Midterm Review

15-213: Introduction to Computer Systems
October 14, 2013
Agenda

- Midterm
- Brief Overview of few Topics
- Practice Questions
Midterm

- **Wed Oct 16th to Sun Oct 20th.**
  - Check timings for each day on the [website](#).
  - Duration - 80 minutes nominal time, but you can have upto 4 hrs to finish the exam.

- **Cheat Sheet – ONE double sided 8½ x 11 paper**
  - No worked out problems in that sheet.

- **What to study?**
  - Chapters 1-3 and Chapter 6
  - Note that in previous years, Chapter 6 (memory hierarchy) was covered by Exam 2

- **How to Study?**
  - Read each chapter 3 times, work practice problems and do problems from previous exams.
Floating Point

- Sign, Exponent, Mantissa
  - \((-1)^s \times M \times 2^E\)

- Bias \((2^{k-1} - 1)\)

- Denormalized
  - \(\text{exp} = 000...000\)
  - \(E = 1 - \text{bias}\)
  - Small values close to zero.

- Special Values
  - \(\text{exp} = 111...111\)
  - +/-inf, NaN
Floating Point (contd)

- **Normalized**
  - exp != 0 and exp != all ones
  - E = exp – bias

- **Rounding**
  - Round-up – if the spilled bits are greater than half
  - Round-down – if the spilled bits are less than half
  - Round to even – if the spilled bits is exactly equal to half
**Problem 3. (6 points):**

*Floating point encoding.* In this problem, you will work with floating point numbers based on the IEEE floating point format. We consider two different 6-bit formats:

**Format A:**

- There is one sign bit $s$.
- There are $k = 3$ exponent bits. The bias is $2^{k-1} - 1 = 3$.
- There are $n = 2$ fraction bits.

**Format B:**

- There is one sign bit $s$.
- There are $k = 2$ exponent bits. The bias is $2^{k-1} - 1 = 1$.
- There are $n = 3$ fraction bits.

For formats A and B, please write down the binary representation for the following (use round-to-even). Recall that for denormalized numbers, $E = 1 - \text{bias}$. For normalized numbers, $E = e - \text{bias}$. 

<table>
<thead>
<tr>
<th>Value</th>
<th>Format A Bits</th>
<th>Format B Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>0 000 00</td>
<td>0 00 000</td>
</tr>
<tr>
<td>One</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11/8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ASSEMBLY REVIEW
Assembly Loop

080483a0 <mystery>:

- push %ebp
- mov %esp,%ebp
- push %ebx
- mov 0xc(%ebp),%ebx
- mov 0x8(%ebp),%ecx
- test %ebx,%ebx
- jle 80483c5 <mystery+0x25>
- xor %eax,%eax
- mov (%ecx,%eax,4),%edx
- test $0x1,%dl
- jne 80483be <mystery+0x1e>
- add $0x1,%edx
- mov %edx,(%ecx,%eax,4)
- add $0x1,%eax
- cmp %ebx,%eax
- jne 80483b0 <mystery+0x10>
- pop %ebx
- pop %ebp
- ret
Assembly Loop

```c
void mystery(int *array, int n)
{
    int i;
    for(_________; __________; ________)
    {
        if(_____________ == 0)
            ____________;
    }
}
```
Assembly – Stack

- How arguments are passed to a function
  - IA-32
  - X86-64

- Return value from a function

- How these instructions modify stack
  - call, leave, ret
  - pop, push
Given assembly code of `foo()` and `bar()`, Draw a detailed picture of the stack, starting with the caller invoking `foo(3, 4, 5)`.

Value of `%ebp` when `foo` is called: `0xffffd858`
Return address in function that called `foo`: `0x080483c9`

<table>
<thead>
<tr>
<th>Stack address</th>
<th>The diagram starts with the arguments for <code>foo()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>0xffffd850</code></td>
<td>5</td>
</tr>
<tr>
<td><code>0xffffd84c</code></td>
<td>+</td>
</tr>
<tr>
<td><code>0xffffd848</code></td>
<td>+</td>
</tr>
<tr>
<td><code>0xffffd844</code></td>
<td>+</td>
</tr>
<tr>
<td><code>0xffffd840</code></td>
<td>+</td>
</tr>
<tr>
<td><code>0xffffd83c</code></td>
<td>+</td>
</tr>
<tr>
<td><code>0xffffd838</code></td>
<td>+</td>
</tr>
<tr>
<td><code>0xffffd834</code></td>
<td>+</td>
</tr>
<tr>
<td><code>0xffffd830</code></td>
<td>+</td>
</tr>
</tbody>
</table>
int bar (int a, int b) {
    return a + b;
}

int foo(int n, int m, int c) {
    c += bar(m, n);
    return c;
}

08048374 <bar>:
8048374:   55              push   %ebp
8048375:   89 e5           mov     %esp,%ebp
8048377:   8b 45 0c        mov     0xc(%ebp),%eax
804837a:   03 45 08        add     0x8(%ebp),%eax
804837d:   5d              pop     %ebp
804837e:   c3              ret

0804837f <foo>:
804837f:   55              push   %ebp
8048380:   89 e5           mov     %esp,%ebp
8048382:   83 ec 08        sub     $0x8,%esp
8048385:   8b 45 08        mov     0x8(%ebp),%eax
8048388:   89 44 24 04     mov     %eax,0x4(%esp)
804838c:   8b 45 0c        mov     0xc(%ebp),%eax
804838f:   89 04 24        mov     %eax,(%esp)
8048392:   e8 dd ff ff ff ff call    8048374 <bar>
8048397:   03 45 10        add     0x10(%ebp),%eax
804839a:   c9              leave
804839b:   c3              ret
Array Access

- Start with the C code
- Then look at the assembly Work backwards!
- Easiest to just do an example
int array1[H][J];
int array2[J][H];

int copy_array(int x, int y) {
    array2[y][x] = array1[x][y];

    return 1;
}

Suppose the above C code generates the following x86-64 assembly code:

# On entry:
#   %edi = x
#   %esi = y
#
copy_array:
    movslq  %edi,%rdi
    movslq  %esi,%rsi
    movq    %rdi, %rax
    leaq    (%rsi,%rsi,2), %rdx
    salq    $5, %rax
    subq    %rdi, %rax
    leaq    (%rdi,%rdx,2), %rdx
    addq    %rsi, %rax
    movl    array1(,%rax,4), %eax
    movl    %eax, array2(,%rdx,4)
    movl    $1, %eax
    ret
Caching Concepts

- Dimensions: S, E, B
  - S: Number of sets
  - E: Associativity – number of lines per set
  - B: Block size – number of bytes per block (1 block per line)

- Given Values for S,E,B,m
  - Find which address maps to which set
  - Is it a Hit/Miss. Is there an eviction
  - Hit rate/Miss rate
Questions/Advice

- Relax!
- Work Past exams!
- Email us - (15-213-staff@cs.cmu.edu)