15-213 Introduction to Computer Systems

Exam 1

February 27, 2007

Name: **Model Solution**

Andrew User ID: **fp**

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
Integers	1	**10**	
Floating Point	2	**15**	
Assembly Language	3	**15**	
Calling Conventions	4	**10**	
Structures and Alignment	5	**10**	
Out-of-Order Execution	6	**15**	
	Total	**75**	

1. Integers (10 points)

For each of the following propositions, write in **all** comparisons that make it true among the four possibilities:

```
<  >  ==  !=
```

If none are guaranteed to hold, please indicate that explicitly by marking it with an **X**. We have filled in the first two for you as examples. Assume the variables are declared with

```
int x,y;
```

and initialized to some unknown values. You may assume that `int`'s are 32 bits wide, `char`'s are 8 bits wide and that right shift is arithmetical on signed numbers and logical on unsigned numbers.

If `x > y` then `y`	{ `<` `!=` } `x`
If `x < 0` then `x+1`	{ **X** } `0`
If `x > 0` then `x+1`	{ `!=` } `0`
If `(x >> 31) < 0` then `x`	{ `<` `!=` } `0`
If `((x << 31) >> 31) < 0` then `x & 1`	{ `>` `!=` } `0`
If `x > y` then `(unsigned)x`	{ `!=` } `y`
If `((unsigned char)x >> 1) < 64` then `(char)x`	{ **X** } `0`

2. Floating Point (15 points)

1. (10 pts) Fill in the blank entries in the following table. We assume an IEEE representation of floating point numbers with 1 sign bit, $k = 3$ bits for the exponent and $n = 4$ bits for the fractional value. This means the bias is $2^{3-1} - 1 = 3$.

Value	Form $M \times 2^E$ for $1 \leq M < 2$	Hexadecimal representation	Decimal fraction
Smallest denorm.	1.0×2^{-6}	`0x01`	$\frac{1}{64}$
Smallest norm.	1.0×2^{-2}	**`0x10`**	$\frac{1}{4}$
One	1.0×2^{0}	**`0x30`**	1
Largest norm.	1.1111×2^{3}	**`0x6F`**	$\frac{31}{2}$
Infinity	XXXXX	**`0x70`**	XXXXX

2. (5 pts) With standard single precision floating point numbers with 1 sign bit, $k = 8$ bits for the exponent, and $n = 23$ bits for the significand, what is the smallest positive value for `n` declared with `int n;` such that

   `(int)(float)n != n ?` $2^{24} + 1$

3. Assembly Language (15 points)

Consider the following recursive C program that sorts a segment of an integer array into ascending order in place.

```
/* qsort(A, low, high) sorts subarray A[low]..A[high] */
void qsort (int* A, int low, int high) {
  int pivot, i, k;
  if (low >= high) return;
  pivot = A[high];
  i = low;
  k = high;
  while (i < k) {
    if (A[i] < pivot) {
      i++;
    } else {
      A[k] = A[i];
      k--;
      A[i] = A[k];
    }
  }
  A[k] = pivot;
  qsort(A, low, k-1);
  qsort(A, k+1, high);
}
```

The `while` loop is compiled to the following assembly language instructions.

```
.L13:
        incl    %edx
        jmp     .L8
.L7:
        movslq  %edx, %rcx
        movl    (%rbp,%rcx,4), %eax
        cmpl    %r8d, %eax
        jl      .L13
        movl    %eax, (%rbp,%rdi,4)
        decl    %ebx

        _____________________________ <first missing instruction>

        _____________________________ <second missing instruction>
        movl    %eax, (%rbp,%rcx,4)
.L8:
        cmpl    %ebx, %edx
        jl      .L7
```

1. (5 pts) Complete the following table associating C variable or expressions with registers.

C Expression	Register
`pivot`	`%r8d`
`A`	**`%rbp`**
`i`	**`%edx`**
`k`	**`%ebx`**
`(long)k`	**`%rdi`**
`(long)i`	**`%rcx`**

2. (5 pts) The register `%eax` holds temporary values during the loop computation. List all the source expressions whose values it holds.

`A[i]`,`A[k]`

3. (5 pts) Fill in the two missing instructions.

`movslq %ebx, %rdi` and `movl (%rbp,%rdi,4), %eax`

4. Calling Conventions (10 points)

After the code of the `while` loop shown before, we find the following instructions to implement the first recursive call.

```
movl    %r8d, (%rbp,%rdi,4)
leal    -1(%rbx), %edx
movq    %rbp, %rdi
call    qsort
```

1. (5 pts) By the x86-64 calling conventions, which of the mentioned registers are guaranteed to have the same value when `qsort` returns as right before the call? Write **yes** or **no** into the space.

Register	Same?
`%r8d`	**no**
`%rbp`	**yes**
`%rdi`	**no**
`%rbx`	**yes**
`%edx`	**no**

We are surprised that following the first recursive call, we find no further recursive call, but instead the instructions

```
leal    1(%rbx), %esi
jmp     .L14
```

where `.L14` is a label near the beginning of the `qsort` procedure. The compiler used an optimization to eliminate the second recursive call in favor of a jump instruction.

2. (5 pts) Complete the following rendering of the compiler's optimization in C in the form of a new `while` loop.

```
void qsort (int* A, int low, int high) {
 int pivot, i, k;
 /* eliminated here: if (low >= high) return; */
 while (____________) {
   pivot = A[high];
   i = low;
   k = high;
   while (i < k) { ... as before ... }
   A[k] = pivot;
   qsort(A, low, k-1);

   ____________________   /* was: qsort(A, k+1, high); */
 }
}
```

`low < high` and `low = k+1`

5. Structures and Alignment (10 points)

This question concerns alignment on an x86-64 architecture and pointer arithmetic. Consider the following C structure declaration.

```
struct box {
  int tag; /* 0 = int, 1 = long, 2 = float, 3 = double */
  union {
    int i;
    long l;
    float f;
    double d;
  } data;
};
```

1. (3 pts) What is the total size of a `box` structure on an x86-64, expressed in bytes?

16

2. (2 pts) An element of type `struct box` must be aligned at 0 modulo **8**.

3. (5 pts) Write a C function `int high4 (long x);` that extracts the high 4 bytes of a `long` as an `int`. For example, `high (0x1234567890abcdef)` should return `0x12345678`. Use a `box` structure and pointer arithmetic. Your code may ignore the `tag` field.

   ```
   int high4 (long x) {
     struct box b;
     b.data.l = x;
     return *(&b.data.i+1);
   }
   ```

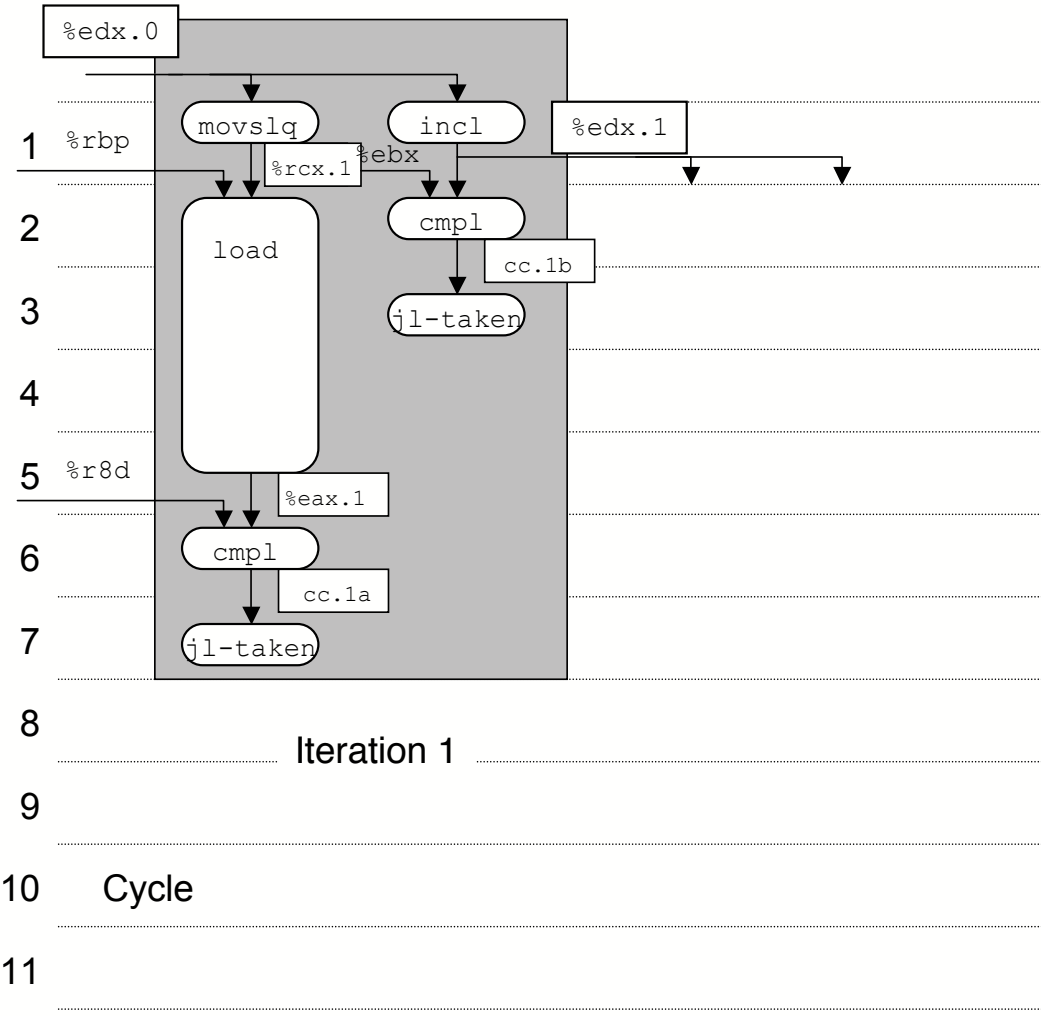
6. Out-of-Order Execution (15 points)

We now return to the earlier quicksort procedure. If the array is already sorted, the first conditional branch instruction in the code fragment below will always be taken.

1. (5 pts) Assume that the processor correctly predicts this, as well as the second branching instruction (for example, because they both go backwards). On the right-hand side, show the translation of the assembly language instructions into execution unit operations with register renaming. Do not rename registers that are invariant throughout multiple loop iterations. We have already filled in the translations of the conditional branches; the unconditional jump is handled by the instruction fetch unit. Consider `.L7` as your loop entry point.

```
.L13:                                |
    incl    %edx                     | incl %edx.0 -> %edx.1
    jmp     .L8                      | (handled in fetch unit)
.L7:                                 |
    movslq  %edx, %rcx               | movslq %edx.0 -> %rcx.1
    movl    (%rbp,%rcx,4), %eax      | load (%rbp,%rcx.1,4) -> %eax.1
    cmpl    %r8d, %eax               | cmpl %r8d, %eax.1 -> cc.1a
    jl      .L13                     | jl-taken cc.1a
    <omitted code>                   |
.L8:                                 |
    cmpl    %ebx, %edx               | cmpl %ebx, %edx.1 -> cc.1b
    jl      .L7                      | jl-taken cc.1b
```

2. (7 pts) Complete the labeling in the following timed data dependency diagram. We have already filled in the registers that are not renamed and one register move operation.



3. (3 pts) What is the theoretical CPE for this loop, assuming perfect branch prediction and no resource limitions?

1.0 CPE