CPSC 313, 04w Term 2 — Midterm Exam 1

Date: February 4, 2005, Instructor: Mike Feeley

This is a closed book exam; no calculators; no notes. Answer in the space provided; use the backs of pages if needed. This exam is optional; you can decide at any time to not hand it in and thereby increase the weight of your final exam by 20 percentage marks (e.g., from 44% to 64% of your final grade).

NAME: **STUDENT NUMBER:**

1	2	3	4	5	6	7	8	total

1. **(8 marks)** Short answers.

1a. Assume memory address 0x1000 stores 0x0, 0x1004 stores 0x4 and so on and that `%ebx` stores the value 0x1000. What is the value of `%eax` after each of the following instructions complete?

```
movl %ebx, %eax                 %eax =
movl (%ebx), %eax               %eax =
movl 8(%ebx), %eax              %eax =
leal 8(%ebx), %eax              %eax =
leal 8(%ebx, %ebx, 4), %eax     %eax =
```

1b. Consider the following assembly-language code.

```
subl    %ebx, %edx
cmpl    %edx, %ebx
jlt     .L0
```

Briefly explain, using C-language-like pseudo code, what this code does.

Is there any simple way to modify this code to make it run faster? If so, what?

1c. Carefully describe what a *procedure activation* is and how it differs from a *procedure*

1d. Transistors are used to implement gates. Give a simple abstract description of the key feature of a transistor that makes this possible. That is, describe *what* a transistor does, *not how it does it*.

2. **(5 marks)** Consider the following C-language data-structure declarations.

```
int A[10][2];
struct { int i; int *p; } S[10];
```

And the following pre-conditions:

1. `%ebx` stores the virtual-memory address of A[0][0]
2. `%ecx` stores the virtual-memory address of S
3. `%edx` stores the value of i

For each of the following C-language statements, give an assembly-language implementation that uses **as few instructions as possible** and leaves the value of `r` in `%eax`, without storing it in memory.

2a. `r = A[0][0]`:

2b. `r = A[i][1]`:

2c. `r = (int) &A[i][1]`:

2d. `r = *S[0].p`:

2e. `r = *S[i].p`:

3. **(5 marks)** Consider the following assembly language code.

```
foo:
        # prologue omitted
        movl    8(%ebp), %ebx
        movl    12(%ebp), %eax
        testl   %ebx, %ebx
        jeq     .L1
.L0:    incl    %eax
        decl    %ebx
        jne     .L0
.L1:    # epilogue omitted
        ret
```

Give the most concise C-language program that **could plausibly have been generated by a C compiler** into assembly language of this form (except for the comments).

4. **(5 marks)** Given the following C-language `switch` statement.

```
    switch (i) {
        10: a+=1;
            break;
        12: a+=a;
        13: a+=b;
            break;
        default:
            a=0;
    }
```

And pre-conditions:

1. `%ebx` stores the value of `i`
2. `%ecx` stores the value of `a`
3. `%edx` stores the value of `b`

Complete the following assembly-language implementation of this statement, using a jump table.

```
    .data
.L5:    .long                # fill in the values
        .long
        .long
        .long
    .text
                                 # insert your code here






.L0:    incl    %ecx
        jmp     .L4
.L1:    addl    %ecx, %ecx
.L2:    addl    %edx, %ecx
        jmp     .L4
.L3:    xorl    %ecx, %ecx
.L4:
```

5. **(12 marks)** Consider the following C program.

```
int A[3][3];
int *B;

int foo (int C[], int i, int j)
{
    int k;
    return A[i][j] + B[i] + C[j] + (int) &k;
}

int main ()
{
    B = (int *) calloc ( sizeof(int), 3*3 );
    printf ("%d\n", foo (B,1,2));
}
```

In the space below give the assembly-language declaration of all variables stored in static data section of the program's executable image (i.e., `.data`); use the "`.skip <number-of-bytes>, 0`" directive to allocate space for each variable.) Then, give a complete assembly language implementation for the procedure `foo` (and don't give anything for `main`). Be sure to include the procedure prologue and epilogue and clearly indicate, using comments, which instructions compute each term of the expression. Comment every line of the assembly-language code you write.

```
    .data:




    .text:
foo:
```

6. **(5 marks)** Consider this implementation of the recursive procedure `bar`, which differs from a typical procedure implementation by not using a prologue or epilogue.

```
foo:    # prologue omitted
        movl    $2, %eax
        call bar
        # epilogue omitted
bar:    decl    %eax
        jne     .L0
        call    bar
.L0:    ret
```

6a. Is this correct implementation okay? That is, does the recursion work and does `bar` return to `foo` after the proper number of recursive steps?

6b. If this code is okay, what restrictions must the compiler place on procedures like `bar` for it to use this approach?

7. **(5 marks)** An n-type mosfet transistor can be implemented by introducing two wells of silicon doped with arsnic (which has five electrons in its valance band compared to Silicon's four), a layer of glass and some wires called the source, drain and gate. What value of the gate closes the transistor (i.e., connects them together so that current flows between them)? Briefly explain how this works.

8. **(5 marks)** Give the HCL description of a circuit that takes two 32-bit integer inputs, A and B, and computes a function that evaluates to:

- 0 if either A or B are 0,
- A if A is odd and B is even,
- B if B is odd and A is even,
- $A + B$ if both are even, and
- $A - B$ if both are odd.

You can use the notation, $A[i]$ to refer to the i^{th} bit of A (where the lowest-order bit is bit 0).