EECS 213: Midterm Exam

From a tour of computer systems to machine level representation of programs.

Spring 2007

Name:
Major/Department/School:

Some words of advice:

• Read all the questions first.
• Start from the easiest one and leave the harder ones for the end.
• Approximate results are almost always a valid answer; for sure I do not need 5-decimal precision answers!
• This is an Open Book exam; you may use any book or notes you like.
• Write clearly; if I can’t read it I can’t grade it.

Good luck!

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Credited</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

Problems ...

1. (10 points) Consider a 5-bit two’s complement representation. Fill in the empty boxes in the following table. Addition and subtraction should be performed based on the rules for 5-bit, two’s complement arithmetic.

**Answers in bold.**

<table>
<thead>
<tr>
<th>Number</th>
<th>Decimal Representation</th>
<th>Binary Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>0</td>
<td>0 0000</td>
</tr>
<tr>
<td>n/a</td>
<td>−2</td>
<td>1 1110</td>
</tr>
<tr>
<td>n/a</td>
<td>9</td>
<td>0 1001</td>
</tr>
<tr>
<td>n/a</td>
<td>−14</td>
<td>1 0010</td>
</tr>
<tr>
<td>n/a</td>
<td>12</td>
<td>0 1100</td>
</tr>
<tr>
<td>n/a</td>
<td>−12</td>
<td>1 0100</td>
</tr>
<tr>
<td>Tmax</td>
<td>15</td>
<td>0 1111</td>
</tr>
<tr>
<td>Tmin</td>
<td>−16</td>
<td>1 0000</td>
</tr>
<tr>
<td>Tmin+Tmin</td>
<td>0</td>
<td>0 0000</td>
</tr>
<tr>
<td>Tmin+1</td>
<td>−15</td>
<td>1 0001</td>
</tr>
<tr>
<td>Tmax+1</td>
<td>−16 (Tmin)</td>
<td>1 0000</td>
</tr>
<tr>
<td>−Tmax</td>
<td>−15</td>
<td>1 0001</td>
</tr>
<tr>
<td>−Tmin</td>
<td>−16 (Tmin)</td>
<td>1 0000</td>
</tr>
</tbody>
</table>
2. (8 points) Consider the following 5-bit floating point representation based on the IEEE floating point format. There is a sign bit in the most significant bit. The next three bits are the exponent, with an exponent bias is 3. The last bit is the fraction. The rules are like those in the IEEE standard (normalized, denormalized, representation of 0, infinity, and NAN).

As discussed in class, we consider the floating point format to encode numbers in a form:

\[ V = (-1)^s \times M \times 2^E \]

where \( M \) is the significand and \( E \) is the exponent.

Fill in missing entries in the table below with the following instructions for each column:

**Description:** Some unique property of this number, such as, “The largest denormalized value.”

**Binary:** The 5 bit representation.

**\( M \):** The value of the Mantissa written in decimal format.

**\( E \):** The integer value of the exponent.

**Value:** The numeric value represented, written in decimal format.

You need not fill in entries marked “—”. For the arithmetic expressions, recall that the rule with IEEE format is to round to the number nearest the exact result. Use “round-to-even” rounding.

**Answers in bold.**

<table>
<thead>
<tr>
<th>Description</th>
<th>Binary</th>
<th>( M )</th>
<th>( E )</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minus Zero</td>
<td>10000</td>
<td>0</td>
<td>-2.0</td>
<td>-0.0</td>
</tr>
<tr>
<td>Positive Infinity</td>
<td>01110</td>
<td>—</td>
<td>—</td>
<td>+\infty</td>
</tr>
<tr>
<td>Largest Number</td>
<td>01101</td>
<td>1.5</td>
<td>3</td>
<td>12.0</td>
</tr>
<tr>
<td>Smallest number &gt; 0</td>
<td>00001</td>
<td>0.5</td>
<td>-2</td>
<td>0.125</td>
</tr>
<tr>
<td>One</td>
<td>00110</td>
<td>1.0</td>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>4.0 - 0.75</td>
<td>01001</td>
<td>1.5</td>
<td>1</td>
<td>3.0</td>
</tr>
<tr>
<td>2.0 + 3.0</td>
<td>11010</td>
<td>1.0</td>
<td>2</td>
<td>4.0</td>
</tr>
</tbody>
</table>
3. (10 points) A C function `looper` and the assembly code it compiles to on an IA-32 machine running Linux/GAS is shown below:

```assembly
looper:
    pushl %ebp
    movl %esp,%ebp
    pushl %esi
    pushl %ebx
    movl 8(%ebp),%ebx
    movl 12(%ebp),%esi
    xorl %edx,%edx
    xorl %ecx,%ecx
    cmpl %ebx,%edx
    jge .L25

.L27:
    movl (%esi,%ecx,4),%eax
    cmpl %edx,%eax
    jle .L28

.L28:
    incl %edx
    incl %ecx
    cmpl %ebx,%ecx
    jl .L27

.L25:
    movl %edx,%eax
    popl %ebx
    popl %esi
    movl %ebp,%esp
    popl %ebp
    ret
```

**Answer between ***

```c
int looper(int n, int *a) {
    int i;
    int x = ______________; *** = 0; ***
    for(i = __________; __________; *** 0; i < n ***
        i++) {
        if (___________________) *** a[i] > x or !(a[i] <= x) ***
            x = _________________; *** x = a[i] ***
        ________________; *** x++ ***
    }
    return x;
}
```

Based on the assembly code, fill in the blanks in the C source code.

**Notes:**

- You may only use the C variable names `n`, `a`, `i` and `x`, not register names.
- Use array notation in showing accesses or updates to elements of `a`. 

4. (10 points) Consider the following incomplete definition of a C struct along with the incomplete code for a function func given below.

Answer between ***

typedef struct node {

_______________ x; *** double ***

_______________ y; *** unsigned short ***

struct node *next;

struct node *prev;

} node_t;

node_t n;

void func() {

node_t *m;

m = ______________________; *** n.next->prev ***

m->y /= 16;

return;
}

When this C code was compiled on an IA-32 machine running Linux, the following assembly code was generated for function func.

```
func:
pushl %ebp
movl n+12,%eax
movl 16(%eax),%eax
movl %esp,%ebp
movl %ebp,%esp
shrw $4,8(%eax)
popl %ebp
ret
```

Given these code fragments, fill in the blanks in the C code given above. Note that there is a unique answer.

The types must be chosen from the following table, assuming the sizes and alignment given.

<table>
<thead>
<tr>
<th>Type</th>
<th>Size (bytes)</th>
<th>Alignment (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>
5. (12 points) The following problem concerns the following, low-quality code:

```c
void foo(int x)
{
    int a[3];
    char buf[4];
    a[0] = 0xF0F1F2F3;
    a[1] = x;
    gets(buf);
    printf("a[0] = 0x%x, a[1] = 0x%x, buf = %s\n", a[0], a[1], buf);
}
```

In a program containing this code, procedure `foo` has the following disassembled form on an IA32 machine:

```
080485d0 <foo>:
080485d0: 55   pushl %ebp
080485d1: 89 e5 movl %esp,%ebp
080485d3: 83 ec 10 subl $0x10,%esp
080485d6: 53   pushl %ebx
080485d7: 8b 45 08 movl 0x8(%ebp),%eax
080485da: c7 45 f4 f3 f2 movl $0xf0f1f2f3,0xfffffff4(%ebp)
080485df: f1 f0  # gets
080485e1: 89 45 f8 movl %eax,0xffffffff8(%ebp)
080485e4: 8d 5d f0 leal 0xffffffff0(%ebp),%ebx
080485e7: 53   pushl %ebx
080485e8: e8 b7 fe ff ff call 80484a4 <_init+0x54>  # gets
080485ed: 53   pushl %ebx
080485ee: 8b 45 f8 movl 0xffffffff8(%ebp),%eax
080485f1: 50   pushl %eax
080485f2: 8b 45 f4 movl 0xffffffff4(%ebp),%eax
080485f5: 50   pushl %eax
080485f6: 68 ec 90 04 08 pushl $0x80490ec
080485fb: e8 94 fe ff ff call 8048494 <_init+0x44>  # printf
08048600: 8b 5d ec movl 0xffffffffec(%ebp),%ebx
08048603: 89 ec movl %ebp,%esp
08048605: 5d popl %ebp
08048606: c3    ret
08048607: 90    nop
```

For the following questions, recall that:

- `gets` is a standard C library routine.
- IA32 machines are little-endian.
- C strings are null-terminated (i.e., terminated by a character with value 0x00).
- Characters '0' through '9' have ASCII codes 0x30 through 0x39.
Fill in the following table indicating where on the stack the following program values are located. Express these as decimal offsets (positive or negative) relative to register %ebp:

**Answers in bold.**

<table>
<thead>
<tr>
<th>Program Value</th>
<th>Decimal Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>-12</td>
</tr>
<tr>
<td>a[2]</td>
<td>-4</td>
</tr>
<tr>
<td>x</td>
<td>+8</td>
</tr>
<tr>
<td>buf</td>
<td>-16</td>
</tr>
<tr>
<td>Saved value of register %ebx</td>
<td>-20</td>
</tr>
</tbody>
</table>