Patrick Woodrum

Homework 4

Due 11:59pm Thursday, March 25

1. [20pts]Consider the following loop.

```
Loop:   lw    r1, 0(r1)
        and   r1, r1, r2
        lw    r1, 0(r1)
        lw    r1, 0(r1)
        beq   r1, r0, loop
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, and that the pipeline has full forwarding support. Also assume that many iterations of this loop are executed before the loop exits.

a. [15pts]Show a pipeline execution diagram for the third iteration of this loop, from the cycle in which we fetch the first instruction of that iteration up to (but not including) the cycle in which we can fetch the first instruction of the next iteration. Show all instructions that are in the pipeline during these cycles (not just those from the third iteration. Some from 2nd and 4th iterations).

cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(2nd)															
(2nd)	W														
(2nd)	D	E	M	W											
Loop: lw r1,0(r1)	F	D	E	M	W										
and r1, r1, r2		F	D	Stall	E	M	W								
lw r1, 0(r1)			F	stall	D	E	M	W							
lw r1, 0(r1)					F	D	stall	E	M	W					
beq r1, r0, loop						F	stall	stall	stall	D	E	M	W		
(4th)										F	D	E	M	W	
(4th)											F	D	stall	E	M
(4th)												F	stall	D	E
(4th)														F	D
(4th)															F

b. [5pts]How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?

1/5 or 20% of the cycles have all five pipeline stages doing useful work.

Patrick Woodrum

2. Branch prediction, accuracy and impact.
   The CPI for an ideal 5-stage pipeline that is always full is 1. Assume a code with no data hazards, and no delay slots are used, but with 25% of branch instruction. Assume that branch outcomes are determined in the ID stage and a misprediction causes a cycle delay. Also assume the following branch predictor accuracies.

Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

   a. [5pts]Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the always-taken predictor?
      Always-taken is 0.45 so extra CPI = 3*(1-0.45)*0.25 = 0.4125
   b. [5pts]What is the extra CPI due to mispredicted branches with the always-not-taken predictor?
      Always-not-taken is 0.55 so extra CPI = 3*(1-0.55)*0.25 = 0.3375
   c. [5pts]What is the extra CPI due to mispredicted branches with the 2-bit predictor?
      2-bit is 0.85 so extra CPI = 3*(1-0.85)*0.25 = 0.1125

3. [20pts]Based on the lecture notes, you are given a standard single issue 5-stage pipeline with forwarding, extended with an integer ALU and a floating-point ALU (integer instructions executing as 'E' and FP instructions executing as 'X' respectively). An independent instruction can start into the execute stage on each cycle. All instructions must complete in program order, so single-cycle instructions cannot enter the memory stage until all previous instruction have passed through the memory stage.

E.g.,

```
addf  F4,F2,F0  //4-cycle fp exec, F4 <- F2 + F0      F D X X X X M W
addf  F8,F6,F0  //4-cycle fp exec, F8 <- F6 + F0        F D X X X X M W
```

and

```
addf  F4,F2,F0  //4-cycle fp exec, F4 <- F2 + F0      F D X X X X M W
sub   R1,R1,16  //1-cycle int exec, R1 <- R1 - 16       F D E - - - M W
```

Patrick Woodrum

a. [15pts]Given the following loop code segment, draw the pipeline diagram in the table by filling the stages and stalls, and mark the data forwarding.

```
Loop: ld    F0, 0(R1)         // 1-cycle latency, F0 <- memory[ R1 + 0 ]
      addf  F4, F2,F0         // 3-cycle latency, F4 <- F2 + F0
      st    F4, 0(R1)         // 1-cycle latency, memory[ R1 + 0 ] <- F4
      ld    F6, 8(R1)         // 1-cycle latency, F6 <- memory[ R1 + 8 ]
      addf  F8, F6,F0         // 3-cycle latency, F8 <- F6 + F0
      st    F8, 8(R1)         // 1-cycle latency, memory[ R1 + 8 ] <- F8
      sub   R1, R1, 16        // 0-cycle latency, R1 <- R1 - 16
      bne   R1, R2, loop      // 0-cycle latency, branch to loop if R1 != R2
```

cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Loop: ld F0, 0(R1)	F	D	E	M	W																
addf F4, F2, F0		F	D	stall	stall	E	E	E	M	W											
st F4, 0(R1)			F	stall	D	E	stall	stall	stall	M	W										
ld F6, 8(R1)					F	D	stall	stall	stall	E	M	W									
addf F8, F6, F0						F	stall	stall	stall	D	stall	E	E	E	E	M	W				
st F8, 8(R1)										F	stall	D	E	stall	stall	stall	M	W			
sub R1, R1, 16												F	D	stall	stall	stall	E	M	W		
bne R1, R2, loop																	F	D	E	M	W

b. [5pts]How many cycles are there between two successive stores of F4 in the loop? Consider there is no branch prediction.

4 cycles

4. 2-issue processors. You are given the following code for a loop:

```
Loop:   lw    r1, 0(r6)
        lw    r2, 4(r6)
        sub   r3, r1, r2
        sw    r3, 0(r7)
        addi  r6, r6, 8
        addi  r7, r7, 8
        addi  r4, r4, 2
        bne   r4, r0, loop
```

Patrick Woodrum

a. [15pt]If the loop exits after executing only two iterations. Show the scheduled instructions for the given MIPS code on a 2-issue processor shown in Figure 4.69. Assume the processor has perfect branch prediction.

loop	ALU/branch	Load/store	cycle
1st iteration		lw $r1, 0($r6)	1
		lw $r2, 4($r6)	2
	sub $r3, $r1, $r2		3
		sw $r3, 0($r7)	4
	addi $r6, $r6, 8		5
	addi $r7, $r7, 8		6
	addi $r4, $r4, 2		7
	bne $r4, $r0, loop		8

b. [15pts]Rearrange the code to achieve better performance on a 2-issue statically scheduled processor from Figure 4.69. Write the sequence below.

```
Loop:   lw    r1, 0(r6)
        lw    r2, 4(r6)
        sub   r3, r1, r2
        lw    r3, r2(r1)
        sw    r3, 0(r7)
        addi  r6, r6, 8
        lw    r6, 8(r6)
        addi  r7, r7, 8
        lw    r7, 8(r7)
        addi  r4, r4, 2
        lw    r4, 2(r4)
        bne   r4, r0, loop
        sw    r4, r0, loop
```

c. [15pts]Repeat step a for the code from b.

loop	ALU/branch	Load/store	cycle
1st iteration		lw $r1, 0($r6)	1
		lw $r2, 4($r6)	2
	sub $r3, $r1, $r2	lw $r3, $r1($r2)	3
		sw $r3, 0($r7)	4
	addi $r6, $r6, 8	lw $r6, 8($r6)	5
	addi $r7, $r7, 8	lw $r7, 8($r7)	6
	addi $r4, $r4, 2	lw $r4, 2($r4)	7
	bne $r4, $r0, loop	sw $r4, $r0, loop	8