15-213 Introduction to Computer Systems

Exam 1

February 22, 2005

Name: ______________________

Andrew User ID: ______________

Recitation Section: ______________

- This is an open-book exam. Notes are permitted, but not computers.
- Write your answer legibly in the space provided.
- You have 80 minutes for this exam.

Problem	Max	Score
1	**15**	
2	**15**	
3	**15**	
4	**15**	
5	**7**	
6	**8**	
Total	**75**	

1. Floating Point (15 points)

Consider the following function in assembly language.

```
problem1:
        pushl   %ebp
        movl    %esp, %ebp
        flds    16(%ebp)
        fadds   12(%ebp)
        fmuls   8(%ebp)
        leave
        ret
```

Recall that `fld` pushes the argument onto the floating point register stack, `fadd` pushes and adds, and `fmul` pushes and multiplies. The suffix `s` determines size of the operands to be 32 bits.

1. (5 points) Write out a corresponding function definition in C.

2. (1 points) Assume the standard representation of single-precision floating point numbers with 1 sign bit, $k = 8$ bits for the exponent, and $n = 23$ bits for the fractional value. What is the bias?

3. (5 points) Give the hexadecimal representation of the number $\frac{1}{4}$.

4. (4 points) Give the representation of `0xBE000000` as a fraction.

2. Pointers and Functions (15 points)

The following somewhat misguided code tries to optimize the exponential function on positive integers by creating an array of function pointers called `table` and using them if the exponent is less than 4.

```
int exp0 (int x) { return 1; }
int exp1 (int x) { return x; }
int exp2 (int x) { return x * x; }
int exp3 (int x) { return x * x * x; }

_______________________________ = {
  &exp0, &exp1, &exp2, &exp3
};

int exp (int x, int n) {
  int y = 1;
  if (n < 4)
    return ____________________ ;
  while (n > 0) {
    y *= x;
    n--;
  }
  return y;
}
```

1. (5 points) Fill in the missing declaration of `table` and complete the `return` statement.

2. (3 points) Note the assignment of program variables to registers in the following assembly code produced by gcc for `exp`. We have elided some alignment instructions and the specialized `exp`n functions.

```
table:
        .long   exp0
        .long   exp1
        .long   exp2
        .long   exp3

exp:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        movl    12(%ebp), %edx
        cmpl    $3, %edx
        movl    8(%ebp), %ecx
        movl    $1, %eax
        jle     _______
        testl   %edx, %edx
        jle     _______
.L11:
        decl    %edx
        imull   %ecx, %eax
        testl   %edx, %edx
        jg      .L11
.L6:
        leave
        ret
.L14:
        subl    $12, %esp
        pushl   %ecx
        call    _________________
        jmp     .L6
```

Variable	Register
x	
n	
y	

3. (3 points) Justify the use of particular registers chosen by gcc.

4. (4 points) Fill in the three missing lines of assembly code.

3. Structures and Alignment (15 points)

Consider the following C declaration:

```
typedef struct {
  unsigned short id;
  char* name;
  char andrew_id[9];
  char year;
  int raw_score;
  double percent;
} Student;
```

1. (5 points) On the template below, show the layout of the `Student` structure. Delineate and label the areas for each component of the structure, cross-hatching the parts that are allocated but not used. Clearly mark the end of the structure. You should assume Linux alignment rules. Do **not** fill in the data fields; you will need them to answer the next question.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

2. (3 points) Show the state of an instance of the `Student` structure after the following code is executed. Write in the assigned values by filling them into the assigned squares above and leave the remaining squares blank. Assume a Linux/x86 architecture and use hexadecimal format.

```
Student jenn;
jenn.id = 16;
jenn.year = -2;
jenn.raw_score = 513;
```

3. (5 points) Rewrite the declaration to use the minimal amount of space for the structure with the same components. You should make sure that the amount of space is minimal for both Linux and Windows alignment rules.

4. (2 points) How many bytes does your new structure require?

4. Optimization (15 points)

Consider the following declaration of a linked list data structure in C:

```
typedef struct LIST {
  struct LIST *next;
  int data;
} List;
```

We use a next pointer of NULL to mark the end of the list, and we assume we have a function `int length (List* p);` to calculate the length of a non-circular linked list. You may assume all linked lists in this problem are non-circular.

1. (3 points) The function `count_pos1` is supposed to count the number of positive elements in the list at `p` and store it at `k`, but it has a serious bug which may cause it not to traverse the whole list. Insert one line and change one line to fix this problem.

```
void count_pos1 (List *p, int *k) {
  int i;
  *k = 0;
  for (i = 0; i < length(p); i++) {
    if (p->data > 0)
      (*k)++;
    p = p->next;
  }
}
```

2. (5 points) Further improve the efficiency of the corrected function from part 1 by eliminating the iteration variable `i`, changing it to a iteration using only pointers instead. Fill in the template below.

```
void count_pos2 (List *p, int *k) {
  *k = 0;
  while (________________) {
    if (p->data > 0)
      (*k)++;
    ______________;
  }
}
```

3. (2 points) Your function from part 2 still has a bug in that it does not always "*count the number of positive elements in the list at* `p` *and stores it at* `k`". Explain the problem.

4. (5 points) Fix the problem you identified in part 3. Your function should also run faster by reducing memory accesses when compared to the function in part 2.

5. Out-of-Order Execution (7 points)

1. (5 points) The inner loop corresponding to our answer to part 4 of the previous problem has the following form:

```
.L48:
        movl    4(%eax), %ecx      # load 4(%eax.0) --> %ecx.1
        testl   %ecx, %ecx
        jle     .L47
        incl    %edx
.L47:
        movl    (%eax), %eax
        testl   %eax, %eax
        jne     .L48
```

   Annotate each line with the execution unit operations for one iteration, assuming the inner branch is **not** taken. To get you started, we have filled in the first operation.

2. (2 points) Assuming most numbers in a list are positive and branch predictions are correct, give a plausible lower bound on the CPE for the inner loop based on the execution unit operations. Optimistically assume memory accesses are cache hits.

6. Cache Memory (8 points)

Assume we have byte-addressable memory with addresses that are 12 bits wide. We have a 2-way set associative cache with with 4 byte block size and 4 sets.

1. (3 points) On the template below, show the portions of an address that make up the tag, the set index, and the block offset.

11	10	9	8	7	6	5	4	3	2	1	0

2. (5 points) Consider the following cache state, where all addresses, tags, and values are given in hexadecimal format.

Set Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	00	1	40	41	42	43
	83	1	FE	97	CC	D0
1	00	1	44	45	46	47
	83	0	–	–	–	–
2	00	1	48	49	4A	4B
	40	0	–	–	–	–
3	FF	1	9A	C0	03	FF
	00	0	–	–	–	–

For each of following memory accesses indicate if it will be a cache hit or miss, when they are **carried out in sequence** as listed. Also give the value of a read if it can be infered from the information in the cache.

Operation	Address	Hit?	Read Value (or Unknown)
Read	0x834		
Write	0x836		(not applicable)
Read	0xFFD		