ECE M116C - CS M151B Computer Architecture Systems - UCLA

Quiz 2 (Upload it to Gradescope) - Open book and note

Fall 2022 Don't forget to sign in before you leave.

Notes:

- You are allowed to access any software, tools, online material, notes, and books during the quiz. But you are not allowed to text, chat, or post anything during the quiz.
- If you have any questions, the preferred method of communication is to DM me on Campuswire, but you can also raise your hand.
- You have 80 minutes to complete the quiz (until 1:30 PM).
- You can write your answers on a laptop, tablet, or paper. If your internet is currently not working, probably paper is a better option.
- Once you are finished, please upload your answers to Gradescope. You have an additional 10 minutes to upload your files. If you upload it after 1:30 PM, on Gradescope, it will show it as late, but you won't lose any points. Note that you won't be allowed to upload anything after 1:40 PM.
- If your internet doesn't work after a few tries, you can hand in your paper answers. If you wrote down your results in PDF, just take a screenshot and timestamp your file. We will let you upload it later.
- Good luck! :)

Q1 [30 points] **Branch Prediction:** *'oh wait, what was RISC-V?'*

Assume the assembly code below:

```
          ADDI x1, x0, 2
BEGIN:    BEQ x0, x1, END
          ADDI x1, x1, -1
          JAL x0, BEGIN
END:      XOR x0, x0, x0
```

Assume a single-issue pipelined 32-bit RISC-V in-order processor. Assume a 4-entry BTB where each entry has a BHT (2-bit bimodal branch predictor). BTB is initially filled only with the BEQ instruction and its BHT is in weak not taken (01) mode. Control-flow instructions are resolved in DE stage and BTB and BHT will be correctly updated for all control-flow instructions. We have forwarding.

a. Draw the timeline (by putting `f,d,e,m,w`) for each instruction. You don't need to show the forwarding. If any instructions needed to be flushed and replaced by bubbles show it with b (e.g., `eb`, `mb`, and `wb` for bubbles in each stage). Show the table until XOR instruction is in writeback.

Correct order of execution, assuming perfect accuracy:
addi (1st)
beq
addi (2nd)
jal
beq
addi (2nd)
jal
beq
xor

1st Loop (From addi -> xor, cycles 1-9):
At the first run through the beq, our BHT state is weak not taken (01). So we execute addi (2nd) right away, which is correct. Our BHT state goes to strong not taken (00).

BHT transition: Weak Not Taken (01) -> Strong Not Taken (00)

When we reach the jal for the first time, we don't know where its target is, so we need to execute the instruction below it despite the fact that this branch is always taken (unconditional).

We resolve the branch at the decode stage, and we can flush the xor.

2nd Loop (From beq -> jal, cycles 6-12):
We then execute the beq. Our prediction is strong not taken, so we execute addi (2nd), which is correct.

BHT transition: Strong Not Taken (00) -> Strong Not Taken (00)

At the jal, we know the target, and it is always taken (unconditional), so we go to beq without any mistakes or flushes.

3rd Loop (From beq -> xor, cycles 9-15):

At the beq for the third time, our prediction is strong not taken, so we execute addi (2nd), which is incorrect based on the correct execution of code. So we need to flush the addi (2nd). We then execute the xor.

BHT Transition: Strong Not Taken (00) -> Weak Not Taken (01)

BHT Results:

Weak Not Taken (01) -> Strong Not Taken (00) -> Strong Not Taken (00) -> Weak Not Taken (01)

Instr	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
addi	f	d	e	m	w										
beq		f	d	e	m	w									
addi			f	d	e	m	w								
jal				f	d	e	m	w							
xor					fb	db	eb	mb	wb						
beq						f	d	e	m	w					
addi							f	d	e	m	w				
jal								f	d	e	m	w			
beq									f	d	e	m	w		
addi										fb	db	eb	mb	wb	
xor											f	d	e	m	w

b. What is the total number of cycles?

15 cycles

Q2 [40 points] **Out-of-order execution:** *'Ok fine, you don't need to fill THAT slide!'*

Each question is independent of the rest. The tables are pre-filled and the values are for the beginning of the cycle.

a. ***Rename*** the following instructions using the attached tables: (no need to update the tables itself.)

`SW x1, x2, 100 → Renamed:` SW p1, p9, 100

`ADD x3, x3, x4 → Renamed:` ADD p10, p3, p8

RAT	
A-reg	PMap
x1	1
x2	9(7)
x3	3
x4	8(4)
x5	6(5)

"Free" Pool
p10

SW: the source registers are x1 and x2, so we rename them to their current mappings in the RAT, which are x1 -> p1, and x2 -> p9.

ADD: the source registers are x3 and x4. We rename them to their current mapping in the RAT, which are x3 -> p3, and x4 -> p8. The destination register is x3, so we need to use a free register from the "Free" Pool, which is p10. So x3 -> p10.

b. ***Dispatch*** the following instructions in the reservation table (update only the reservation station table).

```
LW p3, p2, 200
ADDI p6, p3, 2
```

FU R?	
ALU	1
MEM	1

R?	
1	1
2	0
3	1
4	1
5	1
6	1
7	1
8	0
9	1
10	1

use	OP	Dest Reg	Src Reg 1	Src 1 Ready	Src Reg 2	Src 2 Ready	FU	ROB#
1	XOR	p2	p1	y	P8	n	0	4
0								
0								
0								

use	**OP**	**Dest**	**Src Reg**	**Src 1**	**Src Reg**	**Src 2**	**FU**	**ROB#**

		Reg	1	Ready	2	Ready		
1	XOR	p2	p1	y	p8	n	0	4
1	LW	p3	p2	n	p0	y	1	5
1	ADDI	p6	p3	n	p0	y	0	6

For LW: p3 is the Dest Reg. p2 is the Src Reg 1, which is not ready (XOR makes it not ready). The Src Reg 2 is blank (or p0). This is always ready. The FU is 1 (MEM), and the ROB# is 5.

For ADDI: p6 is the Dest Reg. p3 is the Src Reg 1, which is not ready (LW makes it not ready). The Src Reg 2 is blank (or p0). This is always ready. The FU is 0 (ALU), and the ROB# is 6.

c. Using the given tables, ***issue*** up to 3 instructions from the reservation station. Assume that the following two instructions are moving into the Complete stage starting this cycle (they were in FU0 and FU2, respectively. FU1 is also occupied).

```
LW  p6, p0, 100   (this was in FU 2 and is now completed.)
ADD p8, p0, p1    (this was in FU 0 and is now completed.)
```

Considering the completed instructions above and the table below, show the reservation station AFTER the instructions are issued.
(Change the use bit of issued instructions to zero).

FU	R?
ALU-1	0
ALU-2	0
MEM	0

use	OP	Dest Reg	Src Reg 1	Src 1 Ready	Src Reg 2	Src 2 Ready	FU	ROB#
1	ADD	p3	p1	y	p7	n	0	0
1	SUB	p4	p2	y	p1	y	1	1
1	LW	p5	p6	n	p1	y	2	2
1	ORI	p9	p8	n	p0	y	0	3

As the LW completes, p6 is now ready. As the ADD completes, p8 is now ready.

Looking at the Reservation Station, the ADD's Src Reg 2 (p7) is still not ready, so we can't issue it.

For the SUB, p2 is ready, p1 is ready, FU 1 is not ready, so we can't issue it.

For the LW, p6 is now ready (due to the previous LW completing). p1 is also ready. FU2 (MEM) is ready as well, so we issue the LW with ROB# = 2.

For the ORI, p8 is now ready (due to the previous ADD completing). p0 is always ready. FU0 (ALU-1) is ready as well, so we issue the ORI with ROB# = 3.

Instructions to Issue: LW, ORI

Q3 [30 points] **Cache:** '*You don't need to convert to HEX either ;)*'
Assume we have a direct-mapped cache with 1 block per line and 16 sets. We now change this cache to a 2-way set-associative cache with 4 blocks per line and 16 sets. Comparing the new design with the old, fill the table below by putting increase (I), decrease (D), unchanged (U), or unknown (K). For each row, briefly explain why.

Metric	Change (I, D, U, K)	Why?
Cache size	I	Direct-mapped: total blocks = 1 block per line x 16 sets = 16 blocks. 2-way set-associative: Each set has 2 ways. If there are 4 blocks per line (where a line = a way), then there are 2 ways x 4 blocks per line = 8 blocks in a set. Then, 8 blocks in a set x 16 sets = 128 blocks. 128 blocks > 16 blocks

		Alternatively, we see that we have the same number of sets, with each set having more lines, and more blocks per line.
Miss rate	D	Increasing associativity generally decreases miss rate, as blocks have an extra degree of freedom as to which way they enter (we increased associativity going from one way in 16 sets to 2 ways in 16 sets). We also increased the block size, which makes use of spatial locality and decreases the miss rate. Increasing cache size also generally decreases miss rate. Since we kept the number of sets the same as before while increasing associativity at the same time, this should reduce the number of conflict misses.
AAT	D or K	Increasing associativity generally increases the hit time, as we need to do potentially multiple tag checks, while reducing miss rate. Increasing the cache size generally increases the hit time as there are more lines (and higher delay). It however, reduces the miss rate since bigger cache is better. The decrease in miss rate is more likely to be dominant so we expect AAT to decrease. However, without concrete numbers it is hard to say. We accept both D or K for this answer.
Compulsory miss	D or U	Increasing the line size (4 blocks instead of 1 block), should decrease compulsory misses as we make use of more spatial locality. We also accept U answers if you have argued that compulsory miss is not relevant to cache size and/or associativity.

Q4 [5 points] The most important question: *'This is it! Time to eat!'*

The best ice cream you've ever eaten was? (location and flavor)
(I still remember mine so vividly!)

Justin: Graeter's ice cream found in the midwest. My family really likes the black raspberry chip flavor.

Nader: Mine is milk chocolate at this Ice cream place called Nicecream near the place I used to live in Washington DC.

Shyandeep: I would say Butterscotch Ice Cream, it is a flavor quite popular in india made of brown sugar and butter! There is this chain back home called kwality walls, they had awesome Butterscotch ice cream!