Andrew login ID: _____

Full Name: _____

15-213, Fall 2007

Final Exam

December 13, 2007, 1:00pm-4:00pm

- Make sure that your exam is not missing any sheets. Write your full name and Andrew login ID on the front.
- Write your answers in the space below each problem. If you make a mess, clearly indicate your final answer.
- The point value of each problem and the total possible is indicated below.
- The problems are of varying difficulty. Pile up the easy points quickly and then go back to the harder problems.
- This exam is OPEN BOOK. You may use any books or notes you like. Calculators are allowed, but no other electronic devices (including cell phones). Electronic communication is strictly forbidden. Good luck!

Do not write below this line

Problem	Possible Points	Your Score
1	16	
2	12	
3	9	
4	9	
5	12	
6	12	
7	20	
8	12	
9	12	
10	13	
Total	127	

Problem 1. (16 points):

Consider the following 8-bit floating point representations based on the IEEE-754 format.

Format A:

- There is one sign bit.
- There are $k = 3$ exponent bits. The exponent bias is 3.
- There are $n = 4$ fraction bits.

Format B:

- There is one sign bit.
- There are $k = 4$ exponent bits. The exponent bias is 7.
- There are $n = 3$ fraction bits.

The rules are like those in the IEEE standard (normalized, denormalized, representation, infinity, and NAN).

Part I

A. Let X_A be the largest, finite, positive value that can be represented in Format A, and let X_B be the largest, finite, positive value that can be represented in Format B.

(a) What is the encoding of X_A in Format A? Express your answer as a sequence of 8 bits.

(b) Circle the correct relationship between X_A and X_B.

$$X_A < X_B \qquad X_A == X_B \qquad X_A > X_B$$

B. Let Y_A be the largest, finite, *negative* value that can be represented in Format A, and let Y_B be the largest, finite, *negative* value that can be represented in Format B.

(a) What is the encoding of Y_B in Format B? Express your answer as a sequence of 8 bits.

(b) Circle the correct relationship between Y_A and Y_B. **Remember that Y_A and Y_B are both negative.**

$$Y_A < Y_B \qquad Y_A == Y_B \qquad Y_A > Y_B$$

Part II

Please complete the table below as follows. For a given format in a given row, the bit representation and value fields should correspond exactly to each other. For Format B in the last four rows of the table (where neither the bit representation nor the value are specified for that format), you should indicate:

- the *bit representation* that is closest (including any rounding if necessary) to the value for Format A in that same row;
- the *value* that corresponds exactly to the bit representation that you enter for Format B (which may or may not be equal to the value for Format A in the same row).

If rounding is necessary, you should use the round to even scheme that is the default in the IEEE format. For the value fields, you can give the values either as fractions (e.g., $\frac{17}{64}$, $\frac{7}{2}$), mixed numbers (e.g., $5\frac{1}{2}$) or as an integer times a power of 2 (e.g., 17×2^{-6} or 7×2^{-1}).

Format A		Format B	
Bit Representation	Value	Bit Representation	Value
`1 011 1010`	$-13/8$	`1 0111 101`	$-13/8$
`0 101 0011`			
`1 000 1100`			
	12		
	$3\frac{1}{4}$		

Problem 2. (12 points):

Dr. Evil has returned! He has placed a binary bomb in this exam! Once again, Dr. Evil has made the disastrous mistake of leaving behind some of his source code. Can you save all of mankind (or at least your grade on this question), and tell us what this bomb does?

The C source code Dr. Evil forgot to erase:

```
/* bomb.c:  Use new computer technology to blow up exams! -- Dr. Evil */
#include <stdio.h>
#include <stdlib.h>

extern long secret_unsolvable_puzzle_fn(long input);

void explode_bomb() {
    printf("You fail!  Mwhahahahaha!!!\n");
    exit(8);
}

int main(int argc,char *argv[]) {
    if(argc != 2) {
        printf("Usage: %s <magic password>\n", argv[0]);
        explode_bomb();
    }

    if(secret_unsolvable_puzzle_fn(atol(argv[1])) == 0)
        explode_bomb();
    printf("Curses, foiled again!  You get paid one MEEELLION DOLLARS!\n");
    return 0;
}
```

The IA32 disassembly for the stuff Dr. Evil did erase:

```
        secret_unsolvable_puzzle_fn:
 80485c0:       55              pushl  %ebp
 80485c1:       89 e5           movl   %esp,%ebp
 80485c3:       8b 45 08        movl   0x8(%ebp),%eax
 80485c6:       85 c0           testl  %eax,%eax
 80485c8:       74 1d           je     80485e7
 80485ca:       8d 14 00        leal   (%eax,%eax,1),%edx
 80485cd:       81 fa 42 53 00  cmpl   $21314,%edx
 80485d2:       00
 80485d3:       77 0a           ja     80485df
 80485d5:       01 c2           addl   %eax,%edx
 80485d7:       81 fa 42 53 00  cmpl   $21314,%edx
 80485dc:       00
 80485dd:       76 f6           jbe    80485d5
 80485df:       81 fa 43 53 00  cmpl   $21315,%edx
 80485e4:       00
 80485e5:       74 02           je     80485e9
 80485e7:       31 c0           xorl   %eax,%eax
 80485e9:       89 ec           movl   %ebp,%esp
 80485eb:       5d              popl   %ebp
 80485ec:       c3              ret
```

A. Does the function `secret_unsolvable_puzzle_fn()` contain any of the following (circle either *yes* or *no*):

loops:	*yes*	*no*
if statements:	*yes*	*no*
function calls:	*yes*	*no*
recursion:	*yes*	*no*

B. For each of the following input values, circle whether it *defuses* or *explodes* the bomb:

input = 0:	*defuses*	*explodes*
input = 1:	*defuses*	*explodes*
input = 7105:	*defuses*	*explodes*
input = 10657:	*defuses*	*explodes*

Problem 3. (9 points):

This problem will test your understanding of the memory layout of C structures and unions in IA-32/Windows assembly code. (Recall that in Windows, 8 byte primitive data types must be aligned upon 8-byte boundaries.) Consider the data structure declarations below. (Note that this is a single declaration which includes several data structures; they are shown horizontally to fit on the page.)

```
struct s1 {
  struct s2 a;
  struct s2 *b;
  struct s1 *c;
  double d;
  int e[5];
};
```

```
struct s2 {
  char i[7];
  union u1 *j;
  int k;
};
```

```
union u1 {
  int f;
  struct s1 g;
  struct s2 *h;
};
```

For each of these four C procedures, fill in the missing offsets in the corresponding IA-32 assembly code immediately below it. (If you give the wrong answer below but write the correct sizes next to the structures above, you might get some partial credit.)

A.

```
int proc1(struct s1 *x) {
   return x->e[3];
}

proc1:
   pushl %ebp
   movl  %esp,%ebp
   movl  8(%ebp),%eax
   movl  ____(%eax),%eax
   movl  %ebp,%esp
   popl  %ebp
   ret
```

B.

```
int proc2(struct s2 *x) {
   return x->j->g.e[1];
}

proc2:
   pushl %ebp
   movl  %esp,%ebp
   movl  8(%ebp),%eax
   movl  ____(%eax),%eax
   movl  ____(%eax),%eax
   movl  %ebp,%esp
   popl  %ebp
   ret
```

C.

```
int proc3(union u1 *x) {
   return x->g.c->b->k;
}

proc3:
   pushl %ebp
   movl  %esp,%ebp
   movl  8(%ebp),%eax
   movl  ____(%eax),%eax
   movl  ____(%eax),%eax
   movl  ____(%eax),%eax
   movl  %ebp,%esp
   popl  %ebp
   ret
```

D.

```
int proc4(union u1 *x) {
   return x->h->j->f;
}

proc4:
   pushl %ebp
   movl  %esp,%ebp
   movl  8(%ebp),%eax
   movl  ____(%eax),%eax
   movl  ____(%eax),%eax
   movl  ____(%eax),%eax
   movl  %ebp,%esp
   popl  %ebp
   ret
```

Problem 4. (9 points):

Consider the C code below:

```
int fdplay() {
    int pid;
    int fd1, fd2;

    fd1 = open("/file1", O_RDWR);
    dup2(fd1, 1);
    printf("A");
    if ((pid = fork()) == 0) {
        printf("B");
        fd2 = open("/file1", O_RDWR);
        dup2(fd2, 1);
        printf("C");
        /* POINT X */
    } else {
        waitpid(pid, NULL, 0);
        printf("D");
        close(fd1);
        printf("E");
    }
    exit(2);
}
```

A. How many processes share the open file structure referred to by `fd1` at "`POINT X`" in the code?

B. How many file descriptors (total among all processes) share the open file structure referred to by `fd1` at "`POINT X`" in the code?

C. Assuming that `/file1` was empty before running this code, what are its contents after the execution is complete?

Problem 5. (12 points):

3M decides to make Post-Its by printing yellow squares on white pieces of paper. As part of the printing process, they need to set the CMYK (cyan, magenta, yellow, black) value for every point in the square. 3M hires you to determine the efficiency of the following algorithms on a machine with a 2048-byte direct-mapped data cache with 32 byte blocks.

You are given the following definitions:

```
struct point_color {
    int c;
    int m;
    int y;
    int k;
};

struct point_color square[16][16];
register int i, j;
```

Assume:

- `sizeof(int) = 4`
- `square` begins at memory address 0
- The cache is initially empty.
- The only memory accesses are to the entries of the array `square`. Variables `i` and `j` are stored in registers.

A. What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[i][j].c = 0;
        square[i][j].m = 0;
        square[i][j].y = 1;
        square[i][j].k = 0;
    }
}
```

Miss rate for writes to `square`: _____ %

B. What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[j][i].c = 0;
        square[j][i].m = 0;
        square[j][i].y = 1;
        square[j][i].k = 0;
    }
}
```

Miss rate for writes to `square`: _____ %

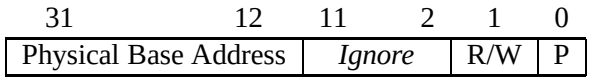
C. What percentage of the writes in the following code will miss in the cache?

```
for (i=0; i<16; i++){
    for (j=0; j<16; j++) {
        square[i][j].y = 1;
    }
}
for (i=0; i<16; i++) {
    for (j=0; j<16; j++) {
        square[i][j].c = 0;
        square[i][j].m = 0;
        square[i][j].k = 0;
    }
}
```

Miss rate for writes to `square`: _____ %

Problem 6. (12 points):

This question focuses on the two-level page table structure that IA-32/Linux machines use to translate virtual to physical addresses. The layout of a *Page Directory Entry* (PDE) and a *Page Table Entry* (PTE) is shown below on the left, and the contents of physical memory for this problem is shown below on the right:



Physical Base Address (bits 31-12): the 20 most significant bits of either the physical PTE address (if this is a PDE), or the physical page address (if this is a PTE). (Note that this forces both page tables and pages to be 4KB aligned.) Note that if this is a PDE, this address represents where the PTE starts; if it is a PTE, then it represents where the physical page being accessed starts.

***Ignore* (bits 11-2):** not pertinent to this problem.

R/W (bit 1): **0** indicates that we have read-only permission for either the PTE or the page, and **1** indicates that we have permission both to read and write.

P (bit 0): indicates whether the PTE (if this is a PDE) or the page (if this is a PTE) is in physical memory; **1** means that it is in memory; **0** means that it is not.

Physical Memory Contents

Physical Address	Data Value
0x000C8000	0x20000025
0x000C8004	0x08000025
0x000C8008	0x00100025
0x000C800c	0x20000001
...	
0x00100000	0xAD34A645
0x00100004	0x12480007
0x00100008	0x001C0A05
0x0010000C	0x8BEEF407
0x00100010	0xBF072627
...	
0x08000000	0x0002C003
0x08000004	0x01000C25
0x08000008	0x0FA00027
0x0800000C	0x824AF667
...	
0x20000000	0x024C8C05
0x20000004	0x0A4F3407
0x20000008	0x023FD225
0x2000000C	0x198AAE27

Assume the following:

- The page size is 4KB.
- All memory accesses are to 4-byte words (using byte addresses, as always).
- The first level of the page table begins at physical address `0x000C8000` (i.e. this is the value of the "`PDBR`" register in the Pentium III processor).

For the memory accesses on the next page, your mission is to answer the following two questions:

1. *Does the access complete normally, or does it result in a fault?* Note that there are multiple reasons why an access may result in a fault. If you believe that a fault occurs, explain why.
2. *What is the final 4-byte physical address that is accessed in the course of performing this access?* If the access completes normally, then this will simply be the physical address upon which the memory operation is performed. If there is a fault, however, then this address will be a 4-byte word within the page table (note that it may be either a PDE or a PTE).

To avoid excessive page turning, the addresses on the next page are:
`0x00803CDC` (write), `0x00000320` (read), `0x00802127` (write), and `0x00401478` (read).

A. **Write to virtual address** `0x00803CDC`**.**

Final physical address accessed:

Does a fault occur? (yes or no):

If yes, explain why:

B. **Read from virtual address** `0x00000320`**.**

Final physical address accessed:

Does a fault occur? (yes or no):

If yes, explain why:

C. **Write to virtual address** `0x00802127`**.**

Final physical address accessed:

Does a fault occur? (yes or no):

If yes, explain why:

D. **Read from virtual address** `0x00401478`**.**

Final physical address accessed:

Does a fault occur? (yes or no):

If yes, explain why:

Problem 7. (20 points):

In each of the following questions, *one or more* of the possible answers is correct. **Clearly indicate all of the correct answers by writing their letter(s) in the blank at the end of each question.**

A. Which of the following x86 instructions can be used to add two registers and store the result without overwriting either of the original values?

(a) `mov`
(b) `add`
(c) `lea`
(d) None of above

Correct answer(s): _____

B. The register `rax` is currently storing a NULL pointer. Which of the following x86 instructions will cause a segmentation fault because of an invalid memory access?

(a) `mov (%rax), %rcx`
(b) `lea (%rax), %rcx`
(c) None of the above

Correct answer(s): _____

C. In buflab, a buffer was allocated on the stack. When the user ran the program and typed something in, it was written into the buffer. If the user entered more characters than the buffer could fit, they could overwrite additional values on the stack. Which of the following regions of the stack could they directly overwrite in this manner?

(a) The part of the stack with *higher* (i.e. larger) addresses than the buffer
(b) The part of the stack with *lower* (i.e. smaller) addresses than the buffer

Correct answer(s): _____

D. A function declares a local variable named `my_int` of type `int`. Which of the following (if any) is/are dangerous in C?

(a) Returning `&my_int`
(b) Setting the value of a global variable to `&my_int`
(c) Printing the address `&my_int` to the screen
(d) None of the above

Correct answer(s): _____

E. A programmer wishes to compare the contents of a string called `my_str` to the string "`GET`". He or she writes the following C code:

```
if (my_str == "GET") ...
```

Which of the following apply?

(a) `my_str` is a pointer to the first character of a string in memory
(b) `my_str` is the ASCII value of the first character of a string in memory
(c) `my_str` is a register containing all of the characters in the string
(d) "`GET`" will compile to a pointer to a string in memory
(e) "`GET`" will compile to the ASCII value for the letter "G"
(f) "`GET`" will compile to a register containing the string "`GET`" represented as an integer
(g) The comparison will always work as expected
(h) The comparison will not necessarily work as expected
(i) The comparison itself will cause the program to crash

Correct answer(s): _____

F. The function `foo()` is declared in a C program as follows:

```
void foo(int int_param, char *str_param);
```

A programmer calls `foo()` from within the function `bar()` as follows:

```
foo(my_int, my_string);
```

Which of the following is/are true:

(a) If `foo()` changes the value of `int_param`, the change will propagate back to the calling function `bar()`, in other words, the value of `my_int` will also change.
(b) If `foo()` changes the second character of `str_param`, the change will propagate back to the calling function `bar()`, in other words, the second character of `my_string` will also change.
(c) If `foo()` changes the address of `str_param` to point to a different string, the change will propagate back to the calling function `bar()`, in other words, `my_string` will now point to a different string.
(d) None of the above.

Correct answer(s): _____

G. A programmer has declared an array in a C program as follows:

```
int my_array[100];
```

Which of the following give(s) the address of the eighth element in the array (bearing in mind that the first element in the array is at index zero):

(a) `my_array[7]`
(b) `&my_array[7]`
(c) `my_array + 7`
(d) `my_array + 28`
(e) None of the above

Correct answer(s): _____

H. A programmer has stored an 8-bit value in memory. The pointer:

```
char *ptr;
```

points to the location where it is stored. He or she now wants to retrieve the value and store it into the variable:

```
int value;
```

Which of the following (if any) will achieve this properly?

(a) `value = ptr;`
(b) `value = *ptr;`
(c) `value = (int)ptr;`
(d) `value = (int *)ptr;`
(e) `value = *(int *)ptr;`
(f) None of the above

Correct answer(s): _____

I. In malloclab, we provided code for an implicit list allocator. Many students improved this code by creating a linked list of free blocks. Why did this increase the performance of the allocator?

(a) Traversing a linked list is significantly faster than moving from block to block in the implicit list.
(b) The implicit list had to include every block in memory, but the linked list could just include the free blocks.
(c) The compiler knows how to optimize the code for a linked list by unrolling loops, but wasn't able to do this for the implicit list.
(d) Having a linked list made coalescing significantly faster.
(e) None of the above.

Correct answer(s): _____

J. A multithreaded program has two global data structures that will be shared among the threads. The data structures are not necessarily accessed at the same time. Which of the following is/are true (if any)?

(a) If the program has only one semaphore, and threads call P on that single semaphore before using either of the data structures, the code will not work correctly.

(b) Having one semaphore will work, but having two, one per shared data structure, may allow for increased performance.

(c) If the machine has only one processor, only one of the threads can run at a time, so semaphores are not necessary in that case.

(d) None of the above.

Correct answer(s): _____

Problem 8. (12 points):

Consider the C code below:

```
void handler (int sig) {
    printf("s");
    exit(7);
}

int forker(int x) {
    int pid, status;

    signal(SIGINT, handler);
    printf("A");
    if (x > 0) {
        pid = fork();
        printf("B");
        if (pid == 0) {
            printf("C");
        } else {
            kill(pid, SIGINT);
            waitpid(pid, &status, 0);
            printf("%d", WEXITSTATUS(status));
        }
    }
    printf("E");
    exit(4);
}
```

Consider each of the following outputs and circle the ones that could be produced by the code above (after all processes are terminated).

ABBCE4E

ABs7E

ABCEsB4E

AB4EBCE

ABCBs7E

ABCEB4sE

Problem 9. (12 points):

Consider the following three threads and four semaphores:

```
/* Initialize x */
x = 1;

/* Initialize semaphores */

s1 =

s2 =

s3 =

s4 =
```

```
void thread1()
{
  while (x != 360) {

    x = x * 2;

  }
  exit(0);
}
```

```
void thread2()
{
  while (x != 360) {

    x = x * 3;

  }
  exit(0);
}
```

```
void thread3()
{
  while (x != 360) {

    x = x * 5;

  }
  exit(0);
}
```

Provide initial values for the four semaphores and add P(), V() semaphore operations (using the four semaphores) in the code for thread 1, 2 and 3 such that the process is guaranteed to terminate.

Problem 10. (13 points):

A. Assume that we want to transmit over the network the contents of a structure of the following type. Circle the structure elements that must be put in network byte order to guarantee that the recipient can interpret what it receives correctly.

```
struct data {

   int foo;

   char name[16];

   short bar;
};
```

B. Consider the following segment of network code:

```
fd = socket (AF_INET, SOCK_STREAM, 0)
...
connect (fd, &serveraddr, sizeof(serveraddr));
write (fd, data, N);
read (fd, buf, N);
...
```

(a) Assume that the `socket()`, `connect()`, and `write()` calls return success. Is the `read()` call guaranteed to return quickly?

(b) When the `read()` call returns, how many bytes of `buf` will have been modified? Indicate either a value or a precise range of values (e.g., "between X and Y"). (Note that an answer of "between zero and infinity" will not receive credit: you need to be precise.)

(c) Based on the code above, will the first byte of `buf` after the `read()` match the first byte of `data`? (Your answer should be one of either "yes", "no", or "maybe".)

C. How many unique socket connections could a web server that listens on a single port (e.g., port 80) and has a single IP address have with clients at once? (Ignore possible limitations imposed by the operating system.)

D. How many concurrent socket connections could the same server have if it listens on all ports?