yxu66

Homework11

Thursday, May 1, 2014 2:07 PM

4.10 In this exercise, we examine how resource hazards, control hazards, and Instruction Set Architecture(ISA) design can affect pipelined execution. Problems in this exercise refer to the following fragment of MIPS code:

```
sw r16, 12(r6)
lw r16, 8(r6)
beq r5, r4, Label # Assume r5!=r4
add r5, r1, r4
slt r5, r15, r4
```

4.10.3 [10] <§4.5> Assuming stall-on-branch and no delay slots, what speedup is achieved on this code if branch outcomes are determined in the ID stage, relative to the execution where branch outcomes are determined in the EX stage? To get the speedup, just find the ratio of the numbers of cycles used by the two implementations: ignore the table of pipeline stage latencies.

If direct use of beq with branching determined in EX stage will result in 2 cycle stall for every branch.

Instruction	1	2	3	4	5	6	7	8	9	10	11
sw	If	Id	Ex	Mem	Wb						
Lw		If	Id	Ex	Mem	Wb					
Beq			If	Id	Ex	Mem	Wb				
Add				()	()	If	Id	Ex	Mem	Wb	
Slt							If	Id	EX(forwarding from EX of Add)	Mem	Wb

If direct use of beq with branching determined in ID stage will result in 1 cycle stall for every branch.

Instruction	1	2	3	4	5	6	7	8	9	10	
sw	If	Id	Ex	Mem	Wb						
Lw		If	Id	Ex	Mem	Wb					
Beq			If	Id	Ex	Mem	Wb				
Add				()	If	Id	Ex	Mem	Wb		
Slt						If	Id	Ex(forwarding from EX of Add)	Mem	Wb	

So speedup = (the numbers of cycles used by branch outcomes are determined in the ID stage)/(the numbers of cycles used by branch outcomes are determined in the EX stage) = 1/10 = 0.1 =10%

4.11 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.

4.11.1 [10] <§4.6> 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).
Predict branch taken

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13
And	Wb												
Lw2	Mem	Wb											
Lw3	Ex	Mem	Wb										
Beq	Id	Ex	Mem	wb									
Lw1	IF	ID	EX	MEM	WB								
And		()	IF	ID	EX (forwarding from lw1 MEM phase)	MEM	WB						
Lw2				IF	ID	EX	MEM (forwarding from And MEM phase)	WB					
Lw3						If	id	Ex MEM (forwarding from Lw2 MEM phase)	Mem	Wb			
Beq							()	()	If	Id	Ex (Ex (forwarding from Lw3 MEM phase)	Mem	Wb
Lw1										If	Id	Ex	Mem
And											()	If	id
Lw2													if

4.11.2 [10] <§4.6> How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?
Lw are all five pipeline stages are doing useful work, so it's 3/5 = 0.6 = 60%

4.14 This exercise is intended to help you understand the relationship between delay slots, control hazards, and branch execution in a pipelined processor. In this exercise, we assume that the following MIPS code is executed on a pipelined processor with a 5-stage pipeline, full forwarding, and a predict-taken branch predictor:

```
   lw r2, 0(r1)
label1: beq r2, r0, label2 # not taken once, then taken
   lw r3, 0(r2)
   beq r3, r0, label1 # taken
   add r1, r3, r1
 label2: sw r1, 0(r2)
```

4.14.1 [10] <§4.8> Draw the pipeline execution diagram for this code, assuming there are no delay slots and that branches execute in the EX stage.

Insturction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Lw1	IF	ID	EX	MEM	WB											
Beq1		()	IF	ID	EX	MEM	WB									
Sw				()	If	X	X	X	X							
Lw2						If	Id	Ex	Mem	Wb						
Beq2							()	If	Id	Ex (forwarding from Lw1 MEM phase)	Mem	wb				
Beq1									()	If	Id	Ex	Mem	Wb		
Sw											()	If	Id	Ex	Mem	Wb

4.14.2 [10] <§4.8> Repeat part a, but assume that delay slots are used. In the given code, the instruction that follows the branch is now the delay slot instruction for that branch.

```
  lw r2, 0(r1)
label1: beq r2, r0, label2 # not taken once, then taken
   lw r3, 0(r2)
   beq r3, r0, label1 # taken
   add r1, r3, r1
 label2: sw r1, 0(r2)
```

Change the position of add and beq.

Insturction	1	2	3	4	5	6	7	8	9	10	11	12	13
Lw1	IF	ID	EX	MEM	WB								
Beq1		()	IF	ID	EX	MEM	WB						
Sw				If	X	X	X	X					
Lw2					If	Id	Ex	Mem	Wb				
Beq2						()	If	Id	Ex (forwarding from Lw1 MEM phase)	Mem	Wb		
Beq1								If	Id	Ex	Mem	Wb	
Sw									If	Id	Ex	Mem	Wb

4.14.5 [10] <§4.8> For the given code, what is the speedup achieved by moving branch execution into the ID stage? Explain your answer. In your speedup calculation, assume that the additional comparison in the ID stage does not affect clock cycle time.

(16-13)/16 = 0.1875 = 18.75%

5.2 Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 32-bit memory address references, given as word addresses. 3, 180, 43, 2, 191, 88, 190, 14, 181, 44, 186, 253

5.2.1 [10] <§5.3> For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with 16 one-word blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

Word address	Cache index(%)	Tag(/)	Binary address	Hit/miss
3	3	0000	00000011	miss
180	4	1011	10110100	miss
43	11	0010	00101011	miss
2	2	0000	00000010	miss
191	15	1011	10111111	miss
88	8	0101	1011000	miss
190	14	1011	10111110	miss
14	14	0000	00001110	miss
181	5	1011	10110101	miss
44	12	0010	00101100	miss
186	10	1011	10111010	miss
253	13	1111	11111101	miss

5.2.2 [10] <§5.3> For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with <u>two-word blocks and a total size of 8 blocks</u>. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

Word address	Cache index(%)	Tag(/)	Binary address	Block(ten)	offset	Hit/miss
3	3	0000	00000011	1	1	miss
180	4	1011	10110100	2	0	miss
43	11	0010	00101011	5	1	miss
2	2	0000	00000010	1	0	hit
191	15	1011	10111111	7	1	miss
88	8	0101	1011000	4	0	miss
190	14	1011	10111110	7	0	hit
14	14	0000	00001110	7	0	miss
181	5	1011	10110101	2	1	hit
44	12	0010	00101100	6	0	miss
186	10	1011	10111010	5	0	miss
253	13	1111	11111101	6	1	miss

5.2.4 [15] <§5.3> Calculate the total number of bits required for the cache listed above, assuming a 32-bit address. Given that total size, find the total size of the closest direct-mapped cache with 16-word blocks of equal size or greater. Explain why the second cache, despite its larger data size, might provide slower performance than the first cache.

total size = line number * [valid bit + tag + line]!
total size(8 block) = 2^8 *[1 + 5 + 32]!
total size(16 block) = 2^16* [1 + 4 + 32]!
2^16* [1 + 4 + 32] > 2^8 *[1 + 5 + 32]!
The total number of take line in cache is so small, that it becomes more likely that an access will be a miss. So larger caches have lower miss rates. With lower miss rate, we save time to access main memory that usually take longer time. However, we still need time to literate and looking for data in cache that may cost longer time. Thus, larger data size cache, might provide slower performance than the smaller cache!

5.3 For a direct-mapped cache design with a 32-bit address, the following bits of the address are used to access the cache.

Tag	Index	Offset
31–10	9–5	4–0

5.3.1 [5] <§5.3> What is the cache block size (in words)?
2^5/4 = 8 words

5.3.2 [5] <§5.3> How many entries does the cache have?
2^5=32

5.3.3 [5] <§5.3> What is the ratio between total bits required for such a cache implementation over the data storage bits?

Address											
0	4	16	132	232	160	1024	30	140	3100	180	2180

Starting from power on, the following byte-addressed cache references are recorded.
Ratiao = [32*(32*8+20+1)]/[32*(32*8)] = 277/256

5.3.4 [10] <§5.3> How many blocks are replaced?
Three blocks.

Address	**0**	**4**	**16**	**132**	**232**	**160**	**1024**	**30**	**140**	**3100**	**180**	**2180**
Line id	**0**	**0**	**1**	**8**	**14**	**10**	**0**	**1**	**9**	**1**	**11**	**8**
Hit/miss	**Miss**	**Hit**	**Miss**	**Miss**	**Miss**	**Miss**	**Miss**	**Hit**	**Hit**	**Miss**	**Miss**	**Miss**
Replace	**N**	**N**	**N**	**N**	**N**	**N**	**Y**	**N**	**N**	**Y**	**N**	**y**

5.3.5 [10] <§5.3> What is the hit ratio?
Ratio = 3/12 = 0.25

5.3.6 [20] <§5.3> List the final state of the cache, with each valid entry represented as a record of <index, tag, data>.

<Index,tag,data>:
<0000012, 00012, mem[1024]>
<0000012, 00112, mem[16]>
<0010112, 00002, mem[176]>
<0010002, 00102, mem[2176]>
<0011102, 00002, mem[224]>
<0010102, 00002, mem[160]>