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>??</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>??</td>
<td></td>
</tr>
</tbody>
</table>

Problems ...
1. (?? points) In the following questions assume the variables a and b are signed integers and that the machine uses two’s complement representation. Also assume that MAX_INT is the maximum integer, MIN_INT is the minimum integer, and W is one less than the word length (e.g., W = 31 for 32-bit integers).

Match each of the descriptions on the left with a line of code on the right (write in the letter). You will be given 2 points for each correct match.

1. One’s complement of a

   a. \((-a | (b \sim (MIN\_INT + MAX\_INT)))\)

2. a.

   b. \(((a \sim b) \& \sim b) | (\sim(a \sim b) \& b)\)

3. a & b.

   c. \(1 + (a << 3) + \sim a\)

4. a * 7.

   d. \((a << 4) + (a << 2) + (a << 1)\)

5. a / 4.

   e. \(((a < 0) ? (a + 3) : a) >> 2\)

6. (a < 0) ? 1 : -1.

   f. \(a = (MIN\_INT + MAX\_INT)\)

   g. \((a | (\sim a + 1)) >> W) \& 1\)

   h. \((a >> W) << 1\)

   i. \(a >> 2\)
2. (?? points) Consider a 6-bit two's complement representation. Fill in the empty boxes in the following table:

<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></td>
</tr>
<tr>
<td>n/a</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td>-10</td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td>01 1010</td>
<td></td>
</tr>
<tr>
<td>n/a</td>
<td>10 0110</td>
<td></td>
</tr>
<tr>
<td>Tmax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tmin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tmax + Tmax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tmin + Tmin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tmin + 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tmin - 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tmax + 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-TMax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-TMin</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. (?? points) Consider the source code below, where \( M \) and \( N \) are constants declared with \#define.

```c
int array1[M][N];
int array2[N][M];

int copy(int i, int j)
{  
array1[i][j] = array2[j][i];  
}
```

Suppose the above code generates the following assembly code:

```assembly
copy:
  pushl %ebp
  movl %esp,%ebp
  pushl %ebx
  movl 8(%ebp),%ecx
  movl 12(%ebp),%ebx
  leal (%ecx,%ecx,8),%edx
  sall $2,%edx
  movl %ebx,%eax
  sall $4,%eax
  subl %ebx,%eax
  sall $2,%eax
  movl array2(%eax,%ecx,4),%eax
  movl %eax,array1(%edx,%ebx,4)
popl %ebx
  movl %ebp,%esp
  popl %ebp
ret
```

What are the values of \( M \) and \( N \)?

\[ M = \]

\[ N = \]
4. (?? points) Consider the following assembly code for a C for loop:

```
loop:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%ecx
  movl 12(%ebp),%edx
  xorl %eax,%eax
  cmpl %edx,%ecx
  jle .L4
  .L6:
    decl %ecx
    incl %edx
    incl %eax
    cmpl %edx,%ecx
    jg .L6
  .L4:
    incl %eax
    movl %ebp,%esp
    popl %ebp
    ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. (Note: you may only use the symbolic variables x, y, and result in your expressions below — do not use register names.)

```
int loop(int x, int y)
{
  int result;

  for (_____________; ___________; result++) {
      __________;
      __________;
  }

  __________;
  return result;
}
```
5. (?? points) This next problem will test your understanding of stack frames. It is based on the following recursive C function:

```c
int silly(int n, int *p)
{
    int val, val2;
    if (n > 0)
        val2 = silly(n << 1, &val);
    else
        val = val2 = 0;
    *p = val + val2 + n;
    return val + val2;
}
```

This yields the following machine code:

```
silly:
pushl %ebp
movl %esp,%ebp
subl $20,%esp
pushl %ebx
movl 8(%ebp),%ebx
testl %ebx,%ebx
    jle .L3
addl $-8,%esp
leal -4(%ebp),%eax
pushl %eax
leal (%ebx,%ebx),%eax
pushl %eax
call silly
jmp .L4
    .p2align 4,,7
.L3:
xorl %eax,%eax
movl %eax,-4(%ebp)
.L4:
    movl -4(%ebp),%edx
    addl %eax,%edx
    movl 12(%ebp),%eax
    addl %edx,%ebx
    movl %ebx,(%eax)
    movl -24(%ebp),%ebx
    movl %edx,%eax
    movl %ebp,%esp
    popl %ebp
    ret
```

(a) Is the variable val stored on the stack? If so, at what byte offset (relative to %ebp) is it stored, and why is it necessary to store it on the stack?
(b) Is the variable `val2` stored on the stack? If so, at what byte offset (relative to `%ebp`) is it stored, and why is it necessary to store it on the stack?

(c) What (if anything) is stored at `-24(%ebp)`? If something is stored there, why is it necessary to store it?

(d) What (if anything) is stored at `-8(%ebp)`? If something is stored there, why is it necessary to store it?