Weiss, S., and J. E. Smith [1984]. "Instruction issue logic for pipelined supercomputers," *Proc. 11th Symposium on Computer Architecture* (June), Ann Arbor, Mich., 110–118.

Exercises

Solutions to the "starred" exercises appear in Appendix B.

A.1 [15/15/15] <A.2> Use the following code fragment:

```
Loop:  LD       R1,0(R2)       ;load R1 from address 0+R2
       DADDI    R1,R1,#1       ;R1=R1+1
       SD       0(R2),R1       ;store R1 at address 0+R2
       DADDI    R2,R2,#4       ;R2=R2+4
       DSUB     R4,R3,R2       ;R4=R3-R2
       BNEZ     R4,Loop        ;branch to Loop if R4!=0
```

Assume that the initial value of R3 is R2 + 396.

Throughout this exercise use the classic RISC five-stage integer pipeline (see Figure A.1) and assume all memory accesses take 1 clock cycle.

a. [15] <A.2> Show the timing of this instruction sequence for the RISC pipeline *without* any forwarding or bypassing hardware but assuming a register read and a write in the same clock cycle "forwards" through the register file, as in Figure A.6. Use a pipeline timing chart like Figure A.6. Assume that the branch is handled by flushing the pipeline. If all memory references take 1 cycle, how many cycles does this loop take to execute?

b. [15] <A.2> Show the timing of this instruction sequence for the RISC pipeline with normal forwarding and bypassing hardware. Use a pipeline timing chart like Figure A.6. Assume that the branch is handled by predicting it as not taken. If all memory references take 1 cycle, how many cycles does this loop take to execute?

c. [15] <A.2> Assume the RISC pipeline with a single-cycle delayed branch and normal forwarding and bypassing hardware. Schedule the instructions in the loop including the branch delay slot. You may reorder instructions and modify the individual instruction operands, but do not undertake other loop transformations that change the number or opcode of the instructions in the loop (that's for Chapter 4!). Show a pipeline timing diagram and compute the number of cycles needed to execute the entire loop.

✪ A.2 [15/15] <A.2, A.4, A.5> Use the following code fragment:

```
Loop:  L.D      F0,0(R2)
       L.D      F4,0(R3)
       MUL.D    F0,F0,F4
       ADD.D    F2,F0,F2
       DADDUI   R2,R2,#8
       DADDUI   R3,R3,#8
       DSUBU    R5,R4,R2
       BNEZ     R5,Loop
```

Assume that the initial value of R4 is R2 + 792.

For this exercise assume the standard five-stage integer pipeline and the MIPS FP pipeline as described in Section A.5. If structural hazards are due to write-back contention, assume the earliest instruction gets priority and other instructions are stalled.

a. [15] <A.2, A.4, A.5> Show the timing of this instruction sequence for the MIPS FP pipeline *without* any forwarding or bypassing hardware but assuming a register read and a write in the same clock cycle "forwards" through the register file. Assume that the branch is handled by flushing the pipeline. If all memory references hit in the cache, how many cycles does this loop take to execute?

b. [15] <A.2, A.4, A.5> Show the timing of this instruction sequence for the MIPS FP pipeline with normal forwarding and bypassing hardware. Assume that the branch is handled by predicting it as not taken. If all memory references hit in the cache, how many cycles does this loop take to execute?

A.3 [15] <A.2> Suppose the branch frequencies (as percentages of all instructions) are as follows:

Conditional branches	15%
Jumps and calls	1%
Conditional branches	60% are taken

We are examining a four-deep pipeline where the branch is resolved at the end of the second cycle for unconditional branches and at the end of the third cycle for conditional branches. Assuming that only the first pipe stage can always be done independent of whether the branch goes and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?

A.4 [15] <A.1, A.2> A reduced hardware implementation of the classic five-stage RISC pipeline might use the EX stage hardware to perform a branch instruction comparison and then not actually deliver the branch target PC to the IF stage until the clock cycle in which the branch instruction reaches the MEM stage. Control hazard stalls can be reduced by resolving branch instructions in ID, but improving performance in one respect may reduce performance in other circumstances. How does determining branch outcome in the ID stage have the potential to increase data hazard stall cycles?

A.5 [12/13/20/20/15/15] <A.2, A.3> For these problems, we will explore a pipeline for a register-memory architecture. The architecture has two instruction formats: a register-register format and a register-memory format. There is a single-memory addressing mode (offset + base register).

There is a set of ALU operations with format

ALUop Rdest, Rsrc_1, Rsrc_2

or

ALUop Rdest, Rsrc_1, MEM