CS M151B - Homework 5

Kuruvilla Cherian, Jahan
UID: 104436427
Section 1B

February 25, 2017

Problem 4.13

1

Using the software solution of no-ops we have to first determine the key data hazard points in the set of instructions. We can see three key data hazards:

1.) The hazard for the first **LW** on **r5** from the ADD instruction. This means that the load word's Decode needs to align withe Write back of the Add instruction so that it can use the updated **r5**.
2.) The hazard wherein the **OR** depends on the **r3** from the first **LW**. This means that the OR's Decode stage requires the written back value from the second load word.
3.) The **SW** requires it such that **r3** be written back by the **OR** instruction.

This then gives us the following pipeline diagram, with a total of **14** cycles (without forwarding) and the following no-ops. Note that the highlights represent the lineup of data dependencies.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
ADD r5, r2, r1	IF	ID	EX	M	WB									
NOP		IF	ID	EX	M	WB								
NOP			IF	ID	EX	M	WB							
LW r3, 4(r5)				IF	ID	EX	M	WB						
LW r2, 0(r2)					IF	ID	EX	M	WB					
NOP						IF	ID	EX	M	WB				
OR r3, r5, r3							IF	ID	EX	M	WB			
NOP								IF	ID	EX	M	WB		
NOP									IF	ID	EX	M	WB	
SW r3, 0(r5)										IF	ID	EX	M	WB

Figure 1: Pipeline diagram with no-ops

2

With forwarding, we see that from Figure 2, we only have a total of **9** cycles. We only have these number of cycles because of the fact that in this particular question, there is no need for any stalls and so all the data can be forwarded such that the overall pipeline remains like a generic pipeline. Note that the red arrows show the forwarding and the colored highlights show the linked dependencies.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
ADD r5, r2, r1	IF	ID	EX	M	WB				
LW r3, 4(r5)		IF	ID	EX	M	WB			
LW r2, 0(r2)			IF	ID	EX	M	WB		
OR r3, r5, r3				IF	ID	EX	M	WB	
SW r3, 0(r5)					IF	ID	EX	M	WB

Figure 2: Pipeline diagram with forwarding

Problem 4.14.1

1

In this section we assume the branch gets resolved in the **EX** stage of the five staged pipeline. The highlights in Figure 3 represent the periods of stalls in order to either conflicting pipeline instructions or to wait for the result from the instruction before it. The forwarding happens between the **LW** and **BEQ** instructions, and because the load word is directly preceding the branch we have to stall to wait for two cycles so that the value from the MEM stage of the LW can be forwarded to the ID section of the BEQ (wherein the comparator lies). Note that the next instruction is essentially acting as a no-op for the instruction after the store word in label2.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19
L1: LW r2, 0(r1)	IF	ID	EX	M	WB														
BEQ r2, r0, L2		IF	ID	ID	ID	EX	M	WB											
SW r1, 0(r2)			IF	IF	IF	ID	EX	M	WB										
Next$_1$						IF	ID	EX	M	WB									
LW r3, 0(r2)							IF	ID	EX	M	WB								
BEQ r3, r0, L1								IF	ID	ID	ID	EX	M	WB					
LW r2, 0(r1)									IF	IF	IF	ID	EX	M	WB				
BEQ r2, r0, L2												IF	ID	ID	ID	EX	M	WB	
SW r1, 0(r2)													IF	IF	IF	ID	EX	M	WB

Figure 3: Pipeline diagram with forwarding and stalling for branching resolved at EX

2

In this section we assume the branch gets resolved in the **ID** stage of the five staged pipeline. This basically just means that we shift everything a cycle earlier since the branch gets resolved one stage

earlier.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18
L1: LW r2, 0(r1)	IF	ID	EX	M	WB													
BEQ r2, r0, L2		IF	ID	ID	ID	EX	M	WB										
SW r1, 0(r2)			IF	IF	IF	ID	EX	M	WB									
LW r3, 0(r2)						IF	ID	EX	M	WB								
BEQ r3, r0, L1							IF	ID	ID	ID	EX	M	WB					
LW r2, 0(r1)								IF	IF	IF	ID	EX	M	WB				
BEQ r2, r0, L2											IF	ID	ID	ID	EX	M	WB	
SW r1, 0(r2)												IF	IF	IF	ID	EX	M	WB

Figure 4: Pipeline diagram with forwarding and stalling for branching resolved at ID

Problem 4.16

1

We essentially match up each prediction from the original that is T, NT, T, T, NT with either all T's or all NT's. We see that:

1.) Always taken - 3/5 match up leading to an accuracy of **60%**.
2.) Always-not-taken - 2/5 match up leading to an accuracy of **40%**.

2

We begin at the lower left corner of the finite state machine (FSM) for the two bit branch predictor.

We begin by starting at NT, which then based on reality branches with a T, taking us to another NT, which then based on reality branches with an NT going back to the bottom left NT, which then branches on the reality of T to go back to NT. Thus considering only the first 4 branches, we have the pattern of NT, NT, NT and NT. This when compared to the original pattern shows our prediction as being right only 1/5 times leading to an accuracy of **25%**.

3

We walk through around three cycles to see if our predictor ever reaches a state of an infinite loop. This happens after about 3 iterations of walking through, to show that the predictor will always switch between the upper layer of the FSM oscillating between a Strongly biased Predict Taken and a weakly biased Predict Taken. This is highlighted in Figure 5.

Based on such a branch prediction cycle, we see that in the third iteration, we reach an always taken stage such that 3/5 predictions are correct and so with infinite cycles the predictor approaches a **60%** accuracy.

Start Cycle:

Real Branches	T	NT	T	T	NT
Predictions	NT	NT	NT	NT	T

Second Cycle:

Real Branches	T	NT	T	T	NT
Predictions	NT	T	NT	T	T

Third Cycle:

Real Branches	T	NT	T	T	NT
Predictions	T	T	T	T	T

Figure 5: Branch predictor accuracy for infinite cycles checked