CPSC 313, 05w Term 1— Midterm 1

Date: October 7, 2005; Instructor: Mike Feeley

This is a closed book exam; no notes; you may use calculators to perform simple arithmetic calculations. Answer in the space provided; use the backs of pages if needed. There are **7** questions on **4** pages, totaling **50** marks. You have **50 minutes** to complete the exam.

NAME: _____ **SCORE:** _____ / 50

STUDENT NUMBER: _____

1. **(10 marks)** Short answers.

1a. What is the advantage of using two different registers (i.e., `%ebp` and `%esp`) to store virtual addresses to the runtime stack?

1b. What does a `call` instruction do that a `jmp` instruction does not?

1c. We discussed two ways to implement a C-language `switch` statement in assembly language. What are they? Under what conditions would one be favoured over the other (both ways)?

1d. Why is it faster to compute the address of an element of an array of structs if the size of each struct is a power of two?

1e. Write the two assembly-language instructions that compute "`if (a<=b) goto X`" where `a` and `b` are signed integers stored in registers `%eax` and `%ebx` respectively and `X` is a label.

_____ / 10

2. **(4 marks)** Indicate whether each of the following values are determined statically or dynamically. For each, write the word **static** or **dynamic**.

2a. The virtual address of a global variable: _____

2b. The virtual address of a local variable: _____

2c. The offset from the register `%ebp` of a local variable: _____

2d. The virtual address to which control is transferred when a procedure is called: _____

2e. The virtual address to which control is transferred when a procedure returns: _____

3. **(4 marks)** Write the assembly-language instructions that compute "`%eax` = `%eax` × 15 + 15" **most efficiently** (i.e., your could should execute faster than any alternative).

4. **(8 marks)** Consider the following C-language procedure. Answer each question with the appropriate IA32 (gas) assembly language. Treat each question in isolation and assume that none of the registers other than `%esp` and `%ebp` hold useful values. **Comment your code.**

```
int A[100];     // a global variable

void foo(int b, int* c)
{
    int d;

    ...
}
```

4a. Give assembly code for `d = b;`

4b. Give assembly code for `d = A[d];`

4c. Give assembly code for `*c = *c + d;`

5. **(9 marks)** Consider the following assembly-language procedure.

```
        # prologue omitted
        movl    8(%ebp), %ebx           #
        movl    12(%ebp), %ecx          #
        movl    $0, %edx                #
        movl    $0, %eax                #
        cmpl    %ecx, %edx              #
        jge     .L2                     #
.L0:    cmpl    $0,(%ebx,%ecx,4)        #
        jle     .L1                     #
        addl    $1, %eax                #
.L1:    addl    $1, %edx                #
        cmpl    %ecx, %edx              #
        jl      .L0                     #
.L2:    # epilogue omitted
```

5a. Comment the code above and then explain what this procedure does (pseudo-code not necessary).

6. **(6 marks)** Now consider this piece of code.

```
        .section .rodata
.L0:    .long   .L1             #
        .long   .L2             #
        .long   .L3             #
        .section .text
        # some stuff left out
        movl    $1, %eax
        jmp     *.L0(,%ebx,4)   #
.L1:    sall    $1, %eax        #
.L2:    sall    $1, %eax        #
.L3:    sall    $1, %eax        #
```

6a. Comment the code above and then explain what this procedure does (pseudo-code not necessary).

7. (9 marks) Consider this C-language code. Assume that one caller-save register (i.e., `%ecx`) and one callee-save register (i.e., `%ebx`) must be saved/restored for `foo()` to call `bar()`. Answer the following three questions with **commented** assembly code.

```
void foo(int i, int j) {
    bar(i,j);
}
```

7a. Give the assembly code of the procedure-call statement `bar(i,j)`.

7b. Give the assembly code of `bar`'s *prologue*.

7c. Give the assembly code of `bar`'s *epilogue*. Assuming nothing about the current value of the stack pointer when the epilogue starts executing.