Introduction to Computer Architecture

Assignment 2 - Solution

Due May 14, 2015

1. [50 = 10 x 5] Use the following code fragment:

```
Loop:   LD          R1, 0(R2)       ; load R1 from address 0+R2

        DADDI       R1, R1, #1      ; R1 = R1 + 1

        SD          R1, 0, (R2)     ; 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.

a. Data hazards are caused by data dependences in the code. List all of the data dependences in the code above. Record the register, source instruction, and destination instruction; for example, there is a data dependency fro register R1 from the LD to DADDI, that is,

   R1 LD DADDI

b. Show the timing of this instruction sequence for the 5-stage RISC pipeline without any forwarding or bypassing hardware but assuming that a register read and a write in the same clock "forwards" through the register file, as shown in Figure C.6. Use a pipeline timing chart like that in Figure C.5. Assume that the branch is handled by flushing the pipelining. If all memory references take 1 cycle, how many cycles does this loop take to execute?

c. Show the time of this instruction sequence for the 5-state RISC pipeline with full forwarding and bypassing hardware. Use a pipeline timing chart like that shown in Figure C.5. Assume that the branch is handled by predicting it as not taken. If all memory references take 1 cycle, how many clock cycles does this loop take to execute?

d. Show the timing of this instruction sequence for the 5-stage RISC pipeline with full forwarding and bypassing hardware. Use a pipeline timing chart like that shown in Figure C.5. Assume that the branch is handled by predicting it as taken. If all memory references take 1 cycle, how many clock cycles does this loop take to execute?

e. High-performance processors have very deep pipelines-more than 15 stages. Imagine that you have a 10-stage pipeline in which every stage of the 5-stage pipeline has been split in two. The only catch is that, for data forwarding, data are forwarded from the end of a pair of stages to the beginning of the two stages where they are needed. For example, data are forwarded from the output of the second execute stage to the input of the first execute stage, still causing a 1-cycle delay. Show the timing of this instruction sequence for the 10-stage RISC pipeline with full forwarding and bypassing hardware. Use a pipeline timing chart like that shown in Figure C.5. Assume that the branch is handled by predicting it as taken. If all memory references take 1 cycle, how many cycles does this loop take to execute?

Solution

a.

R1	LD	DADDI
R1	DADDI	SD
R2	LD	DADDI
R2	SD	DADDI
R2	DSUB	DADDI
R4	BNEZ	DSUB

b. Forwarding is performed only via the register file. Branch outcomes and targets are not known until the end of the execute stage. All instructions introduced to the pipeline prior to this point are flushed.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
`LD    R1, 0(R2)`	F	D	X	M	W													
`DADDI R1, R1, #1`		F	s	s	D	X	M	W										
`SD    0(R2), R1`					F	s	s	D	X	M	W							
`DADDI R2, R2, #4`								F	D	X	M	W						
`DSUB  R4, R3, R2`									F	s	s	D	X	M	W			
`BNEZ  R4, Loop`												F	s	s	D	X	M	W
`LD    R1, 0(R2)`																	F	D

Since the initial value of R3 is R2 + 396 and equal instance of the loop adds 4 to R2, the total number of iterations is 99. Notice that there are 8 cycles lost to RAW hazards including the branch instruction. Two cycles are lost after the branch because of the instruction flushing. It takes 16 cycles between loop instances; the total number of cycles is $98 \times 16 + 18 = 1584$. The last loop takes two addition cycles since this latency cannot be overlapped with additional loop instances.

c. Now we are allowed normal bypassing and forwarding circuitry. Branch outcomes and targets are known now at the end of decode.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
`LD    R1, 0(R2)`	F	D	X	M	W													
`DADDI R1, R1, #1`		F	D	s	X	M	W											
`SD    R1, 0(R2)`			F	s	D	X	M	W										
`DADDI R2, R2, #4`					F	D	X	M	W									
`DSUB  R4, R3, R2`						F	D	X	M	W								
`BNEZ  R4, Loop`							F	s	D	X	M	W						
(incorrect instruction)									F	s	s	s	s					
`LD    R1, 0(R2)`										F	D	X	M	W				

Again we have 99 iterations. There are two RAW stalls and a flush after the branch since the branch is taken. The total number of cycles is $9 \times 98 + 12 = 894$. The last loop takes three addition cycles since this latency cannot be overlapped with additional loop instances.

d. See the table below.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
`LD    R1, 0(R2)`	F	D	X	M	W													
`DADDI R1, R1, #1`		F	D	s	X	M	W											
`SD    R1, 0(R2)`			F	s	D	X	M	W										
`DADDI R2, R2, #4`					F	D	X	M	W									
`DSUB  R4, R3, R2`						F	D	X	M	W								
`BNEZ  R4, Loop`							F	s	D	X	M	W						
`LD    R1, 0(R2)`									F	D	X	M	W					

Again we have 99 iterations. We still experience two RAW stalls, but since we correctly predict the branch, we do not need to flush after the branch. Thus, we have only $8 \times 98 + 12 = 796$.

e. See the table below.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
`LD    R1, 0(R2)`	F1	F2	D1	D2	X1	X2	M1	M2	W1	W2										
`DADDI R1, R1, #1`		F1	F2	D1	D2	s	s	s	X1	X2	M1	M2	W1	W2						
`SD    R1, 0(R2)`			F1	F2	D1	s	s	s	D2	X1	X2	M1	M2	W1	W2					
`DADDI R2, R2, #4`				F1	F2	s	s	s	D1	D2	X1	X2	M1	M2	W1	W2				
`DSUB  R4, R3, R2`					F1	s	s	s	F2	D1	D2	s	X1	X2	M1	M2	W1	W2		
`BNEZ  R4, Loop`									F1	F2	D1	s	D2	X1	X2	M1	M2	W1	W2	
`LD    R1, 0(R2)`										F1	F2	s	D1	D2	X1	X2	M1	M2	W1	W2

We again have 99 iterations. There are three RAW stalls between the LD and ADDI, and one RAW stall between the DADDI and DSUB. Because of the branch prediction, 98 of those iterations overlap significantly. The total number of cycles is $10 \times 98 + 19 = 999$.

3. [40 = 10 x 4] Virtual machines can lose performance from a number of events, such as the execution of privileged instructions, TLB misses, traps, and I/O. These events are usually handled in system code. Thus, one way of estimating the slowdown when running under a VM is the percentage of application execution time in a system versus user mode. For example, an application spending 10% of its execution in a system mode might slow down by 60% when running on a VM. Figure 2.32 lists the early performance of various system calls under native execution, pure virtualization, and paravirtualization for LMbench using Xen on an Itanium system with times measured in microseconds (courtesy of Matthew Chapman of the University of New South Wales).

a. What types of programs would be expected to have smaller slowdowns when running under VMs?
b. If slowdowns were linear as a function of system time, given the slowdown above, how much slower would a program spending 20% of its execution in system time be expected to run?
c. What is the median slowdown of the system calls in the table above under pure virtualization and paravirtualization/
d. Which functions in the table above have the largest slowdowns? What do you think the cause of this could be?

Benchmark	Native	Pure	Para
Null call	0.04	0.96	0.50
Null I/O	0.27	6.32	2.91
Stat	1.10	10.69	4.14
Open/close	1.99	20.43	7.71
Install sighandler	0.33	7.34	2.89
Handle signal	1.69	19.26	2.36
Fork	56.00	513.00	164.00
Exec	316.00	2084.00	578.00
Fork + exec sh	1451.00	7790.00	2360.00

Solution

a. Programs that do a lot of computation but have small memory working sets and do little I/O or other system calls.

b. The slowdown above was 60% for 10%, so 20% system time would run 120% slower.

c. The median slowdown using pure virtualization is 10.3, while for para virtualization the median slowdown is 3.76.

d. The null call and null I/O call have the largest slowdown. These have no real work to outweigh the virtualization overhead of changing protection levels, so they have the largest slowdowns.