CMSC 133

Introduction to Computer Organization, Architecture and Assembly Language

Laboratory Report

Names of all Authors and Student IDs	Kienth Matthew B. Tan
Lab No.	2
Task/Step No.	1-3
Lab Title	Building a MIPS Processor
Email Addresses of all Authors	kbtan@up.edu.ph
Date of Document Issue	08/06/2022
Document Version Number	1.0
Address of Organisation Preparing this report:	Department of Computer Science University of the Philippines Cebu Gorordo Avenue, Lahug, Cebu City Tel: (032) 232 8185
Course Name	CMSC 133 Introduction to Computer Organization, Architecture and Assembly Language
Course Units	3

Summary

Laboratory 2 Step 1 – 3 is all about creating a simplified MIPS processor that can successfully run MIPS code using Logisim. Tasks included completing the control logic, hooking up the datapath, and finally is by testing a simple code to check if the processor works correctly.

There is a lot of things that you can learn in doing this laboratory activity, specifically in how the MIPS processor works. In completing the control logic, I learned how the control logic sends the correct signals to the processor in order to perform the needed instructions. However, I had a bit of trouble in this step especially in completing the control decode, as though my logic seems correct according to my table, it sometimes led to having errors in the processor as shown in how the type instructions would turn red. In completing the second step, I also learned how to wire each signal correctly to the appropriate components of the processor. This step was quite easy as we only needed to follow the wiring of the datapath from the provided figure, however even without the figure, it would still be quite simple to hook up the datapath. We only needed to know what each component of the processor needs or uses and simply hook it up. The last and final step which is in testing the processor by loading a code was one of the most confusing steps in the activity as we had to check multiple times if the inputs, outputs, MUXes, control bits, and the value and register being written is correct or not.

Over the course of this laboratory, we would truly learn and apply what we learned about MIPS and processors. It is here to strengthen and cement our learnings from the past few months. That each part of the MIPS processor plays a key part in being able to run a code. When one of the components isn't wired correctly or if the logic is incorrect then it would affect the whole processor. It is through a step by step understanding of the MIPS processor do we be capable of building a working and simple processor. From building and understanding the logic behind each key component through wiring the datapath and understanding what component needs and finally is through testing it, in which we finally verify if what we did was right and precise.

TABLE OF CONTENTS

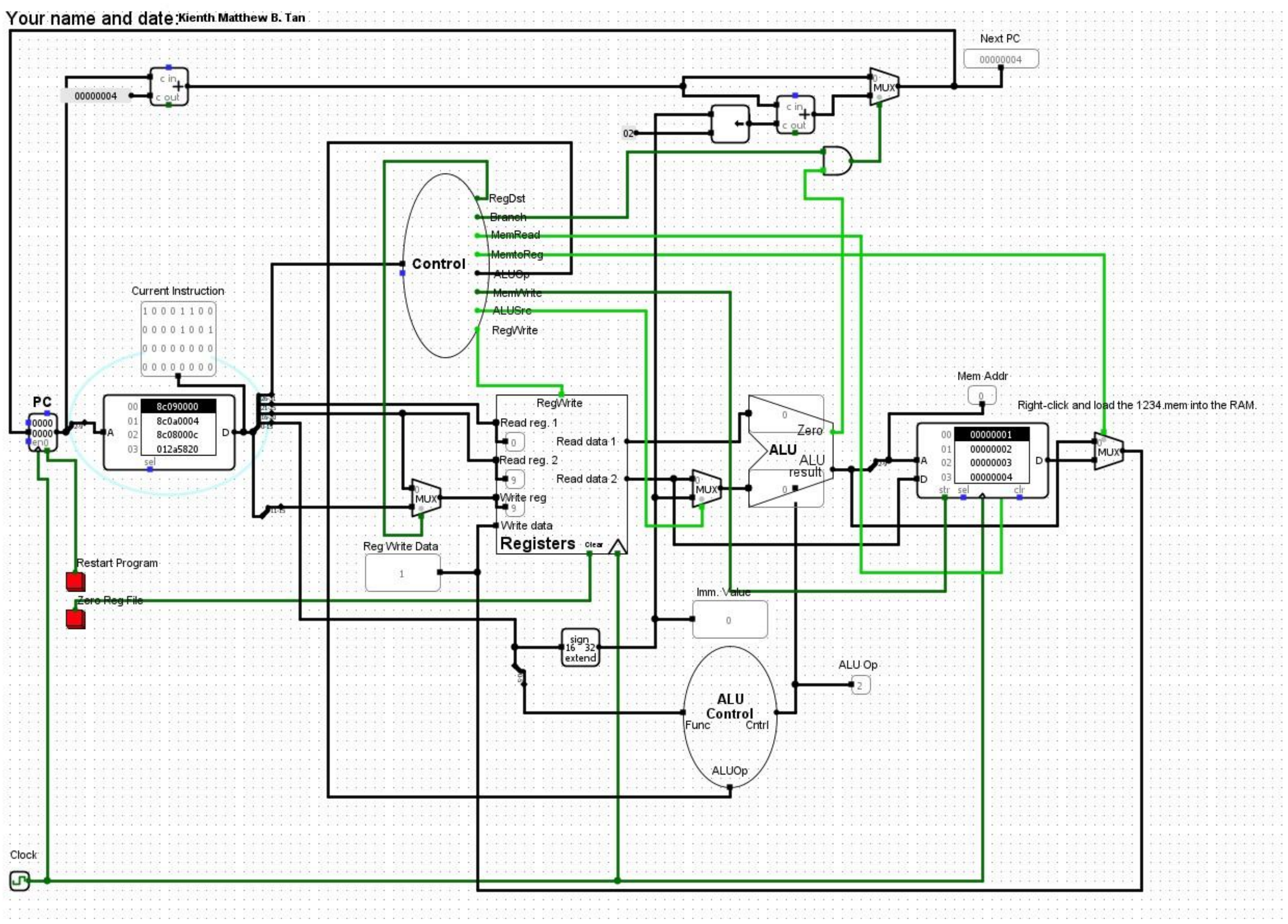
1 The Lab Problem

In this laboratory activity, we are tasked to build a simple MIPS Processor using Logisim and there are three steps to finish in doing so. The first step involves finishing the control logic, this includes the Type Decode unit, the Control Decode unit, and the ALU Control unit. After finishing the first step, the next task is to hook up the datapath and make sure that the MIPS processor is wired correctly. Lastly, is to test the processor with the provided MIPS code. It is a step by step laboratory activity where to be able to complete and understand the task, you have to complete the task before it.

2 The MIPS Processor

Put here an image of each of the following processor component. Explain the function of each component.

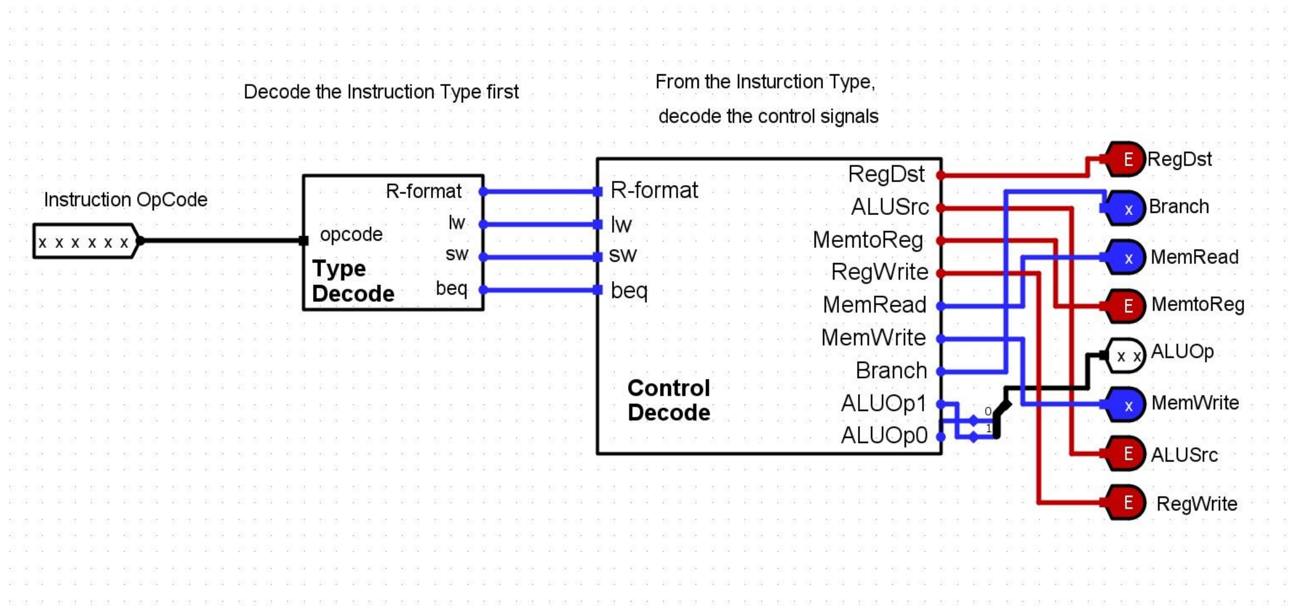
2.1 The Whole MIPS Processor



The MIPS processor makes use of all of its key components to run a code. The components being the PC, instruction memory, control logic, registers, the ALU and ALU control, and the final component is the data memory. To execute an instruction using the MIPS processor, it needs to go through five stages:

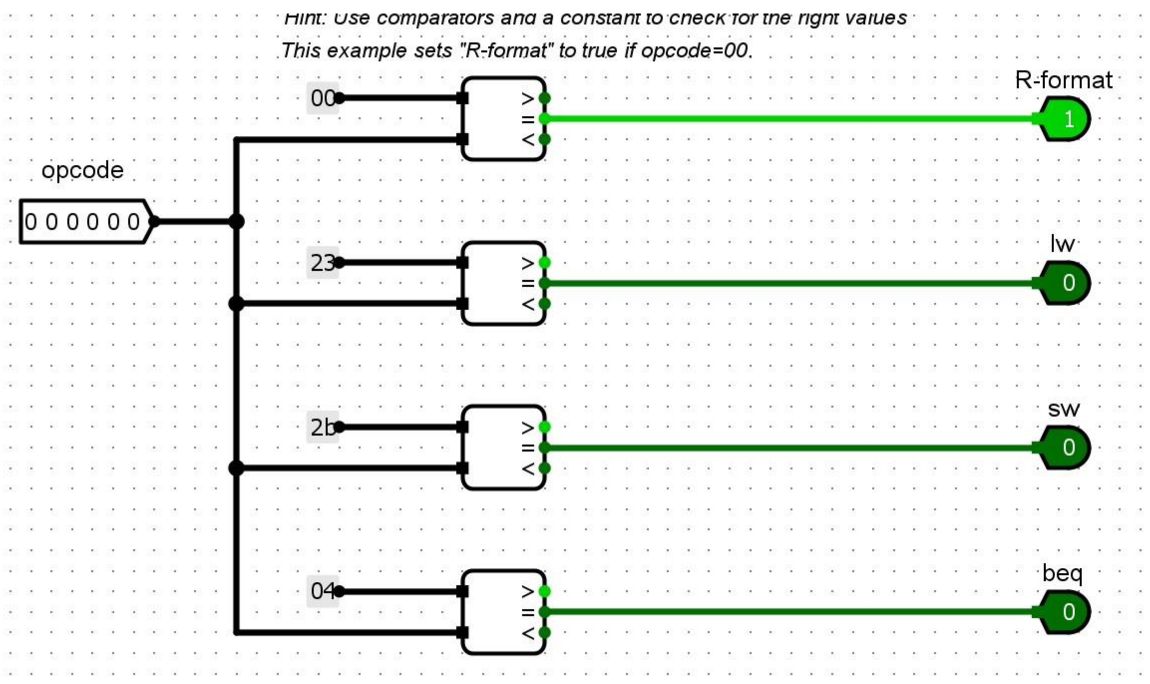
- **IF** or the instruction fetch stage fetches the next instruction from memory using the address in the PC register and stores the instruction in the Instruction Register.
- **ID** or the instruction decode stage decodes the instruction, calculates the next PC, or reads any operands required from the register.
- **EX** or the execute stage makes use of the ALU to perform operations such as addition and subtraction.
- **MA** or the memory access stage performs any other memory access required by the instruction such as loading from memory and storing into memory. It makes use of the data memory component.
- **WB** or the write back stage writes the result back to the register file.

2.2 The Control Unit



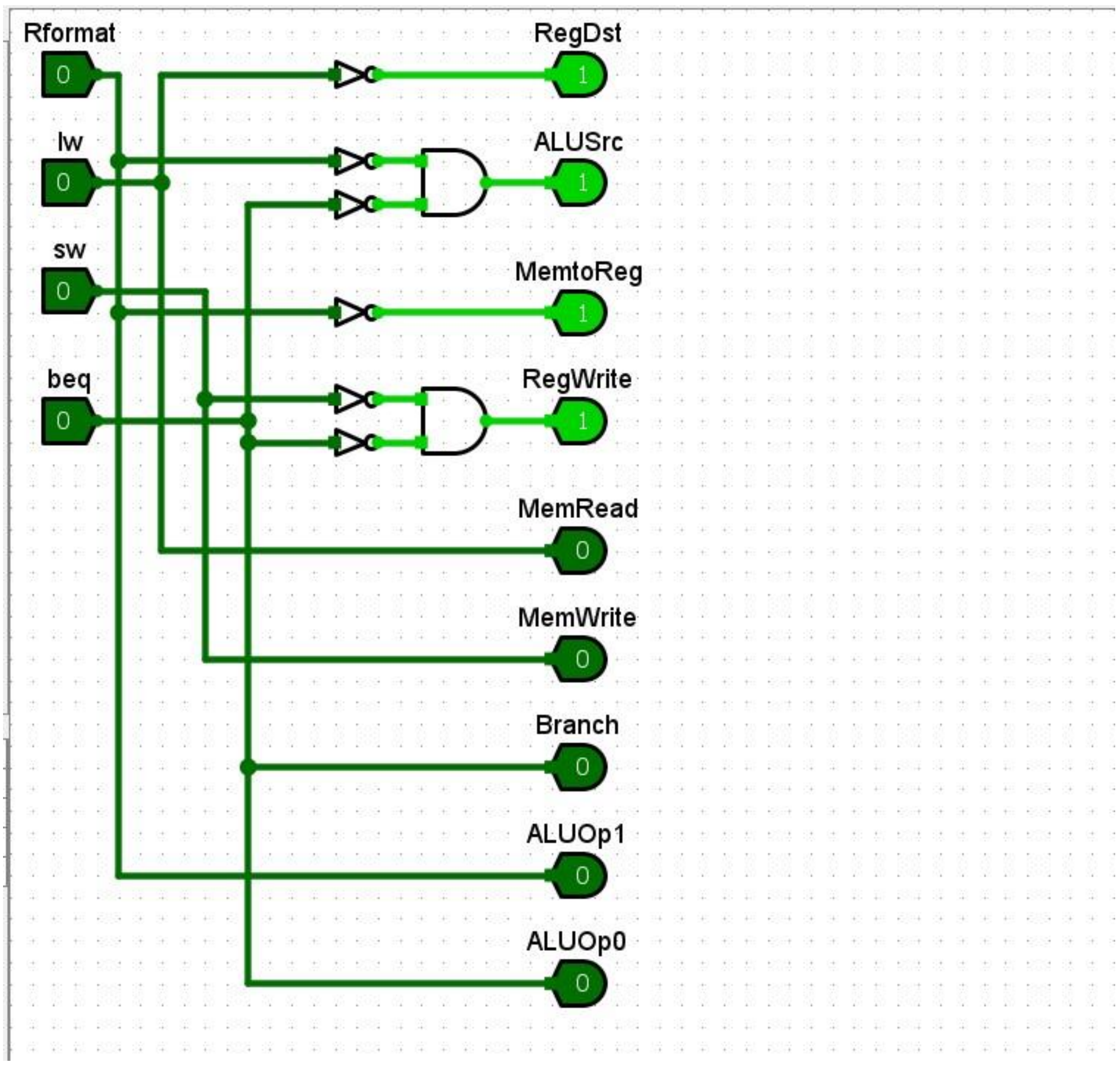
The Control Unit is responsible for setting all the control signals so that each instruction is executed properly. It is composed of the Type Decode unit and the Control Decode unit. It tells the datapath what to do, based on the instruction that is currently being executed.

2.2.1 The Type Decode Unit


Hint: Use comparators and a constant to check for the right values
This example sets "R-format" to true if opcode=00.
opcode
0 0 0 0 0 0
00
23
2b
04
R-format
1
lw
0
sw
0
beq
0

The Type Decode unit is a component of the control unit. It accepts an opcode and depending on the opcode, it would determine the instruction type whether it is an R-type instruction such as in the use of add, sub, and slt, or whether it is a load instruction, store instruction, or a branch instruction.

2.2.2 The Control Decode Unit

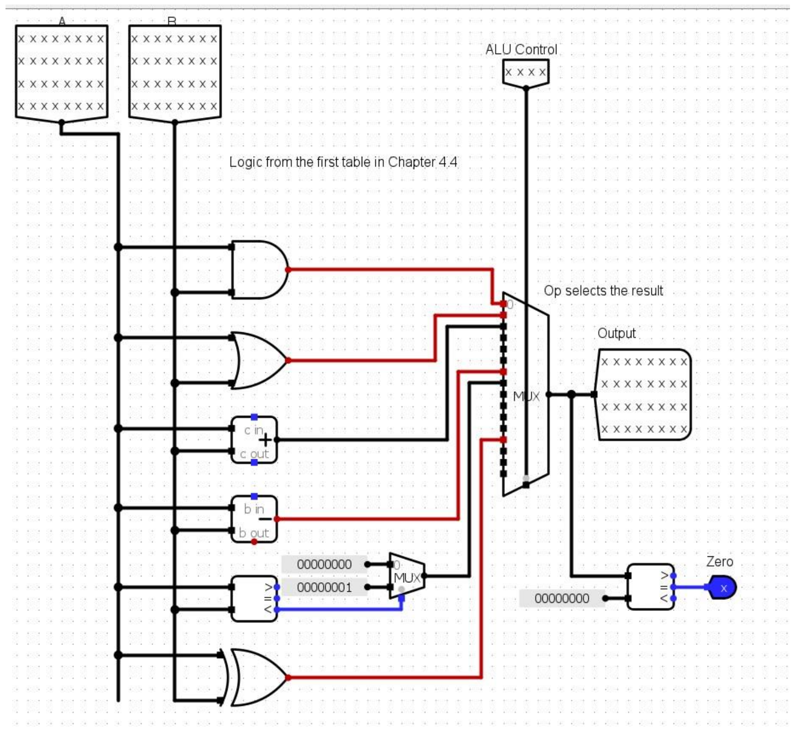


Include the completed table for the Control Decode Unit.

Output	R-format	Lw	Sw	Beq
RegDst	1	0	X	X
ALUSrc	0	1	1	0
MemtoReg	0	1	X	X
RegWrite	1	1	0	0
MemRead	0	1	0	0
MemWrite	0	0	1	0
Branch	0	0	0	1
ALUOp1	1	0	0	0
ALUOp0	0	0	0	1

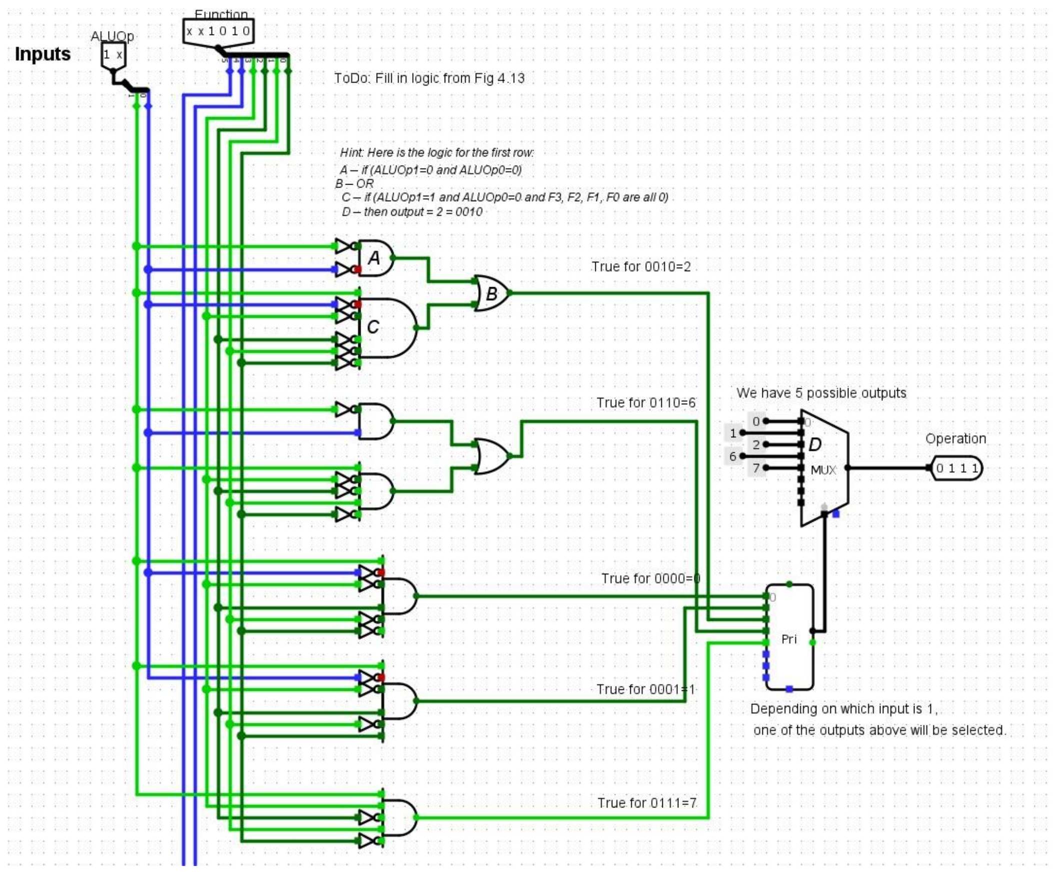
The control decode unit is another component of the Control unit. It decodes the instruction and depending on the type instruction, it would determine the control signals that will be used by the instruction.

2.3 The ALU



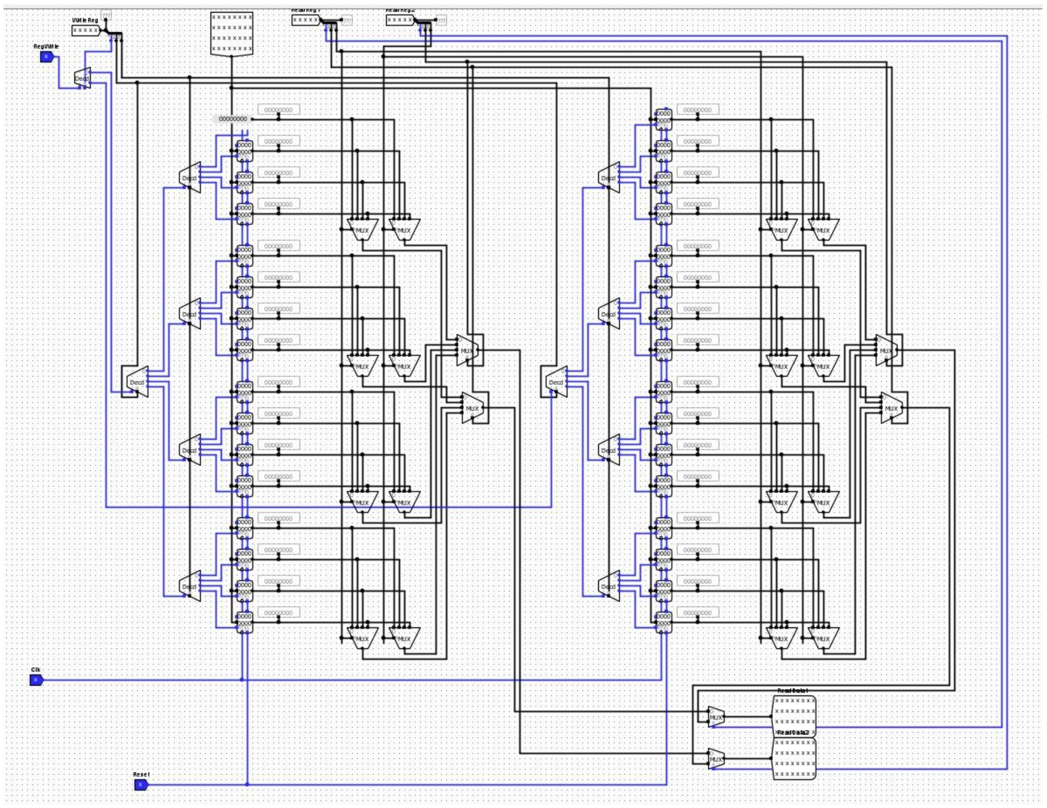
The ALU is used to calculate an arithmetic result, compute a memory address, or perform a comparison for a branch. It takes two values A and B as shown in the circuit, the ALU control then chooses what operation to perform with the two values. The result of the execution is then written back into the register file.

2.4 The ALU Control Unit


Inputs
ALUOp
1 x
Function
x x 1 0 1 0
ToDo: Fill in logic from Fig 4.13
Hint: Here is the logic for the first row:
A – if (ALUOp1=0 and ALUOp0=0)
B – OR
C – if (ALUOp1=1 and ALUOp0=0 and F3, F2, F1, F0 are all 0)
D – then output = 2 = 0010
A
B
C
True for 0010=2
True for 0110=6
We have 5 possible outputs
0
1
2
6
7
D
MUX
Operation
0 1 1 1
True for 0000=0
Pri
True for 0001=1
Depending on which input is 1,
one of the outputs above will be selected.
True for 0111=7

The ALU control unit determines what operation to be used in the ALU. It accepts two inputs, ALUOp and Function which is then passed into the subsequent logic, and depending on which of the following logic results in 1, one of the outputs in the MUX will be selected and then passed into the ALU.

2.5 The Register File



The register file contains the 32 MIPS general-purpose registers. This is the component where we can read from registers or write to registers. It makes it possible to read from two registers and write into one register depending on the MIPS instruction. It acts as a means of memory storage, containing instructions, addresses, or any kind of data.

3 Answer to Questions

3.1 What would happen if more than one output from the Type Decode unit was true at once?

- It would affect the output outcome and the R-format output variation, it would also crash as because you would have two instructions going on at the same time.

3.2 What would this mean about the input?

- It would mean that the opcode would have two values at the same time which is impossible in this processor.

3.3 Why doesn’t the truth table for the Control Decode Unit you filled in have an entry for 0011?

- This is because there is no instruction that uses both sw and beq at the same time only. If it was 1011 or 0111 then there would be instructions as Rformat and lw usually have don’t cares for the last two bits.

3.4 Write down for each cycle (until the program runs out of instructions) what value is in each used register and what value and register will be written to for each instruction.

Cycle	Instruction	Register Written	Register Used
1	lw $t1, 0($zero)	$t1 = 1	
2	lw $t2, 4($zero)	$t2 = 2	
3	lw $t0, 12($zero)	$t0 = 4	
4	add $t3, $t1, $t2	$t3 = 3	$t1 = 1, $t2 = 2
5	add $t4, $t3, $t1	$t4 = 4	$t3 = 3. $t1 = 1
6	sub $t0, $t0, $t2	$t0 = 2	$t0 = 4, $t2 = 2
7	or $t5, $t4, $t1	$t5 = 5	$t4 = 4, $t1 = 1
8	slt $t6, $t4, $t5	$t6 = 1	$t4 = 4, $t5 = 5
9	beq $t2, $t0, loop		$t2 = 2, $t0 = 2
10	add $t3, $t1, $t2	$t3 = 3	$t1 = 1, $t2 = 2
11	add $t4, $t3, $t1	$t4 = 4	$t3 = 3. $t1 = 1
12	sub $t0, $t0, $t2	$t0 = 0	$t0 = 2, $t2 = 2
13	or $t5, $t4, $t1	$t5 = 5	$t4 = 4, $t1 = 1
14	slt $t6, $t4, $t5	$t6 = 1	$t4 = 4, $t5 = 5
15	beq $t2, $t0, loop		$t2 = 2, $t0 = 0
16	PROGRAM RUNS OUT OF INSTRUCTIONS		

3.5 What is the value of the Write data and the Write reg when the beq instruction is executed? Why? Is this a problem?

- The value of the write data is '???' and Write reg is 31 when the beq instruction is executed. This is not a problem. Write data does not have any value simply because beq does not write into any registers, that's why it has '???' as a value in write data. As for why write reg is 31 when beq is executed, it is because register 31 is the return address register, it is used in jumps and branches.

4 Learning & Insights

In the first laboratory task, I learned a lot about the control logic. Essentially there are two parts to the control logic, the first part is the Type decode unit where it compares the hexadecimal constant with the appropriate opcode to generate the correct instruction type, and when both opcode and the constant are equal, the type instruction corresponding to them would be true. From looking at the logic where we are using comparators, it is simple to see that there can only be one output that would return true, otherwise there would be complications as only one instruction can be running at a time. The second part however makes use of these instruction types. It would decode the output of the type decode and turn the corresponding control signals to true as well. For example, if R-format is true then RegDst, RegWrite, and ALUOp1 would return true as well. To complete the control logic for the control decode however, I didn't use the combinational analysis as it was a bit complicated, instead I completed the control logic manually by following the table I made. The last part to complete the control logic is the ALU control, and from what I understood by completing its control logic is that the ALU controls the operation that is needed by the current instruction. By looking into the ALU component, we can see that it takes two data's from the registers and performs numerous operations such as add, sub, and, and or which then leads into a MUX. The output from the ALU control then leads into the MUX and depending on the output, it would select the result from the operations.

There are also a few things I've learned from wiring the datapath especially on what signals the key components such as the ALU and registers need from the Control.

5 Conclusion

In this laboratory activity, we made use of everything we learned about MIPS and through doing this activity, we learned about the ins and outs of a MIPS processor. In this way, we also learn about what happens when we load a MIPS code into the processor. We can see how and why the program counter changes, we can learn and see how control signals are sent from the control unit, we can also learn and see how the ALU works. Whenever a problem arises in the processor, the tips on how to debug present in the starter files also helped in truly learning what we are doing, what the problem was, why it happened, and finally is how we can fix it.

In Laboratory 1, we learned how to use the MIPS assembly language to make programs that fit our objectives such as getting the length of a string. In this laboratory activity however, we learn about how the MIPS processor runs our code. The experience and knowledge provided by doing this laboratory will help in strengthening our current knowledge on processors and would hopefully help in our future endeavors when we encounter computer architecture.