Andrew login ID: ______________________________

Full Name: ______________________________

Recitation Section: ______________________________

CS 15-213, Fall 2008

Exam 1

Thurs. September 25, 2008

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name, Andrew login ID, and recitation section (A–H) on the front.
- Write your answers in the space provided for the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 72 points.
- 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 OPEN BOOK. You may use any books or notes you like. No calculators or other electronic devices are allowed.
- Good luck!

1 (8):	
2 (10):	
3 (12):	
4 (9):	
5 (6):	
6 (8):	
7 (11):	
8 (8):	
TOTAL (72):	

Problem 1. (8 points):

For this problem, assume the following:

- We are running code on an 8-bit machine using two's complement arithmetic for signed integers.
- `short` integers are encoded using 4 bits.
- Sign extension is performed whenever a `short` is cast to an `int`

The following definitions are used in the table below:

```
short sa = -6;
int b = 2*sa;
short sc = (short)b;
int x = -64;
unsigned ux = x;
```


Fill in the empty boxes in the table. If the expression is cast to or stored in a `short`, use a 4-bit binary representation. Otherwise assume an 8-bit binary representation. The first 2 lines are given to you as examples, and you need not fill in entries marked with "—".

Expression	Decimal Representation	Binary Representation
Zero	0	`0000 0000`
`(short)0`	0	`0000`
—	-17	
—		`0010 1001`
`sa`		
`b`		
`sc`		
`ux`		
TMax		
TMax $-$ TMin		

Problem 2. (10 points):

Assume we are using a machine where data type `int` uses a 32-bit, two's complement representation, and right shifting is performed arithmetically. Data type `float` uses a 32-bit IEEE floating-point representation.

Consider the following definitions.

```
int i = hello();
float fi = i;
```

Answer the following questions. For each C-language expression in the first column, either

1. Mark that it is TRUE of all possible values returned by function `hello()`, and *provide an explanation of why it is true*.
2. Mark that it is possibly FALSE, and provide a counter-example.

Puzzle	True/False	Explanation/Counter-example
`(i ^ ~(i >> 31)) < 0`		
`-(i | (~i + 1)) > 0`		
`i > 0` ⇒ `i + (int) fi > 0`		
`fi > 0` ⇒ `fi + (float) i > 0`		
`i & 1 == ((int) fi) & 1`		

Problem 3. (12 points):

Consider the following two 8-bit floating point representations based on the IEEE floating point format. Neither has a sign bit—they can only represent nonnegative numbers.

1. Format A
   - There are $k = 3$ exponent bits. The exponent bias is 3.
   - There are $n = 5$ fraction bits.
2. Format B
   - There are $k = 5$ exponent bits. The exponent bias is 15.
   - There are $n = 3$ fraction bits.

Fill in the blanks in the table below by converting the given values in each format to the closest possible value in the other format. Express values as whole numbers (e.g., 17) or as fractions (e.g., 17/64). If necessary, you should apply the round-to-even rounding rule.

Format A		Format B	
Bits	Value	Bits	Value
`011 00000`	1	`01111 000`	1
			15
	$\frac{53}{16}$		
		`10100 110`	
`000 00001`			

Problem 4. (9 points):

Consider the following x86_64 assembly code:

```
# On entry: %rdi = M, %esi = n
# Note: nopl is simply a nop instruction for alignment purposes
0000000000400500 <func>:
  400500: 85 f6                 test   %esi,%esi
  400502: 7e 2a                 jle    40052e <func+0x2e>
  400504: 31 c0                 xor    %eax,%eax
  400506: 48 8b 0f              mov    (%rdi),%rcx
  400509: 31 d2                 xor    %edx,%edx
  40050b: 0f 1f 44 00 00        nopl   0x0(%rax,%rax,1)
  400510: 44 8b 01              mov    (%rcx),%r8d
  400513: 45 85 c0              test   %r8d,%r8d
  400516: 7f 18                 jg     400530 <func+0x30>
  400518: 83 c2 01              add    $0x1,%edx
  40051b: 48 83 c1 04           add    $0x4,%rcx
  40051f: 39 c2                 cmp    %eax,%edx
  400521: 7e ed                 jle    400510 <func+0x10>
  400523: 83 c0 01              add    $0x1,%eax
  400526: 48 83 c7 08           add    $0x8,%rdi
  40052a: 39 c6                 cmp    %eax,%esi
  40052c: 7f d8                 jg     400506 <func+0x6>
  40052e: 31 c0                 xor    %eax,%eax
  400530: f3 c3                 repz retq
```


Fill in the blanks of the corresponding C function:

```
int func(______ M, int n) {
    int i, j;
    for (i = 0; ______; i++) {
        for (j = 0; ______; j++) {
            if (________________)
                return ____;
        }
    }
    return ____;
}
```

Problem 5. (6 points):

Consider the C code below, where `H` and `J` are constants declared with `#define`.

```
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
```

What are the values of `H` and `J`?

```
H =

J =
```

Problem 6. (8 points):

Consider the following data structure declarations:

```
struct node {
  struct entry e;
  struct node *next;
}
```

```
struct entry {
  char a;
  char b;
  long c[2];
}
```

Below are given four C functions and five x86-64 code blocks.

```
char *one(struct node *ptr){
  return &(ptr->e.a)+1;
}
```



```
long two(struct node *ptr){
  return ((ptr->e.c)[0] = ptr->next);
}
```



```
char *three(struct node *ptr){
  return &(ptr->next->e.a);
}
```



```
char four(struct node *ptr){
  return ptr->e.b;
}
```

```
A   mov     0x18(%rdi), %rax
```

```
B   lea     0x18(%rdi), %rax
```

```
C   lea     0x1(%rdi), %rax
```

```
D   mov     0x18(%rdi), %rax
    mov     %rax, 0x8(%rdi)
```

```
E   movsbl  0x1(%rdi), %rax
```

In the following table, next to the name of each C function, write the name of the x86-64 block that implements it.

Function Name	Code Block
one	
two	
three	
four	

Problem 7. (11 points):

The next problem concerns code generated by GCC for a function involving a switch statement. The code uses a jump to index into the jump table:

```
  400519: jmpq   *0x400640(,%rdi,8)
```

Using GDB, we extract the 8-entry jump table as:

```
0x400640:    0x0000000000400530
0x400648:    0x0000000000400529
0x400650:    0x0000000000400520
0x400658:    0x0000000000400529
0x400660:    0x0000000000400535
0x400668:    0x000000000040052a
0x400670:    0x0000000000400529
0x400678:    0x0000000000400530
```


The following block of disassembled code implements the branches of the switch statement:


```
# on entry: %rdi = a, %rsi = b, %rdx = c
  400510: mov    $0x5,%rax
  400513: cmp    $0x7,%rdi
  400517: ja     400529
  400519: jmpq   *0x400640(,%rdi,8)
  400520: mov    %rdx,%rax
  400523: add    %rsi,%rax
  400526: salq   $0x2,%rax
  400529: retq
  40052a: mov    %rsi,%rdx
  40052d: xor    $0xf,%rdx
  400530: lea    0x70(%rdx),%rax
  400534: retq
  400535: mov    $0xc,%rax
  400538: retq
```

Fill in the blank portions of C code below to reproduce the function corresponding to this object code. You can assume that the first entry in the jump table is for the case when `a` equals 0.

```
long test(long a, long b, long c)
{
  long answer = ______;
  switch(a)
  {
    case ___:
      c = ______;
      /* Fall through */
    case ___:
    case ___:
      answer = ______;
      break;
    case ___:
      answer = ______;
      break;
    case ___:
      answer = ______;
      break;
    default:
      answer = ______;
  }

  return answer;
}
```

Problem 8. (8 points):

```
struct {
    char *a;
    short b;
    double c;
    char d;
    float e;
    char f;
    long g;
    void *h;
} foo;
```

A. Show how the struct above would appear on a 32-bit Windows machine (primitives of size k are k-byte aligned). Label the bytes that belong to the various fields with their names and clearly mark the end of the struct. Use hatch marks to indicate bytes that are allocated in the struct but are not used.


```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```



```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```



```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

B. Rearrange the above fields in `foo` to conserve the most space in the memory below. Label the bytes that belong to the various fields with their names and clearly mark the end of the struct. Use hatch marks to indicate bytes that are allocated in the struct but are not used.

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

C. How many bytes of the struct are wasted in part A?

D. How many bytes of the struct are wasted in part B?