EE 4720 **Homework 5** **Due: 22 April 1998**

The code below is referred to in the problems.

```
LOOP:   ! LOOP = 0x5000
 lw   r1, 0(r2)
 addi r2, r2, #4
 beqz r1, TARG
 sub  r4, r4, r1
 j    LOOP
TARG:
 subi r5, r5, #1
 bnez r5, LOOP
 and  r4, r4, r6
 or   r4, r4, r7
 sw   0(r8), r4
 addi r8, r8, #4
 jr   r31
```

Problem 1: Show the execution of the code above up to IF of the third iteration on a 4-way superscalar implementation of DLX which is statically scheduled, instruction fetches are 4-instruction aligned, is fully bypassed including ID stage bypassing for branch conditions, and in which hardware is fully duplicated (including writeback). Assume that the first conditional branch is not taken in the first iteration, but taken in the second iteration, and that the code executes for many iterations. Branches do not have delay slots, hardware cannot detect branch target/fall through overlap, there is no branch prediction hardware, and no branch target prediction hardware. Indicate the cancelling of an instruction by an x in the earliest cycle that the hardware could cancel it. (Do not assume the implementation can predict the future using unspecified hardware or any other means.) For example in a single-issue implementation:

```
Time              0   1   2   3   4   5   6
 beqz r0, TARG    IF  ID  EX  MEM WB
 add  r1, r2, r2      IF  x
...
TARG:
 sub  r1, r3, r4          IF  ID  EX  MEM WB
```

Problem 2: Compute the CPI of the execution of a large number of iterations of the loop above when 30% of the words starting at the initial value of `r2` hold zero.

Problems on next page.

Problem 3: How effective and how practical would each of the following be in speeding execution of the loop (compared to the problem-1 DLX implementation):

- Branch delay slots.
- Dynamic scheduling.
- Branch target buffer.
- Predicated execution.
- Loop unrolling. (See section 4.1)

The answer should specify how each technique would avoid specific bubbles (if possible) present in the solution to problem 1.

Problem 4: A 4-way VLIW ISA (derived from DLX) includes predicated execution in the following way: the first instruction of a bundle can be a *bundle execution specifier* (BX) instruction which specifies whether the remaining three instructions execute. The instruction has three register operands, corresponding to the second, third, and fourth instruction in the bundle. Each register operand has a negation bit, indicated in assembly language by an exclamation point. If the negation bit is zero then the corresponding instruction executes if the register contents is non-zero. If the negation bit is one then the corresponding instruction executes if the register contents is zero.

For example, consider the following bundle:

```
bx  r1,!r2,!r0
add r4, r5, r6
sub r7, r8, r9
div f0, f1, f2
```

The `add` executes if `r1` is non-zero, the `sub` executes if `r2` is zero, and the divide always executes.

If the first instruction of a bundle is not `bx` all instructions will execute (including the first, as an ordinary instruction). Source registers in a bundle refer to values produced in preceding bundles.

Convert the DLX program below to this VLIW ISA.

```
 beqz r1, ELSE
 add  r2, r3, r4
 j    ENDIF
ELSE:
 add  r2, r3, r5
ENDIF:
 sub  r6, r6, r2
 bnez r1, SKIP
 addi r7, r7, #1
SKIP:
 addi r8, r2, #12
 slt  r1, r8, r9
 and  r10, r8, r11
```