Andrew login ID: _____
Full Name: _____
Section: _____

15-213/18-243, Summer 2011
Exam 1
Tuesday, June 28, 2011

Instructions:
• Make sure that your exam is not missing any sheets, then write your Andrew login ID, full name, and section on the front.
• This exam is closed book, closed notes. You may not use any electronic devices.
• Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
• The exam has a maximum score of 100 points.
• The problems are of varying difficulty. The point value of each problem is indicated. Good luck!

1 (12):
2 (12):
3 (9):
4 (20):
5 (15):
6 (20):
7 (12):

TOTAL (100):

Question 1. (12 points)
Bits, bytes, and floats.

For the conversions below, please make sure that you show all your work and your intermediate steps. Answers that only contain end results will not be graded even if they are correct.

A-) Convert 11.6875 to IEEE 32-bit floating point format.

11.6875 = 11 11/16 = 11 + ½ + 3/16 = 1011.1011

1011.1011 = 1.0111011 x 2^3

Bias = 127
3+bias = exp ➔ Exp = 130 = 10000010

0100 0001 0011 1011 0000 0000 0000 0000 (413B0000)

B-) Perform the following addition in the form of 8-bit, twos complement binary addition. Note if there is an overflow or not.

-72 + 61 = 11110101, No overflow
-72 = 10111000
61 = 00111101

C-) An 8-bit location in memory contains value c9 (in hex). This value would have different interpretations when it represents an 8-bit floating point number, or an 8-bit signed integer, or an 8-bit unsigned integer. Find its numerical interpretations in decimal. Again, show your work clearly.

8-bit floating point number =
S = 1
Exp =1001 = 9
Bias = 2^(4-1)-1 = 7
Frac = 001
1.01 * 2^(9-7) = 100.1 = 4.5; Sign was 1 ➔ Answer = -4.5

Signed Integer = -55

Unsigned Integer = 201

Question 2. (12 points)
Structs.

Consider the following struct:

```
typedef struct
{
        char a[5];
        short b[3];
        double c;
        long double d;
        int* e;
        int f;
        float* g;

} MYSTR;
```

Part 1.
Show how the struct above would appear on a 64-bit ("x86 64") Linux machine. Label the bytes that belong to the various fields with their names and clearly mark the end of the struct. Use x's to indicate bytes that are wasted in the struct.

```
aaaaaxbb    bbbbxxxx
cccccccc    xxxxxxxx
dddddddd    dddddddd
eeeeeeee    ffffxxxx
gggggggg
```

Part 2.
Rearrange the above fields in the above struct such that it would consume the most space in memory.

The struct is maximal in its current form however; other arrangements which result in maximum number of bytes are also accepted.

Part 3.
Rearrange the above fields in the above struct such that it would consume the least space in memory.

```
dddddddd    dddddddd
cccccccc    eeeeeeee
gggggggg    ffffbbbb
bb aaaaax
```

Question 3. (9 points)
Assembly to C.

Consider the 64-bit assembly code for a simple sort function. We provide parts of the corresponding C code on the next page. Please fill in the missing parts of the C code.

```
mysterysort:
        mov $1, %r8
        jmp L1
L2:
        movl (%rdi, %r8, 4), %r11d
        mov $0, %r9
        jmp L3
L4:
        movl (%rdi, %r9, 4), %eax
        cmp %eax, %r11d
        jge L7
        mov %r8, %r10
        jmp L6
L5:
        mov 0xfffffffffffffffc(%rdi, %r10, 4), %eax
        mov %eax, (%rdi, %r10, 4)
        sub $1, %r10d
L6:
        cmp %r9, %r10
        jg L5
        movl %r11d, (%rdi, %r9, 4)
        jmp L1
L7:
        add $1, %r9
L3:
        cmp %r8, %r9
        jl L4
        add $1, %r8
L1:
        cmp %rsi, %r8
        jl L2
        retq
```

```
void mysterysort(int *arr, int len)
{
   int i, j, k, temp;
   for(i = _1_; i < len; i++) {
      temp = arr[i];
      for(j = 0; j < i; j++) {
         if(temp < arr[j]) {
            for(k = i; k > j; k--) {
               arr[k] = arr[k-1];
            }
            arr[j] = temp;
            break;
         }
      }
   }
}
```

Question 4. (20 points)
Stacks.

Consider the C code for calculating Fibonacci numbers and the corresponding 32-bit assembly code. On the chart on the next page, please document the entire state of the stack as detailed as possible just before a call to fib(4) would return, but before it has popped its stack frame, noting that higher addresses are closer to the top of the page. Your trace should begin with the arguments from the stack frame of the caller of fib(4). Please also document where the relevant registers would point on your diagram. Note that you may not find it necessary to use all of the blanks. If you find a need, you may refer to the addresses on which the marked lines of code would be with the letter with which they are marked.

```
int fib(int n)
{
   /* computes the nth fibonacci number */
   if(n < 2) {
      return 1;
   }
   return fib(n-1) + fib(n-2);
}
```

```
fib:
        push    %ebp
        mov     %esp,%ebp
        sub     $0xc,%esp
        mov     %ebx,0xfffffff8(%ebp)
        mov     %esi,0xfffffffc(%ebp)
        mov     0x8(%ebp),%esi
        mov     $0x1,%eax
        cmp     $0x1,%esi
        jle     cleanup
        lea     0xffffffff(%esi),%eax
        mov     %eax,(%esp)
        call    fib                     # Address "A"
        mov     %eax,%ebx               # Address "B"
        lea     0xfffffffe(%esi),%eax
        mov     %eax,(%esp)
        call    fib                     # Address "C"
        add     %ebx,%eax               # Address "D"
cleanup:
        mov     0xfffffff8(%ebp),%ebx
        mov     0xfffffffc(%ebp),%esi
        mov     %ebp,%esp
        pop     %ebp
        ret
```

Higher addresses

4
Return address to caller
Old %ebp ← %ebp, and X
Old %esi
Old %ebx
2 ← %esp
D
X ← Y
4 (would accept esi)
3 (would accept ebx)
0
D
Y ← Z
2 (same)
1 (same)
0
D
Z
2 (same)
1 (same)

Lower Addresses

Question 5 (15 points)

Jump Table.

Consider the assembly dump for a switch and jump table given below. The switch expression and all case and break statements have been removed from the C code below it. Using what you know, fill in the switch expression, as well as case statements and break statements on the lines below, noting that you may not use every line.

```
L1:
        .quad L6
        .quad L4
        .quad L5
        .quad L3
        .quad L4
        .quad L4
        .quad L2
        .quad L6

scramble:
        cmpq $7, %rdi
        ja L4
        jmp *L1(,%rdi, 8)
L2:
        movl (%rdx), %r8d
        addl %r8d, (%rsi)
        jmp L7
L3:
        movl (%rsi), %r8d
        mov $5, %r9
        movl %r8d, (%rdx, %r9, 4)
        jmp L7
L4:
        movl $15213, (%rsi)
        jmp L7
L5:
        movl (%rdx), %r8d
        movl %r8d, (%rsi)
L6:
        movl (%rsi), %r8d
        lea (%r8, %r8, 2), %r8
        movl %r8d, (%rsi)
        jmp L7
L7:
        ret
```

```
void scramble(unsigned a, int * b, int * c)
{
   switch(a) {
_________case 3:_____________________________________
                c[5] = *b;
_____________break;_____________________________
_____________________________________________
_________case 6:_____________________________________
                *c += *b;
_____________break;_____________________________
_____________________________________________
_________case 2:_____________________________________
                *b = *c;
_____________________________________________
_________case 0:_____________________________________
_________case 7:_____________________________________
                *b *= 3;
_____________break;_____________________________
_____________________________________________
_________default:____________________________________
                *b = 15213;
_____________break; (optional)_____________________________
   }
}
```

Question 6. (20 points)
Caches.
Part 1.
Assume that we have an initially empty 16 byte cache and there is a sequence of accesses to this cache. Write down the miss/hit sequence for a 4-byte blocks, direct-mapped cache. Describe your reasoning/work for credit.

Address	Hit/Miss
0x00	Miss
0x0c	Miss
0x01	Hit
0x11	Miss
0x0f	Hit
0x03	Miss

Part 2.
Assume that we have an initially empty 16 byte cache and there is a sequence of accesses to this cache. Write down the miss/hit sequence for a 4-byte blocks, 2-way set-associative cache. Describe your reasoning/work for credit.

Address	Hit/Miss
0x00	Miss
0x0c	Miss
0x01	Hit
0x11	Miss
0x0f	Hit
0x03	Hit

Part 3.
For a cache with 128 byte cache lines, give the address of the first word in the line containing the following addresses:

0x3892ae4f = 0x3892ae00

0x4637e20c = 0x4637e200

Part 4.
A 16KB cache has a line length of 64 bytes. How many sets does the cache have if it is 2-way associative, or if it is 8-way associative?

#(Sets) = #(Lines) / Associativity
16KB = 2^14
#(Lines) = 2^8

2-way-associative = (2^8) / 2 = 2^7 = 128

8-way-associative = (2^8) / (2^3) = 2^5 = 32

Question 7. (12 points)
Cache Performance.

Part 1.
Given the data below, what is the impact of cache associativity on cache performance? Compare the performance of the direct mapped cache with its 2-way associative version in terms of the average miss penalty.

- Hit time for direct mapped cache = 7.9 clock cycles.
- Hit time for 2-way associative version = 8.2 clock cycles.
- Miss rate for direct mapped cache = 17%
- Miss rate for 2-way associative version = 12%
- Miss penalty for both versions = 180 clock cycles.

Miss_Rate * Miss_Penalty
17/100 *180 = 30.6
12/100 *180 = 21.6

I have also given points to:
Hit_Time + Miss_Rate * Miss_Penalty

Direct: 7.9 + 17/100 *180 = 38.5
2-way: 8.2 + 12/100 *180 = 29.8

Associative cache has better performance.

Part 2.
Assume that in 500 memory references there are 35 misses in the first-level L1 cache, 15 misses in the second-level L2 cache. What are the miss rates of L1 and L2 caches? Miss penalty of L2 cache is 150 clock cycles while its hit time is 12 clock cycles. The hit time of L1 cache is 2 clock cycles. What is the average memory access time?

Miss Rate for L1 = 35 misses / 500 references = 70/1000 = 7%
Local Miss Rate for L2 = 15/35 = 3/7
Global Miss Rate for L2 = 15/500 = 3/100 = 3%

Avg Mem Access =
Hit_Time_L1 + Miss_Rate_L1 x (Hit_Time_L2 + Miss_Rate_L2 x Miss_Penalty_L2)

2 + 7/100 (12 + 15/35 * 150) = 7.34