

**Problem 1:** Complete a pipeline execution diagram for the following code running on a two-way statically scheduled superscalar processor. Show execution until the second fetch of the first `add`. The processor fetches instructions in aligned groups and is fully bypassed. The branch will be taken. There is no branch prediction hardware.

What is the CPI for a large number of iterations?

```
LOOP: ! LOOP = 0x1004
  add r1, r2, r3
  add r4, r5, r6
  add r9, r4, r7
  lw  r10, 0(r4)
  add r11, r11, r10
  or  r12, r11, r13
  xor r15, r16, r17
  bnez r10, LOOP
```

**Problem 2:** Schedule the code from the problem above so that it executes efficiently. The solution can contain added `nop` instructions. Do not try to unroll the loop. A correct solution contains two stalls plus the branch delay.

Now, what is the CPI for a large number of iterations?

**Problem 3:** Show the execution of the code from Problem 1 on a two-way superscalar dynamically scheduled machine using Method 1. The number of reservation stations, functional units, and reorder buffer entries is unlimited. Do not show reservation station numbers or reorder buffer entry numbers in the diagram. Do show where instructions commit. Assume that the machine has perfect branch and branch target prediction and so a branch target will be fetched when the branch is in ID. Complete the diagram to the point where all instructions in the first iteration commit, showing what happens to instructions in the second iteration up to that point.

Now, what is the CPI for a large number of iterations?

*More problems on the next page.*

**Problem 4:** Convert the code below to VLIW DLX as described in the notes. The maximum lookahead value is 15, use that for bundles that do not modify any registers. Set the lookahead values and serial bits for maximum performance. (The lookahead values will mostly be small.) How would a modification of the end-of-loop test improve performance on a VLIW implementation?

```
j TEST
LOOP:
lw r1, 0(r10)
lw r2, 4(r10)
lw r3, 8(r10)
lw r4, 12(r10)
andi r1, r1, #15
andi r2, r2, #15
andi r3, r3, #15
andi r4, r4, #15
sw 0(r10), r1
sw 4(r10), r2
sw 8(r10), r3
sw 12(r10), r4
addi r10, r10, #16
TEST:
slt r11, r10, r12
bnez r11, LOOP
```

**Problem 5:** Insert the minimum number of IA-64-style stops in the DLX code below. Do not convert the instructions themselves to IA-64, just insert the stops.

The material on stops was covered in class and will be in the notes. A primary reference is Appendix A of the IA-64 Application Developer's Architecture Guide, available at <http://developer.intel.com/design/ia64/downloads/adag.pdf>. Appendix A describes how stops affect the execution of code.

```
j TEST
LOOP:
lw r1, 0(r10)
lw r2, 4(r10)
lw r3, 8(r10)
lw r4, 12(r10)
andi r1, r1, #15
andi r2, r2, #15
andi r3, r3, #15
andi r4, r4, #15
sw 0(r10), r1
sw 4(r10), r2
sw 8(r10), r3
sw 12(r10), r4
addi r10, r10, #16
TEST:
slt r11, r10, r12
bnez r11, LOOP
```