Faculty of Information Technology Computer Systems Engineering Department

ENCS437

Computer Architecture

Project num.2

Name: Ibrahim Injass **ID: 1171148**

Name: Yousef Ghanem **ID: 1172333**

Name: Mahmoud Sub Laban **ID: 1172144**

Instructor: DR. Aziz Qaroush

Date: 12/6/2021

ABSTRACT:

In this project We go to learn how Designing and testing a simple 16-bit Pipelined RISC processor with eight 16-bit general-purpose registers Using the Logisim simulator.

Table of Contents

ABSTRACT:.....2
Part1: Design and Implementation.....4
ALU Control.....6
Controller unit.....7
Execution Buffer.....8
Extender Block.....9
Pc Control.....9
Forwarding Unit.....10
Final Datapath:.....11
1) Fetching Stage.....11
2) Decoding Stage.....12
Executing Stage.....13
MEMORY STAGE.....14
Write Back stage.....15
Simulation and test.....16
Teamwork:.....21
Conclusion.....22

Part1: Design and Implementation

Pipeline structure processor has five stages:

1. Fetching.
2. Decoding.
3. ALU.
4. Memory.
5. Write Back stage.

We designed these main blocks that is we will use it in full Datapath pipeline:

1. ALU.
2. Register file.
3. Memory.
4. Controller unit.

moreover, we use other blocks like multiplexers.

Firstly, we will talk about ALU block. from **fig1**

that is show deign ALU we notice Our ALU has five operations (ADD, SUB, AND, OR and a Comparator) and we used the output of the comparator to determine the max value is SLT instruction, we also used the output of the Adder in LWS

flags: the zero flag for the Branch Equal.

JR Flag is used to send the result of the ALU to the PC register. We used it in the PC control block.

A MUX is used to multiplex the needed operation according to the 3-bit ALU control (Function), Both the sources and the output are 16-bit.

As an optimization that we did was that we gave two inputs from the MUX (3 & 4) to the ADD operation so that we could minimize the truth table of the control unit.

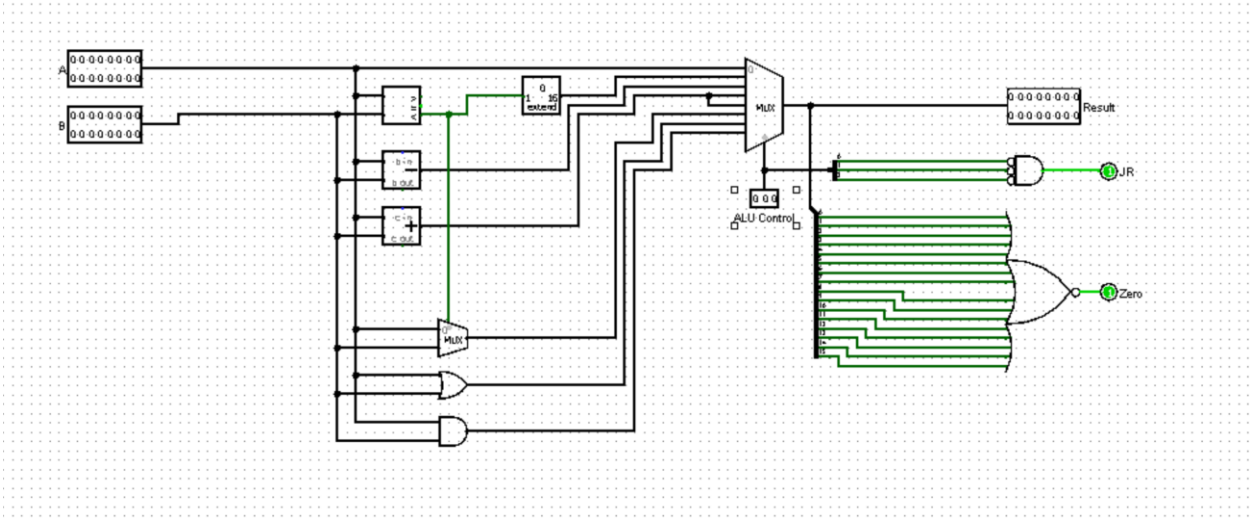


Fig1

Secondly, we will talk about Register file block. from **fig2**

we mad a register file block to load and store data to registers, it contains 7 registers.

for the load process we select which register from the values of (RA) and (RB) and write its data on Bus A and Bus B.

for the storing process we take data from Bus W and store it in a register that has been selected by RW and it would only write if the RegWrite is enabled (1).

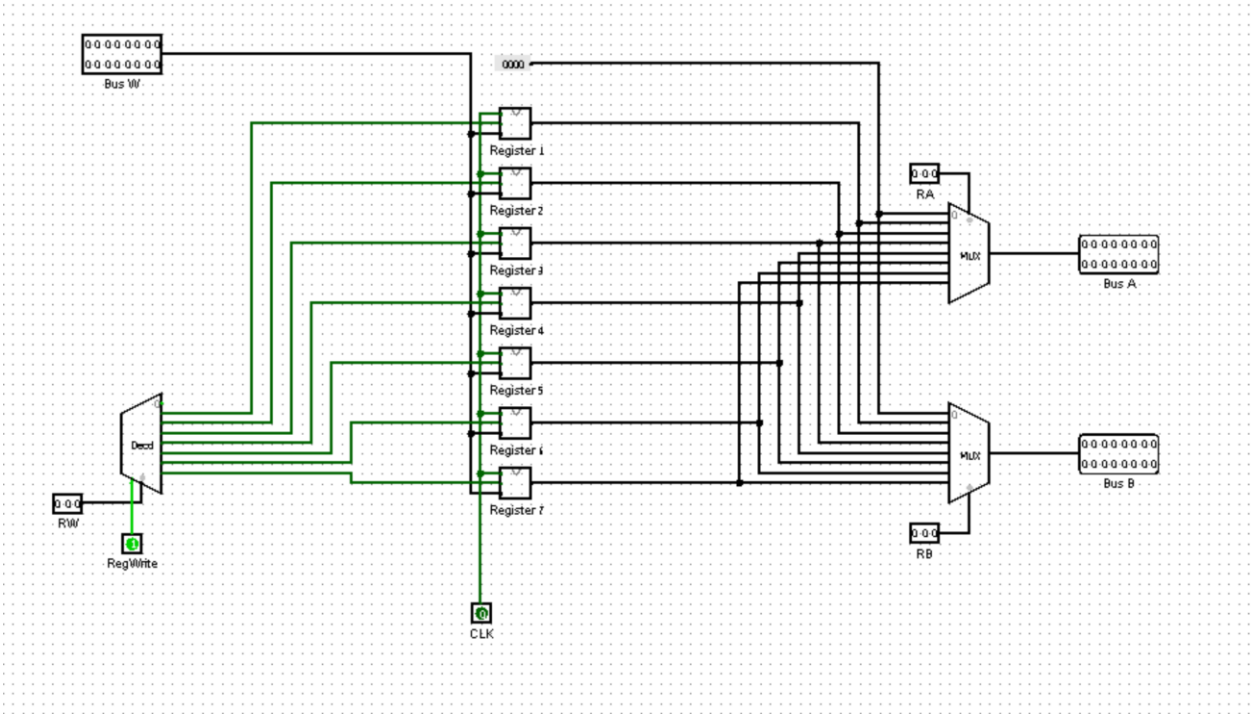


Fig2

ALU Control, from **fig3**

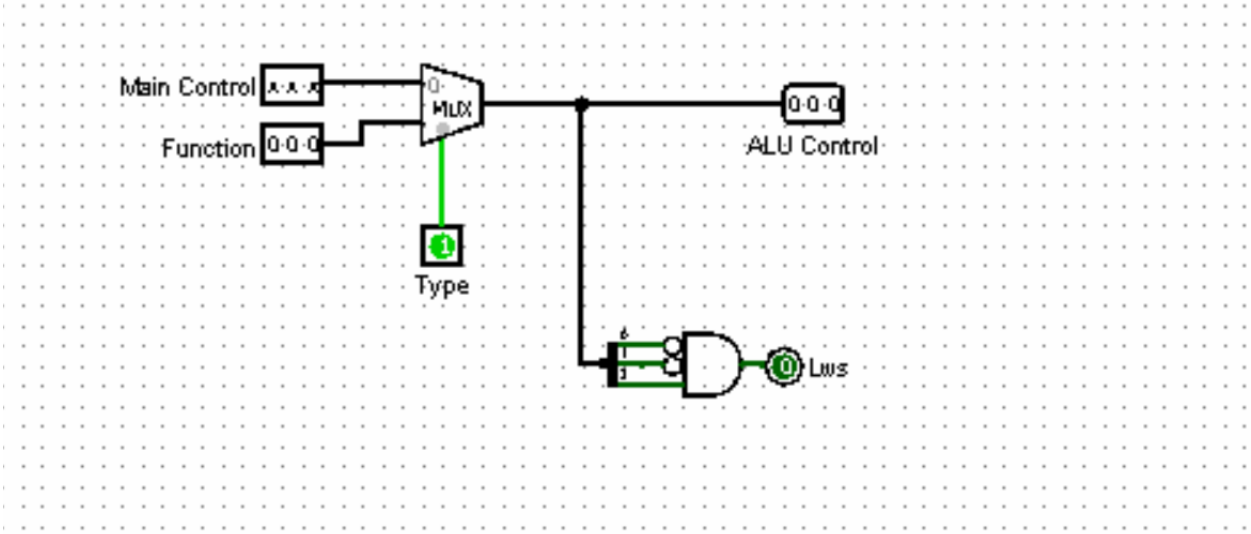


Fig3

This block is used to control the function of the ALU Block we talked about previously.
We have three inputs: -

1)Main control Which we got from the control Unit that is used
In None R type instructions.

2) Function that we take it from the first three bits of the instruction to decide which ALU operation.

3)Type input that works as a select line that we take it from the Control Unit
To determine if the instruction is R-type or I-Type.

We have Two outputs: -

1) ALU Control is a selected input from the Mux that will be used in the ALU block.
2) LWS Flag is to determine if the instruction is LWS (load) or not.

Controller unit block. from **fig4** that is show deign of Controller unit we notice Our Controller unit has one 4-bit input that is (Opcode)
To decide which instruction out of the 11 instruction we have.
Every instruction has an output flags and we did a truth table to know which operation are enabled (selected)
And we used an encoder to get the function to the ALU.

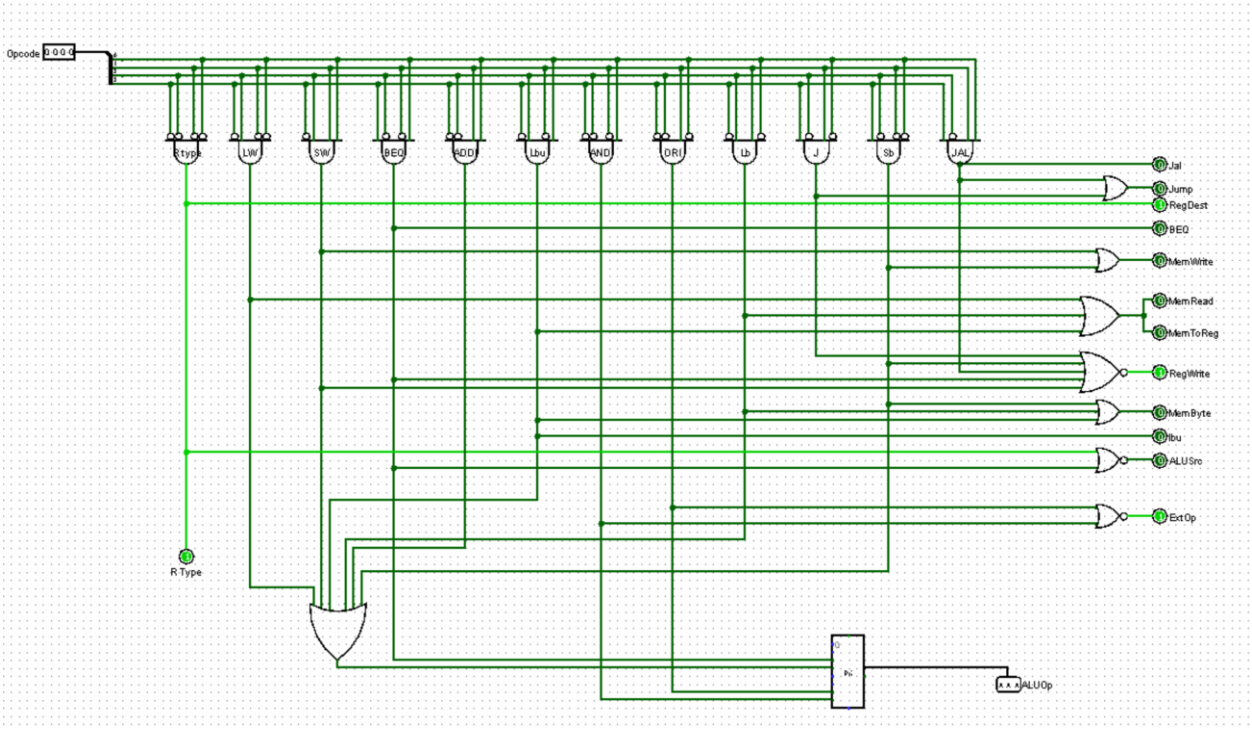


Fig4

we will talk about **Byte extender block.** from **fig5**
we used the Byte extender when we use LB/LBU instructions to extend the value from 8-bit to 16-bit (from 0-7 was taken using a splitter)

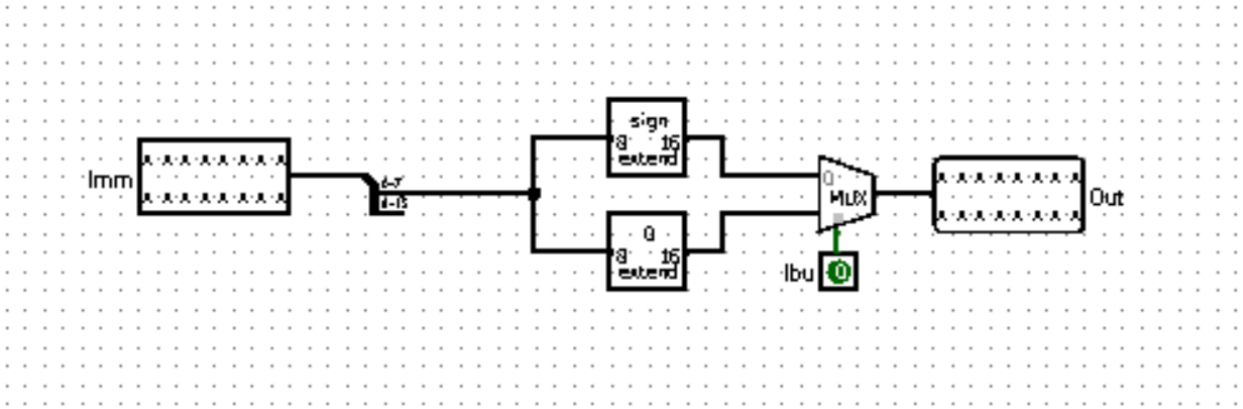


Fig5

Execution Buffer **in Fig 6** for an example to specify the four buffers we used (IF buffer, MEM buffer, ID BUFFER and the EX-buffer we want to specify).
We Used the buffers as a gate between stages to prevent each stage information to cross to the next stage before the current procedure ends and then it will open on the rising edge of the clock. that will be the enable for the D flip flop.
Every buffer works the same so this one is a demonstration of them all.

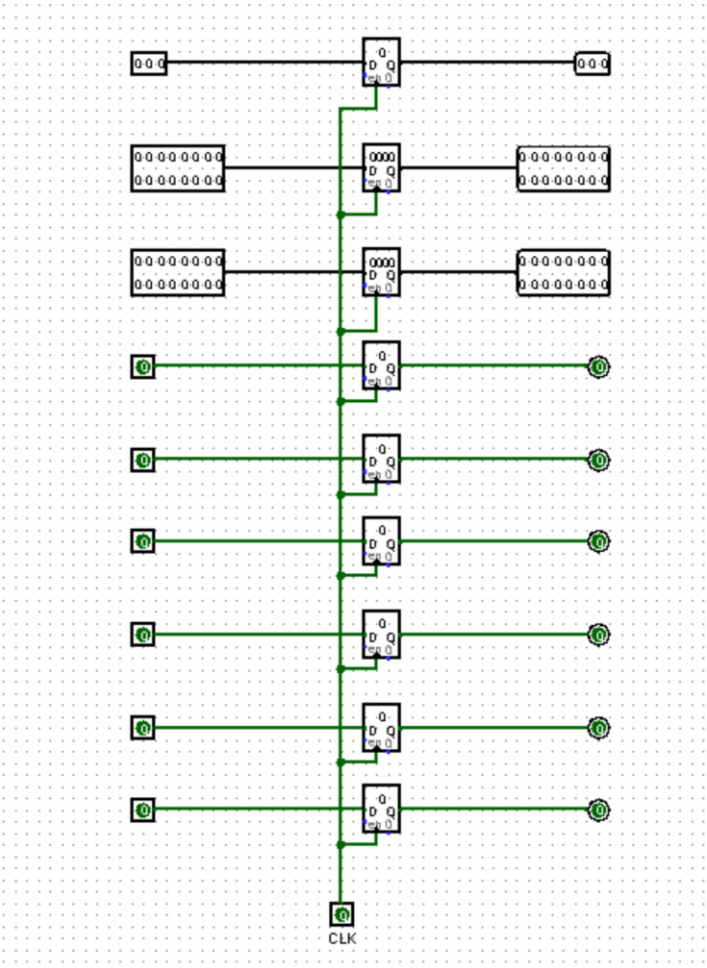


Fig6

Extender Block **in fig 7 is used to extend the immediate value to 16 bit for I-type instruction For both sign and Unsigned which is chosen through the select line (EXtOp flag) that we get from the control unite.**

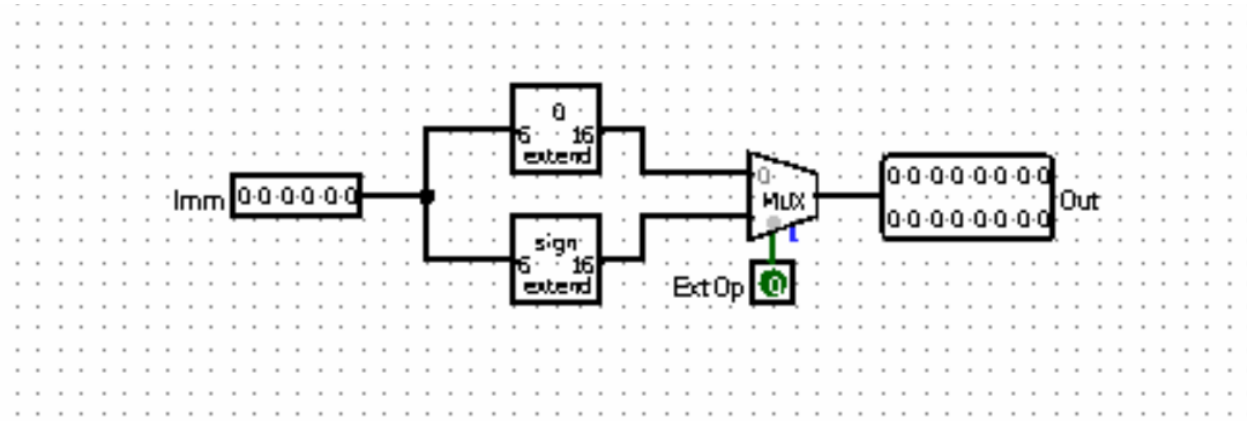


Fig7

Pc Control **in fig 8**

In the PC control block we used multiple inputs, jump and BEQ are taken from the control unit and zero and JR are taken from the ALU.
we used the Rs input to separate the call of the JR instruction from a zero instruction.
The output of the block goes to a mux which determines what is the next PC address

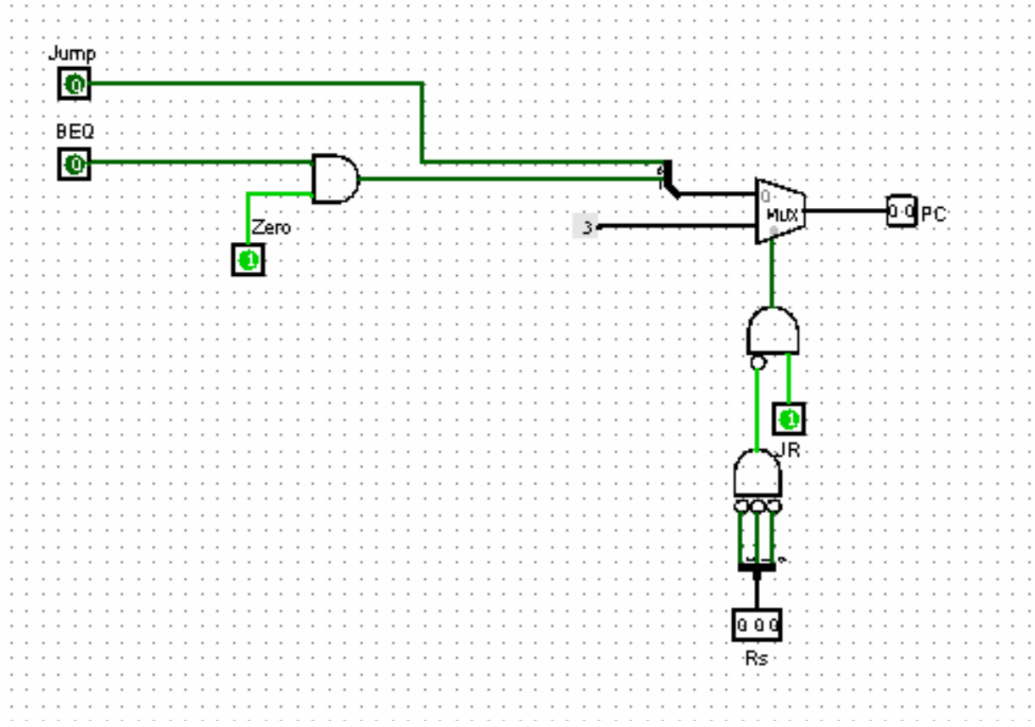


fig 8

Forwarding Unit:

We used the forwarding unit to detect any data hazards, we used Rs and Rt taken from the ID buffer to compare them to the Rd taken from memory stage and write back stage. So instead of using any stall cycles we used the output of this block to determine what input goes into the ALU (if there were any dependencies and on which inputs).

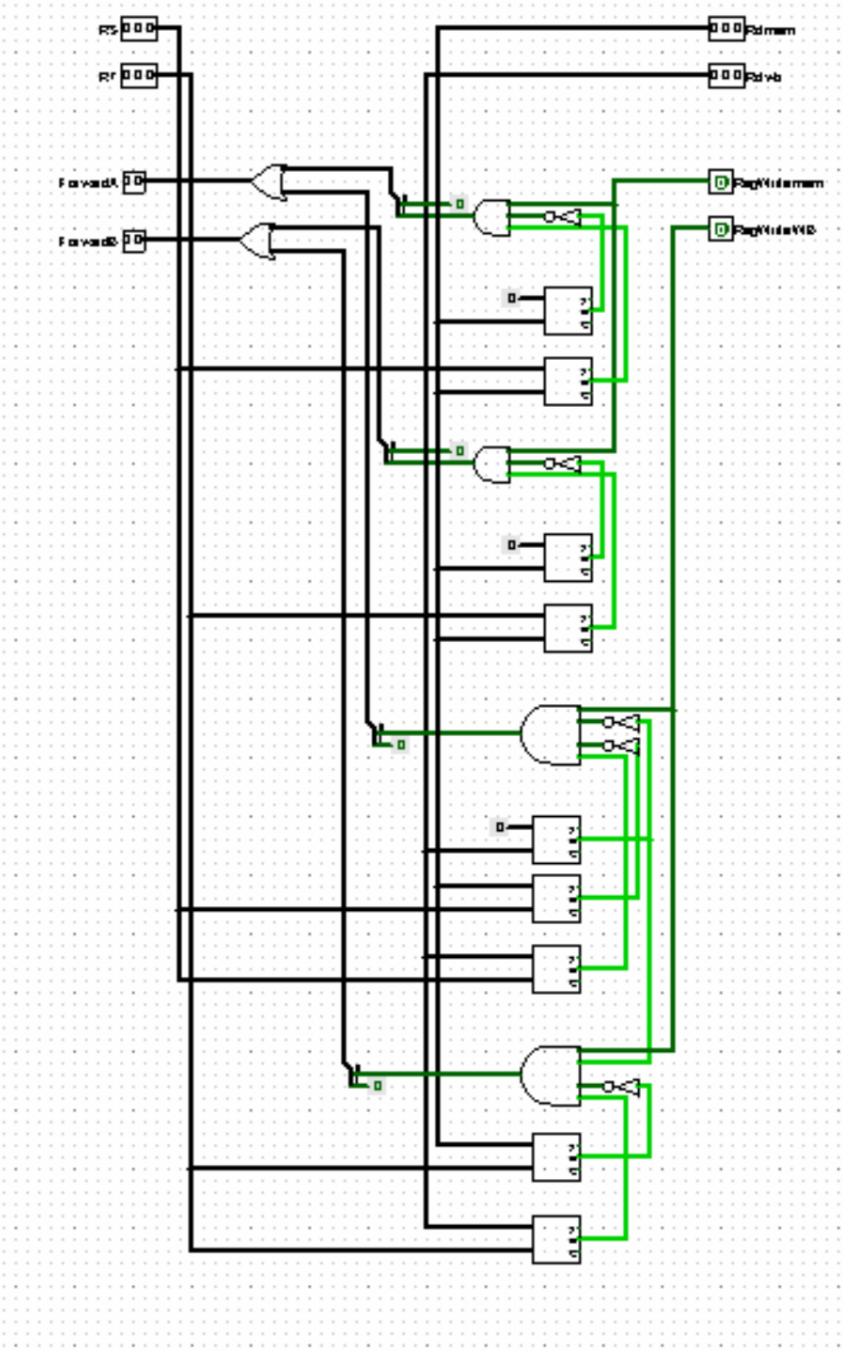


Fig 9

Final Datapath:

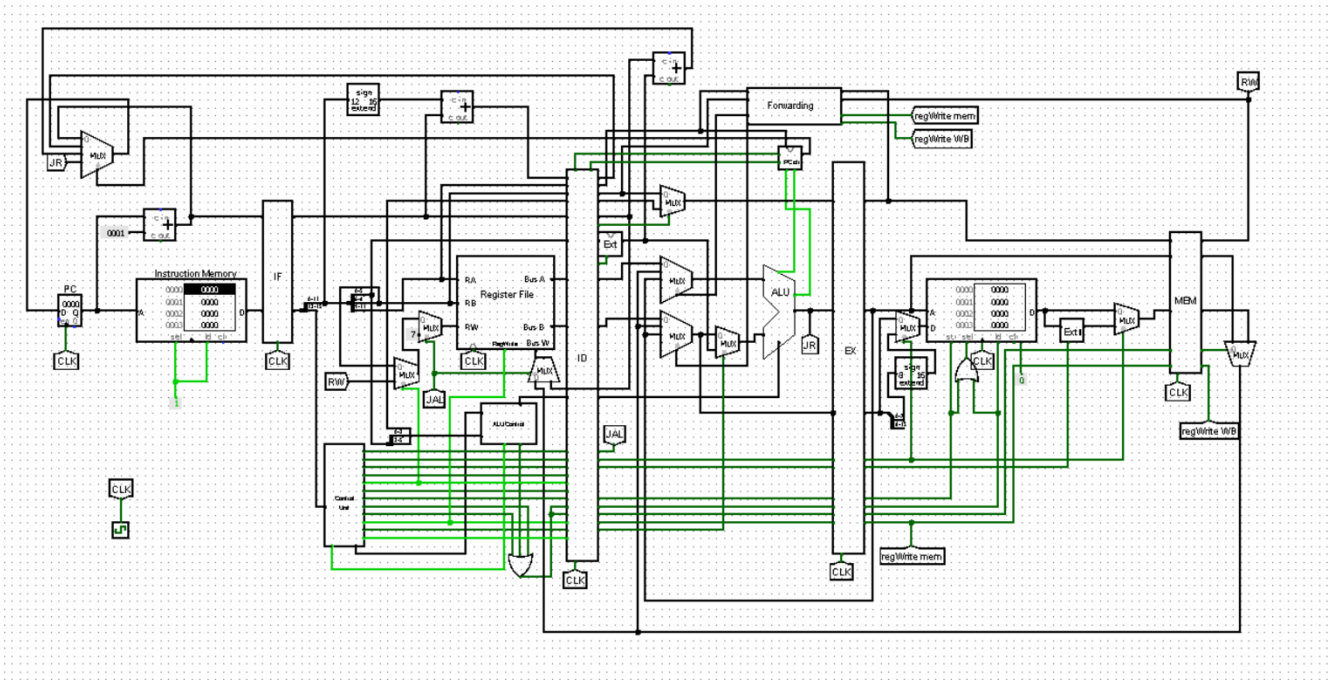
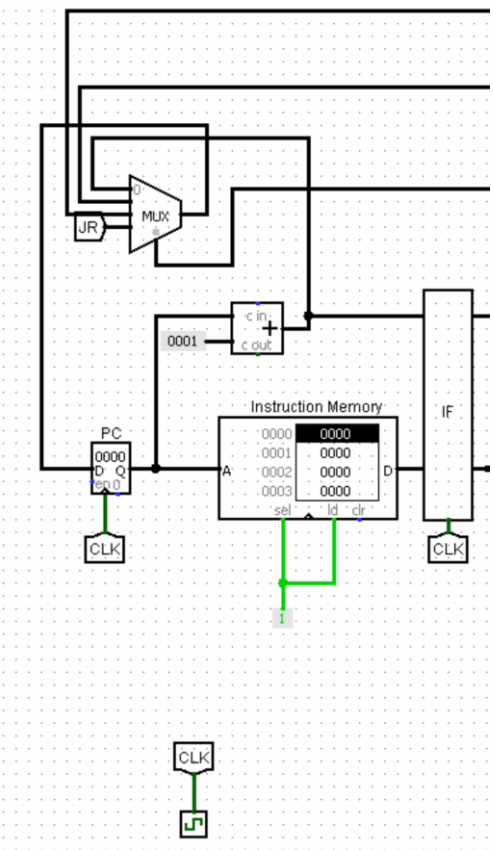


Fig 10

This project is built on five stages that will be specified here.

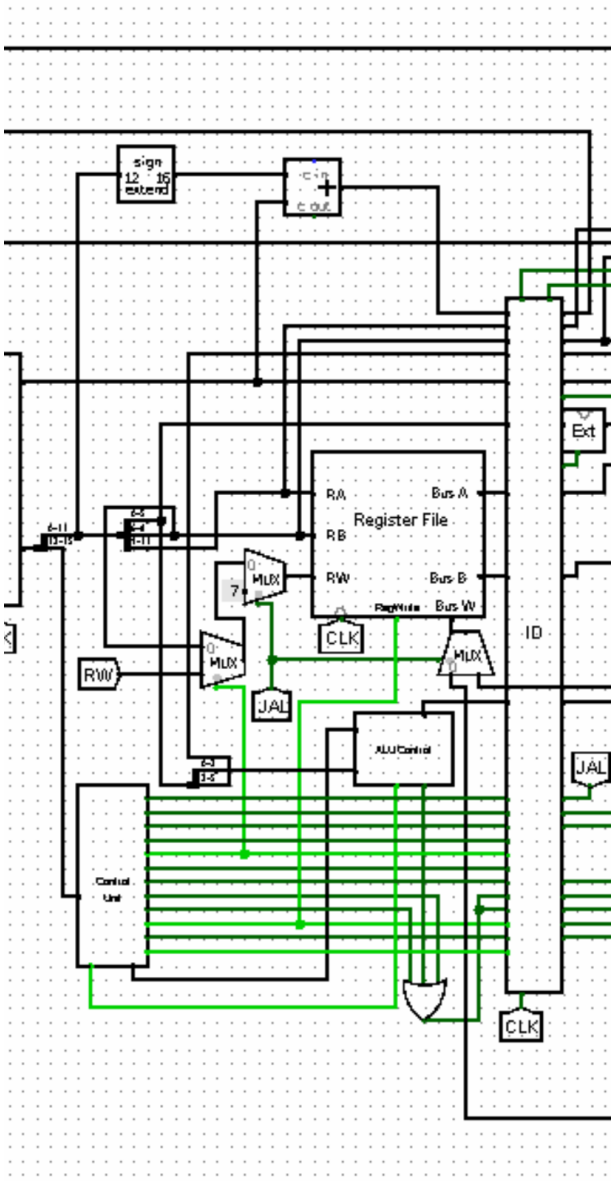
1) Fetching Stage

In this stage we take the address from the PC to the Instruction memory, in the same time the PC is added to 1 and returned to a mux (it has inputs from other stages and it select come from PC controller). the instruction memory loads the instruction to the buffer to get it into the next stage as well as the added PC



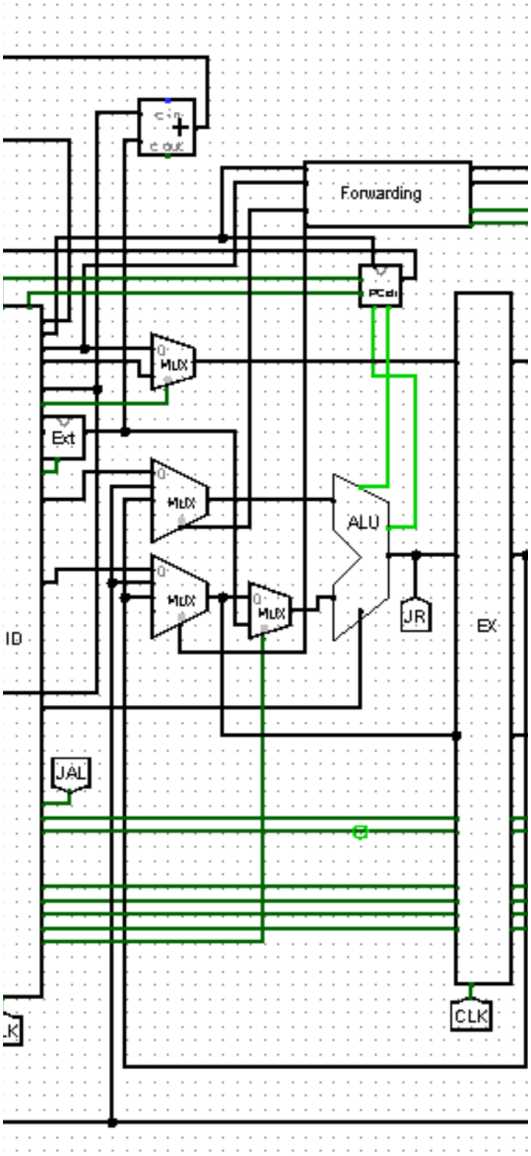
2) Decoding Stage

In this stage we took the instruction from the previous stage and spitted it taking in case all the types, in all instructions the last 4 bits are opcode and sent to the control unit to decode it. The rest of the instruction is spitted and as partially sent to register file and to the next stage using the buffer.
in addition, we took in consideration the Jal instruction which saves the current pc to R7, as well as we took one input of RW from the Write back stage to store a value to a register when needed.



Executing Stage

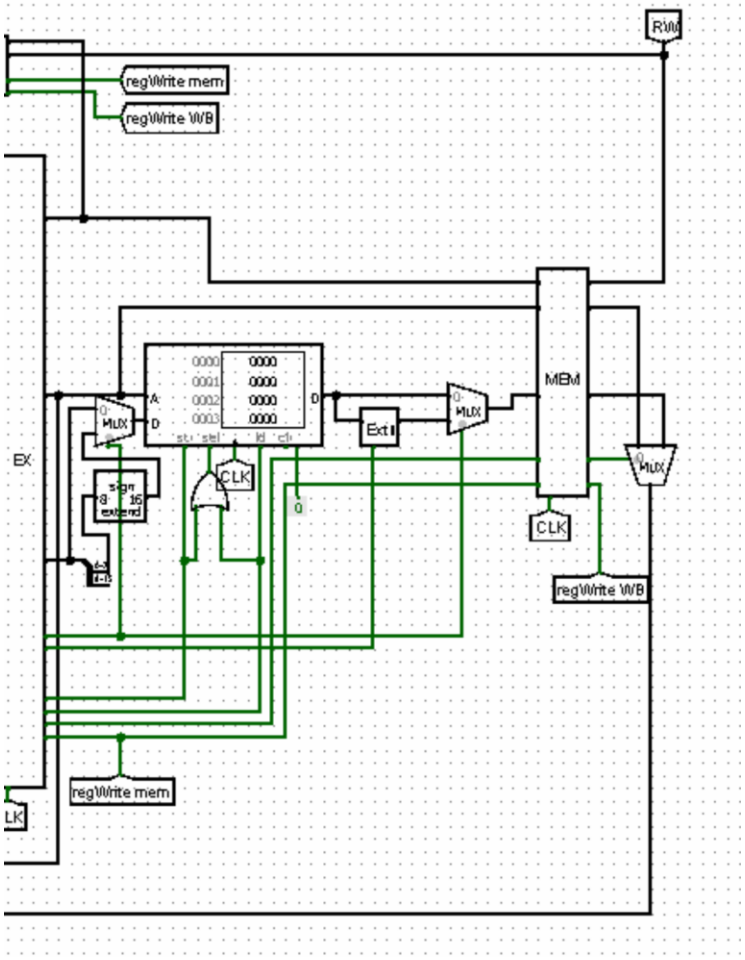
In this stage we took the outputs of the register file into two muxes which takes inputs for the ALU, the other outputs of the muxes are the output of the ALU of the previous stage and the output of the write back stage for both inputs, the selection is done by the forwarding unit.
to output are sent from the ALU to the pc control block.
the second input of the ALU take another possibility from the output of the extended immediate and selected by ALUsrc from the control unit.
other outputs of the buffer are forwarded to the next stage.



MEMORY STAGE

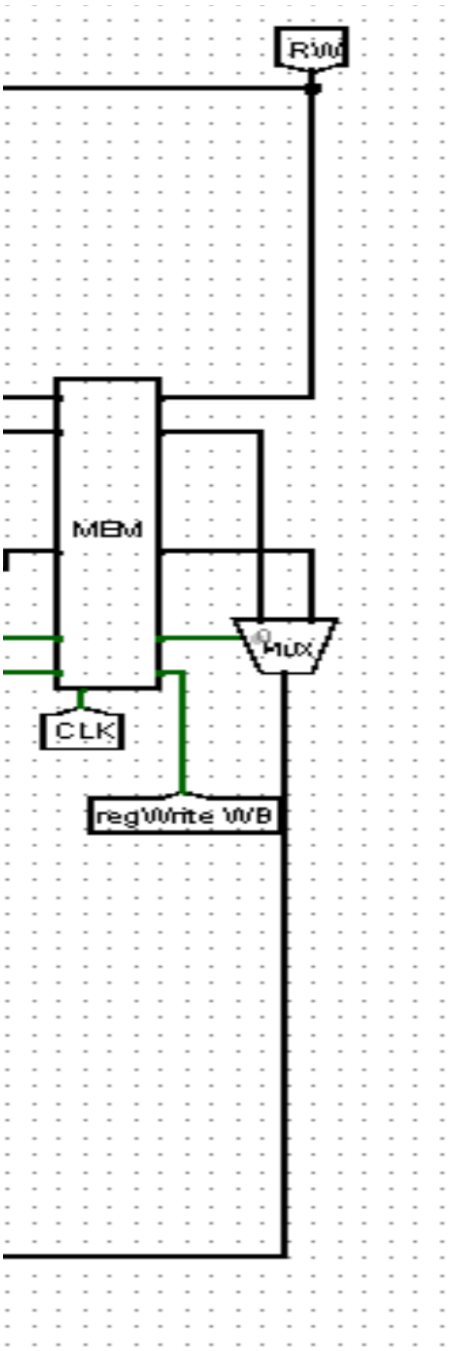
In this stage we took the output of the ALU of the previous stage as the address to Data memory (used in load and store instructions), the 2nd input of the ALU is forwarded as data in the data memory which is extended if we Used Sb instead of Sw. the data memory takes str and LD flags from the control unit and works if one of them is enabled.
for the output data from the memory, it is extended if the instruction used is LB or LBU instead of LW and selected by a flag from the control unit.
if the instruction is arithmetic (not store or load) the ALU result is forwarded to the next stage.

the regWrite in this stage is took as an input in the forwarding block.



Write Back stage

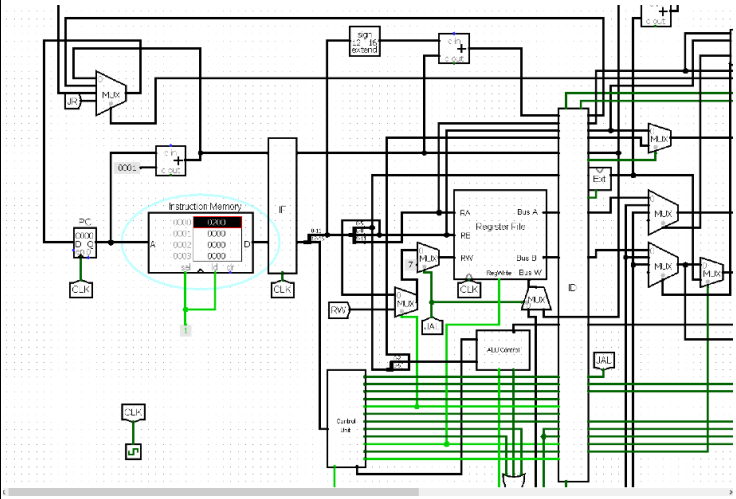
In this stage we take the output of the ALU from the previous stages and the output of the data memory into a mux which is selected by memToReg from the control unit. The output of the mux is sent to Bus W in the Register file and also as inputs in the ALU. a Rd value forwarded from previous stages called RW is sent to the register file to write to a specific register if needed. Also, the regWrite flag forwarded from previous stages is used in the forwarding unit

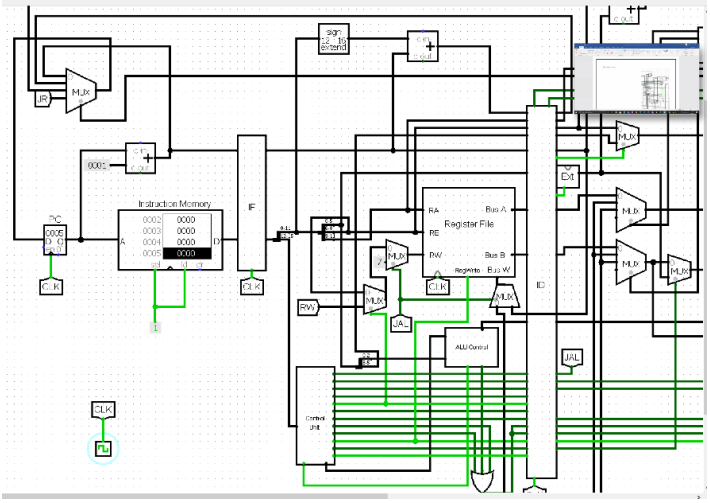


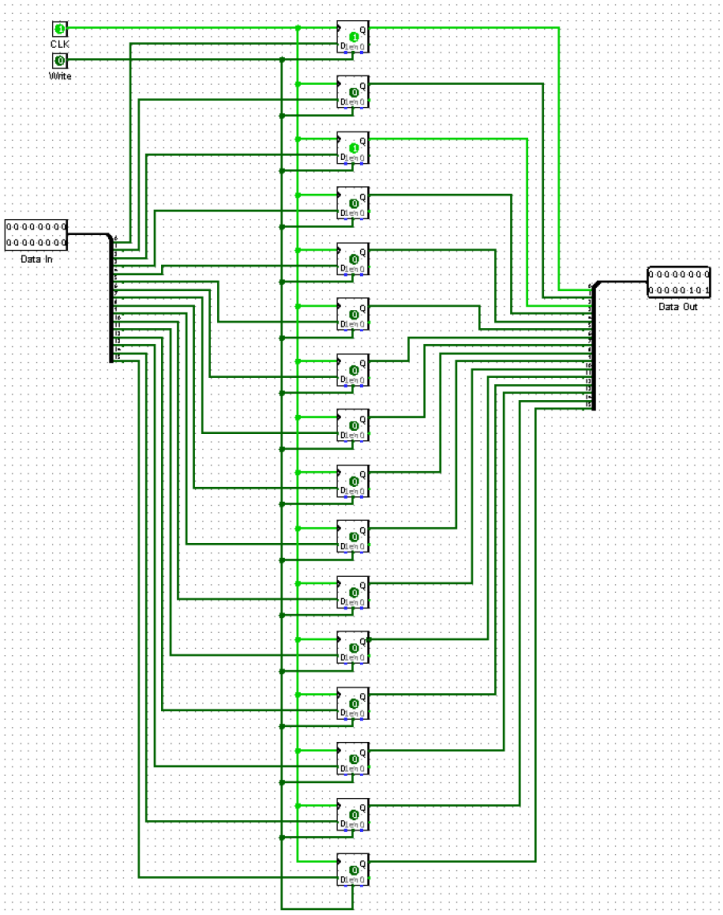
Simulation and test

We have made some tests with these cases:

1- JR instruction is (0000 0010 0000 0000)

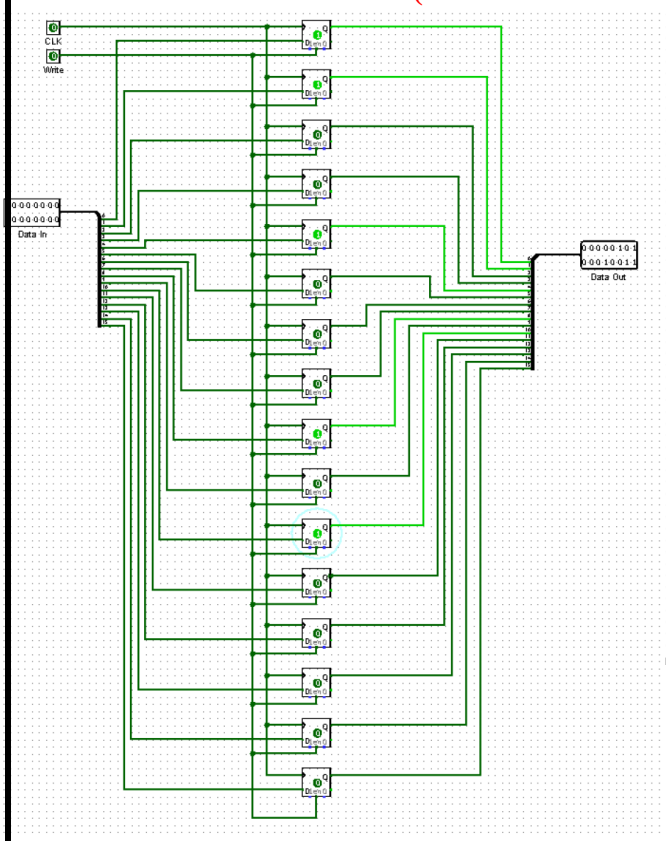


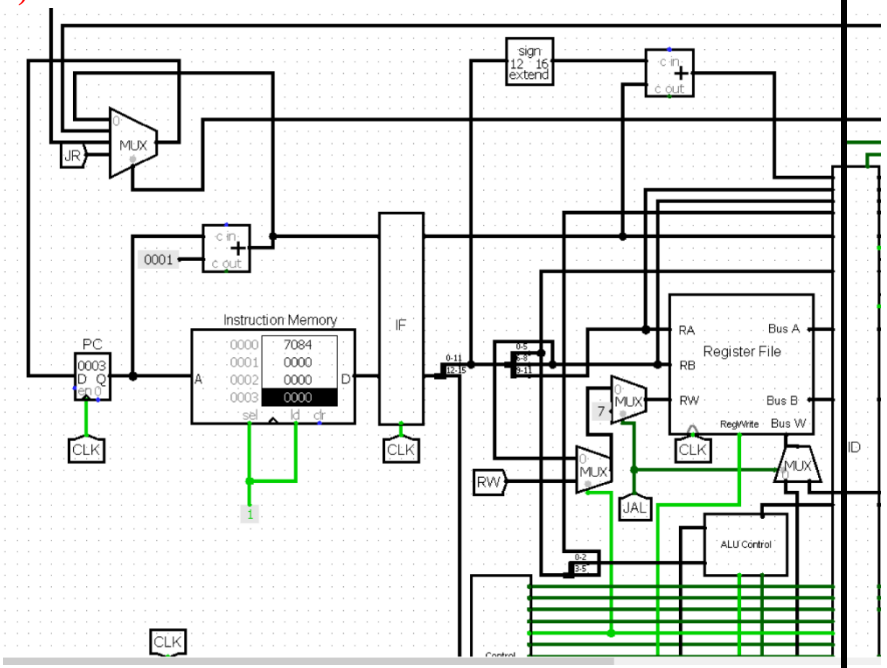


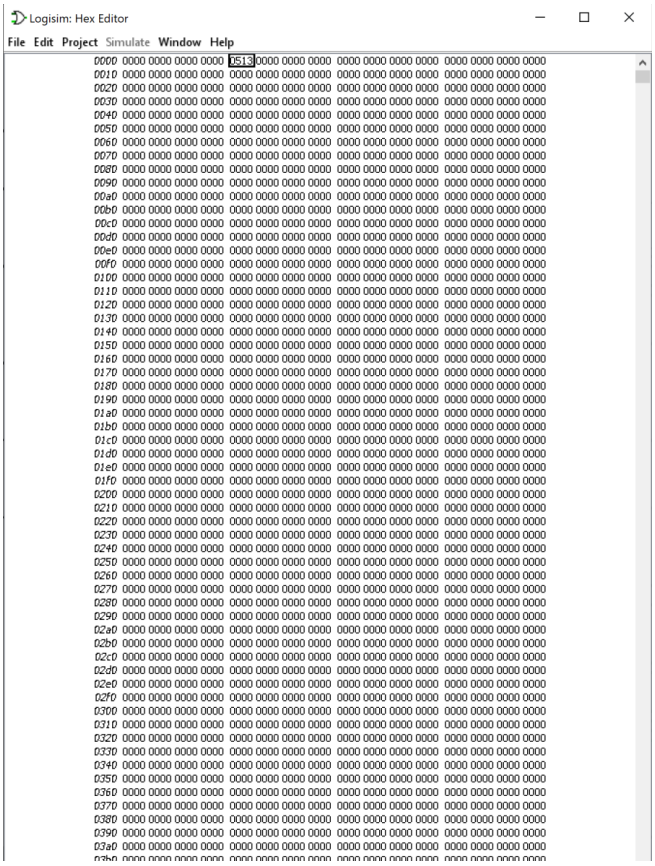


We used JR instruction and returned the Register R1 as the PC.

2- SW instruction (0111 0000 1000 0100)

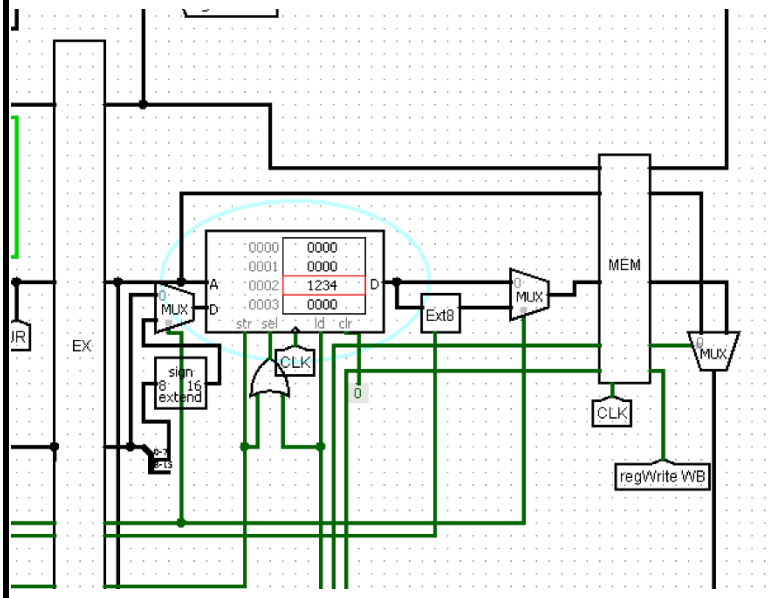


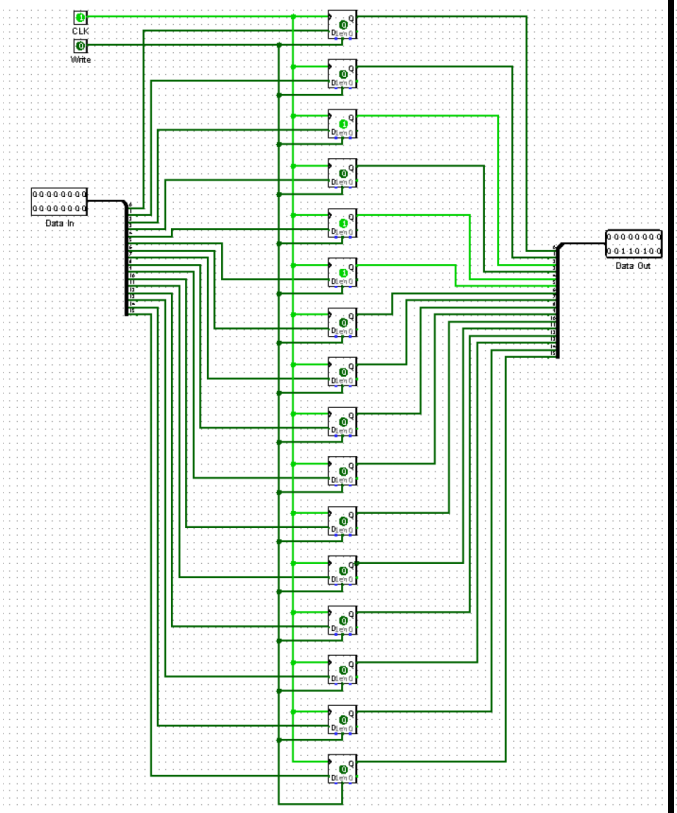


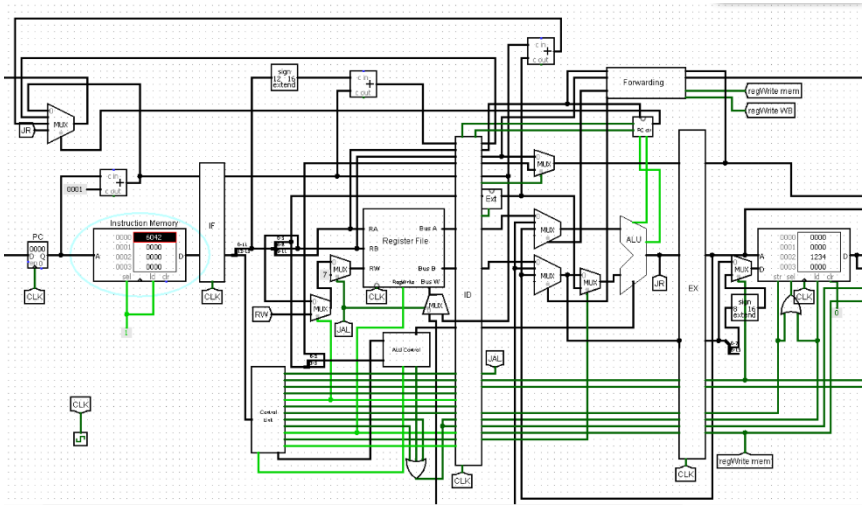


We used SW instruction to store the contents of Register R2 into memory of address 0004.

3- LB Instruction (0110 0000 0100 0010)





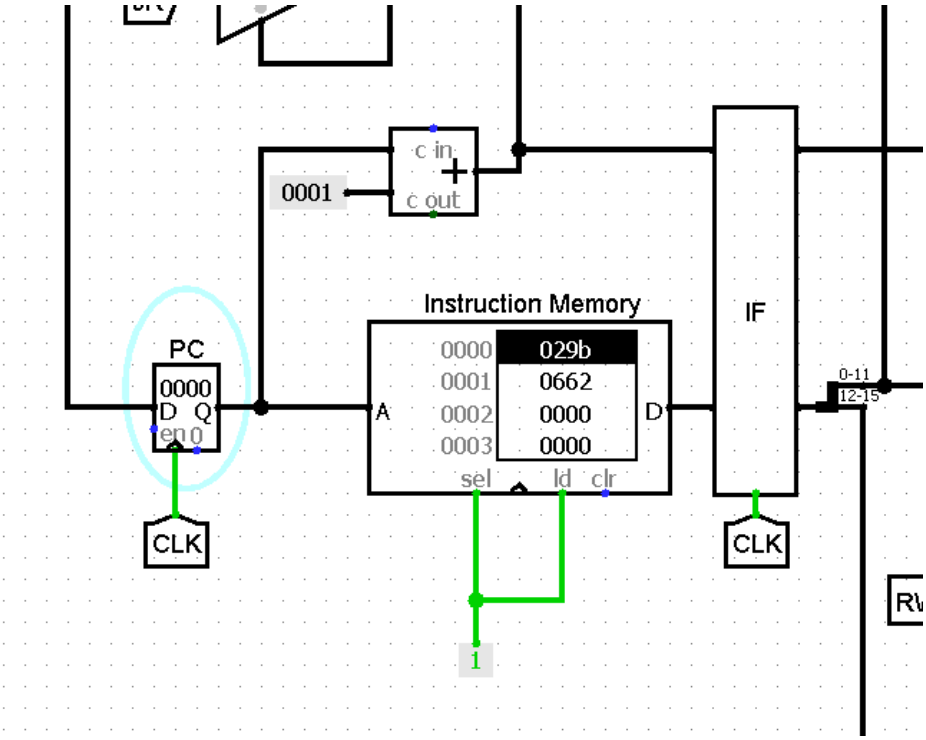


We used LB Instruction to load contents of memory of address 0002 into Register R1 of bits (7:0)

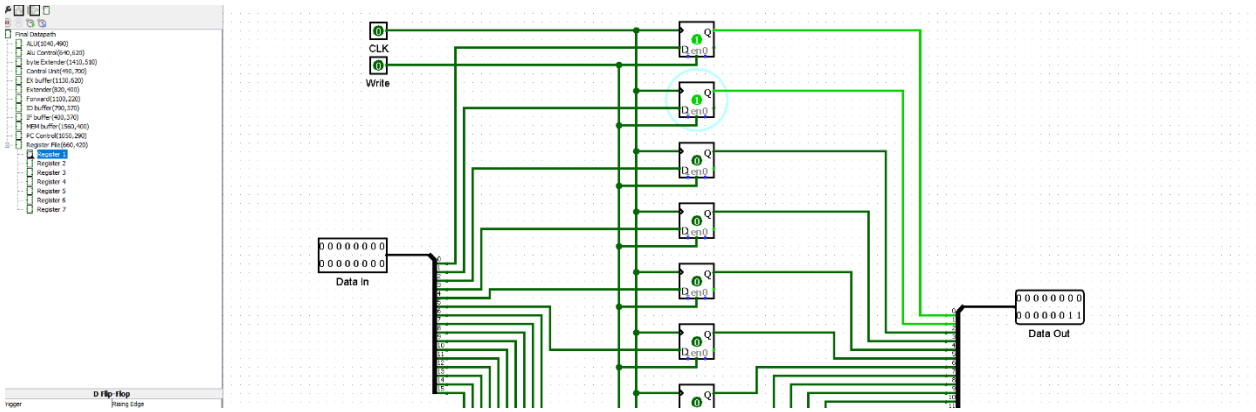
4- We did two dependent instructions to test forwarding case:

Add instruction (0000 0010 1001 1011)

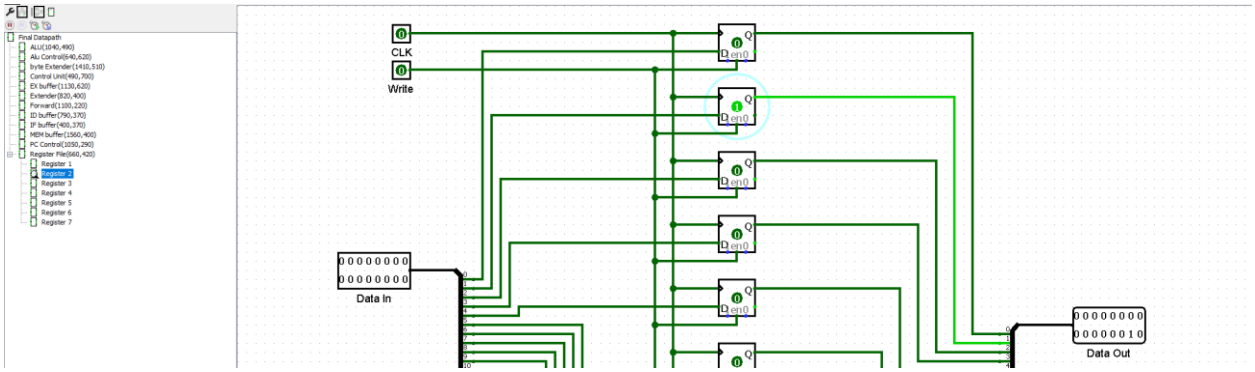
Sub instruction (0000 0110 0110 0010)



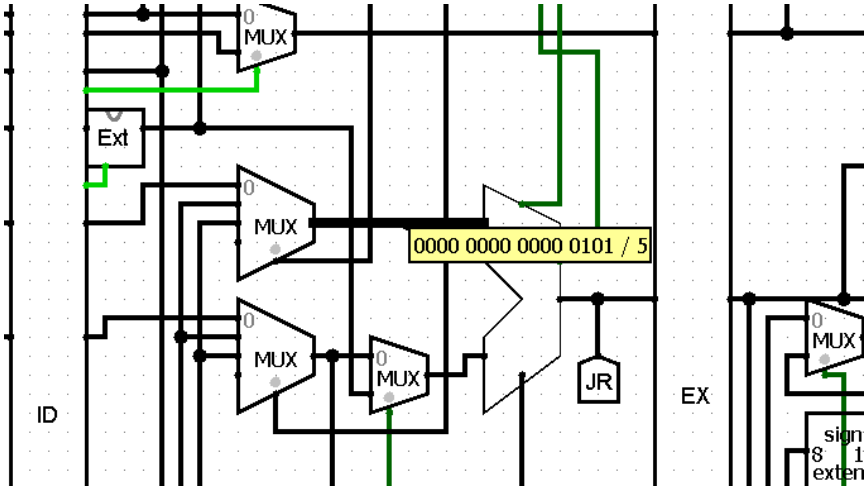
R1:



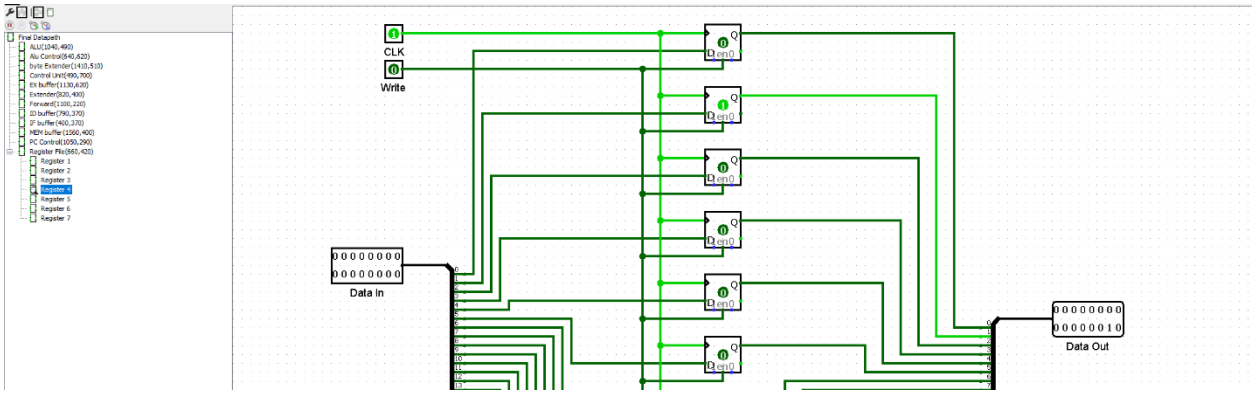
R2:



Input of register R3 in Sub instruction:



R4 after both instructions:



Teamwork:

In general, most of the project was done as a team (Zoom meetings), some of the units were done separately with help of partners.

The units done separately:

- Control Unit is done by Mahmoud Sublaban
- ALU is done by Ibrahim Injass
- Forwarding Unit is done by Yousef Ghanem.

Simulations was done by Ibrahim Injass and Yousef Ghanem simultaneously with Mahmoud adding them to the report.

Conclusion

In this project we learned a lot about designing in Logisim simulator and we designed and simulated a simple 16-bit pipeline RISC processor that is a little like MIPS processor we learned in the course. with eight 16-bit general-purpose registers: R0 through R7. The architecture allows predicated instruction execution and multiple load and store execution. There are 3 types of instruction formats (namely R, I and J).