CS330 Architecture and Organization
Assignment Chapter 4

Your Name Here

Problem 7 — (Sections 4.5–4.7) Consider the following code segment.

```
i0:    or $t2, $t0, $t1
i1:    or $t3, $t1, $t2
i2:    or $t4, $t2, $t3
```

We want to execute this stream of instructions on a 5-stage pipelined MIPS processor.

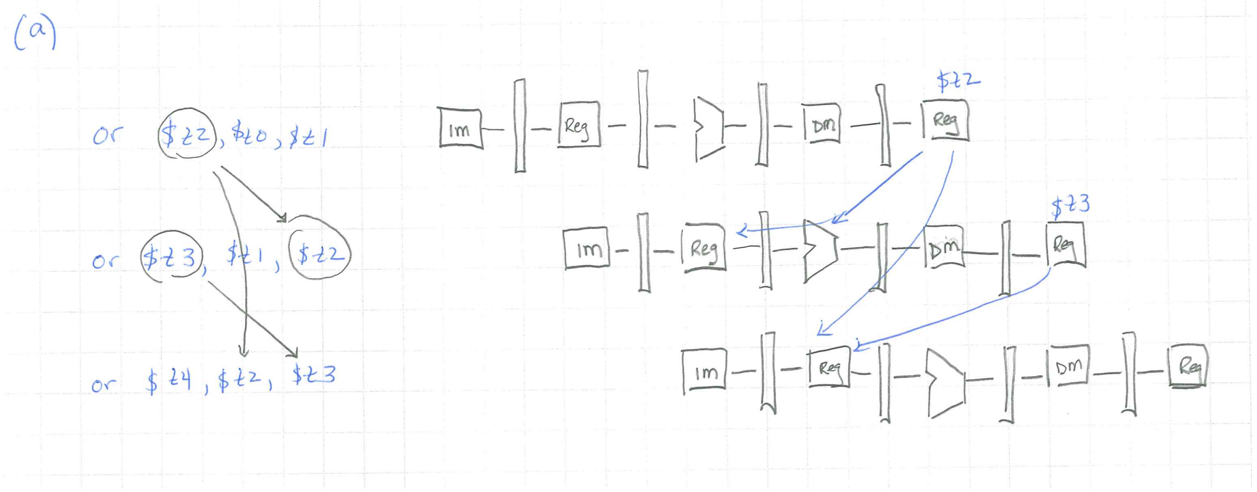
(a) (5 points) Assume the processor does not implement forwarding. Draw a stage diagram similar to Figure 4.34 on p. 288 showing the execution of each stage of the three instructions, assuming the processor fetches the next instruction each clock cycle. Identify all the data hazards.

(b) (5 points) Assume the processor does not implement forwarding. Draw a stage diagram, only this time insert nops between regular instructions, as necessary, in order to stall the pipeline long enough to ensure that the instructions execute correctly.

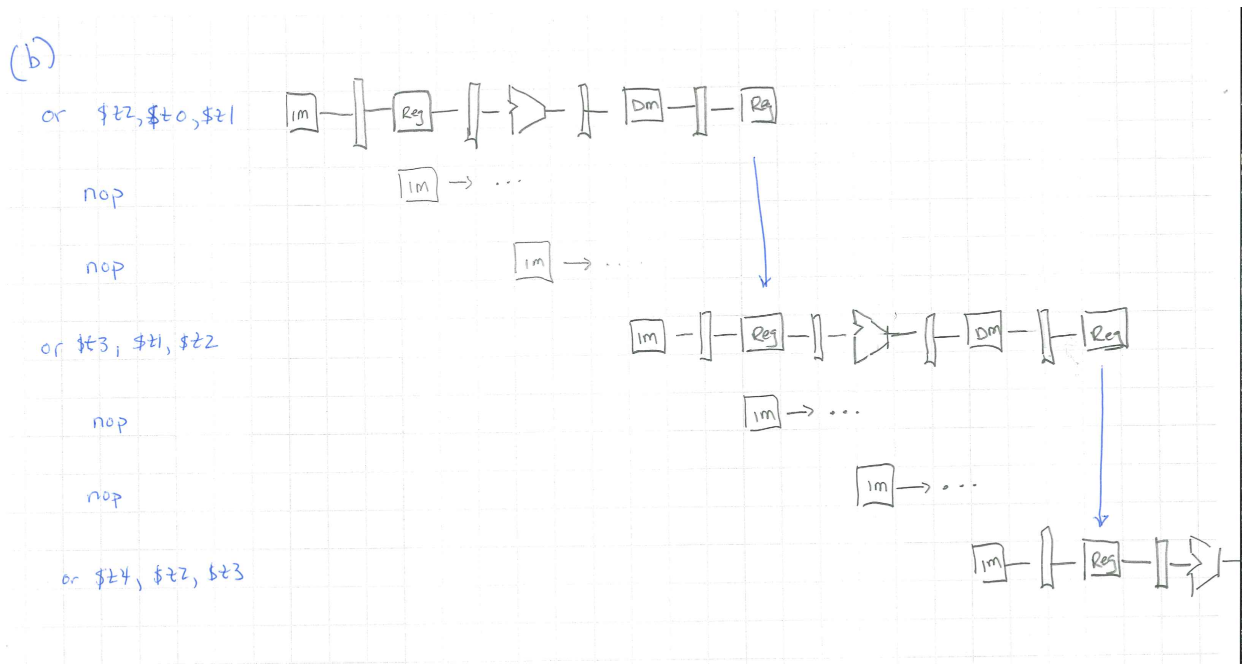
(c) (5 points) In ALU-to-ALU forwarding, the output of the ALU for a particular instruction is available as an input of the ALU of the following instruction at the beginning of the next clock cycle. Draw another diagram like you did in question 1 showing the execution of each stage of the pipeline, this time assuming that the processor uses ALU-to-ALU forwarding. Show the forwarding by drawing a line from the ALU output of one instruction to the ALU input of the instruction which needs it.

Answer:

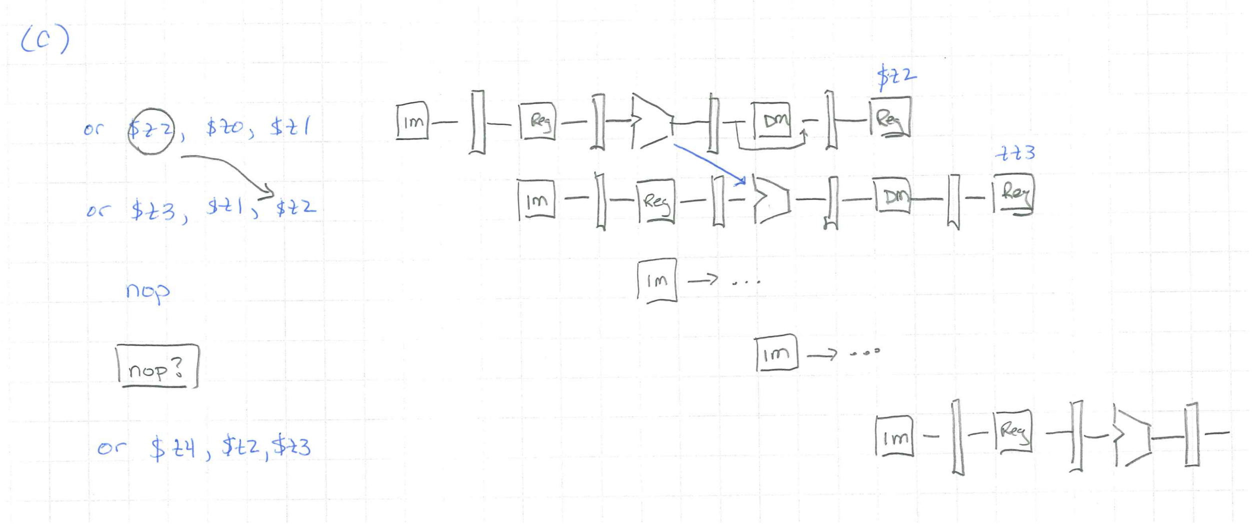
(a) Register $t2 is a hazard in the second instruction and the third instruction. It has not yet been written back when the value is needed again. Register $t3 is a hazard in the third instruction for the same reason.



(b) Without forwarding, it requires two nop instructions of separation to allow register writeback and register read to coincide in the same clock cycle.



(c) This is the trickiest situation. Register forwarding works between the first and second instruction to forward the value of register $t2. However, we can't forward $t2 on to the third instruction because the ALU has had a new output value. That leaves us in the situation of needing a nop to give time for register writeback to occur on $t2. However, even though a nop doesn't create a value, it still consumes the output of the ALU which now prevents $t3 from being forwarded as well. Thus, a second nop ends up being required to allow for writeback of $t3.



Problem 8 — (Section 4.8) Assume a particular branch in code is encountered several times and the taken/not-taken behavior at that branch turns out to be: T, NT, T, T, NT

(a) (2 points) What is the accuracy of the always-taken predictor for this sequence of branch outcomes?

(b) (2 points) What is the accuracy of the never-taken predictor for this sequence of branch outcomes?

(c) (3 points) Assume we use a 1-bit predictor, where 0 means the previous branch was not taken and 1 means it was. Assuming the state is initialized to 0, what is the accuracy of the of a 1-bit predictor for this sequence of branch outcomes?

(d) (3 points) Assume we use a 1-bit predictor initialized to 0, but this block of code is revisited 10 times, so the sequence at the branch becomes { T, NT, T, T, NT } $\times 10$. What is the accuracy now?

(e) (3 points) Assume we use a 2-bit predictor, where 00 means the previous branch was not taken twice (or more) and 11 means it was taken twice (or more). Assuming the state is initialized to 00, what is the accuracy of the of a 2-bit predictor for this sequence of branch outcomes?

(f) (3 points) Assume we use a 2-bit predictor initialized to 00, but this block of code is revisited 10 times, so the sequence at the branch becomes { T, NT, T, T, NT } $\times 10$. What is the accuracy now?

Answer:

(a) The always-taken predictor will be right for the three Taken branches and wrong for the two Not-taken branches for an accuracy of $3/5 = 60\%$.

(b) The never-taken predictor will be right for the two Not-taken branches and wrong for the three Taken branches for an accuracy of $2/5 = 40\%$.

(c) We need to track the 1-bit predictor to answer this:

State	0	1	0	1	1
Prediction	NT	T	NT	T	T
Branch	T	NT	T	T	NT
Correct				x	

So, the accuracy is $1/5 = 20\%$.

(d) If we consider the state of the 1-bit predictor the second time this block of code is entered, it will again start in state 0 (because the final branch was Not-taken the last time we were in this code). So, the table we used in part (c) is appropriate for each of the remaining 9 times this code section is encountered. Each round the 1-bit predictor is correct on a single Taken branch and incorrect on the other four. So, the accuracy over all 50 branches is $10/50 = 20\%$.

(e) We need to track the 2-bit predictor to answer this:

State	00	01	00	01	10
Prediction	NT	NT	NT	NT	T
Branch	T	NT	T	T	NT
Correct		x			

So, the accuracy is $1/5 = 20\%$.

(f) The second time through this block of code is different than in part (e) because the initial state of the predictor is now 01 (having dropped back from 10 when the final branch was not taken in the previous run. If we redo the table we get these results:

State	01	10	01	10	11
Prediction	NT	T	NT	T	T
Branch	T	NT	T	T	NT
Correct				x	

The third time through this block of code is again different, because the inital state of the predictor is now 10 (having dropped back from 11 when the previous branch was Not-taken). If we redo the table we get these results:

State	10	11	10	11	11
Prediction	T	T	T	T	T
Branch	T	NT	T	T	NT
Correct	x		x	x	

Every time after this will be the same. So the final number of accurate predictions is $1 + 1 + 3 + \cdots + 3 = 1 + 1 + 3(8) = 26$ for an accuracy of $26/50 = 52\%$.