Introduction to Computer Systems
15-213/18-243, spring 2009
Recitation, Jan. 14th
Agenda

- News
- Stack discipline review
- lea vs. mov
- Buflab introduction
News

- Datalab feedback is available on autolab and on paper
- Bomblab is due tomorrow (Tuesday)
- Buflab comes out tomorrow and is due next Thursday (more on that later)
- Exam 1 is two weeks from tomorrow
Stack Layout and Discipline
IA32 Stack

- Region of memory managed with stack discipline
- “Grows” toward lower addresses
- Register %esp indicates lowest stack address
  - address of “top” element
  - stack pointer
IA32 Stack Pushing

Pushing

- `pushl Src`
- Fetch operand from `Src`
  - A register: `%ebp`
  - Memory: `8(%ebp)`
- Decrement `%esp` by 4
- Store operand in memory at address given by `%esp`
IA32 Stack Popping

**Popping**

- `popl Dest`
- Read operand at address given by `%esp`
- Increment `%esp` by 4
- Write to `Dest`
Stack Operation Examples

pushl %eax

popl %edx

%eax 213
%edx 555
%esp 0x108
Procedure Control Flow

- Use stack to support procedure call and return

**Procedure call:**
- `call label` Push return address on stack; Jump to `label`

**Return address value**
- Address of instruction *after* call
- Example from disassembly
  - `804854e: e8 3d 06 00 00`  `call 8048b90 <main>`
  - `8048553: 50`  `pushl %eax`
  - Return address = 0x8048553

**Procedure return:**
- `ret` Pop address from stack; Jump to address
Procedure Call Example

```
804854e:   e8 3d 06 00 00              call   8048b90 <main>
8048553:   50                           pushl  %eax

804854e:           call   8048b90
                  50 50 0x108 0x10c 0x110 0x104

%esp  0x108
0x108  123
0x10c
0x110
%eip  0x804854e
%eip  0x804854e
%esp  0x108
%esp  0x104
%esp  0x8048553
%eip  0x8048b90
%eip  0x8048b90
%eip  is program counter
```
Procedure Return Example

8048591: c3  ret

%esp  0x104
%eip  0x8048591

0x110 0x10c  0x108  0xc  0x104
0x8048553
123

%esp  0x104
%eip  0x8048591

ret

0x110 0x10c  0x108  0xc  0x108
0x8048553
123

%esp  0x108
%eip  0x8048553

%eip is program counter
IA32/Linux Stack Frame

Current Stack Frame ("Top" to "Bottom")
- Parameters for function about to call
  - "Argument build"
- Local variables
  - If don't all fit in registers
- Caller's saved registers
- Caller's saved frame pointer

Caller's Stack Frame
- Return address
  - Pushed by call instruction
- Arguments for this call
lea vs. mov
**lea vs. mov**

- Problem point for many in bomblab
- lea = 'Load effective address'
  - First operand must be in the form of a memory reference – lea 0x4(%eax), %ecx
  - The 'effective address' of the first operand is calculated (in this case, %eax + 4).
  - Instead of reading Mem[%eax + 4], the value %eax + 4 is copied into %ecx

- Remember the memory addressing forms:
  - (reg) => reg
  - d(reg) => reg + d
  - (reg, reg2) => reg + reg2
  - d(reg, regi, scale) => reg + regi * scale + d (with scale = 1, 2, 4 or 8)
  - All forms are generalizations of this one: if you leave out d or regi it's assumed to be 0, if you leave out scale it's assumed to be 1.
Demo time!
**Buflab Summary**

- The bad news: you only have 9 days for this lab
- The good news: it can be completed in about an hour, *if you really understand stacks and function call conventions*
- New this semester: The last stage, nitroglycerin, is now a required part of the lab and worth a significant amount of points
- buflab and this recitation will be entirely 32-bit, but you're still responsible for knowing how stacks and function calls work on 64-bit machines (read the handout on the website)
Demo time!
Disclaimer

That demo was very simple, you'll have to construct strings with real information in them for buflab:

- Encode a special value in the string
- Avoid corrupting the stack
- The last two stages require hand-writing a few lines of assembly

**DRAW PICTURES OF STACKS**

- Seriously. You'll need to keep track of the position and value of a number of different things that are on the stack in this lab, and it won't all reliably fit in your short-term memory.
- If you're having trouble with this lab, we will be much more willing and able to help you if you bring a picture of what you think the stack looks like before and after your string is entered.
Useful Additional Reading

  - “Smashing the Stack for Fun and Profit”
  - Very long, detailed, explanation of buffer overflows
  - Much more information than you need for the lab, but still fun and useful
Questions?
(stacks, buflab, bomblab)