CSE 560 – Practice Problem Set 3 Solution

Three of these problems come from Hennesy & Patterson's *Computer Architecture: A Quantitative Approach*, 3rd edition.

1. I said in class that one pipeline stage name was poorly chosen. Which one is poorly chosen, and what would you propose as an alternative name?

   I don't care for the name of the decode stage. Yes, that is when we are decoding the meaning of the instruction, but it seems to me that the more important thing (i.e., more computationally difficult thing) that is going on in that stage is reading the register file.

   To be sure, deeper pipelines regularly separate instruction decode from register file read, making those two things two different pipeline stages.

2. This exercise asks how well hardware can find and exploit instruction level parallelism (i.e., pipelining). Consider the following four RISC machine code fragments each containing two instructions:

   i. addi r1 ← r1, #4
      load r2, 7(r1)

   ii. add r3 ← r1, r2
       store r2, 7(r1)

   iii. breq r1, place
        store r1, 7(r1)

   iv. store r3, 17(r10)
       load r2, 12(r8)

   (a) For each code fragment (i) to (iv) identify each dependence that exists or that may exist (a fragment may have no dependencies).

   (b) For each code fragment, indicate whether data forwarding is sufficient to resolve the dependence or if stall cycles are required. Indicate the number of stall cycles.

Code Fragment	Dependence (a)	Resolution (b)
i	True dependence on r1	Data forwarding sufficient
ii	No dependence	None required
iii	No data dependence (but control dependence)	Stall cycles to resolve branch depends upon microarchitecture
iv	Potential dependence on memory	Will execute in order in M stage

3. Consider the following RISC assembly code.

```
load    r1,45(r2)        (1)
add     r7 ← r1, r5      (2)
sub     r8 ← r1, r6      (3)
or      r9 ← r5, r1      (4)
brneq   r7, target       (5)
add     r10 ← r8, r5     (6)
xor     r2 ← r3, r4      (7)
```

(a) Identify each dependence; list the two instructions involved; identify which instruction is dependent; and, if there is one, name the storage location involved (register or memory).

- Inst (2) is dependent on inst (1) through r1
- Inst (3) is dependent on inst (1) through r1
- Inst (4) is dependent on inst (1) through r1
- Inst (5) is dependent on inst (2) through r7
- Inst (6) is dependent on inst (3) through r8
- Inst (6) is dependent on inst (5) through control
- Inst (7) is dependent on inst (5) through control

(b) Using the 5-stage pipeline from class, which of the dependences that you found in part (a) become hazards and which do not.

The first dependency, (2) dependent on (1), is a load-use dependency that will necessitate a one clock cycle delay so that the results of the load (available after stage M) are available to the execution of the add (in stage X). None of the rest are hazards.

(c) Draw a pipeline diagram for the sequence, including stalls needed to rectify the hazards.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
(1)	F	D	X	M	W											
(2)		F	D	d*	X	M	W									
(3)			F	p*	D	X	M	W								
(4)					F	D	X	M	W							
(5)						F	D	X	M	W						
(6)							F	D	s*							
(7)								F	s*							

In the above diagram, I assumed that the branch was predicted NT and was taken, just as an example to show the insertion of stall cycles.

4. Increasing the size of a branch prediction buffer means that it is less likely that two branches in a program will share the same predictor. A single predictor predicting a single branch instruction is generally more accurate than is that same predictor serving more than one branch instruction.
   (a) List a sequence of branch taken and not taken actions to show a simple example of 1-bit predictor sharing that reduces misprediction rate.

   Consider two branches, B1 and B2, that are executed alternately. During the execution of the program, B1 and B2 each alternate taken/not taken. If they each had a 1-bit predictor, each would always be mispredicted.

   In the table below, columns labeled P show the value of a 1-bit predictor shared by B1 and B2. Columns labeled B1 and B2 show the actions of the branches (each alternating taken/not taken). Time increases to the right. T stands for taken, NT for not taken. The predictor is initialized to NT.

	P	B1	P	B2	P	B1	P	B2	P	B1	P	B2	P	B1	P	B2
	NT	T	T	NT	NT	NT	NT	T	T	T	T	NT	NT	NT	NT	T
Correct?		no		no		yes		no		yes		no		yes		No

   Because a single predictor is shared, prediction accuracy improves from 0% to 50%.

   (b) List a sequence of branch taken and not taken actions that show a simple example of how sharing a 1-bit predictor increases misprediction.

   Here, B1 is always taken, B2 is always not taken, and they are interleaved as in (a).

	P	B1	P	B2	P	B1	P	B2	P	B1	P	B2	P	B1	P	B2
	NT	T	T	NT	NT	T	T	NT	NT	T	T	NT	NT	T	T	NT
Correct?		no		no		no		no		no		no		No		No

   If each had a 1-bit predictor, each would be correctly predicted after the initial startup transient. Because a single predictor is shared, accuracy is 0%.

   (c) Discuss why the sharing of branch predictors can be expected to increase mispredictions for the long instruction sequences of actual programs.

   If a predictor is being shared by a set of branch instructions, then over the course of program execution set membership will likely change (i.e., which specific branches are being shared). When a new branch enters the set or an old one leaves the set, the branch action history represented by the state of the predictor is unlikely to predict the set behavior as well as it did old set behavior, which had some time to affect predictor state. The transient intervals following set changes likely will reduce long-term accuracy.