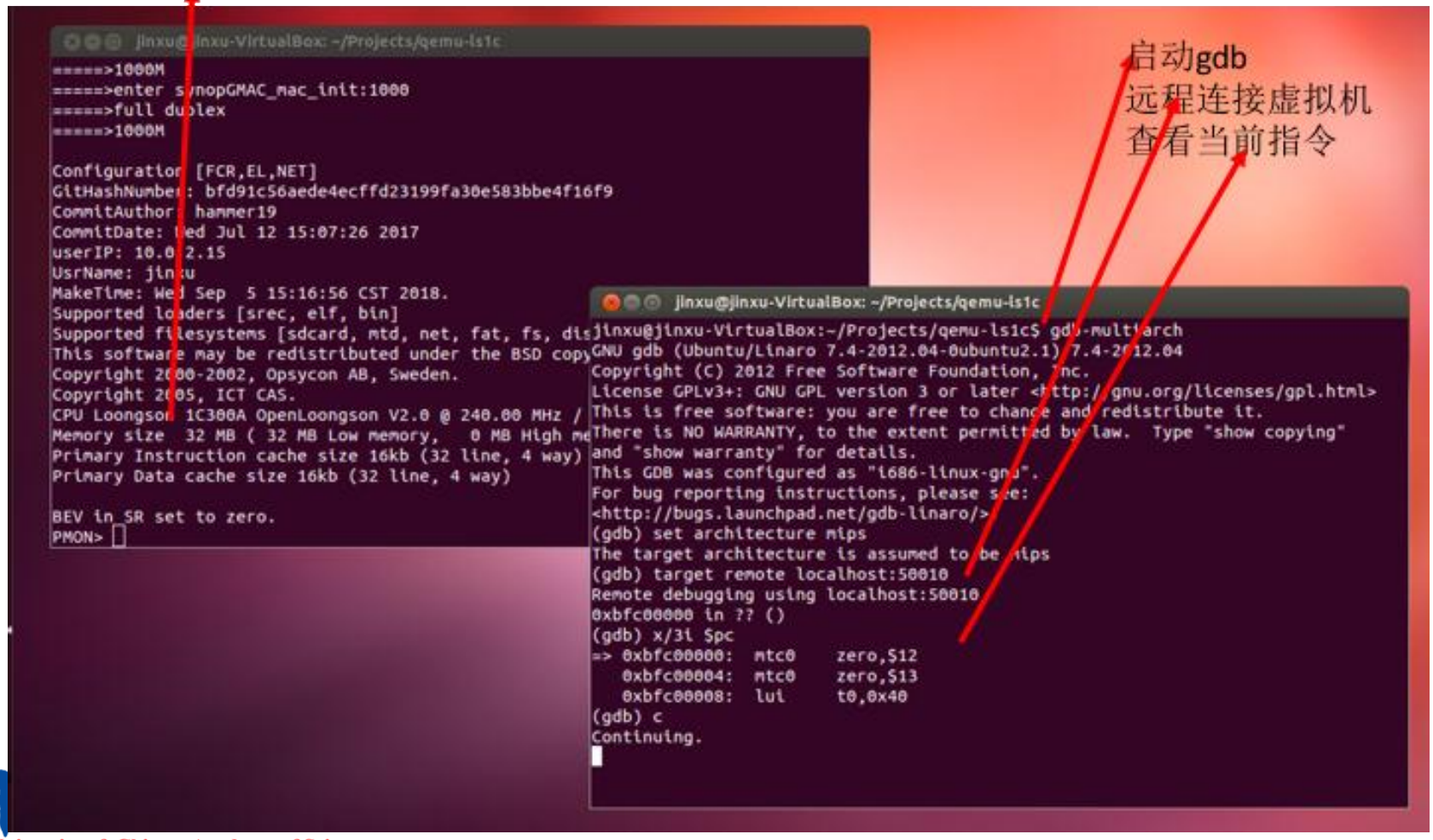
```
jlnxu@jlnxu-VirtualBox:~/.../tools/2/ejtag-debug$ sudo ./ejtag_debug_usb -t
cpu0 -hb 0xa0800000
cpu0 -cont
cpu0 -disas 0xa0800000
0xffffffffa0800000: 3c04a080 lui     a0,0xa080    # 41088
0xffffffffa0800004: 24840080 addiu   a0,a0,0x80   # 128
0xffffffffa0800008: 3c118007 lui     s1,0x8007    # 32775
0xffffffffa080000c: 3639b980 ori     t9,s1,0xb980 # 47488
0xffffffffa0800010: 0320f809 jalr    t9
0xffffffffa0800014: 00000000 nop
0xffffffffa0800018: 00000000 nop
0xffffffffa080001c: 3c08a080 lui     t0,0xa080    # 41088
0xffffffffa0800020: 8d08009c lw      t0,156(t0)   # 0x9c
0xffffffffa0800024: 3c04a080 lui     a0,0xa080    # 41088
cpu0 -si
cpu0: 0xffffffffa0800004: 24840080 addiu   a0,a0,0x80   # 128   # step exp
cpu0 -si
cpu0: 0xffffffffa0800008: 3c118007 lui     s1,0x8007    # 32775        # step exp
cpu0 -si
cpu0: 0xffffffffa080000c: 3639b980 ori     t9,s1,0xb980 # 47488        # step exp
cpu0 -set

zero:0x0 at:0xfffffffc0 v0:0xa0800000 v1:0x80056184
a0:0xa0800080 a1:0x1 a2:0x801ffd28 a3:0x8000b3e0
t0:0x8008a770 t1:0xa11c3800 t2:0x800a484c t3:0xffffffff
t4:0xffffffff t5:0xffffffff t6:0x80000000 t7:0x8000b43e
s0:0x200 s1:0x80070000 s2:0xf s3:0x80090000
s4:0x8000b9c0 s5:0x1 s6:0x8000bce8 s7:0xffffffff
t8:0xa11bd4c0 t9:0xa8 k0:0x1 k1:0x8000b8d8
gp:0x810510c0 sp:0x8000b920 s8:0x0 ra:0x8007a248
status:0x0 lo:0x0 hi:0x0 badvaddr:0x0
cause:0x40000000 pc:0xa080000c epc:0x0
cpu0 -
```

si:单步执行

set:查看寄存器值

Lecture 0: Introduction

- Additional debug tools - qemu



Lecture 0: Introduction

- Grouping with different teachers
  - P1 ~ P2: grouping I
  - P3 ~ P4: grouping II
  - P5 ~ P6: grouping III
  - Group students randomly
  - Group presentation + individual submission

Lecture 0: Introduction

- Project submission
  - Design documents
  - Source code + README
  - Submission site: course web site
    - http://sep.ucas.ac.cn/

Lecture 0: Introduction

- Grading
  - Grading per project
    - design review: 40 points
    - code development: 60 points
  - Final grading
    - Final grades = Basic * 0.9 + Bonus * 0.1
    - Basic

P1	P2	P3	P4	P5	P6
10%	15%	10%	15%	25%	25%

    - Bonus: depends on projects

Lecture 0: Introduction

- Grading
  - Grading individually depends on
    - group presentation and Q&A
    - project submission
  - Submit your project on time: 100%
  - Submit one week after deadline: -30%
  - Copying others' code is ABSOLUTELY prohibited
    - NO points will be given

Lecture 0: Introduction

- Daily Q&A
  - WeChat

Lecture 0: Introduction

- Any question?

Lecture 1 Bootloader

2018.09.12

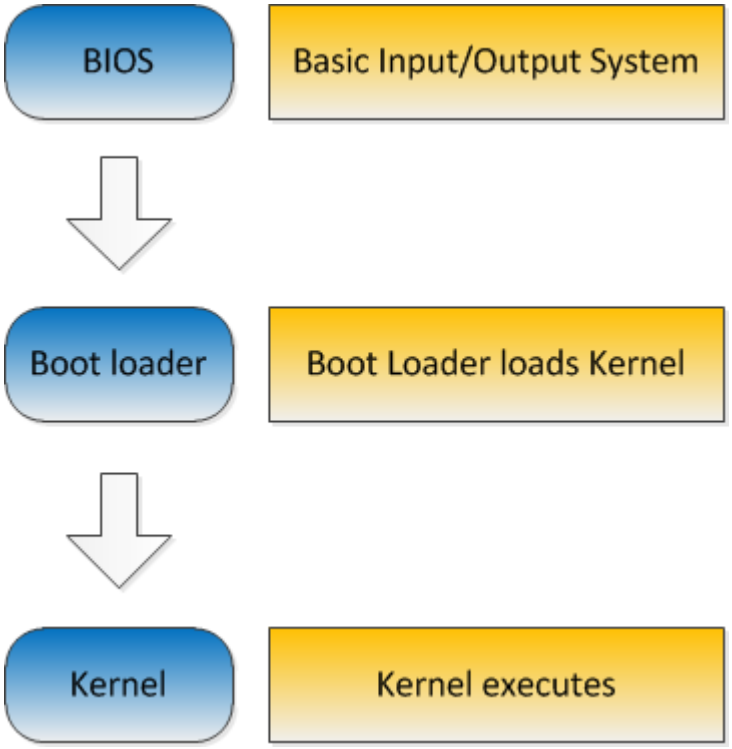
中国科学院大学
University of Chinese Academy of Sciences

Project 1 – Bootloader

- Requirements
  - Write a bootloader to start a simple kernel based on Openloongson SoC board
    - kernel.c
    - bootblock.s
    - createimage.c

Project 1 – Bootloader

- Booting procedure



Project 1 – Bootloader

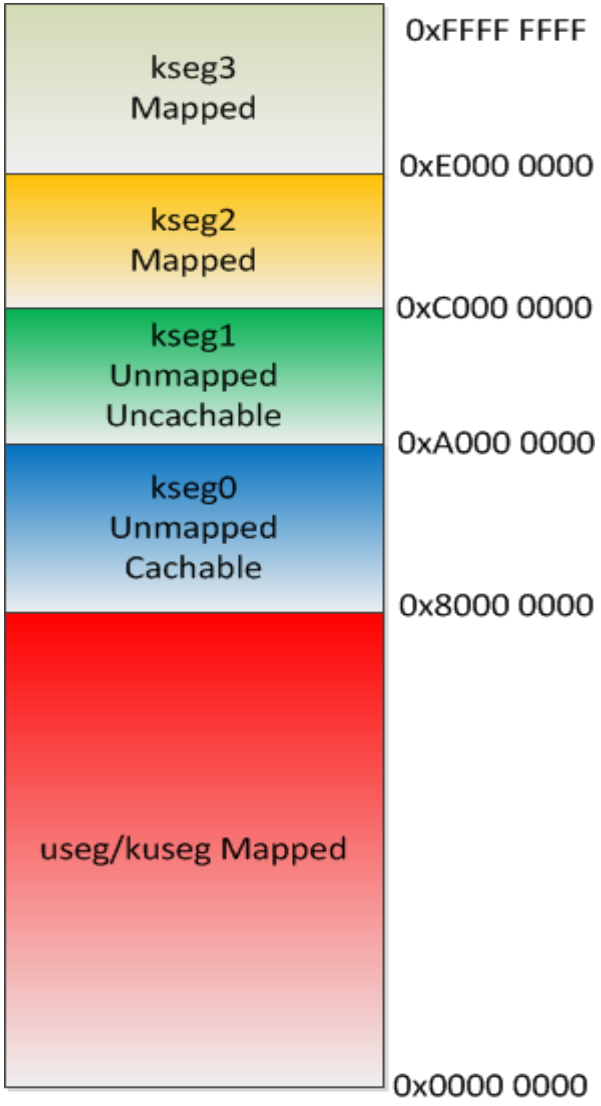
- BIOS
  - Basic Input/Output System
  - Firmware used to perform hardware initialization after power-on
  - Load bootloader
- Bootblock
  - Loaded by BIOS
  - Hard disk

Project 1 – Bootloader

- Bootloader
  - A small program to enable operating system
    - Load the kernel
    - Switch control to the kernel

Project 1 – Bootloader

- Memory mapping



Project 1 – Bootloader

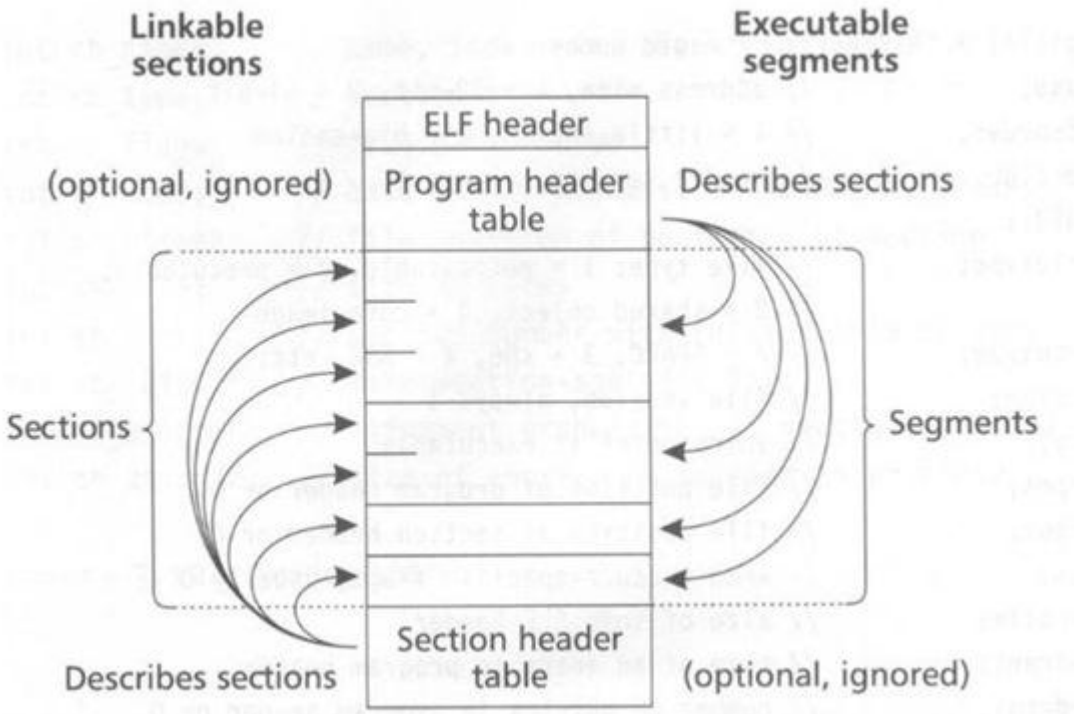
- Createimage
  - Executable file
    - gcc
  - Bootable OS image
    - createimage tool

Project 1 – Bootloader

- ELF object file format
  - Executable and Linking Format (ELF)
  - Object file
    - Binary representation of programs
  - Created by assembler and link editor

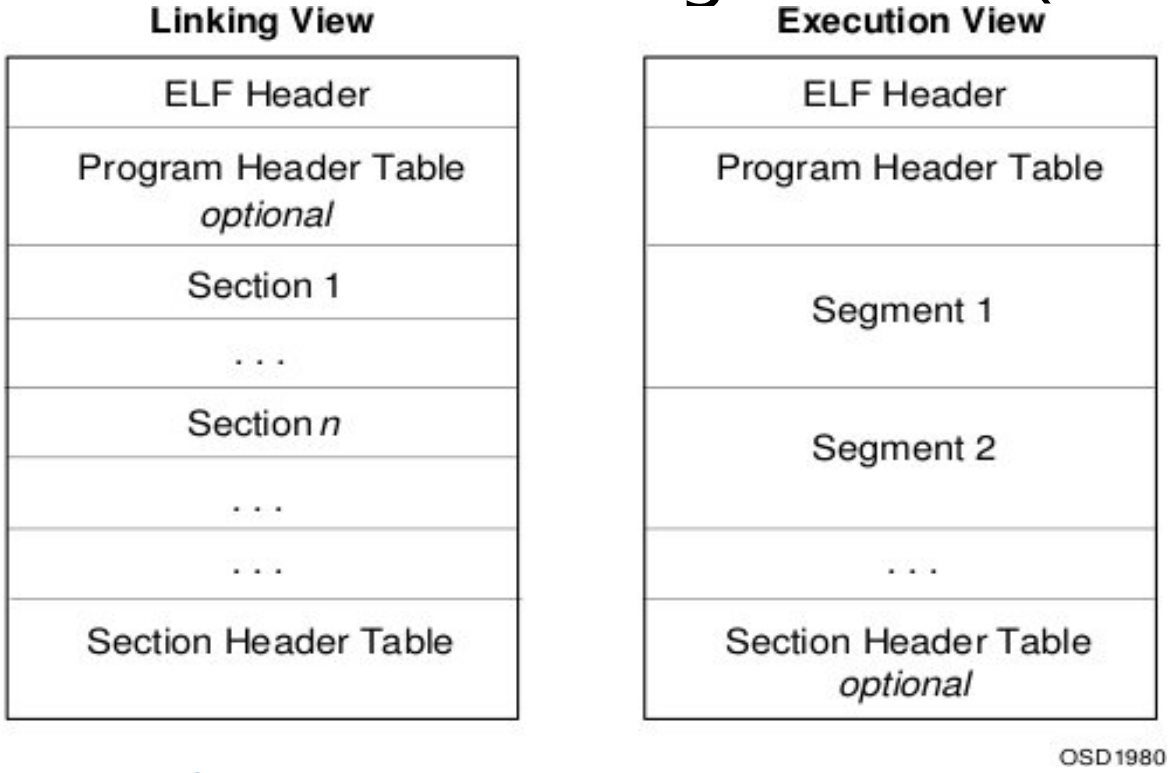
Project 1 – Bootloader

- ELF object file format
  - Executable and Linking Format (ELF)



Project 1 – Bootloader

- ELF object file format
  - Executable and Linking Format (ELF)



Project 1 – Bootloader

- ELF object file format
  - ELF header

```
typedef struct
{
  unsigned char e_ident[EI_NIDENT];   /* Magic number and other info */
  Elf32_Half    e_type;               /* Object file type */
  Elf32_Half    e_machine;            /* Architecture */
  Elf32_Word    e_version;            /* Object file version */
  Elf32_Addr    e_entry;              /* Entry point virtual address */
  Elf32_Off     e_phoff;              /* Program header table file offset */
  Elf32_Off     e_shoff;              /* Section header table file offset */
  Elf32_Word    e_flags;              /* Processor-specific flags */
  Elf32_Half    e_ehsize;             /* ELF header size in bytes */
  Elf32_Half    e_phentsize;          /* Program header table entry size */
  Elf32_Half    e_phnum;              /* Program header table entry count */
  Elf32_Half    e_shentsize;          /* Section header table entry size */
  Elf32_Half    e_shnum;              /* Section header table entry count */
  Elf32_Half    e_shstrndx;           /* Section header string table index */
} Elf32_Ehdr;
```

Project 1 – Bootloader

- ELF object file format
  - Section header

```
typedef struct
{
  elf32_word    sh_name;        /* Section name (string tbl index) */
  elf32_word    sh_type;        /* Section type */
  elf32_word    sh_flags;       /* Section flags */
  elf32_addr    sh_addr;        /* Section virtual addr at execution */
  elf32_off     sh_offset;      /* Section file offset */
  elf32_word    sh_size;        /* Section size in bytes */
  elf32_word    sh_link;        /* Link to another section */
  elf32_word    sh_info;        /* Additional section information */
  elf32_word    sh_addralign;   /* Section alignment */
  elf32_word    sh_entsize;     /* Entry size if section holds table */
} elf32_shdr;
```

Project 1 – Bootloader

- ELF object file format
  - Program header

```
typedef struct
{
  Elf32_Word    p_type;         /* Segment type */
  Elf32_Off     p_offset;       /* Segment file offset */
  Elf32_Addr    p_vaddr;        /* Segment virtual address */
  Elf32_Addr    p_paddr;        /* Segment physical address */
  Elf32_Word    p_filesz;       /* Segment size in file */
  Elf32_Word    p_memsz;        /* Segment size in memory */
  Elf32_Word    p_flags;        /* Segment flags */
  Elf32_Word    p_align;        /* Segment alignment */
} Elf32_Phdr;
```

Project 1 – Bootloader

- MIPS32 assembly language
  - 32 registers

Registers	Alternative name	Usage
$0	zero	Constant 0
$1	$at	Reserved by the assembler
$2 - $3	$v0 - $v1	Values from function results
$4 - $7	$a0 - $a3	Arguments, first four parameters for subroutine
$8 - $15	$t0 - $t7	Temporaries
$16 - $23	$s0 - $s7	Saved value
$24 - $25	$t8 - $t9	Temporaries

Project 1 – Bootloader

- MIPS32 assembly language
  - 32 registers

Registers	Alternative name	Usage
$26 - $27	$k0 - $k1	reserved for use by the interrupt/trap handler
$28	$gp	Global pointer
$29	$sp	Stack pointer
$30	$s8/$fp	Saved value / frame pointer
$31	$ra	Return address

Project 1 – Bootloader

- MIPS32 assembly language
  - Data types
    .ascii
    .byte: 8bit
    .half-word: 16bit
    .word: 32bit
  - a character requires 1 byte of storage
  - an integer requires 1 word of storage

Project 1 – Bootloader

- MIPS32 assembly language
  - Literals
    - numbers entered as is., e.g. 4
    - characters enclosed in single quotes. e.g. 'b'
    - strings enclosed in double quotes. e.g. "A string"

Project 1 – Bootloader

- MIPS32 assembly language
  - Assembler directives
    - Segment the program
  - .data
    - begins data segment
    - declares variable names used in program
    - storage allocated in main memory

```
name:  storage_type  values
val1:  .word  0x33
msg:  .ascii   “hello world\n”
```

Project 1 – Bootloader

- MIPS32 assembly language
  - .text: begins code segment, read-only, executable
    - contains instructions
    - starting point for code, e.g. given label main:

```
# Template.s
# Bare-bones outline of MIPS assembly language program

          .data       # variable declarations follow this line
                      # ...

          .text       # instructions follow this line

main:                 # indicates start of code (first instruction to execute)
                      # ...
```

Project 1 – Bootloader

- MIPS32 assembly language
  - RAM access
    - lw register_destination, RAM_source
    - lb register_destination, RAM_source
    - sw register_source, RAM_destination
    - sb register_source, RAM_destination
    - li register_destination, value

Project 1 – Bootloader

- MIPS32 assembly language
  - Indirect and Based Addressing
    - la $t0 val1
    - lw $t2, ($t0)
      - load word at RAM address contained in $t0 into $t2
    - sw $t2, ($t0)
      - store word in register $t2 into RAM at address contained in $t0
    - lw $t2, 4($t0)

Project 1 – Bootloader

- MIPS32 assembly language
  - Arithmetic instructions
    - add $t0,$t1,$t2
      - $t0 = $t1 + $t2
    - sub $t2,$t3,$t4
      - $t2 = $t3 - $t4
    - addi $t2,$t3, 5
      - $t2 = $t3 + 5

Project 1 – Bootloader

- MIPS32 assembly language
  - Control instructions – branches
    - beq $t0,$t1,target# branch to target if $t0 = $t1
    - blt $t0,$t1,target# branch to target if $t0 < $t1
    - ble $t0,$t1,target# branch to target if $t0 <= $t1
    - bgt $t0,$t1,target# branch to target if $t0 > $t1
    - bge $t0,$t1,target# branch to target if $t0 >= $t1
    - bne $t0,$t1,target# branch to target if $t0 <> $t1

Project 1 – Bootloader

- MIPS32 assembly language
  - Control instructions – jump
    - j target
      - Unconditional jump
    - jr $t3
      - Jump to address contained in $t3
    - jal sub_label
      - copy program counter to register $ra
      - jump to program statement at sub_label
    - jalr

Project 1 – Bootloader

- BIOS functions
  - printch(ch)
    - Print character to serial port
    - address: 0x8007ba00
  - printstr(str)
    - Print string to serial port
    - address: 0x8007b980
  - read_sd_card(address, offset, size)
    - Read SSD card
    - address: 0x8007b1cc

Project 1 – Bootloader

- Step by step
  - Task 1: setup the environment (Project 0)
  - Task 2: develop a simple bootloader to only print characters
  - Task 3: given createimage, develop kernel.c and bootblock.s to start a kernel
  - Task 4: develop your own createimage.c

Project 1 – Bootloader

- Requirement for design review
  - Answer following questions
    - Where do you place your bootblock
    - How to move kernel from disk to memory
    - Where do you place your kernel in the memory
    - Where is your kernel entry point
    - How to create disk image

Project 1 – Bootloader

- Requirement for developing
  - Finish following codes
    - Using read_sd_card function to place kernel image into memory: 15
    - Give control to the kernel: 15
    - Create image: 20
    - Extended flag: 5
    - Kernel runs: 5

Project 1 – Bootloader

- Bonus (1 point)
  - Imagine you have limited memory space, after executing bootloader, you need to reuse the memory space occupied by the bootloader starting from the address 0xa080 0000, please write a kernel.c to save your memory space.

Tips

- Learn to work on Linux
  - Watch out the outputs
- Read the task assignments carefully
- Pay attention to the memory address when you place kernel images

Tips

- About asking questions
  - Think and try to describe your problem clearly
    - Pls. do not just show us a screenshot
  - Discuss with your groupmate/classmate
  - Google search is a good way to help you