Eric Tashji

Homework Assignment 3

Exercise 14.7.3: Consider the following instruction mix:

R-type	I-Type	LDUR	STUR	CBZ	B
24%	28%	25%	10%	11%	2%

14.17.3(a): What fraction of all instructions use data memory?

Solution: Only the STUR and LDUR instructions use the data memory. 25% (LDUR) + 10% (STUR) = 35%. Therefore, 35% of the instructions use data memory. This would be 35/100 or 7/20 instructions.

14.17.3(b): What fraction of all instructions use instruction memory? (don't all of them?)

Solution: 100% of instructions use the instruction memory, as all instructions must be retrieved from the instruction memory.

14.17.3(c): What fraction of all instructions use the sign extend?

Solution: The I-type, LDUR, STUR, and CBZ type instructions use sign-extension. Therefore, 74% of the instructions use sign extension, as 28 + 25 + 10 + 11 = 74%. Therefore, 74/100 or 37/50 of the instructions use sign extension.

Exercise 4.17.4: When silicon chips are fabricated, defects in materials and manufacturing errors can result in defective circuits. A very common defect is for one signal wire to get "broken" and always register a logical 0. This is often called a "stuck-at-0" fault.

14.17.4(a): Which instructions fail to operate correctly if the **MemtoReg** wire is stuck at 0?

Solution: The STUR and LDUR instructions would fail to operate correctly if the MemtoReg wire is stuck at 0 because the MemtoReg wire needs to be at 1 for these operations to work.

14.17.4(b): Which instructions fail to operate correctly if the **ALUSrc** wire is stuck at 0?

Solution: The LDUR and STUR instructions, as well as any branch instructions, fail to operate correctly if the ALUSrc wire is stuck at 0.

14.17.4(c): Which instructions fail to operate correctly if the **Reg2Loc** wire is stuck at 0?

Solution: The LDUR, STUR, and branch-type instructions would fail to operate if the Reg2Loc wire is stuck at 0.

Exercise 14.17.19: Assume that **X1** is initialized to 11 and **X2** is initialized to 22. Suppose you executed the code below on a version of the pipeline from COD Section 4.5 (An overview of pipelining) that does not handle data hazards (i.e. the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). What would the final values of register **X5** be? Assume the register file is written at the beginning of the cycle and read at the end of a cycle. Therefore, an ID stage will return the results of a WB state occurring during the same cycle.

ADDI X1, X2, #5
ADD X3, X1, X2
ADDI X4, X1, #15
ADD X5, X1, X1

Solution: The final value of **X5** is **54**. This is because **X5** is the sum of **X1** added to itself, and at this point, **X1** is equal to **27**. Although **X1** began as being equal to **11**, it will become equal to **27** as a result of the initial **ADDI** instruction. Furthermore, X1 will be updated at the beginning of clock cycle 5, which is when the **ADD X5, X1, X1** instruction will access **X1**. Therefore, it creates a forward dependency (instead of backwards), which is not a data hazard and X1 will be updated to 27 by the time the **ADD X5, X1, X1** instruction will access it.

Eric Tashji

Exercise 4.17.30

Instruction 1	Instruction 2
CBZ X1, LABEL	LDUR X1, [X2, #0]

14.17.30(a): Which exceptions can each of these instructions trigger? For each of these exceptions, specify the pipeline stage in which it is detected. (Which exceptions are possible?)

Solution: Instruction 1 could throw the hardware malfunction or undefined instruction exceptions. The hardware malfunction exception would occur if there was a defect in the hardware, and would be thrown in the EX pipeline stage. The undefined instruction would occur if LABEL was not present in the instruction set, or the computer could not accurately read the instruction. This would occur in the IF pipeline stage.

Instruction 2 could throw the hardware malfunction exception, which would occur if there was a defect in the hardware. This would occur during the MEM stage of the pipeline, data memory is accessed.

Exercise A: Consider the code in exercise 4.17.25.

1. **LDUR X10, [X1, #0]**
2. **LDUR X11, [X1, #8]**
3. **ADD X12, X10, X11**
4. **SUBI X1, X1, #16**
5. **CBNZ X12, LOOP**

A.1: Assume that perfect branch prediction is used (no stall due to control hazard), that has full forwarding support, and that branches are resolved in the ID stage. Show a pipeline execution time diagram for the first two iterations of this loop.

Solution:

Clock cycle 1 contains the IF stage of the first instruction.

Clock cycle 2 contains the ID stage of the first instruction and the IF stage of the second instruction.

Clock cycle 3 contains the EX stage of instruction 1, ID stage of instruction 2, and IF stage of instruction 3.

Eric Tashji

Clock cycle 4 contains the MEM stage of instruction 1, the EX stage of instruction 2, and a stall for instruction 3's ID stage. Instruction 3 requires a stall because the instruction needs to wait for X11 and X11 to be written to by the load orders of instructions 1 and 2. Because the previous instructions are load instructions loading into a register that the third instruction has to use, forwarding cannot be used to prevent the stall, so a stall must be used.

Clock cycle 5 contains the WB stage of instruction 1, the MEM stage of instruction 2, the ID stage of instruction 3, and the IF stage of instruction 4.

Clock cycle 6 contains the WB stage of instruction 2, the EX stage of instruction 3, the ID stage of instruction 4, and the IF stage of instruction 5.

Clock cycle 7 contains the MEM stage of instruction 3, the EX stage of instruction 4, the ID stage of instruction 5, and the IF stage of instruction 1. Note that instruction 5's ID stage relies on instruction 3's WB stage, which will not occur until the next clock cycle. Therefore, instruction 3 will forward from its MEM stage to instruction 5's EX stage. Furthermore, instruction 1's IF stage is in this cycle because the branching was resolved in the ID stage of instruction 5.

Clock cycle 8 contains the WB stage of instruction 3, the MEM stage of instruction 4, the EX stage of instruction 5, the ID stage of instruction 1, and instruction 2's IF stage. Note that instruction 1's ID stage relies on completion of instruction 3's WB stage, so instruction 3 is going to forward the data from its MEM stage to instruction 1's EX stage in the next clock cycle.

Clock cycle 9 contains instruction 4's WB stage, instruction 5's MEM stage, instruction 1's EX stage, instruction 2's ID stage, and instruction 3's IF stage. Note that instruction 2's ID stage relies on instruction 3's WB stage, but that stage was completed this clock cycle, so no forwarding is required.

Clock cycle 10 contains instruction 5's WB stage, instruction 1's MEM stage, instruction 2's EX stage, and a stall for instruction 3's ID stage. Note that a stall is necessary because instruction 3 relies on registers that are being written to by instructions 1 and 2 in their WB stage. Forwarding cannot be used because the register is written to by a load instruction.

Clock cycle 11 contains instruction 1's WB stage, instruction 2's MEM stage, instruction 3's ID stage, and instruction 4's IF stage.

Clock cycle 12 contains instruction 2's WB stage, instruction 3's EX stage, instruction 4's ID stage, and instruction 5's IF stage. Note that instruction 4's ID stage relies on the previous iteration of instruction 4's WB stage,

Eric Tashji

Clock cycle 13 contains the MEM stage of instruction 3, the EX stage of instruction 4, and the ID stage of instruction 5. Note that instruction 5's ID stage relies on instruction 3's WB stage, which will not occur until clock cycle 14. Therefore, instruction 3 will forward the data from its MEM stage to instruction 5's EX stage in the next clock cycle. Also note that no instructions begin here because the branch instruction is (correctly) asserting that the loop will not continue.

Clock cycle 14 contains instruction 3's WB stage, instruction 4's MEM stage, and instruction 5's EX stage.

Clock cycle 16 contains instruction 5's WB stage.

	Clock Cycle 1	Clock Cycle 2	Clock Cycle 3	Clock Cycle 4	Clock Cycle 5	Clock Cycle 6	Clock Cycle 7	Clock Cycle 8	Clock Cycle 9	Clock Cycle 10	Clock Cycle 11	Clock Cycle 12	Clock Cycle 13	Clock Cycle 14	Clock Cycle 15	Clock Cycle 16
LDUR X10, [X1, #0]	IF	ID	EX	MEM	WB											
LDUR X11, [X1, #8]		IF	ID	EX	MEM	WB										
ADD X12, X10, X11			IF	Stall	ID	EX	MEM	WB								
SUBI X1, X1, #16					IF	ID	EX	MEM	WB							
CBNZ X12, LOOP						IF	ID	EX	MEM	WB						
LDUR X10, [X1, #0]							IF	ID	EX	MEM	WB					
LDUR X11, [X1, #8]								IF	ID	EX	MEM	WB				
ADD X12, X10, X11									IF	Stall	ID	EX	MEM	WB		
SUBI X1, X1, #16											IF	ID	EX	MEM	WB	
CBNZ X12, LOOP												IF	ID	EX	MEM	WB

Eric Tashji

Answer to A.2 on Next Page

Eric Tashji

A.2: Assume branches are predicted as not-taken (but actually taken) the first time and predicted taken (but actually not-taken) the second time. Assume full forwarding support and that branches are resolved in EX stage. Show the execution time diagram for the first two iterations of this loop.

Solution:

Clock cycle 1 contains instruction 1's IF stage.

Clock cycle 2 contains instruction 1's ID stage and instruction 2's IF stage.

Clock cycle 3 contains instruction 1's EX stage, instruction 2's ID stage, and instruction 3's IF stage.

Clock cycle 4 contains instruction 1's MEM stage, instruction 2's EX stage, and a stall for instruction 3's ID stage. This is because instruction 3 relies on writebacks from instructions 1 and 2, which will not occur until the next clock cycle.

Clock cycle 5 contains instruction 1's WB stage, instruction 2's MEM stage, instruction 3's ID stage, and instruction 4's IF stage.

Clock cycle 6 contains instruction 2's WB stage, instruction 3's EX stage, instruction 4's ID stage, and instruction 5's IF stage.

Clock cycle 7 contains instruction 3's MEM stage, instruction 4's EX stage, and instruction 5's ID stage. Note that instruction 5's ID stage relies on completion of instruction 3's WB stage, which will not occur until the next clock cycle. Therefore, instruction 3 will forward the data from its MEM stage to instruction 5's EX stage in the next clock cycle. Also note that no other instructions are added at this stage because the branch prediction is (incorrectly) predicting that the loop will not be taken, and branch resolution does not occur until instruction 5's EX stage, which will not occur until the next clock cycle.

Clock cycle 8 contains instruction 3's WB stage, instruction 4's MEM stage, instruction 5's EX stage, and instruction 1's IF stage. Instruction 1 is now inserted here because the branch prediction was resolved in instruction 5's EX stage, in this clock cycle.

Clock cycle 9 contains instruction 4's WB stage, instruction 5's MEM stage, instruction 1's ID stage, and instruction 2's IF stage. Note that instruction 1's ID stage relies on completion of instruction 4's WB stage, which occurs in this clock cycle, so no forwarding is needed.

Eric Tashji

Clock cycle 10 contains instruction 5's WB stage, instruction 1's EX stage, instruction 2's ID stage, and instruction 3's IF stage. Note that instruction 2's ID stage relies on completion of instruction 4's WB stage, which occurred in the last cycle.

Clock cycle 11 contains instruction 1's MEM stage, instruction 2's EX stage, and a stall for instruction 3's ID stage. A stall is necessary here because instruction 3's ID stage relies on registers being written to by instructions 1 and 2, which has not occurred yet. Therefore, a stall is necessary for instruction 3's ID stage because forwarding cannot be used in this instance.

Clock cycle 12 contains instruction 1's WB stage, instruction 2's MEM stage, instruction 3's ID stage, and instruction 4's IF stage.

Clock cycle 13 contains instruction 2's WB stage, instruction 3's EX stage, instruction 4's ID stage, and instruction 5's IF stage. Note that instruction 4's ID stage relies on completion of the previous iteration of instruction 4's ID stage, which has already occurred. This means forwarding is not needed.

Clock cycle 14 contains instruction 3's MEM stage, instruction 4's EX stage, instruction 5's ID stage, and instruction 1's IF stage. Note that instruction 1 is added back in because the branch prediction (incorrectly) believes that the branch will be taken.

Clock cycle 15 contains instruction 3's WB stage, instruction 4's MEM stage, instruction 5's EX stage, and a flush for instruction 1. At this point, branch resolution has occurred and determined that the loop will not be taken again, so instruction 1 will be flushed from the system.

Clock cycle 16 contains instruction 3's WB stage and instruction 4's MEM stage.

Clock cycle 17 contains instruction 5's WB stage.

	Clock Cycle 1	**Clock Cycle 2**	**Clock Cycle 3**	**Clock Cycle 4**	**Clock Cycle 5**	**Clock Cycle 6**	**Clock Cycle 7**	**Clock Cycle 8**	**Clock Cycle 9**	**Clock Cycle 10**	**Clock Cycle 11**	**Clock Cycle 12**	**Clock Cycle 13**	**Clock Cycle 14**	**Clock Cycle 15**	**Clock Cycle 16**	**Clock Cycle 17**
LDUR X10, [X1, #0]	IF	ID	EX	MEM	WB												
LDUR X11, [X1, #8]		IF	ID	EX	MEM	WB											
ADD X12, X10, X11			IF	Stall	ID	EX	MEM	WB									
SUBI X1, X1, #16					IF	ID	EX	MEM	WB								
CBNZ X12, LOOP						IF	ID	EX	MEM	WB							
LDUR X10, [X1, #0]								IF	ID	EX	MEM	WB					
LDUR X11, [X1, #8]									IF	ID	EX	MEM	WB				
ADD X12, X10, X11										IF	Stall	ID	EX	MEM	WB		
SUBI X1, X1, #16												IF	ID	EX	MEM	WB	
CBNZ X12, LOOP													IF	ID	EX	MEM	WB
LDUR X10, [X1, #0]														IF	Flushed		

Eric Tashji