Name: Georges Hatem

CS 550 Homework Assignment 3

Due Date: November 14, 2021

Exercise 4.3.1

For R-type -> Data Memory is skipped:

The result obtained from the ALU for any R-type instruction is fed into the Multiplexer (MUX) controlled by MemtoReg, skipping the data memory and writing the result into the corresponding register.

For I-type -> Data Memory is skipped:

The result obtained from the ALU for any I-type instruction is fed into the Multiplexer (MUX) controlled by MemtoReg, skipping the data memory and writing the result into the corresponding register.

For LDUR -> Data Memory is used:

The sum from the ALU is used as the address for data memory for LDUR.

For STUR -> Data Memory is used:

The store register instruction (STUR) writes to the data memory.

For CBZ -> Data Memory is not used:

CBZ instruction does not reach data memory. In CBZ instruction, the ALU passes the data value read from the register file. The value of PC is added to the sign-extended, 19 bits of the instruction (offset) are shifted left by two; the result is the branch target address. The branch control is set to 1 for CBZ instruction, and the zero status information from the ALU is used to determine which adder result to store in the PC (PC + 4 or branch target address).

For B -> Data Memory is not used:

For B instruction, we need to go to the address in the B instruction. The Unconditional Branch Control is set to 1, therefore, making PC the target address in the B instruction instead of PC + 4. Unconditional branch (B) does not reach data memory.

Based on the above findings, my answer for this exercise is as follows:

35 percent of all instructions in this exercise use Data Memory.

This is done by adding the LDUR and STUR percentage instructions since they are the ones who use data memory in this exercise.

Exercise 4.3.2:

For R-type -> Use Instruction Memory:

The instruction is fetched first before doing any other operation (such as read registers, ALU operation, etc). So, yes R-type instruction uses instruction memory.

For I-type -> Use Instruction Memory

The instruction is fetched first before doing any other operation (such as read registers, ALU operation, etc). So, yes, I-type instruction uses instruction memory.

For LDUR -> Use Instruction Memory

The instruction is fetched first before doing any other operation (such as read registers, ALU operation, etc). So, yes, LDUR instruction uses instruction memory.

For STUR -> Use Instruction Memory

The instruction is fetched first before doing any other operation (such as read registers, ALU operation, etc). So, yes, STUR instruction uses instruction memory.

For CBZ -> Use Instruction Memory

The instruction is fetched first before doing any other operations (sign extend, etc). So, yes, CBZ instruction uses instruction memory.

For B -> Use Instruction Memory

The instruction is fetched first before doing any other operations (sign extend, etc). So, yes, B instruction uses instruction memory.

Based the above findings, my answer for Exercise 4.3.2 is as follows:

100 percent of all instructions in this exercise use instruction memory.

Exercise 4.3.3

For R-type -> Does not use Sign Extend

R-type instructions (ADD, SUB, etc) do not need to branch to a target address. So, it does not use the sign extend. The PC + 4 is used to move to the instruction right after the R-type instruction. So, R-type does not use the sign extend.

For I-type -> Use the Sign Extend

I-type instructions (ADDI, SUBI, etc) use the sign extend to extend instructions [31-0] to 64 bits and then pass it to the ALU second input

through the ALUSrc Multiplexer (MUX). Given for example, ADDI X0, X1, #3 the ALU adds the Rn (X1) passed from Read data 1, and #3 passed from the ALUSrc Multiplexer as described above. Therefore, I-type instructions use the sign extend.

For LDUR -> Use the Sign Extend

LDUR instruction use the sign extend to extend instructions [31-0] to 64 bits and then pass it to the ALU second input through the ALUSrc Multiplexer (MUX). Given for example, LDUR X1, [X0, #8], the ALU adds Rn (X0) from Read data 1 and #8 passed from the ALUSrc Multiplexer as described above. That sum from the ALU is used as the address for the data memory. Therefore, LDUR uses the sign extend.

For STUR -> Use the Sign Extend

STUR instruction use the sign extend to extend instructions [31-0] to 64 bits and then pass it to the ALU second input through the ALUSrc Multiplexer (MUX). Given for example, STUR X1, [X0, #8], the ALU adds Rn (X0) from Read data 1 and #8 passed from the ALUSrc Multiplexer as described above. That sum from the ALU is used as the address for the data memory that we want to store the value X1 in. Therefore, STUR uses the sign extend.

For CBZ -> Use the Sign Extend

The sign extend in CBZ instruction is used to branch to the target address written in CBZ instruction. Therefore, CBZ uses the sign extend.

For B -> Use the Sign Extend

The sign extend in B instruction is used to branch to the target address written in B instruction. Therefore, B uses the sign extend.

Based on the analysis provided above, my response to Exercise 4.3.3 is as follows:

76 percent of all instructions use the sign extend.

This is done by subtracting 100 from 24, which is equal to 76 percent, since R-type instruction is the only type of instruction in this Exercise that does not use sign extend.

Exercise 4.4.1

When the MemtoReg is deasserted (0), the MemtoReg feeds the value coming from the ALU to the register write data input. When the MemtoReg is asserted (1), the MemtoReg feeds the value coming from the data memory to the register write data input.

Any instruction that has the MemtoReg asserted (1) (e.g. LDUR) will be affected. In the case of LDUR, the value that we need to load from the data memory will not be loaded and the data memory address will try to be written to the register write data input, which will most likely cause an error.

These are the instructions provided in Chapter 4 with their respective normally MemtoReg set bit (I added some instructions based on Exercises 4.3.1, 4.3.2, and 4.3.3):

Instruction	MemtoReg
R-format	0
LDUR	1
STUR	X
CBZ	X
B	X
I-format (ADDI, SUBI, etc)	0

From all the instructions that were talked about in Chapter 4, LDUR instruction is the only one that gets affected when MemtoReg is stuck at 0. LDUR will get affected as spoken above.

Exercise 4.4.2

Below are the normal ALUSrc bit for instructions provided in Chapter 4 and instructions in Exercises 4.3.1, 4.3.2, and 4.3.3:

Instruction	ALUSrc
R-format	0
LDUR	1
STUR	1
CBZ	0
B	X
I-format (ADDI, SUBI, etc)	1

Based on the instructions that were talked about in Chapter 4 and the instructions that were talked about in Exercise 4.3.1, 4.3.2, and 4.3.3, the instructions that will be affected are the LDUR, STUR (basically any D-format instruction since they all have an address and

that address must be passed through the sign extend before entering the ALU), and any I-format instruction.

Exercise 4.4.3

Below are the normal Reg2Loc bit for instructions provided in Chapter 4 and instructions in Exercises 4.3.1, 4.3.2, and 4.3.3:

Instruction	Reg2Loc
R-format	0
LDUR	X
STUR	1
CBZ	1
B	X
I-format (ADDI, SUBI, etc)	X

Based on the instructions that were talked about in Chapter 4 and the instructions that

were talked about in Exercise 4.3.1, 4.3.2, and 4.3.3, the instructions that will be affected by Reg2Loc stuck at 0 are the STUR and CBZ instructions.

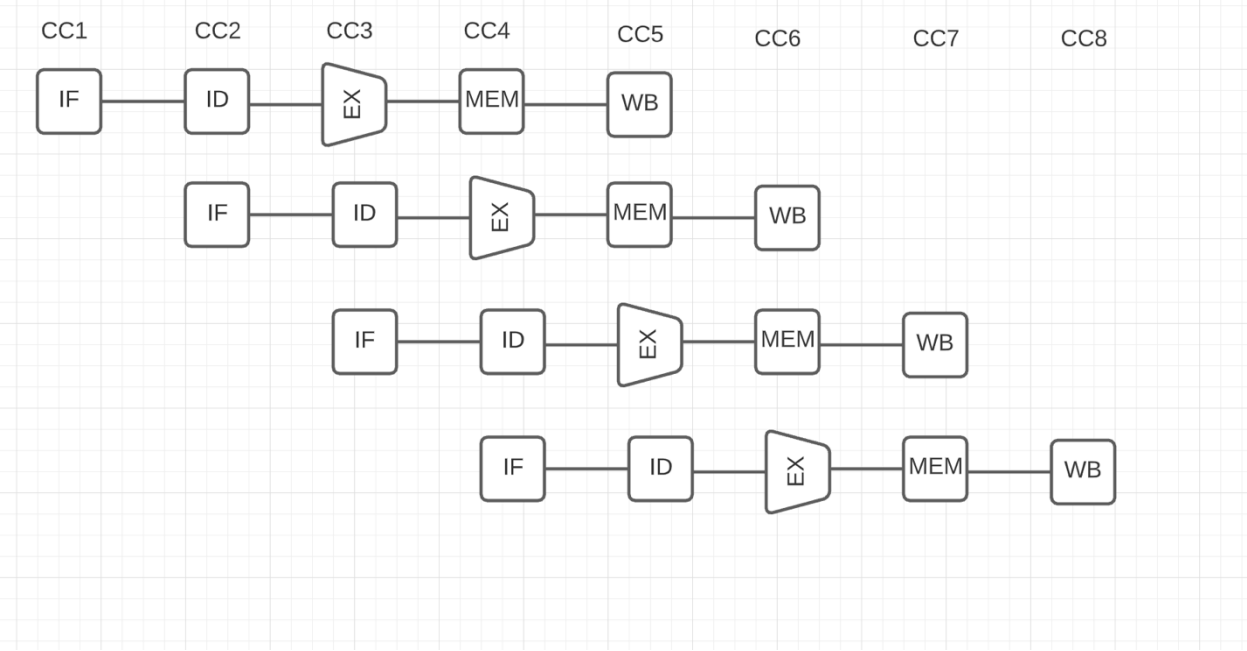
Exercise 4.19:

The value in X5 is 54. The reasoning behind this is as follows:

First, we start with X1 being 11 and X2 being 22.

The first instruction computes (X2 + 5) and write it in register X1. However, since the program is executed on a version of the pipeline that does not handle data hazard, the new value of X1, which is 22 + 5 = 27, will not be passed to the second ADD instruction right after it (ADD X3, X1, X2) and to the ADDI instruction (ADDI X4, X1, #15). However, by the time ADD X5, X1, X1, the following will be

happening. In the same cycle (CC5 in the pipeline execution diagram below), when X1 new value of 27 is being written, X1 value will be read (during ADD X5, X1, X1 implementation) and so the value that will be stored in X5 is 27 + 27 = 54.



Exercise 4.30.1:

Instruction 1: CBZ X1, LABEL

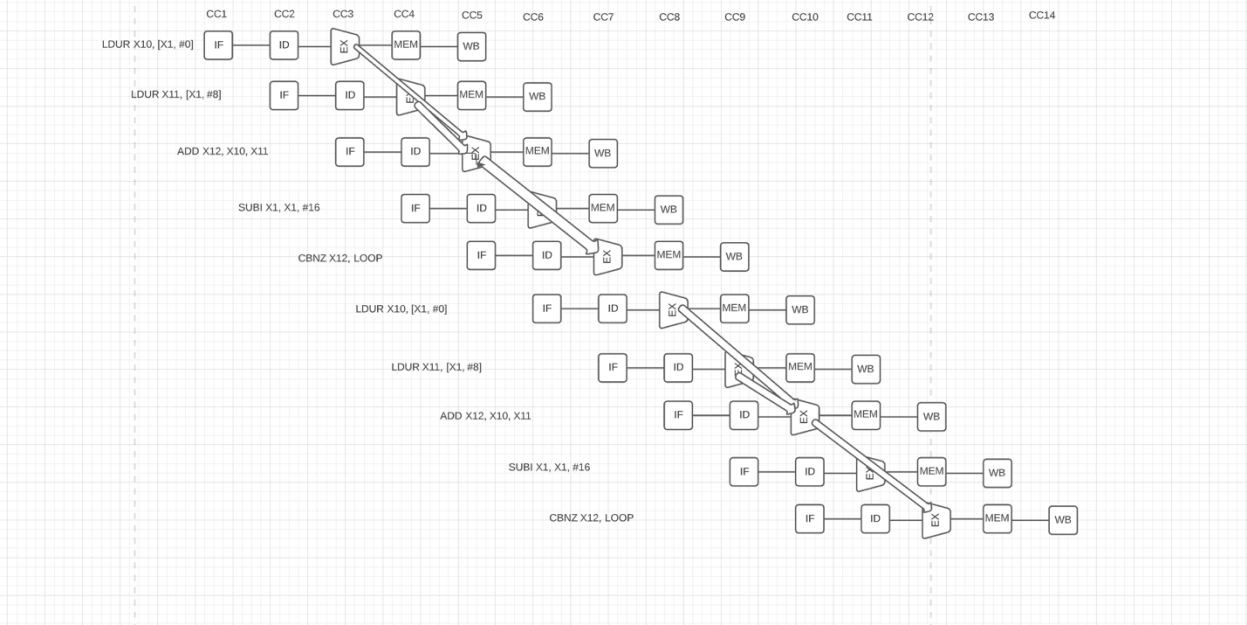
When this instruction is executed, the PC shifts to the branch target address. However, there might be an Invalid Target Exception in case the branch target address is not valid. This is detected during the Execute (EX) pipeling stage.

Instruction 2: LDUR X1, [X2, #0]

This instruction can cause an Invalid Data Address Exception. It is detected during the MEM Stage of the pipeline.

Exercise 4.25.A.1

Below is the pipeline execution diagram for A.1



Exercise 4.25.A.2