Pipelined Processor Report

Instructor: Salah Al-Salah / COE301 / Term 212

By Group #23

Name	ID	Lab Section
Mahammed Al Sabaa	201960790	51
Ahmed El Tayeb	201763170	52
Sajjad Al-Saeed	201630160	51

- **Objectives**
    1. Creating a Pipelined 32-bit processor with 32-bit instructions.
    2. Practicing the Logisim simulator to model and analyze the processor.
    3. Teamwork with group members.

- **Introduction**

In this project, we are going to build a processor that will be able to calculations, comparisons, and more. To build a pipelined processor, a single-cycle processor is needed first. Necessary components are required to design a processor. First component is File register which contains 32 registers. Those registers are to store 32 bits. ALU or Arithmetic-Logic Unit is the second component that will be responsible to do operations such as addition, multiplication, and comparisons. Third part is Control-Unit. This component is going to send signals to certain components depending on what operation is used, and it will also send a signal to ALU to select what operation to execute. Another component is going to be Program counter or what is called in the project next PC. It will be responsible to do jump and branches and change the PC. Other components like memory are already available in Logisim such as RAM, ROM. ROM is used to load instructions in hexadecimals, and RAM is going to be used to load and store values. After single-cycle processor is built, the pipelined processor can be built based on the single-cycle processor design. Forwarding is first the step to build from single cycle to pipelined processor. We need to divide our processor into 5 stages, fetch instructions, Instruction Decode and Register Read, Execute, Memory access, and write back. Registers are needed to be between these stages to save the income instruction before the cycle pulse came. This step will allow the processor to do more instructions in a single cycle because each instruction is in a certain stage. The disadvantage of Forwarding is causing Hazard where if the second instruction depends on the first instruction, for example “addi R1, R0, 10 add R2, R1, R1”, the second instruction may get the old value of R1 instead of the updated R1. Thus, a new component called Hazard detector will check if any of instructions depends on other instructions that are is not finished from all stages. What the Hazar Detector will do is take the needed value from other instructions and use it in instructions that need that value. It also has a condition called Stall to stop Load instruction by bubbling if it depends on other instructions. Stall condition is also going to disable PC and instruction registers. Bubbling is useful on other things like Jump and branches instructions. With these modifications, our Pipelined processor is ready to work.

- **Design and Implementation**
  - **Single-Cycle Processor**

Register File

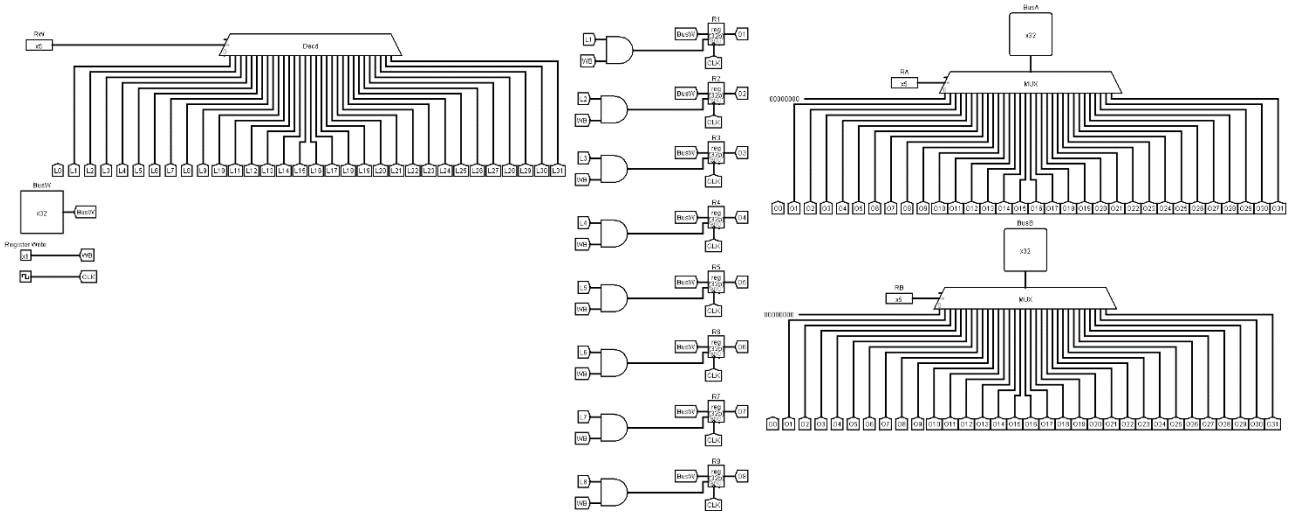

Figure 1: from left, there is a decoder to select which register to write on. Down the decoder is a 32 bits input for the chosen register. Another input is a button to let the input enter the register, so the register would not be written without write enable. Last input is clock that will be connected to registers. In the mid of the circuit, there is a column of 32 registers (only shown 6 registers because the picture will be too long). Finally, in the right side is where we select two buses or a bus to read from registers. The chosen register will be input for a multiplexer to provide the value in the chosen register as an output.

Arithmetic-Logic Unit (ALU)

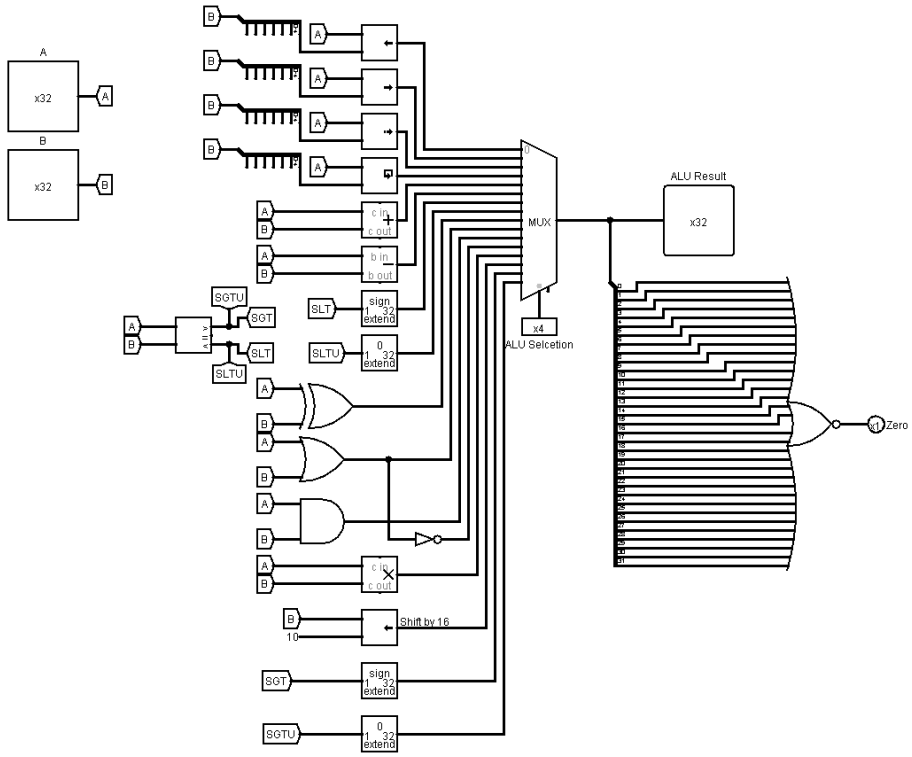

Figure 2: ALU will take two inputs that are 32 bits and execute them into operations. From top or first input, shift logic to left, shift logic to right, shift arithmetic right, addition, subtraction, SLT which check if the first input is less than the second input, SLTU is just SLT but with unsigned numbers, XOR logic gate, OR logic gate, AND logic gate, NOR logic gate, multiplication, shifting by immediate 16 is special case for LUI instruction, SGT check if the first input is bigger than the second input, and SGTU is just SGT but with unsigned numbers. All these operations enter a multiplexer, and, from ALU selection, we will choose which output we would like to get. Zero output is going to be for next PC for BEQ.

Control Unit

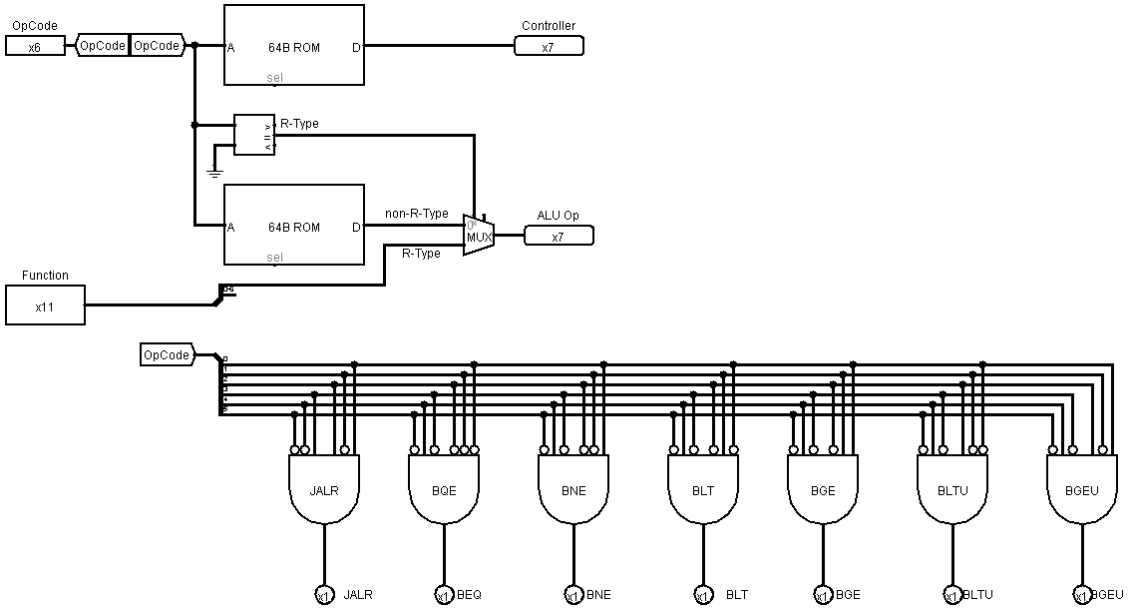

Figure 3: Control Unit will take Operation code (Op Code) from the hexadecimal code. If the Op Code is zero, this means this is an R-type instruction and has something called function that will be go into ALU to select the right operation. A signal will be output to enable File register. If the op code is not zero, it can be either a I-type instruction or a SB-type instruction. The above ROM contains needed signals for every op code. The second ROM is for ALU selection for every op code. There is a simple decoder for branches that are needed in Next PC. This decoder will select the right output depending on op code. for example, op code equals 16 will be BQE.

Next PC

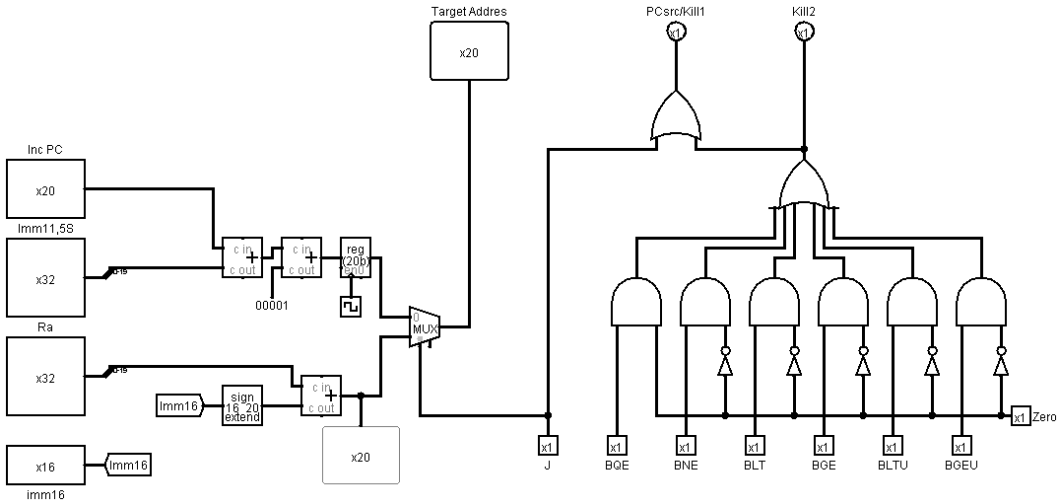

Figure 4: Next PC's job is to check if a jump instruction or a branch instruction is executed. If it is executed, the jump instruction address will be an addition of first register and immediate 16, or the branch address will an addition of PC and signed immediate 11, and 5.

- **Pipelined Processor**

Hazard Detector

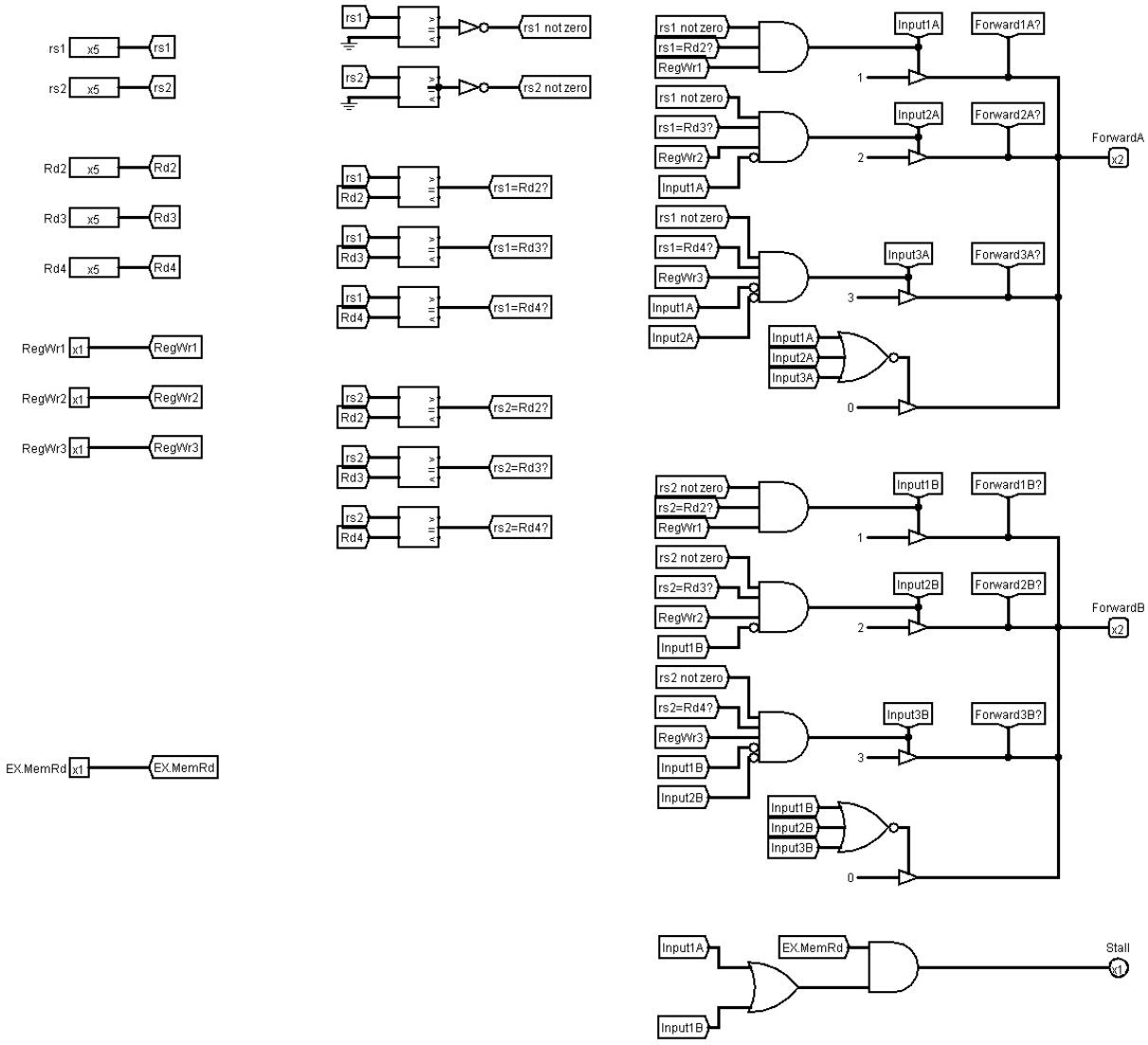

Figure 5: Hazard Detector or Hazard control is responsible to provide any instructions depending on other instructions. it will get any needed before Memory Stage (after ALU, and before ALU register), before Write back-stage, and in write back-stage, and it will forward the needed value to be Bus A or Bus B based on what value it was needed in the instruction. It will prioritize Forward1 (result from ALU), then Forward2 (in Memory Stage), then Forward3 (in Write Back stage) for both A and B. Stall output is a condition if some instructions depend on a load instruction, these instructions will enter bubbling condition (gives no signal control) until load instruction is almost finished and is in Memory stage.

- **Simulation and Testing**
  - **Testing Program 1: All instructions**

Assembly Instruction	HEX	Comment/Remark
I:		
addi R1, R0, 8	00080045	R1 = R0 + 8
addi R8, R0, 15	000f0205	R8 = R0 + 15
addi R30,R0, 1	00010785	R30 = R0 + 1
slli R2, R1, 1	00010881	R2 = R1 shifted left
srli R3, R8, 1	000140c2	R3 = R8 shifted right logical
srai R4, R8, 1	00014103	R4 = R8 shifted right arithmetic
rori R5, R8, 1	00014144	R5 = R8 rotate to right
slti R6, R8, 12	000c4186	R6 = set R8 less than 12
sltiu R7, R1, 10	000a09c7	R7 = set R8 less than 12
xori R8, R8, 2	00024208	R8= R8 xor with 2
ori R9, R1, 2	00020a49	R9 = R1 or with 2
andi R10, R1, 8	00080a8a	R10 = and with 8
nori R11,R1, 1	00010acb	R11 = R1 nor with 1
lui R12, 13	000d030c	R12 = loading upper 16 bits
sw R2,9(R0)	0002024f	Store R2 in the memory
lw R14,0(R0)	0000038e	R14 = load from the memory
jalr R0,R0,r	0011000d	Jump to r
r:		
add R1, R30, R30	009ef040	R1 = R30 + R30 = 2
add R7, R1, R1	008109c0	R7 = R1 + R1 = 4
sll R2, R1, R1	00010880	R2 = R1 shifted left
srl R3, R1, R30	003e08c0	R3 = R1 shifted right logical
sra R4,R1, R30	005e0900	R4 = R1 shifted right arithmetic
ror R5,R1, R1	00610940	R5 = R8 rotate to right
sub R6,R7,R1	00a13980	R6 = R7 – R1 = 2
slt R7, R1,R7	00c709c0	R7 = set R1 less than R7
sltu R8, R6,R7	00e73200	R8 = set R6 less than R7
xor R9, R1, R2	01020a40	R9= R1 xor with R2
or R10, R1, R2	01220a80	R10 = R1 or with R2
and R11, R2, R1	014112c0	R11 = R1 and with R1
nor R12,R1,R2	01620b00	R12 = R1 nor with R2

mul R13, R1,R1	01810b40	R13 = R1 * R1 = 4
sb:		
lw R1,9(R0)	0009004e	R1 = load from the memory
lw R7,0(R0)	000001ce	Loading element number “0” into R7
beq R1,R7,next	00070890	If R1 is equal to R7 branch
addi R1,R7,1	00013845	R1 = R7+1
next:		
lw R7,1(R0)	000101ce	Loading the second memory element into R7
bne R1,R7,next1	00070891	If R1 is not equal to R7, branch
addi R1,R7,1	00013845	R1 = R7+1
next1:		
lw R7,2(R0)	000201ce	Loading the third memory element into R7
blt R1,R7,next2	00070892	If R1 is less than R7, branch
addi R1,R7,1	00013845	R1= R7+1
next2:		
lw R7,3(R0)	000301ce	Loading fourth memory element into R7
bge R1,R7,next3	00070893	If R1 greater or equal to R7, branch
addi R1,R7,1	00013845	R1 = R7+1
next3:		
lw R7,4(R0)	000401ce	Loading the fifth element in memory to R7
bltu R1,R7,next4	00070894	If R1 is less than R7, branch
addi R1,R7,1	00013845	R1 = R7+1, which is “6”
next4:		
lw R7,5(R0)	000501ce	R7= sixth element in memory, “4”
bgeu R1,R7,exit	00070895	If R1 greater than or equal to R7, exit.
addi R1,R7,1	00013845	R1 = R7+1
exit:		

- **Testing Program 2: Counter for a binary bit 1 in a chosen number**

Assembly Instruction	HEX	Comment/Remark
addi R3, R0, 15	000f00c5	Initializing the number we want to count the ones in.
addi R31,R0,31	001f07c5	Number of times we want to traverse.
addi R1,R0,1	00010045	To compare the bit with 1.
addi R5,R0,R0	00000145	Initializing the counter for the loop
beq R5,R31, exit #loop starts before this instruction	001f29d0	Loop break statement.
andi R7,R3,R1	000119ca	Anding the least significant with one to check its value
sra R3,R3,1	004118c0	Shifting right to check the next bit
bne R7, R1, exit1	00013891	If bit is not equal 1 we increment loop counter but not “1’s counter”
addi R6,R6,1	00013185	Incrementing 1’s counter
addi R5,R5,1 #exit1 starts before this instruction	00012945	Incrementing loops counter
jalr R0,R0,loop #exit is after this instruction	0004000d	Jumping to loop

- **Testing Program 3: Use loops to verify Hazard Detector and Forwarding**

Assembly Instruction	HEX	Comment/Remark
addi R1,R0,0	00000045	R1 will have the value 0

addi R2,R0,0	00000085	R2 will have the value 0
addi R3,R0,9	000900c5	R3 will have the value 9
beq R1,R3,exit #Loop starts here	00030910	Branch if R1 equals R3, but R1 = 0 and R3 = 9, so branch will be executed for first time
addi R1,R1,1	00010845	R1 will be an addition of R1 and integer 1. This is for counting the loop.
add R2,R1,R2	00820880	R2 will be an addition of R1 and R2. This is to test Hazard control.
jalr R0,R0,loop #exit jump is after this	0003000d	Jump back and do the branch instruction again until it success, then the program will exit and bubble the addition instructions.

- **Testing Program 4: Bubbling Sort**

Assembly Instruction	HEX	Comment/Remark
addi R12,R0,8	00080305	Here we store the number of elements *minus one*
addi R1,R0, 0	00000045	Storing the index of *i*
addi R1,R1,1	00010845	Incrementing the value of i after each iteration.
blt R12,R1,end	000164d2	When i reaches the last element, exit the program
addi R2,R12, 0	00006085	Storing the number of elements for the inner loop
beq R1,R2,loop1	ffe20f10	If the counter is equal to the number of elements in the inner loop, go back to the first one.
addi R15,R0, 1	000103c5	Inserting 1 so we can use it for subtraction
sub R2,R2, R15	00af1080	Subtracting for the iteration
sll R5, R2, R15	000f1140	Shifting through the “array”

sub R4, R5, R15	00af2900	R15 is 1, so we are iterating through each element
addi R11,R0, 0	000002c5	
add R5,R5,R11	008b2940	Storing the address of the array in R5, since the first location in memory is 0
add R4,R4,R11	008b2100	Calculating next memory location
lw R6,0(R5)	0000298e	Saving the value of the first element in memory to R6
lw R7,0(R4)	000021ce	Saving the value of the first element in memory to R7
blt R7,R6,loop2	ffe63d52	If the first element is smaller, dont swap and go back to inner loop.
sw R6,0(R4)	0006200f	Storing the first element in the second location
sw R7,0(R5)	0007280f	Storing the second element in the first location
jalr R0,R0,loop2	0006000d	Jumping back to inner loop
end:		

- **Teamwork**
  - **Single-cycle Processor**

	Mohammed Al Sabaa	Ahmed El Tayeb	Sajjad Al Saeed
Design	**30**	**30**	**40**
Implantation	**30**	**30**	**40**
Simulation	**33**	**33**	**33**
Testing	**40**	**40**	**20**

  - **Pipelined Processor**

	Mohammed Al Sabaa	Ahmed El Tayeb	Sajjad Al Saeed
Design	**30**	**30**	**40**
Implantation	**30**	**30**	**40**
Simulation	**33**	**33**	**33**
Testing	**40**	**40**	**20**

- **Conclusion**

  It was such a great project to work on. We learned many things and most importantly the project helps us to under our lectures better based on our work. Logisim is great application that has many prebuilt circuits such as registers, Decoders, and ROM. We did not need to build any of those, and we could focus on building processor components. We tried to consider simplicity on our designs, so they can be easy to understand by every group member and the instructor. Using tunnels was the key to make designs clear of spaghetti wiring. However, some deigns were needed wiring to keep track where an input is going. We face a problem where the Register File was not working, and it appeared the problem we copied an input and use it as output without changing it to output. We spent several hours divided by two days to understand how the Control unit exactly works. After we understood it and studied together, we started fill ROMs the needed number. I-type instructions were having a problem of skipping to the next immediate instruction. We found out we were missing a register for the immediate 16. Going back to the control unit, we noticed shift immediate instructions were not working correctly, and the problem was an incorrect code signal. In Hazard detector, we implemented a very hard circuit to output the number of the forwarding. Thanks to our instructor, Mr. Al Salah, we learned how to do a simple way than what we did, and it was by either using Tri-state buffers or using multiplexers. The project was fun and exciting to work on especially with great teammates that are cooperative and ready to learn.