Full Name: ________________________________

CS201 Section 001, Fall 2014

Exam A

Instructions:

• Make sure that your exam is not missing any sheets, then write your full name on the front.

• Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.

• The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.

• This exam is CLOSED BOOK. You may NOT use any books or notes. Good luck!

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (10pts):</td>
<td></td>
</tr>
<tr>
<td>2 (5pts):</td>
<td></td>
</tr>
<tr>
<td>3 (3pts):</td>
<td></td>
</tr>
<tr>
<td>4 (8pts):</td>
<td></td>
</tr>
<tr>
<td>5 (8pts):</td>
<td></td>
</tr>
<tr>
<td>6 (10pts):</td>
<td></td>
</tr>
<tr>
<td>7 (8pts):</td>
<td></td>
</tr>
<tr>
<td>TOTAL ( ):</td>
<td></td>
</tr>
</tbody>
</table>

Page 1 of 10
Problem 1. (12 points):
In the following questions assume the variables \( x \) and \( y \) are signed integers and that the machine uses two’s complement representation. Also assume that \( \text{MAX\_INT} \) is the maximum integer, \( \text{MIN\_INT} \) 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.

<table>
<thead>
<tr>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. One’s complement of ( x )</td>
<td>f</td>
</tr>
<tr>
<td>2. ( x )</td>
<td>b or e</td>
</tr>
<tr>
<td>3. ( x &amp; y )</td>
<td>a or j</td>
</tr>
<tr>
<td>4. ( x \times 7 )</td>
<td>c or j</td>
</tr>
<tr>
<td>5. 1</td>
<td>h</td>
</tr>
</tbody>
</table>

1. One’s complement of \( x \)  
Answer: f  

2. \( x \)  
Answer: b or e  

3. \( x \& y \)  
Answer: a or j  

4. \( x \times 7 \)  
Answer: c or j  

5. 1  
Answer: h  

---

\[
\begin{align*}
\text{a. } &\frac{(x \ll 2) \& (y \ll 2)}{4} \\
\text{b. } &\lor\left(\neg x \& y\right) \lor \neg x \\
\text{c. } &1 + (x \ll 3) + \neg x \\
\text{d. } &((1 + (x \ll 4)) \gg 1) - 1 \\
\text{e. } &x \lor (y \& \neg y) \gg \text{MIN\_INT} + 3 \\
\text{f. } &x \hat{\lor} (\text{MIN\_INT} + \text{MAX\_INT}) \\
\text{g. } &\text{MIN\_INT} \& (\text{MIN\_INT} - \text{MAX\_INT}) \\
\text{h. } &\text{MAX\_INT} \& (\text{MIN\_INT} - \text{MAX\_INT}) \\
\text{i. } &\left(\left(x \hat{\lor} y\right) \lor \neg\left(x \hat{\lor} y\right)\right) \\
\text{j. } &\text{None of the above.}
\end{align*}
\]
Problem 2. (8 points):
Consider the source code below, where M and N are constants declared with \texttt{#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:

```
copy:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
    movl 8(%ebp),%ecx
    movl 12(%ebp),%ebx
    leal (,%ecx,4),%edx
    sall $1,%edx
    movl %ebx,%eax
    sall $5,%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?

\[ M = \]

\[ N = \]
Answer: M=31 N=4

copy:
    pushl %ebp          #setup
    movl %esp,%ebp     #setup
    pushl %ebx          #setup
    movl 8(%ebp),%ecx  # ecx = i
    movl 12(%ebp),%ebx # ebx = j
    leal (,%ecx,4),%edx # edx = 4 * i
    sall $1,%edx        # edx = (4 * i) * 2
    movl %ebx,%eax      # eax = j
    sall $5,%eax        # eax = (2^5 * j) = 32 * j
    subl %ebx,%eax      # eax = 32 * j - j = 31 * j
    sall $2,%eax        # eax = (31 * j) * 2
    movl array2(%eax,%ecx,4),%eax # array2 + eax + ecx * 2
    # array2 + (31 * j) * 2 + i * 2
    # array2 + COLS * j * sizeof(type) + i * sizeof(type)
    # COLS in array2 -> 31
    # M = 31
    movl %eax,array1(%edx,%ebx,4) # array1 + edx + ebx * 2
    # array1 + (4 * i) * 2 + j * 2
    # array1 + COLS * i * sizeof(type) + j * sizeof(type)
    # COLS in array1 -> 4
    # N = 4

popl %ebx
movl %ebp,%esp
popl %ebp
ret
Problem 3. (3 points):
Consider the following C functions and assembly code:

```c
int fun1(int a, int b)
{
    if (a < b)
        return a;
    else
        return b;
}

int fun2(int a, int b)
{
    if (b < a)
        return b;
    else
        return a;
}

int fun3(int a, int b)
{
    unsigned ua = (unsigned) a;
    if (ua < b)
        return b;
    else
        return ua;
}
```

```assembly
pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%edx
movl 12(%ebp),%eax
cmpl %eax,%edx
jge .L9
movl %edx,%eax
.L9:
    movl %ebp,%esp
    popl %ebp
    ret
```

Which of the functions compiled into the assembly code shown?
pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%edx  # edx = a
movl 12(%ebp),%eax  # eax = b
cmpl %eax,%edx  # if a-b >= 0:
jge .L9  # return eax // return b
movl %edx,%eax  # if a-b < 0:
                 # return edx // return a
.L9:
movl %ebp,%esp
popl %ebp
ret
Problem 4. (8 points):
Consider the following assembly representation of a function foo containing a for loop:

```assembly
foo:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
    movl 8(%ebp),%ebx
    leal 2(%ebx),%edx
    xorl %ecx,%ecx
    cmpl %ebx,%ecx
    jge .L4
    .L6:
    leal 5(%ecx,%edx),%edx
    leal 3(%ecx),%eax
    imull %eax,%edx
    incl %ecx
    cmpl %ebx,%ecx
    jl .L6
    .L4:
    movl %edx,%eax
    popl %ebx
    movl %ebp,%esp
    popl %ebp
    ret
```

Fill in the blanks to provide the functionality of the loop:

```c
int foo(int a)
{
    int i;
    int result = ____________;

    for( _______; _______; i++ ) {
        ______________;
        ______________;
    }
    return result;
}
```
Answer:

```c
int foo(int a)
{
    int i;
    int result = a + 2;

    for (i=0; i < a; i++) {
        result += (i + 5);
        result *= (i + 3);
    }
    return result;
}
```
The next problem concerns the following C code:

```c
/* copy string x to buf */
void foo(char *x) {
    int buf[1];
    strcpy((char *)buf, x);
}

void callfoo() {
    foo("abcdefghi");
}
```

Here is the corresponding machine code on a Linux/x86 machine:

```
080484f4 <foo>:
080484f4: 55 pushl %ebp
080484f5: 89 e5 movl %esp,%ebp
080484f7: 83 ec 18 subl $0x18,%esp
080484fa: 8b 45 08 movl 0x8(%ebp),%eax
080484fd: 83 c4 f8 addl $0xffffffff8,%esp
08048500: 50 pushl %eax
08048501: 8d 45 fc leal 0xfffffffc(%ebp),%eax
08048504: 50 pushl %eax
08048505: e8 ba fe ff ff call 80483c4 <strcpy>
0804850a: 89 ec movl %ebp,%esp
0804850c: 5d popl %ebp
0804850d: c3 ret

08048510 <callfoo>:
08048510: 55 pushl %ebp
08048511: 89 e5 movl %esp,%ebp
08048513: 83 ec 08 subl $0x8,%esp
08048516: 83 c4 f4 addl $0xffffffff4,%esp
08048519: 68 9c 85 04 08 pushl $0x804859c # push string address
0804851e: e8 d1 ff ff ff call 80484f4 <foo>
08048523: 89 ec movl %ebp,%esp
08048525: 5d popl %ebp
08048526: c3 ret
```
Problem 5. (8 points):
This problem tests your understanding of the stack discipline and byte ordering. Here are some notes to help you work the problem:

• `strcpy(char *dst, char *src)` copies the string at address `src` (including the terminating `\0` character) to address `dst`. It does **not** check the size of the destination buffer.

• Recall that Linux/x86 machines are Little Endian.

• You will need to know the hex values of the following characters:

<table>
<thead>
<tr>
<th>Character</th>
<th>Hex value</th>
<th>Character</th>
<th>Hex value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'a'</td>
<td>0x61</td>
<td>'f'</td>
<td>0x66</td>
</tr>
<tr>
<td>'b'</td>
<td>0x62</td>
<td>'g'</td>
<td>0x67</td>
</tr>
<tr>
<td>'c'</td>
<td>0x63</td>
<td>'h'</td>
<td>0x68</td>
</tr>
<tr>
<td>'d'</td>
<td>0x64</td>
<td>'i'</td>
<td>0x69</td>
</tr>
<tr>
<td>'e'</td>
<td>0x65</td>
<td>'\0'</td>
<td>0x00</td>
</tr>
</tbody>
</table>

Now consider what happens on a Linux/x86 machine when `callfoo` calls `foo` with the input string “abcdefggh”:

A. List the contents of the following memory locations immediately after `strcpy` returns to `foo`. Each answer should be an unsigned 4-byte integer expressed as 8 hex digits.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x__________</td>
<td>0x__________</td>
<td>0x__________</td>
</tr>
</tbody>
</table>

Solution: `buf[0] = 0x64636261 buf[1] = 0x68676665 buf[2] = 0x08040069`

B. Immediately **before** the `ret` instruction at address 0x0804850d executes, what is the value of the frame pointer register `%ebp`?

<table>
<thead>
<tr>
<th>%ebp</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x__________</td>
<td></td>
</tr>
</tbody>
</table>

Solution: ebp = 0x68676665 (previously undetermined)

C. Immediately **after** the `ret` instruction at address 0x0804850d executes, what is the value of the program counter register `%eip`?

<table>
<thead>
<tr>
<th>%eip</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x__________</td>
<td></td>
</tr>
</tbody>
</table>

Solution: eip = 0x08040069 (previously 0x08048523)
Problem 6. (10 points):
Consider the following C declaration:

```c
struct Node{
    char c;
    double value;
    struct Node* next;
    int flag;
    struct Node* left;
    struct Node* right;
};

typedef struct Node* pNode;

/* NodeTree is an array of N pointers to Node structs */
pNode NodeTree[N];
```

A. Using the template below (allowing a maximum of 32 bytes), indicate the allocation of data for a Node struct. Mark off and label the areas for each individual element (there are 6 of them). Cross hatch the parts that are allocated, but not used (to satisfy alignment).

Assume the Linux alignment rules discussed in Class 9. **Clearly indicate the right hand boundary of the data structure with a vertical line.**

Answer

```plaintext
| c | value | next | flag | left | right |
```

Page 11 of 10
B. For each of the four C references below, please indicate which assembly code section (labeled A – F) places the value of that C reference into register %eax. If no match is found, please write “NONE” next to the C reference.
The initial register-to-variable mapping for each assembly code section is:

%eax = starting address of the NodeTree array
%edx = i

---------------------------------------------------------------

C References:

1. ______ NodeTree[i]->flag
2. ______ NodeTree[i]->left->left->c
3. ______ NodeTree[i]->next->next->flag
4. ______ NodeTree[i]->right->left->left

---------------------------------------------------------------

Linux/IA32 Assembly:

A. sall $2, %edx
   leal (%eax,%edx),%eax
   movl 16(%eax),%eax
B. sall $2, %edx
   leal (%eax,%edx),%eax
   movl (%eax),%eax
   movl 20(%eax),%eax
C: sall $2,%edx
   leal (%eax,%edx),%eax
   movl 20(%eax),%eax
   movl 20(%eax),%eax
D: sall $2,%edx
   leal (%eax,%edx),%eax
   movl (%eax),%eax
   movl 16(%eax),%eax
E: sall $2, %edx
   leal (%eax,%edx),%eax
   movl (%eax),%eax
   movl 16(%eax),%eax
   movl 16(%eax),%eax
F: sall $2, %edx
   leal (%eax,%edx),%eax
   movl (%eax),%eax
   movl 12(%eax),%eax
   movl 16(%eax),%eax

ANSWER: 1:D, 2:NONE, 3:F, 4:B

sall $2, %edx # edx = sizeof(a pointer) = 4
leal (%eax,%edx),%eax # eax = address of node[i]
movl (%eax),%eax # eax = address of struct
movl offset(%eax) %eax # eax = element from struct offset
Problem 7. (8 points):
This problem tests your understanding of C data types and pointers. Consider the following C code:

```c
#include <stdlib.h>

#define ROW_SIZE = 5
#define COL_SIZE = 5

int main(int argc, char** argv) {
    long x[ROW_SIZE][COL_SIZE];
    long (*a)[ROW_SIZE] = &(x[3]);
    long (*b)[ROW_SIZE] = a;
    b++;
    long* c = &(x[3][4]);
    long* d = &c[-1];
    char* e = x;
    e++;
    unsigned long f = x[3] - x[1];
    unsigned long g = &(x[3]) - &(x[1]);
    long* h = x[3] - 2;
    return 0;
}
```

Assume that this program runs on an architecture where pointers are 8 bytes large, and the sizes of char, int, and long are 1, 4, and 8 bytes, respectively. Further assume that the value of pointer variable x is the address 1000. What are the values of variables a thru h just before the return statement?

a = ____________________
b = ____________________
c = ____________________
d = ____________________
e = ____________________
f = ____________________
g = ____________________
h = ____________________
Solution:

// a is a pointer to an array of 5 longs and points to:
// &x[0][0] + 3*sizeof(long[ROW_SIZE]) <- type that a points to
// 1000 + 3*8*ROW_SIZE
// 1120
long (*a)[ROW_SIZE] = &x[3];

// b is a pointer to an array of 5 longs and points to:
// &a[0]
// 1120
long (*b)[ROW_SIZE] = a;

// b points to:
// 1120 + sizeof(long[ROW_SIZE]) <- type that b points to
// 1120 + 40
// 1160
b++;

// c is a pointer to a long and points to:
// &x[0][0] + 3*ROW_SIZE*sizeof(long)
// + 4*sizeof(long)
// 1000 + 3*40 + 4*8
// 1152
long* c = &(x[3][4]);

// d is a pointer to a long and points to:
// &c[0] - sizeof(long) <- type that d points to
// 1144
long* d = &(c[0]);

// e is a pointer pointing to x
// e points to:
// &x[0] -> 1000
char* e = x;

// 1000 + sizeof(char) <- type that e points to
// 1001
e++;
// f is number of addressible long elements from x[3] to x[1]  
// type of x[3]/x[1] is array of longs
unsigned long f = x[3] - x[1];

// g is number of addressible long[5] elements from x[3] to x[1]  
// type of &x[3]/&x[1] is pointer to array of longs
unsigned long g = &x[3] - &x[1];

// h is a pointer to a long and points to:
// &x[3][0] – 2*sizeof(long) <- type that e points to
// 1104
long* h = x[3] - 2;