CS 422/522 Design & Implementation of Operating Systems

Lecture 3: Project Overview

Zhong Shao
Dept. of Computer Science
Yale University

1

Debugging as engineering

- Much of your time in this course will be spent debugging
  - In industry, 50% of software dev is debugging
  - Even more for kernel development
- How do you reduce time spent debugging?
  - Produce working code with smallest effort
- Optimize a process involving you, code, computer

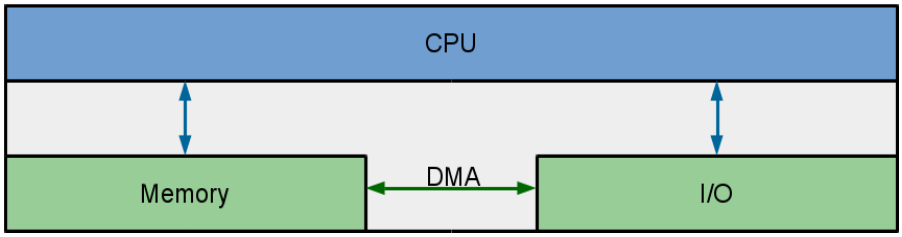
2

Debugging as science

- Understanding -> design -> code
  - not the opposite
- Form a hypothesis that explains the bug
  - Which tests work, which don't. Why?
  - Add tests to narrow possible outcomes
- Use best practices
  - Always walk through your code line by line
  - Module tests – narrow scope of where problem is
  - Develop code in stages, with dummy replacements for later functionality

3

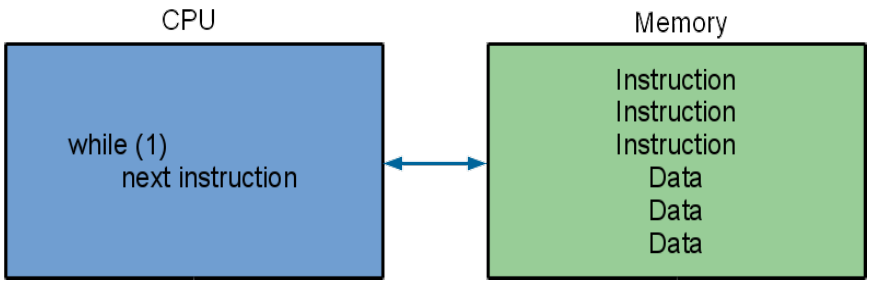
x86 abstract model



- I/O: Communicating data to and from devices
- CPU: Logic for performing computation
- Memory: Storage

4

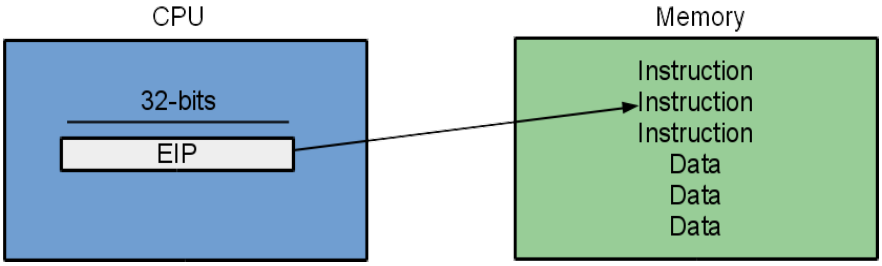
x86 CPU/memory interaction



- Memory stores instruction and data
- CPU interprets instructions

5

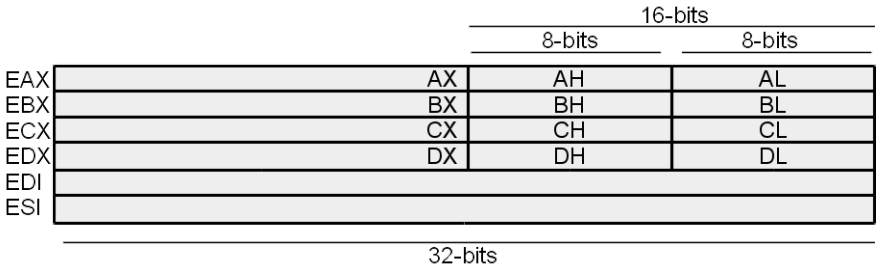
x86 implementation



- EIP points to next instruction
- Incremented after each instruction
- x86 instructions are not fixed length
- EIP modified by CALL, RET, JMP, and conditional JMP

6

x86 general purpose registers (GPR)



- Temporary registers
- Contents may be changed by instructions
- Contents not changed by interrupts / exceptions / traps
- EDI/ESI used by string ops but also as GPR

7

x86 memory models

- **Real mode** with segmentation (16-bit mode)
  - Used by early OSes
  - All x86 still boots in Real Mode for "compatibility" reasons
  - You can only use 1MB memory (4-bit segment + 16-bit address)

    PhysicalAddress = segment * 16 + offset

- **Protected mode** w. segmentation & paging (32-bit)
  - 4GB memory
  - Segmentation done via GDT (Global Descriptor Table)
    - A code segment descriptor holding a base address
    - A data segment descriptor holding a base address
    - A TSS segment descritor …

8

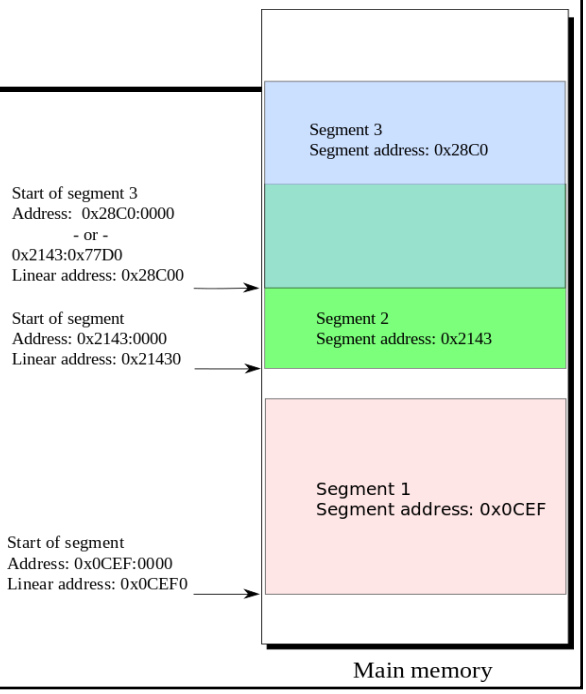
x86 segmentation registers

- 8086 registers 16-bit w/20-bit bus addresses
- Solution: segment registers
  - CS: code segment, EIP
  - SS: stack segment, ESP and EBP
  - DS: data segment, register mem ops
  - ES: string segment, string ops
- Linear address computation:
  - EIP => CS:EIP = 0x8000:0x1000 = 0x81000
  - ESP => SS:ESP = 0xF800:0x1000 = 0xF9000
  - (EAX) => DS:EAX = 0xC123:0x1000 = 0xC2230

9

x86 real mode

- 8086 16-bit with 20-bit address bus
- Stored in segment registers CS, DS, ES, FS
- Logical address: segment:offset
- Physical address: segment*0x10 + offset



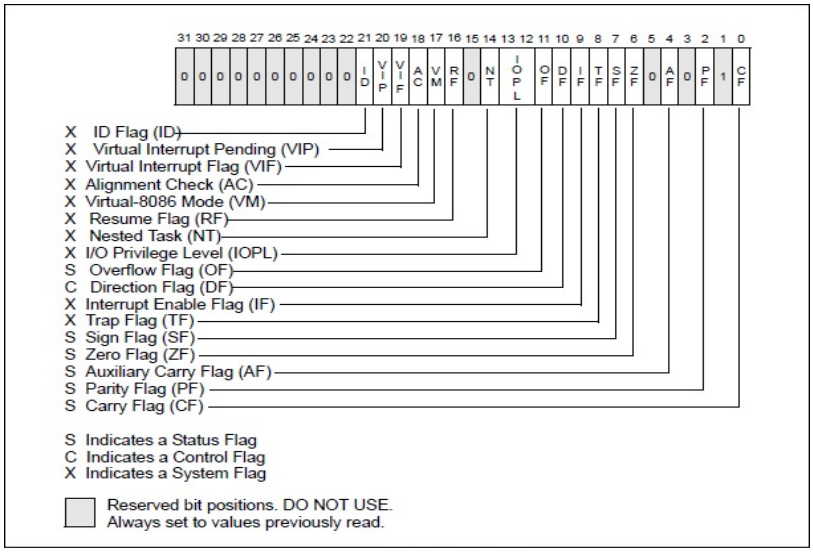
10

x86: the runtime stack

- Additional (temporary) storage
- Stack registers --- 32-bits long
- ESP – stack pointer
- EBP – base pointer

11

x86 EFLAGS register



EFLAGS Register

12

Using EFLAGS register

- Lots of conditional jumps
  en.wikibooks.org/wiki/X86_Assembly/Control_Flow

```
mov $5, %ecx
mov $5, %edx
cmp %ecx, %edx # ZF = 1
je equal
...
equal:
...
```

13

x86 assembly

We will use AT&T syntax

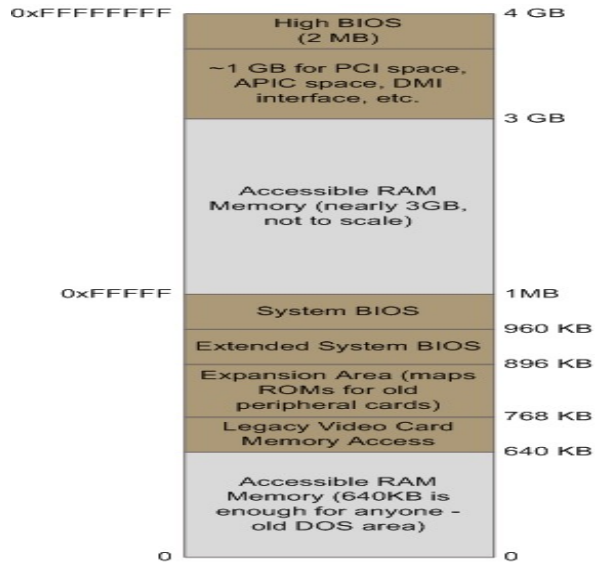
```
int main(void)
{
  return f(3) + 1;
}

int f(int x)
{
  return x + 4;
}
```

```
_main:
  pushl %ebp              # prologue
  movl %esp, %ebp
  pushl $3                # body
  call _f
  addl $1, %eax
  movl %ebp, %esp
  popl %ebp
  ret
_f:
  pushl %ebp
  movl %esp, %ebp
  pushl %ebx        # don't clobber registers
  movl 8(%ebp), %ebx      # access argument
  addl $4, %ebx
  movl %ebx, %eax
  popl %ebx               # restore
  movl %ebp, %esp         # epilogue
  popl %ebp
  ret
```

14

x86 memory layout



15

CS422/522 Lab 1: Bootloader & Physical Memory Management (due 9/16/2021)

- Learn how to use git
- Part 1: PC Bootstrap
  - x86 assembly & QEMU & BIOS
- Part 2: Bootloader
  - Learn how to use QEMU & GDB & read ELF file
- Part 3: Physical Memory Management
  - The MATIntro Layer
  - The MATInit Layer
  - The MATOp Layer
- Enrichment (optional)

16