Name DAN PORTS

Computer System Architecture
6.823 Quiz #3
November 3rd, 2006
Professor Krste Asanovic
Dr. Joel Emer

Name: DAN PORTS

This is a closed book, closed notes exam.
80 Minutes
10 Pages

Notes:

- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully
- Please carefully state any assumptions you make
- Please write your name on every page in the quiz
- You must not discuss a quiz's contents with other students who have not yet taken the quiz

Writing name on each sheet	2	2 Points
Question 1	29	30 Points
Question 2	32	36 Points
Question 3	12	12 Points
TOTAL	75	**80 Points**

Name Dan Ports

Question 1: Local and Global History Bits (30 points)

This problem will investigate the effects of adding local and global history bits to a standard branch prediction mechanism. In this problem we assume that the MIPS ISA has no delay slots.

In the first 3 parts of the question, we will be working with the following program.

```
loop:
     LW   R4, 0(R3)
     ADDI R3, R3, 4
     SUBI R1, R1, 1
b1:
     BEQZ R4, b2
     ADDI R4, R4, 1
     SW   R4, -4(R3)
b2:
     BNEZ R1, loop
```

Assume that the initial value of R1 is n, where n>0. Also assume that the initial value of R3 is p, where p points to the beginning of an array of 32-bit integers.

Part A (5 points) 5

Briefly explain what the program does?

For each of the first n elements in the array at p, if the value is non-zero, it is incremented by 1.

Name D Ponts

Part B (12 points) 11

We will use a 2-bit saturating counter as our predictor state machine. It is shown below.

↓ on not taken	↑ on taken			
		1	1	Strongly taken
		1	0	Weakly taken
		0	1	Weakly not taken
		0	0	Strongly not taken

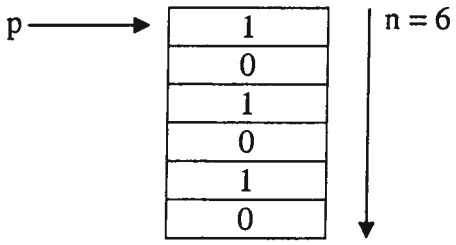
In state 1X we will guess that the branch is taken and in state 0X we will guess that it is not taken.

Assume that b1 and b2 do not conflict in the BHT.

We have added a single global history bit to the branch predictor and use it to select a set associated with a branch. Complete the table on the following page. The table contains an entry for every branch (b1 and b2) that is executed. The Branch Predictor (BP) bits in the table are the bits from the BHT. For each branch, check the corresponding BP bits (indicated by the bold entries in the examples) to make a prediction, and then update the BP bits in the following entry (indicated by the italic entries in the examples).

Name D Pan

Assume that the inputs to the program are shown by the structure below.



b1: branch if R4 = 0

b2: if R1 ≠ 0 (taken except last)

System State			Branch Predictor				Behavior	
PC	R3/R4	History bit	b1 bits		b2 bits			
			Set 0	Set 1	Set 0	Set 1	Predicted	Actual
b1	4/1	1	10	**10**	01	01	T	N
b2	4/2	0	10	*01*	**01**	01	N	T
b1	8/0	1	10	**01**	*10*	01	N	T
b2	8/0	1	10	10	10	01	N	T
b1	12/1	1	10	10	11	10	T	N
b2	12/2	0	10	01	10	10	T	T
b1	16/1	1	10	01	11	10	N	T
b2	16/2	1	10	10	11	10	T	T
b1	20/1	1	10	10	11	11	T	N
b2	20/2	0	10	01	11	11	T	T
b1	24/0	1	10	01	11	11	N	T
b2	24/0	0 X	10	10	11	11	T	N

Part C (5 points)

If we use a local history bit for each branch instead of using the global history bit, as in part B, how does the number of mispredicts change? And why?

The number of mispredicts at steady-state will decrease. Here, the branch conditions depend only on their local history, not the global history. A single local history bit for b_1 correctly determines the branch condition, while a single global history bit does not help. b_2 is always taken except in the last iteration which will always be mispredicted, so both history schemes perform equally well.

Name D PORTS

Part D (8 points) 8

Consider a pipelined processor with 10 pipeline stages. It uses the same branch prediction mechanism as in part B, that is, it uses a single global history bit and a BHT with a 2-bit saturating counter as the predictor state machine. The direction of the branch is predicted in the second stage of the pipeline but the global history bit and the BHT are only updated when the branch commits in the final stage of the pipeline. Consider the following MIPS code.

```
      SLTI R2, R1, 9     # if R1 < 9 then R2 = 1 else R2 = 0
b1:
      BEQZ R2, b1t       # jump to b1t if R2 = 0  ⇒ R1 ≥ 9
      ADDI R3, R3, 10    # R3 = R3 + 10
b1t:
      SLTI R4, R1, 7     # if R1 < 7 then R4 = 1 else R4 = 0
b2:
      BEQZ R4, b2t       # jump to b2t if R4 = 0  ⇒ R1≥7
      ADDI R5, R5, 5     # R5 = R5 + 5
b2t:
```

We can see that there is a high correlation between b1 and b2: the result of b2 is likely to be the same as b1. The BHT reflects this correlation and has the following status.

Global History Bit	BHT			
	b1		b2	
	Set0	Set1	Set0	Set1
0	10	01	00	11

The processor starts another execution of the code with R1 = 10. Briefly describe how b2 will be predicted. Can you suggest a fix to achieve higher prediction accuracy?

b1 will be predicted taken, correctly. But the global history bit is not updated to 1 until several cycles later, so when b2 is predicted, it will still be 0. b2 will be mispredicted not-taken. We can fix this by speculatively updating the global history based on the prediction in stage 2 rather than waiting for the correct value in the last stage. ✓

Name D Patel

Question 2: Reorder Buffer with Unified Physical Register File (36 Points)

Part A (8 Points)

Please identify the potential RAW, WAR, WAW hazards for the following MIPS code and complete the table at the bottom of the page.

```
I1    LW    R5, 0(R1)
I2    LW    R6, 0(R2)
I3    ADD   R5, R5, R6
I4    LW    R7, 0(R3)
I5    MUL   R7, R7, R5
I6    ADD   R3, R3, R1
I7    SW    R7, 0(R3)
I8    LW    R8, 0(R4)
I9    MUL   R8, R8, R5
I10   ADD   R4, R4, R2
I11   SW    R8, 0(R4)
```

Issuing Instruction	RAW	WAR	WAW
I1	-	-	-
I2	-	-	-
I3	I1, I2	-	I1
I4	—	—	—
I5	I3, I4	—	I4
I6	—	I4	—
I7	I5, I6	—	—
I8	—	—	—
I9	I8, I3	—	I8
I10	—	I8	—
I11	I9, I10	—	—

8

Name DPORTS

Part B (9 Points)

The MIPS ISA provides 31 registers (remember that R0 always returns 0). Can you modify the registers used by the code in Part A so that the new code will have no WAR or WAW hazard?

	Opcode	Dest	Src1	Src2
I1	LW	R5	R1	-
I2	LW	R6	R2	-
I3	ADD	R9	R5	R6
I4	LW	R7	R3	
I5	MUL	R10	R7	R9
I6	ADD	R11	R3	R1
I7	SW		R10	R11
I8	LW	R8	R4	
I9	MUL	R12	R8	R5
I10	ADD	R13	R4	R2
I11	SW		R12	R13

R5	R9
R7	R10
R3	R11
R8	R12
R4	R13

9

Name D PORTS

Part C (9 Points)

Most modern machines perform register renaming in hardware similar to what you have done manually in Part B to eliminate WAW and WAR hazards. In this part, we will study the number of cycles it takes to complete the code on a machine with unified physical registers. The machine has the following features.

- Issue Stage (I)
  - Issue width: infinite
  - Issue latency: 1 cycle (including the time for reading physical registers)
- Execution (E)
  - Infinite functional and memory units
  - Adder latency: 1 cycles
  - Multiplier latency: 3 cycles
  - Load/Store unit latency: 1 cycles
- Write-back (W)
  - Infinite write-back ports
  - Write-back latency: 1 cycle
  - SW does not write back
  - Instruction results can only be used after the write-back stage
- Commit (C)
  - In-order commit
  - Infinite commits allowed in the same cycle
  - Commit latency: 1 cycle

Assume that the scheduler does not issue an instruction until after its dependencies finish writing back, (Not the same cycle they are writing back)

Assume that 1) all instructions (I1 - I11) are in the ROB at the beginning, 2) no memory dependency exists between I7 and I8, and 3) the machine has a perfect memory dependency predictor. Please complete the following timing diagram.

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15	t16	t17
I1	I	E	W	C													
I2	I	E	W	C													
I3				I	E	W	C										
I4	I	E	W			C	C										
I5							I	E	E	E	W	C					
I6	I	E	W									C					
I7								I				I	E	C			
I8	I	E	W											C			
I9							I	E	E	E	W			C			
I10	I	E	W											C			
I11												I	E	C			

9

Name DPorts

Part D (5 Points)

Ben Bitdiddle thinks that it is not necessary to allocate a new physical register to an instruction before it enters the ROB, but only a tag to name the value. He proposes to delay allocation until the issue stage, where each tag is then associated with a new physical register. As with the scheme presented in class, his proposal allows a physical register to be reused when the next write to the same architectural register commits. He thinks that his proposal will reduce the number of physical registers required by the machine.

According to Ben's proposal, what is the minimum number of physical registers required so that the machine can execute the code with exactly the same schedule you obtained in Part C? (Assume that R0 always return 0 and doesn't need to be renamed)

14 registers are needed: we must store the 8 values corresponding to architectural registers R1 - R8, plus 6 registers to hold the values generated by issued but uncommitted instructions. There are at most 6 issued but uncommitted instructions in the schedule in part C. 3

→ 3&1

→ R1-R31

Part E (5 Points)

After reading Ben's proposal, Alyssa P. Hacker immediately points out that Ben's scheme may not work, at least not with the intention of saving physical registers. Explain briefly what may go wrong if the machine has less physical registers than the sum of ROB entries and architectural registers? Can you think of a solution that does not require more physical registers?

Suppose there are p physical registers, r architectural registers and e ROB entries; p < r+e. If there are p-r entries in the ROB that have been issued but uncommitted, and another entry is ready to issue, there will not be a physical register available for it. We could stall issuing until a register becomes available, but this would limit the ability of the processor to execute out-of-order.

+2

and cause deadlock

+1

6

Name D Bolts

Question 3 Memory Dependencies (12 Points)

Part A (6 Points)

For each of the following 3 instruction sequences, please give the condition for a memory dependency to exist or explain why there cannot be a dependency. Assume that there is no memory aliasing (i.e. all virtual memory pages are mapped to unique physical pages).

Instruction Pair	Condition under which Memory Dependency occurs
SW R2, 0(R3) LW R5, 0(R4)	R3 = R4
SW R2, 0(R3) LW R5, 4(R3)	R3 = R3+4 Cannot occur, so no possible dependency
SW R2, 0(R3) LW R5, 4096(R3)	R3 = R3 +4096 Cannot occur, since the two pages cannot be aligned No dependency possible

6

Name DPorts

Part B (6 Points)

If we allow memory aliasing to occur, how will it affect your answer in part A? Assume that the page size is 4KB and that the machine is byte addressed.

Instruction Pair	Condition under which Memory Dependency occurs
SW R2, 0(R3) LW R5, 0(R4)	R3 = R4, or R3 = R4 mod 4KB and R3 and R4 point to two virtual pages that map to the same physical page
SW R2, 0(R3) LW R5, 4(R3)	Cannot occur. If R3 and R3+4 are in the same page, the offsets will be different. Even if R3 is at a page boundary and the next page is aliased, the ~~Offsets of R3 and R3+4~~ will differ.
SW R2, 0(R3) LW R5, 4096(R3)	If the page pointed to by R3 and the next page map to the same physical page.

6