WHY DO YOU HAVE TO TAKE THIS STUPID CLASS

Abstraction is good, but don’t forget reality:

Most CS classes emphasize abstraction. Not this one.
WHY DO YOU HAVE TO TAKE THIS STUPID CLASS

- People don’t just write programs in one language for one platform anymore. Real projects have lots of parts.
WHY DO YOU HAVE TO TAKE THIS STUPID CLASS
WHY DO YOU HAVE TO TAKE THIS STUPID CLASS

- People don’t just write programs in one language for one platform anymore. Real projects have lots of parts.
- Computers are changing: parallelism is much more important today than it was in the 90s.
- Stuff you learn here will be used in security, OS, compilers, architecture, IoT, etc.
40 Years of Microprocessor Trend Data

- Transistors (thousands)
- Single-Thread Performance (SpecINT x 10^3)
- Frequency (MHz)
- Typical Power (Watts)
- Number of Logical Cores

Year
MY GOALS FOR YOU

1. Have a gut feeling for what memory is.
2. Write a few bare metal programs that aren’t constrained by an OS.
3. Understand how the computer runs your program.
COURSE OUTLINE

- 1st Five Weeks: Assembly Language Programming
- 2nd Five Weeks: C Programming
- Last Five Weeks: Final Project
ABSTRACTIONS IN A COMPUTER

- Application
- Libraries
- Operating System
- Hypervisor
- Instruction Set Architecture
- Register Transfer Level (RTL)
- Logic
- Circuits
- Devices

This Class
LABS

- Lab is a time when you can do your homework (with help from Neil and others).
- Lab sessions will be held Thursdays from 4-6 PM in Doyle 314.
REQUIRED MATERIALS

- Book: Computer Systems: A Programmer’s Perspective
- You need a laptop with at least 8 GB RAM to run VMWare.
- Download VMWare (or VirtualBox), link on course website.
THE BOOK

- Not Required.
- Buy it if you like books.
- I have a PDF version.
GRADING

- No quizzes or exams. Your whole grade is based on homework and final project.
- No partial credit for code that doesn’t compile.
- Start homework on Tuesday/ Wednesday so you can get help on Thursday in lab if you get stuck.

<table>
<thead>
<tr>
<th>Category</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework</td>
<td>30%</td>
</tr>
<tr>
<td>Participation</td>
<td>10%</td>
</tr>
<tr>
<td>Progress</td>
<td>10%</td>
</tr>
<tr>
<td>Final Project</td>
<td>40%</td>
</tr>
</tbody>
</table>
DOING YOUR OWN WORK

- Do not share code.
- Do not copy code from the internet.
- You might want to save them for the end of the semester.
SLOP DAYS

- Each student gets five slop days to use during the semester.
- Can’t use more than two slop days on one assignment.
CODING STYLE

1. Every function should have a header explaining what it does. For example:

```c
/*
 * memcpy()
 *
 * Copies count bytes from src to dest. Returns
 * the number of bytes copied or a negative number
 * in case of error.
 */

int memcpy(void *dest, void *src, unsigned int count) {
```
Every function should have a header explaining what it does.

Functions written in assembly language also need a stack frame diagram. For example:

```
; memcpy
; ------------
; | count       | 2 bytes
; ------------
; | src         | 2 bytes
; ------------
; | dest        | 2 bytes
; ------------
; | Ret Addr    | 2 bytes
; ------------
; | Caller’s BP | 2 bytes
; ------------
; Copies count bytes from src to dest. Returns...
memcpy:
```
CODING STYLE

1. Every function should have a header explaining what it does.
2. Functions written in assembly language also need a stack frame diagram. For example:
3. Indent properly.

```c
for(k = 0; k < PAGE_SIZE; k++){
    if(page->next != NULL){
        page = page->next;  // NOOOOOO!!!!!!!
    }
}
```
CODING STYLE

1. Every function should have a header explaining what it does.
2. Functions written in assembly language also need a stack frame diagram.
   For example:
3. Indent properly.
4. Comment your code

```c
for(k = 0; k < PAGE_SIZE; k++){ // Loop thru each page...
    if(page->next != NULL){ // Don’t dereference NULL ptr.
        page = page->next; // Get next element of list
    }
}
```
PROGRAMMER’S MODEL OF X86
<table>
<thead>
<tr>
<th>Data Registers</th>
<th>Address Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX</td>
<td>SI</td>
</tr>
<tr>
<td>BX</td>
<td>DI</td>
</tr>
<tr>
<td>CX</td>
<td>BP</td>
</tr>
<tr>
<td>DX</td>
<td>SP</td>
</tr>
<tr>
<td></td>
<td>IP</td>
</tr>
</tbody>
</table>
### Programmer’s Model of X86: Inside the CPU

<table>
<thead>
<tr>
<th>Data Registers</th>
<th>Address Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX</td>
<td>SI</td>
</tr>
<tr>
<td>BX</td>
<td>DI</td>
</tr>
<tr>
<td>CX</td>
<td>BP</td>
</tr>
<tr>
<td>DX</td>
<td>SP</td>
</tr>
<tr>
<td></td>
<td>IP</td>
</tr>
</tbody>
</table>

```
mov ax, 100h
mov bx, 200h
add ax, bx
cmp ax, 200h
```
PROGRAMMER’S MODEL OF X86: INSIDE THE CPU

Data Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX</td>
<td>0100</td>
</tr>
<tr>
<td>BX</td>
<td></td>
</tr>
<tr>
<td>CX</td>
<td></td>
</tr>
<tr>
<td>DX</td>
<td></td>
</tr>
</tbody>
</table>

Address Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td></td>
</tr>
<tr>
<td>DI</td>
<td></td>
</tr>
<tr>
<td>BP</td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td></td>
</tr>
</tbody>
</table>

mov ax,100h
mov bx,200h
add ax,bx
cmp ax,200h
PROGRAMMER’S MODEL OF X86: INSIDE THE CPU

Data Registers

AX: 0100
BX: 0200
CX
DX

Address Registers

SI
DI
BP
SP
IP

Instructions:

- mov ax,100h
- mov bx,200h
- add ax,bx
- cmp ax,200h
PROGRAMMER’S MODEL OF X86: INSIDE THE CPU

**Data Registers**
- AX: 0300
- BX: 0200
- CX: 
- DX: 

**Address Registers**
- SI: 
- DI: 
- BP: 
- SP: 
- IP: 

**Code Snippet**
```assembly
mov ax, 100h
mov bx, 200h
add ax, bx
cmp ax, 200h
```
THE ONLY THING A COMPUTER KNOWS HOW TO DO IS EXECUTE INSTRUCTIONS.

if( a < 5 ) {
    b += a;
    a++;
}

cmp ax,5
jge .not_less_than
add bx,ax
inc ax

.not_less_than:
...
KINDS OF INSTRUCTIONS

- Arithmetic
  - Add, subtract, multiply, divide
- Logic
  - AND, OR, NOT, XOR
- Shifts
  - Left shift, right shift, rotate, etc.

- Control
  - Branch/Jump
  - Procedure calls
- Memory Accesses
  - Load/store
THE ONLY THING A COMPUTER KNOWS HOW TO DO IS EXECUTE INSTRUCTIONS.
HOMEWORK

• Download and install emu8086.
  • You need Windows: use VMWare if you have a mac.
  • If you need help, come to lab on Thursday.
• Sign up for GitHub if you don’t have an account.
• Send me you GitHub username. neil@cs.luc.edu