EE 4720 Homework 3 Due: 14 November 2001

Problem 1: The code below executes on a dynamically scheduled four-way superscalar DLX implementation that uses reorder buffer entry numbers to name destination registers.

- Loads and stores use the load/store unit, which consists of segments L1 and L2.
- The floating-point multiply unit is fully pipelined and consists of six segments.
- The usual number of instructions (for a 4-way superscalar machine) can be fetched, decoded, and committed per cycle.
- An unlimited number of instructions can complete per cycle. (This makes the solution easier.)
- There are an unlimited number of reservation stations and reorder buffer entries.
- The target of a branch is fetched in the cycle after the branch is in ID, *whether or not the branch condition is available.* (We'll cover that later.)

(*a*) Show a pipeline execution diagram for the code below until the beginning of the fourth iteration. Show where instructions commit.

(*b*) What is the CPI for a large number of iterations? *Hint: There should be less than six cycles per iteration.*

(*c*) Show the entries in the register map for registers `f0` and `r1` for each cycle. (Make up reorder buffer entry numbers.)

```
LOOP:  ! LOOP = 0x1000
 ld f0, 0(r1)
 muld f0, f0, f2
 sw 0(r1), f0
 addi r1, r1, #8
 sub r2, r1, r3
 bnez r2, LOOP
```

(*d*) The first two instructions of the code below are different than the code above, the other instructions are identical. It runs on a system identical to the one above except that there are only 1000 reorder buffer entries. (That's actually a lot, but it's not unlimited.) What is the CPI for a large number of iterations? Is the CPI really lower in the period before reorder buffers are used up? If you can, solve the problem without drawing a complete pipeline execution diagram.

```
LOOP:  ! LOOP = 0x1000
 ld f4, 0(r1)
 muld f0, f0, f4
 sw 0(r1), f0
 addi r1, r1, #8
 sub r2, r1, r3
 bnez r2, LOOP
```

Problem 2: When the MIPS program below starts register `$t0` holds the address of a string, the program converts the string to upper case.

(For MIPS documentation see `http://www.ece.lsu.edu/ee4720/mips32v1.pdf` and `http://www.ece.lsu.edu/ee4720/mips32v2.pdf`. Here are the relevant differences with DLX: branches and jumps are delayed (1 cycle). Some branch instructions compare two registers. Register `$0` works like DLX `r0`.)

```
LOOP:
        lbu $t1, 0($t0)
        addi $t0, $t0, 1
        beq $t1, $0, DONE
        slti $t2, $t1, 97 # < 'a'
        bne $t2, $0 LOOP
        slti $t2, $t1, 123 # 'z' + 1
        beq $t2, $0, LOOP
        addi $t1, $t1, -32
        j LOOP
        sb $t1,-1($t0)
DONE:
```

Convert the program to IA-64 assembly language using predicated instructions. (You're not expected to know it at this point.) IA-64 is described in the IA-64 Application Developer's Architecture Guide, available at `http://www.ece.lsu.edu/ee4720/ia-64.pdf`.

For this problem one can ignore alot of IA-64's features. Here is what you will need to know: IA-64 has 64 1-bit predicate registers, `p0` to `p63`, which are written by `cmp` (compare) and other instructions. Predicates can be specified for most instructions, including `cmp` itself. See 11.2.2 for a description of how to use IA-64 predicates.

To solve the problem look at the following sections: 9.3, 9.3.1, and 9.3.2 (a brief description of where to place stops); 11.2.2 (predicate description and some more information on stops); and Chapter 7 (for instruction descriptions). The following instructions will be needed: `cmp` (compare, look at the normal [none] and `and` comparison types), `br` (branch), load, store, and add.

- Use general-purpose registers r0-r31 and predicate registers p1-p63 in your solution. (There are 128 general-purpose registers, but those above r31 must be allocated.)
- Minimize the number of instructions per iteration assuming about half the characters are lower case.
- Use predicates to eliminate some branches.
- Make use of post-increment loads or stores.
- Pay attention to data type sizes.
- Show stops but do not show bundle boundaries.