Faculty of Engineering

Computer Engineering

Pipelined MIPS Processor Implementation

Instructor:

DR. Gihan Naghib

Eng.Jihad Awad

presented by:

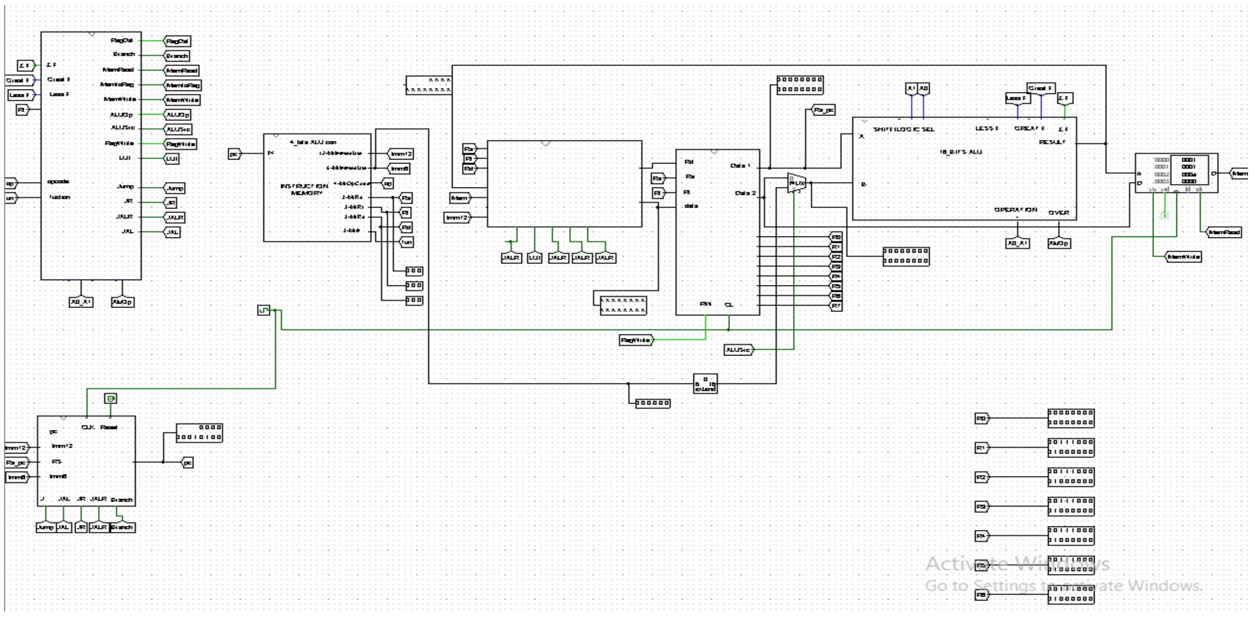
Hamad Sayed Mariam Mohamed Heba Sabri

phase 1: Single Cycle Processor:

1. *Design and Implementation* :

1.1 Data Path :

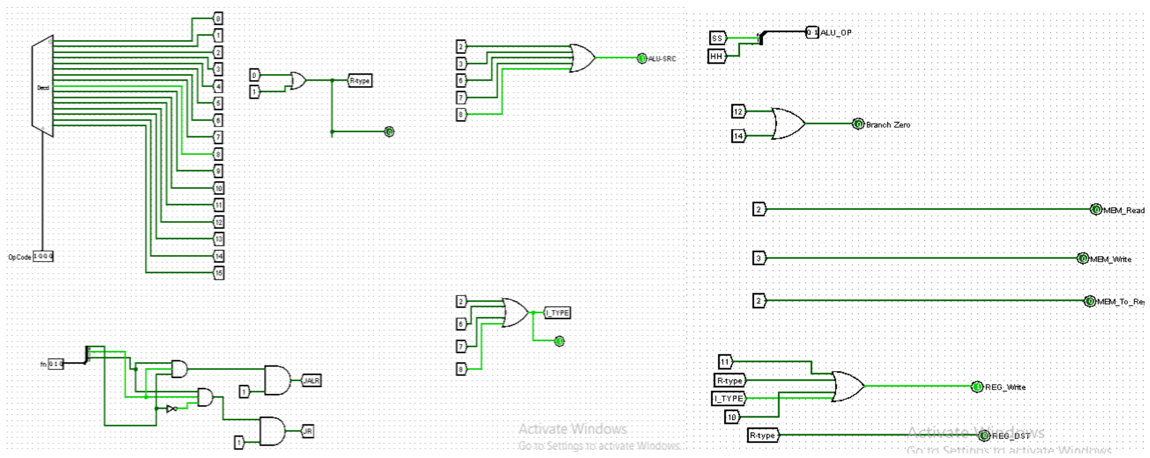
The design of the CPU is a basic MIPS design with a subset of the instruction set. As our designed ISA, Our CPU should be able to do 32 instructions, 16 of which are R-type, 13 are I-type, and 3 are J-type. The general data path starts with fetch instruction word from memory address in PC , then fetch the register file to read the source register(s) used for the instruction. Next, it depends on the instruction type, with R-type both values come from the registers and go through the ALU to compute the instruction operation and then stores it into . With I-type, one of the operands is an immediate value instead of register like the R-type's instructions. In both R and I types, they write the result of the ALU into the designated destination register except for the "sw" which saves into data memory. With J-type instructions, after the result of the ALU (the new PC address) is computed, it sent back around to the PC at which point the PC control unit sends the selector for the mux to accept the branched or jump PC address.

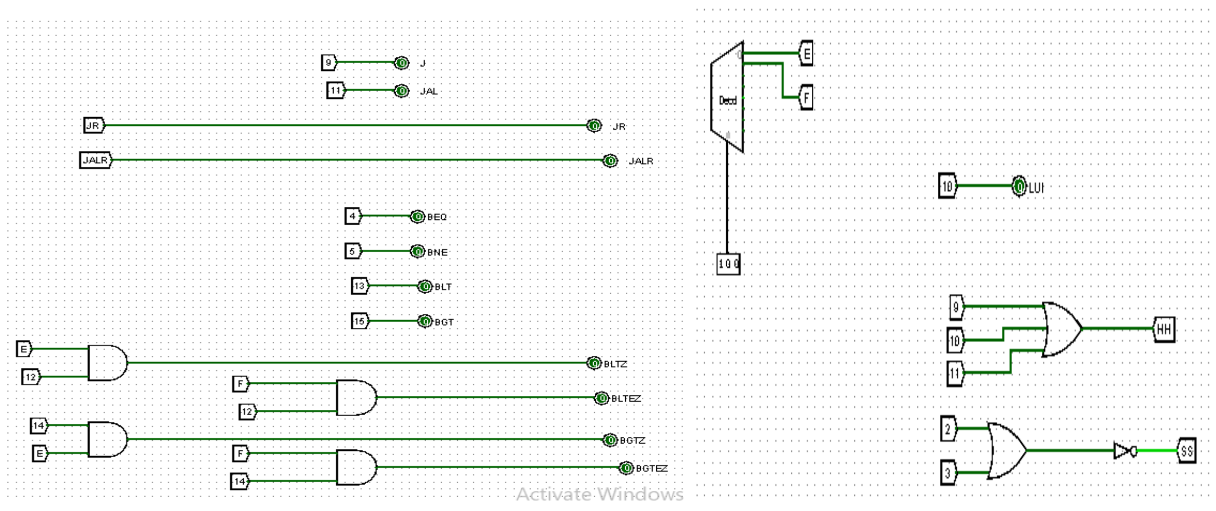


1.2 Control Unit :

There are four units for controlling the CPU: The Main Control unit, the ALU Control, and the PC Control. The Main Control unit handles the opcodes from the current instruction and enabling all of the relevant signals so that the rest of the processor will work as expected as described by the MIPS spec.Some of the signals of the Main Control unit are also fed into the PC Control unit (J, JR, JAL, JALR, BEQ, BNE) and the ALU unit (ALUSrc, ALUOp) . The PC Control unit handles getting the next address that the PC register should save for the next cycle of the processor . The PC Control unit takes the signals of Jump, BEQ, and BNE, as well as Zero flag signal and computes from it a selection value to activate a channel in its multiplexer to select whether the PC will increment, jump, or branch. The ALU Control unit takes the signals ALUSrc and ALUOp from the Main control unit, and based on those signals, retrieves two data pieces A and B, where A is always from a register, and B could be either from a register or directly from the immediate value embedded in the instruction as selected by ALUSrc, and then based on the ALUOp value the ALU Operation is selected and then performed.

Main Control

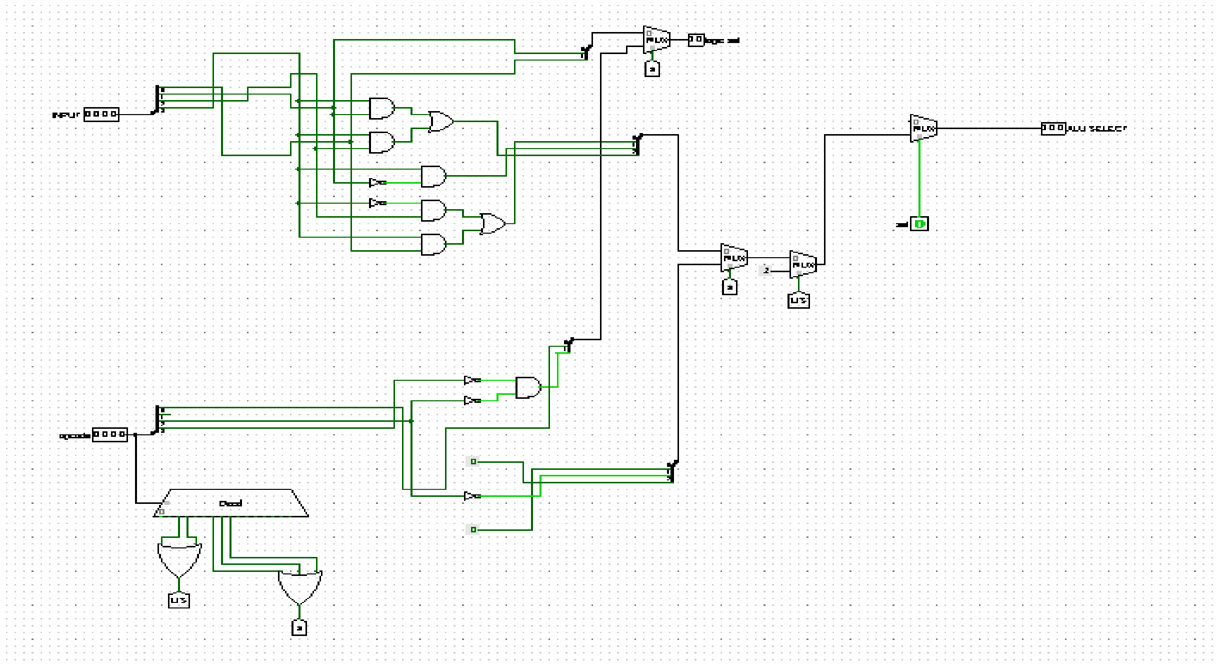




Control Signal

	opcode	Reg1	Reg2	Reg3	EX	Imm	RegWr
AND	0	0	0	0	x	x	1
OR	0	0	0	0	x	x	1
XOR	0	0	0	0	x	x	1
SLL	0	0	x	0	0	0	1
ADD	1	0	0	0	x	x	1
SUB	1	0	0	0	x	x	1
SLT	1	0	0	0	x	x	1
JR	1	1	1	x	x	1	0
JALR	1	x	x	1	x	1	1
Lw	10	0	x	0	1	0	1
Sw	11	0	1	x	1	0	0
Andi	110	0	x	0	0	0	1
ORi	111	0	x	0	0	0	1
BEQ	100	0	0	0	x	x	1
BNQ	101	0	1	x	1	0	0
BLTZ	1100	1	x	x	1	1	0
BLEZ	1100	1	x	x	1	1	0
BGTZ	1110	1	x	x	1	1	0
BGEZ	1110	1	x	x	1	1	0
J	1001	x	x	x	x	x	x
JAL	1011	1	x	1	x	x	1
LUI	1010	x	x	1	0	1	1

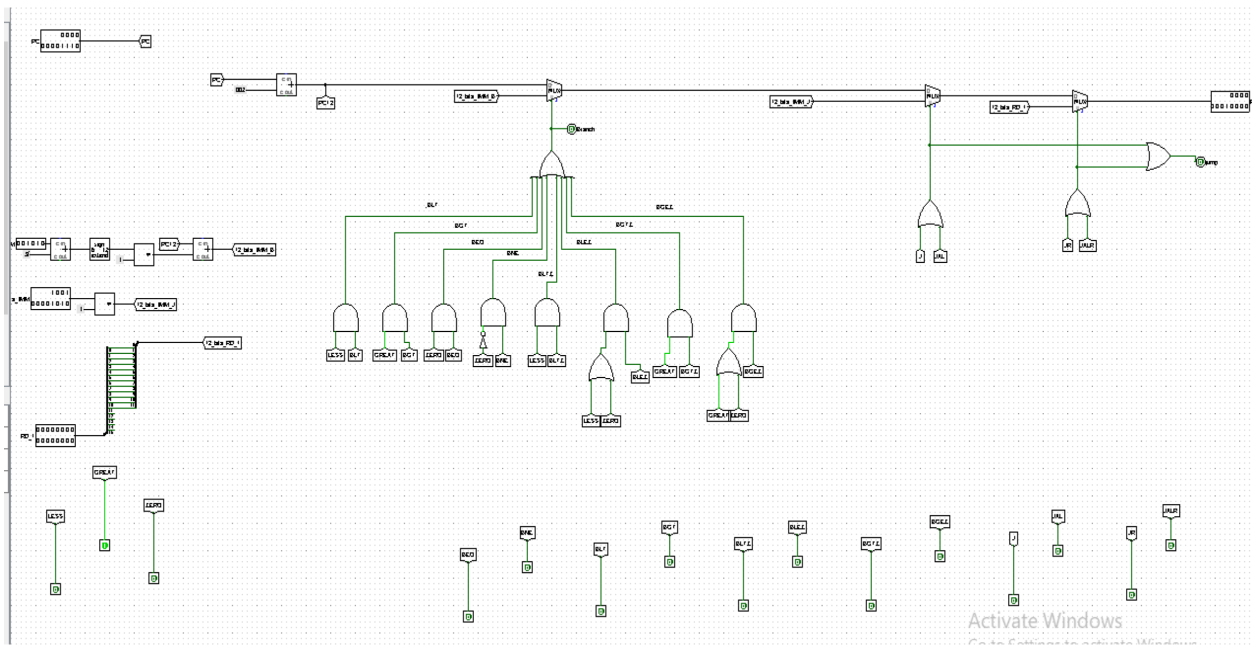
ALU Control

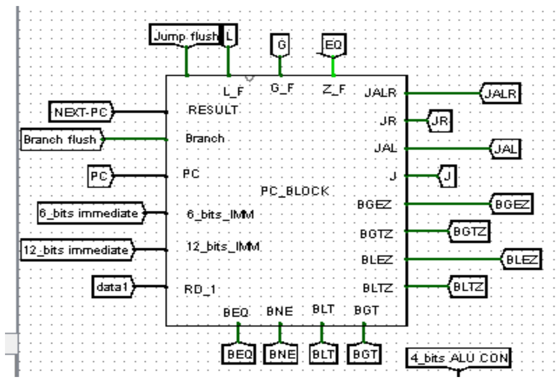


1.3 PC Counter:

The PC in the default case is incremented by Two, but in some cases, such as in branching and jumping, the PC jumps into a specified PC address to fetch a specific instruction. The next Pc holds this task where it finds the next PC if any of the jump or branch instructions are fetched.

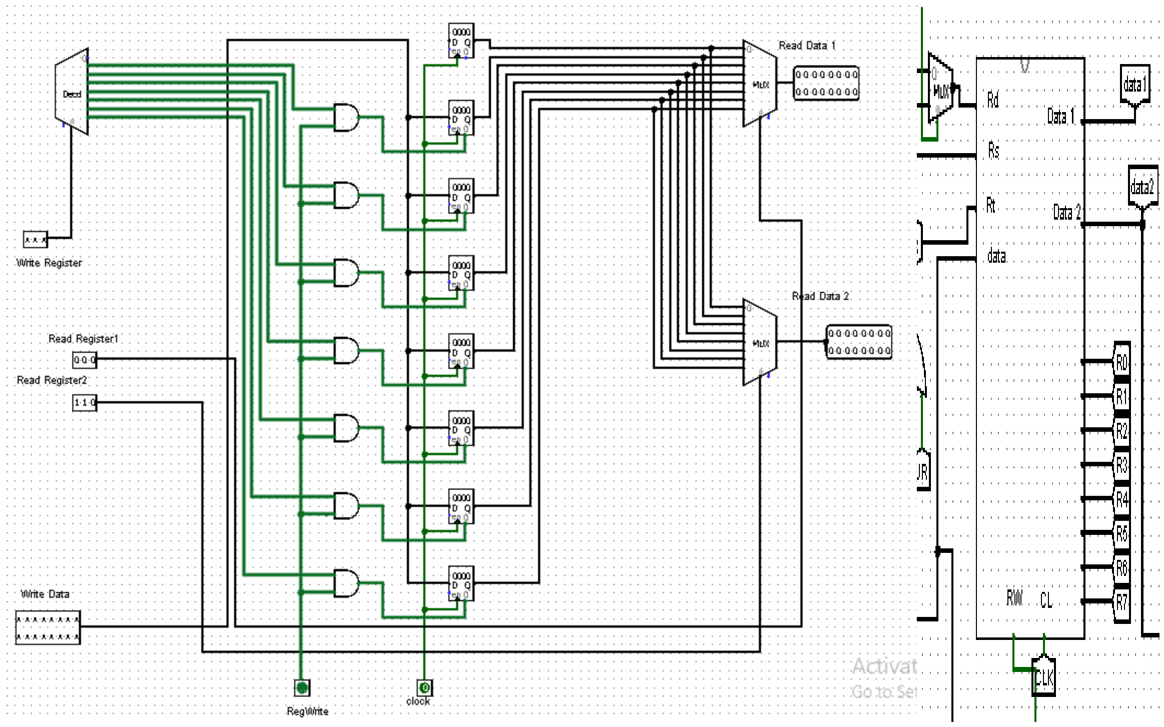
- In this Addressing mode, we take the lower 12 bits of the instruction shift left 1 bit then write them back to the PC register. This mode is used for J-type instructions.
- In this Addressing mode, we take the lower 6 bits of the instruction then add 3f then sign extend to 16 bits, then they will be added to PC+2 finally write them back to the PC register. This mode is used for Branch (in I Type) instructions.
- For LW and SW instructions, base- addressing mode is used. The base address in register Rs contains the memory address.





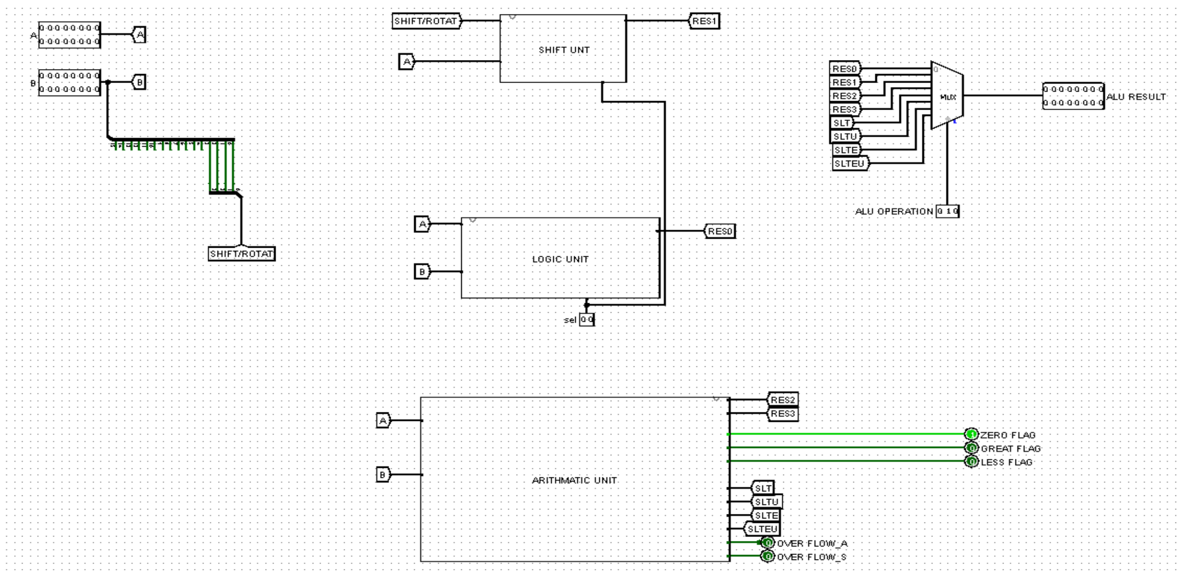
1.4 Register File:

This Block contains 16-bit registers from R1 to R7 and R0 is hardwired to 0. It also contains 2 read ports to read Rs and (Rt or Rd), and a write port to write input to Rd.



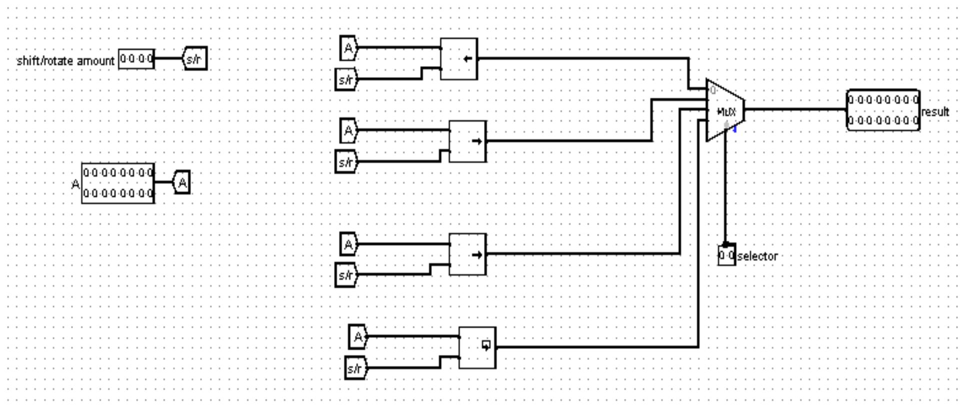
1.5 Arithmetic and Logical Unit (ALU):

The ALU is responsible for doing arithmetic operations on operand whether they are immediate values or registers. It consists of three blocks: shift, arithmetic and logic blocks.



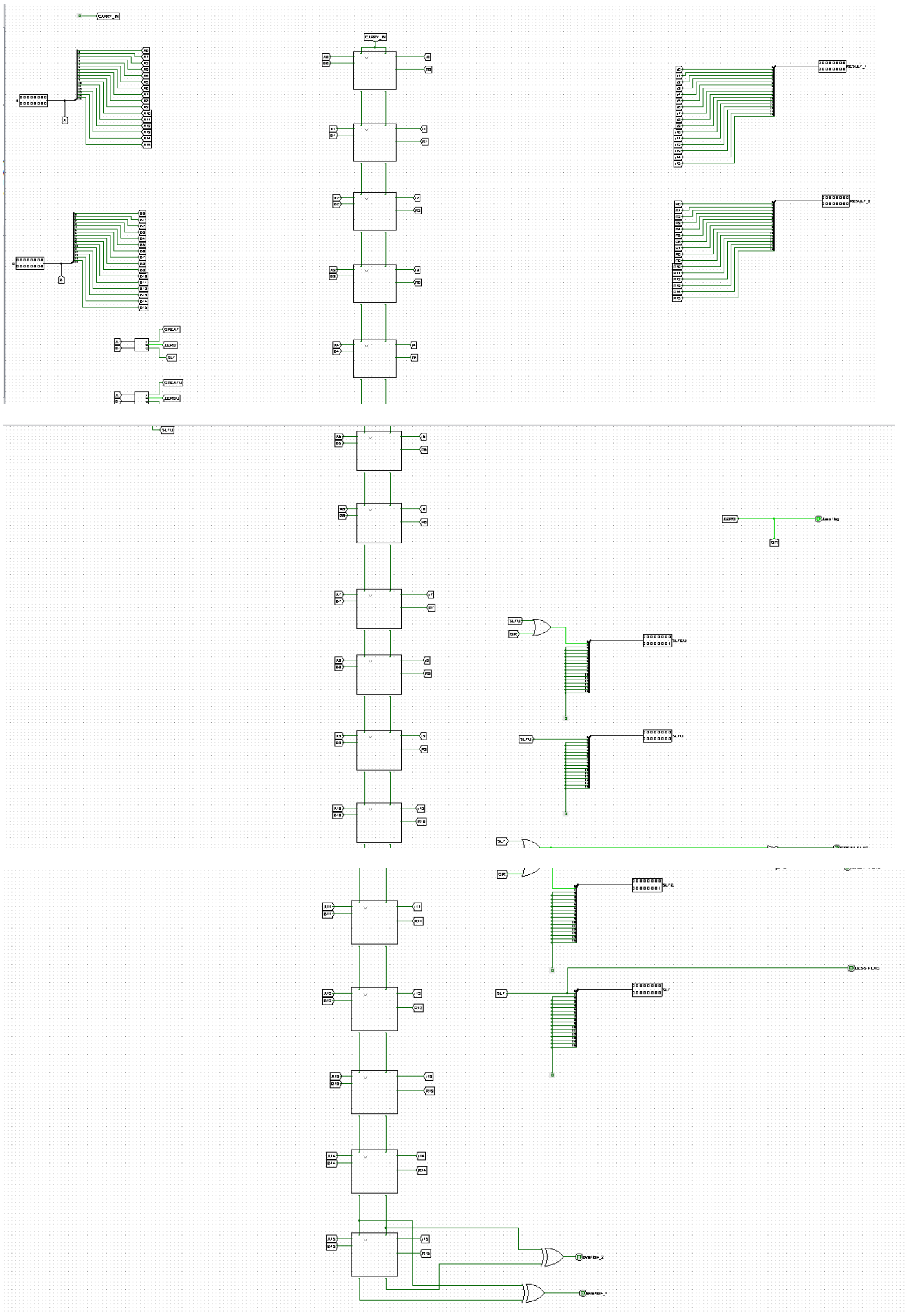
1. Shift Unit:

Responsible for shifting and rotating as required in the project description.



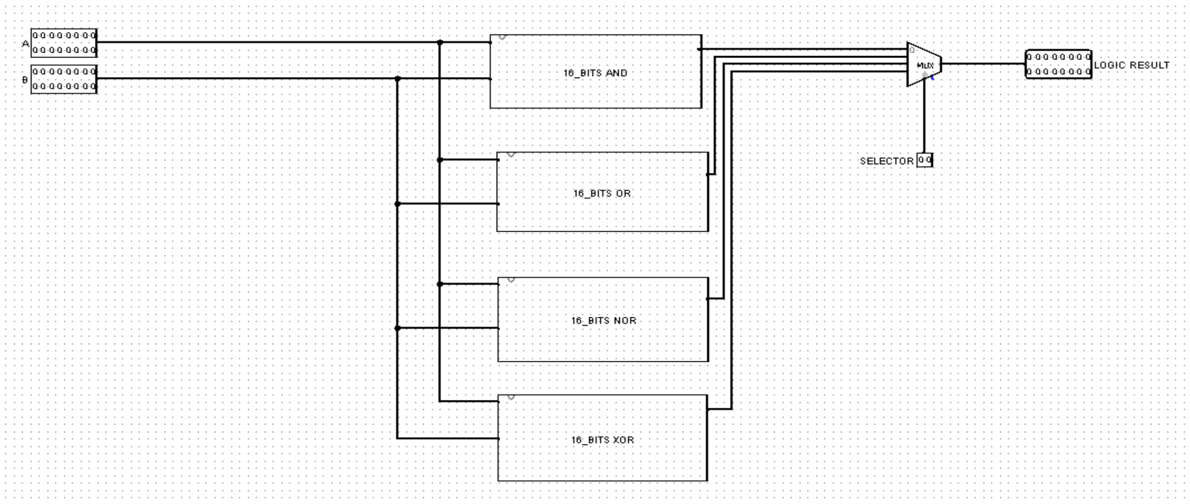
2. Arithmetic Unit:

Responsible for adding, subtracting, slt,slte,slteu and sltu instructions.


CARRY_IN
A
B
GREAT
ZERO
SLT
GREATU
ZEROU
SLTU
RESULT_1
RESULT_2
Zero flag
SLTEU
SLTE
LESS FLAG
overflow_2
overflow_1

3. Logic Unit:

Responsible for doing logic operations such as: and, nor, xor and or.

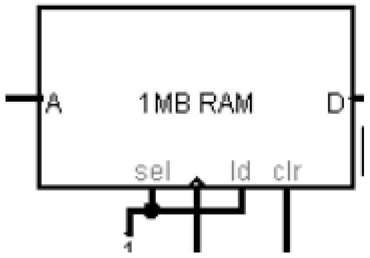


1.6 Instruction Memory:

As shown, it has 5 Pins

Sel: This input enables or disables the entire RAM module, based on whether the value is 1/floating or 0. Here we need to enable it every cycle, so we connect constant 1 to this pin.

Ld: Selects whether the RAM should emit (on D) the value at the current address (A). This output behavior is enabled if out is 1 or undefined; if out is 0, then no value is pushed onto D. Here we need to load the instruction every cycle, so we connect constant 1 to this pin.



Clr: When this is 1, all values in memory are pinned to 0.

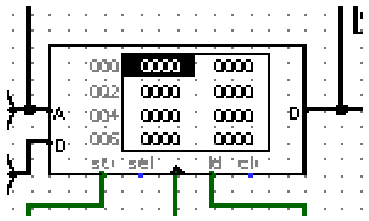
A: The address of the instruction. This pin is connected to the PC register.

D: The data loaded from address A. We take the instruction as output from this pin.

1.7 Memory Data:

The memory data is a Random-access memory (RAM) device that allows data items to be read and written in approximately the same amount of time regardless of the order in which data items are accessed, and the block is already implemented in Logisim.

As shown, it has 7 Pins :



Sel: This input enables or disables the entire RAM module, based on whether the value is 1/floating or 0. Here we need to enable it every cycle, so we connect constant 1 to this pin.

Ld: Selects whether the RAM should emit (on D) the value at the current address (A). This output behavior is enabled if out is 1 or undefined; if out is 0, then no value is pushed onto D on the east edge. Here we need to load the data from address (A) in case if the instruction is (LW), So we connect a signal from the control unit to this pin.

Str: This input is present only if "separate load and store ports" is selected for the Data Port attribute. When it is 1 or floating, a clock pulse will result in storing the data found on the west edge(D) into memory, here we need to store the data in address (A) in case if the instruction is (SW), So we connect a signal from the control unit to this pin.

Clr: When this is 1, all values in memory are pinned to 0.

A: The address of the location to load/store data. This pin is connected to the ALU to take the lower 24 bits of the calculated address.

D on the east edge: The data loaded from address A.

D on the west edge: The data to be stored in address A.

2. *Simulation and Testing* :

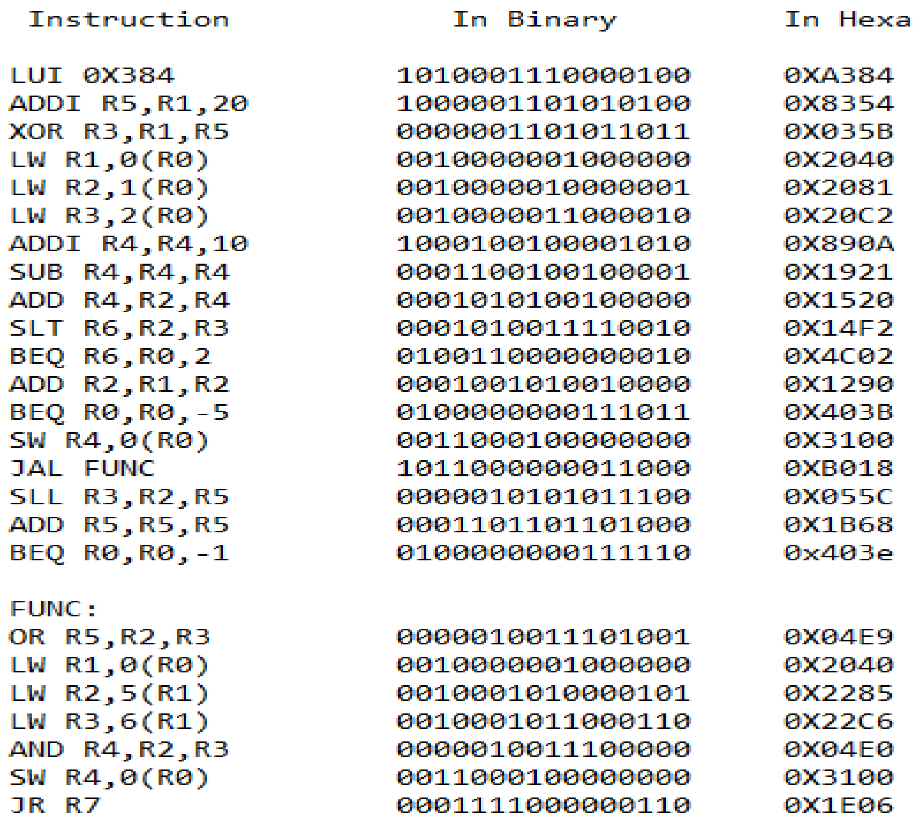
2.1 Carry out the simulation of the processor developed using Logisim:

Shown in the video

2.2 Describe the test programs that you used to test your design with enough comments describing the program, its inputs, and its expected output:

For testing we use a program check the following 14 instructions: LUI,Addi,Xor,lw,sub,Add,slt,Beq,sw,JAL,sll,or,And,Jr. We included hex text files with preloaded memory files that are compatible with logisim.

The Test Code :

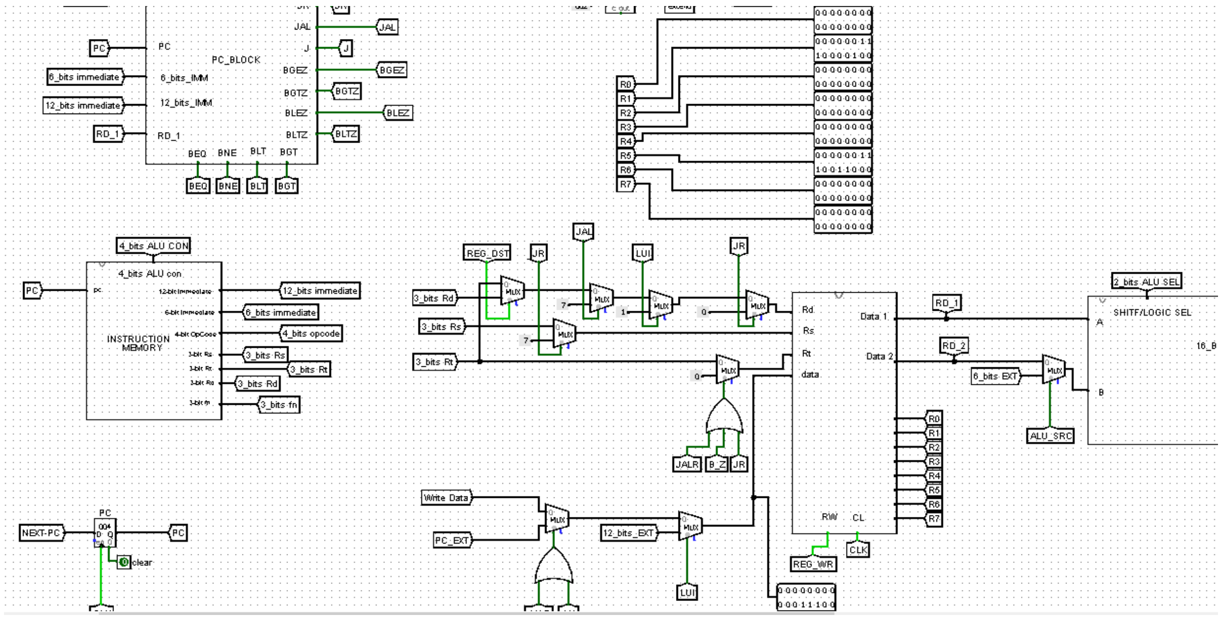


```
 Instruction            In Binary           In Hexa

LUI 0X384             1010001110000100     0XA384
ADDI R5,R1,20         1000001101010100     0X8354
XOR R3,R1,R5          0000001101011011     0X035B
LW R1,0(R0)           0010000001000000     0X2040
LW R2,1(R0)           0010000010000001     0X2081
LW R3,2(R0)           0010000011000010     0X20C2
ADDI R4,R4,10         1000100100001010     0X890A
SUB R4,R4,R4          0001100100100001     0X1921
ADD R4,R2,R4          0001010100100000     0X1520
SLT R6,R2,R3          0001010011110010     0X14F2
BEQ R6,R0,2           0100110000000010     0X4C02
ADD R2,R1,R2          0001001010010000     0X1290
BEQ R0,R0,-5          0100000000111011     0X403B
SW R4,0(R0)           0011000100000000     0X3100
JAL FUNC              1011000000011000     0XB018
SLL R3,R2,R5          0000010101011100     0X055C
ADD R5,R5,R5          0001101101101000     0X1B68
BEQ R0,R0,-1          0100000000111110     0x403e

FUNC:
OR R5,R2,R3           0000010011101001     0X04E9
LW R1,0(R0)           0010000001000000     0X2040
LW R2,5(R1)           0010001010000101     0X2285
LW R3,6(R1)           0010001011000110     0X22C6
AND R4,R2,R3          0000010011100000     0X04E0
SW R4,0(R0)           0011000100000000     0X3100
JR R7                 0001111000000110     0X1E06
```

2.3 List all the instructions that were tested and work correctly:

all of the instructions have been implemented and operate correctly.

2.4 Provide snapshots of the Simulator window with your test program loaded and showing the simulation output results:



This instruction is testing Addi, which should add 20 and the value of R1 in R5 .

phase 2: Pipelined Processor:

The CPU is divided into four stages saving in every stage the required values and control signals, every stage has its own data and control register except the first stage it has now control signal register.

This stages are:

1. **Instruction Fetch Stage :**

This is the first stage in the Pipeline . Instructions are fetched from the memory . The function of the instruction fetch is to obtain an instruction from the instruction memory using the current value of the PC and increment the PC value for the next instruction.

The instruction fetch stage is also responsible for reading the instruction memory and sending the current instruction to the next stage in the pipeline, or a stall if a branch has been detected in order to avoid incorrect execution.

2. **Instruction Decode Stage:**

It is the second stage in the pipeline. Branch targets will be calculated here and the Register File. The forwarding units, solving the data hazards in the pipeline. Their function is to detect if the register to be fetched in this stage is written to in a later stage. In that case the data is forward to This stage and the data hazard is solved. This stage is where the control unit determines what values the control lines must be set to depending on the instruction. In addition, hazard detection is implemented in this stage, and all necessary values are fetched from the register file.

3. **Execution Stage :**

The third stage in the pipeline is where the arithmetic- and logic-instructions will be executed. All instructions are executed with 16- bit operands and the result is a 16-bit word. The execution unit of the MIPS processor contains the arithmetic logic unit (ALU) which performs the operation determined by the ALUop signal. The branch address is calculated by adding the PC+4 to the sign extended immediate field shifted left 2 bits by a separate adder.

4. **Memory Access Stage :**

The memory access stage is the fourth stage of pipeline. This is where load and store instructions Will access data memory. During this stage, single cycle latency instructions simply have their results forwarded to the next stage. This forwarding ensures that both single and two cycle instructions always write their results in the same stage of the pipeline, so that just one write port to the register file can be used, and it is always Available. If the instruction is a load, the data is read from the data memory.

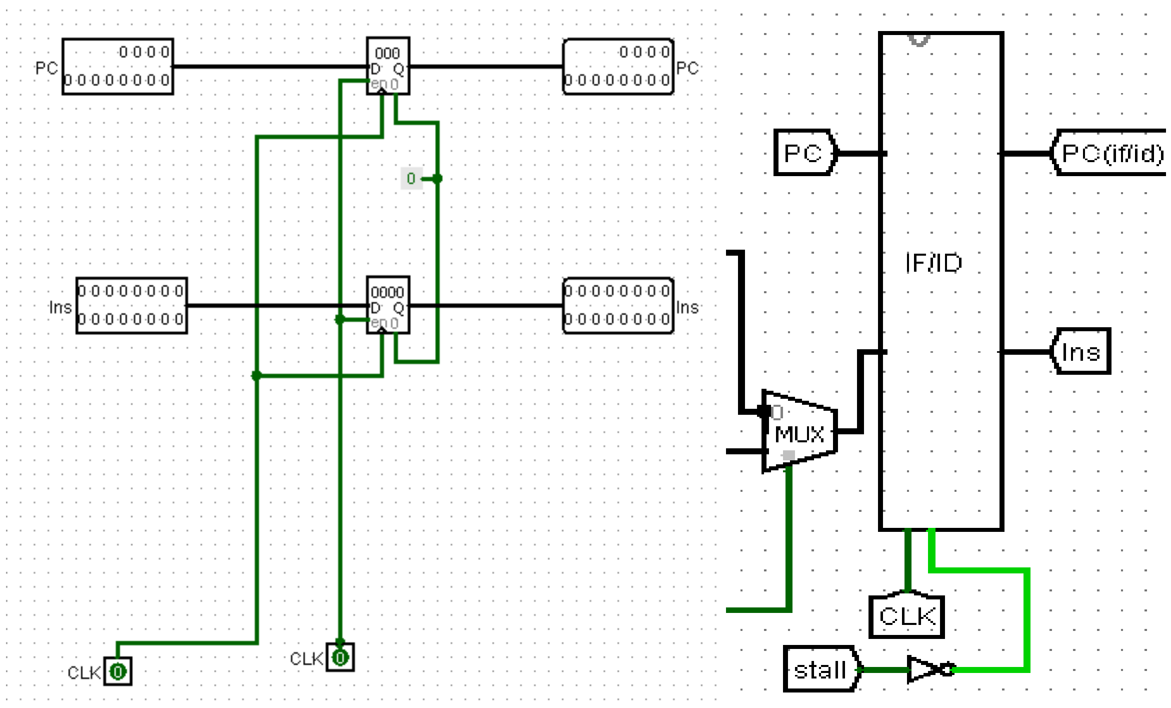
5. **Write Back Stage:**

During this stage, instructions write their results into the register file.

And We can implement those stages by 4 Registers:

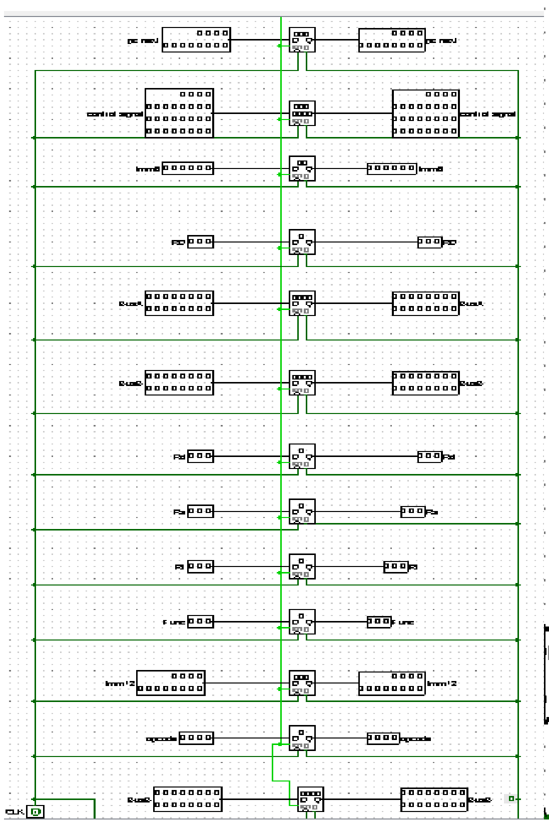
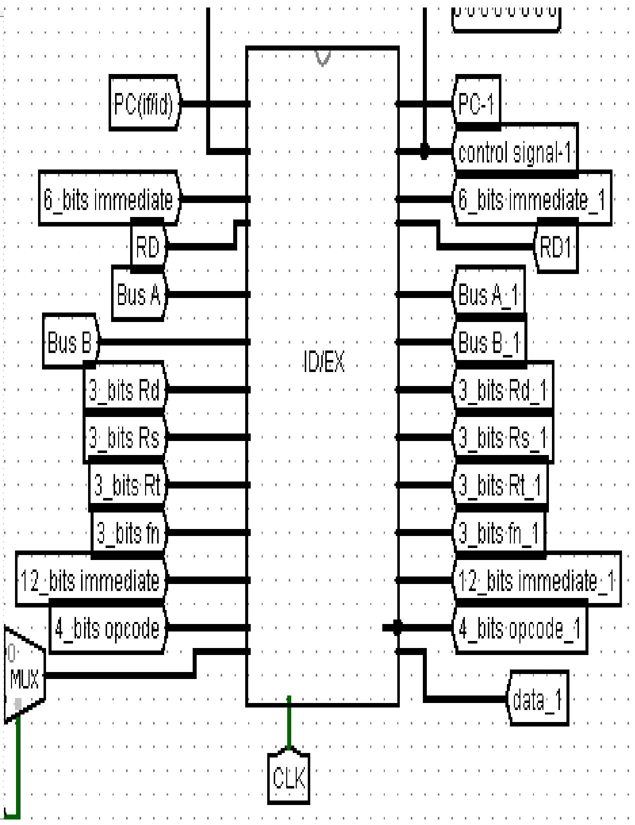
1. **IF/ID Register:**

- Data input : PC , Inst
- Control Signals : none



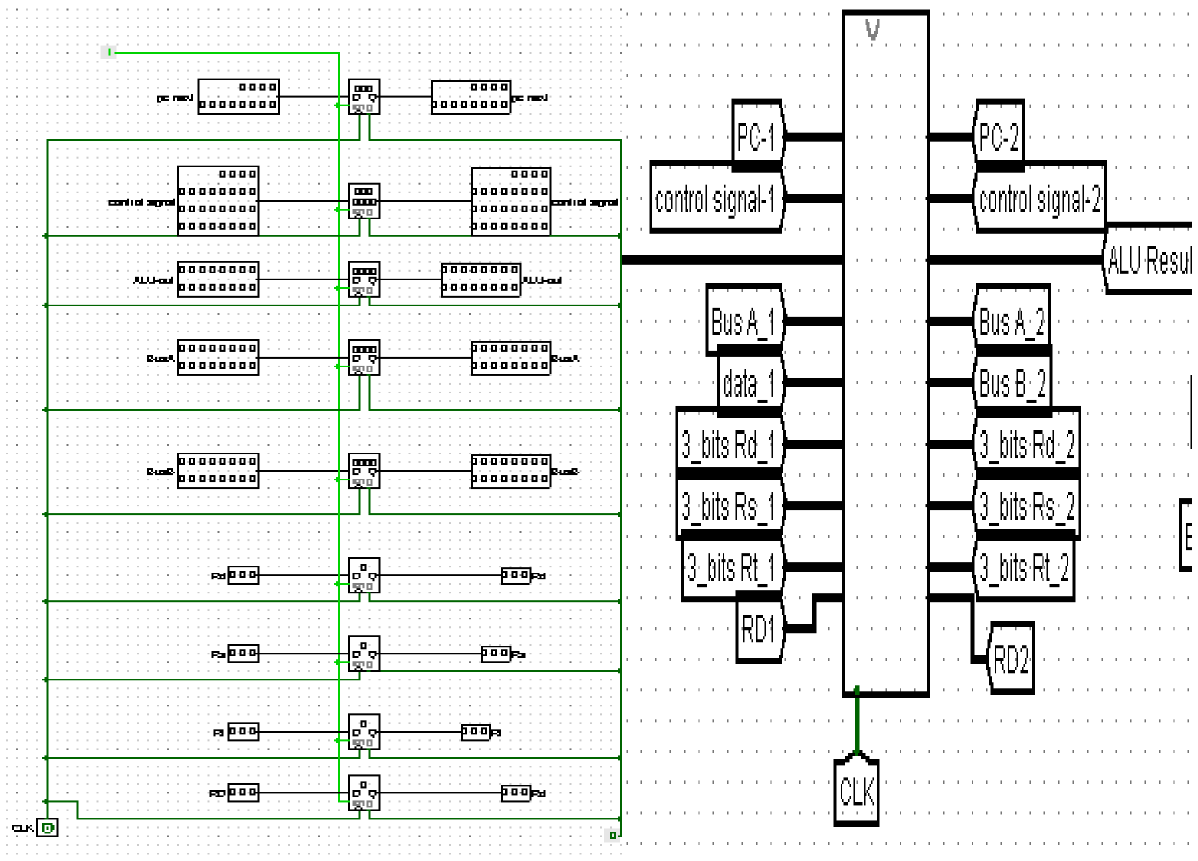
2. **ID/EX Register:**

- Data input : Imm6, Imm12, IF/ID PC, BusA, BusB, Rd, Rs, Rt, Func, Opcode, RD.
- Control Signals : ALUop , MemRead, MemWrite, MemToReg, RegWrite, Reg Dst, ALU CON, Opcode, ALU Src, BEQ, BNQ, BLT, BLTZ, BGT, BGTZ, BLEZ, BGEZ,JAL, JALR, J, JR.

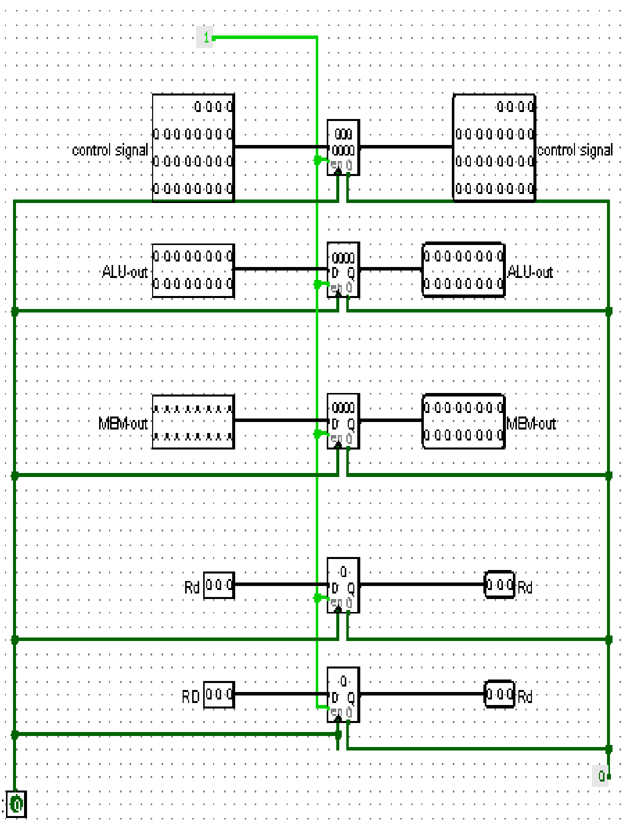



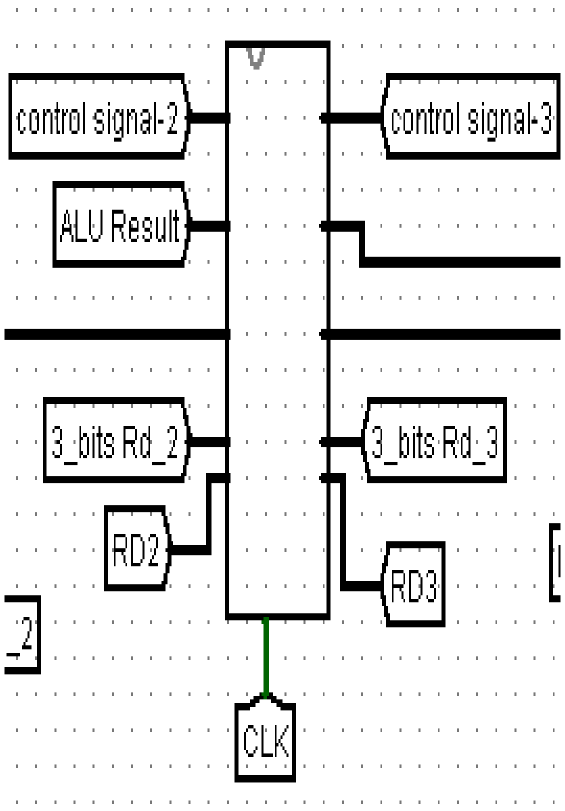
3. **EX/MEM Register:**

- Data input : BusA, data1, Rd, Rs, Rt, RD, PC, .
- Control Signals : ALUop , MemRead, MemWrite, MemToReg, RegWrite, Reg Dst, ALU CON, Opcode, ALU Src, BEQ, BNQ, BLT, BLTZ, BGT, BGTZ, BLEZ, BGEZ,JAL, JALR, J, JR.



4. **MEM/WB Register:**
- Data input : ALU Result, Rd, RD.
- Control Signals : ALUop , MemRead, MemWrite, MemToReg, RegWrite, Reg Dst, ALU CON, Opcode, ALU Src, BEQ, BNQ, BLT, BLTZ, BGT, BGTZ, BLEZ, BGEZ,JAL, JALR, J, JR.

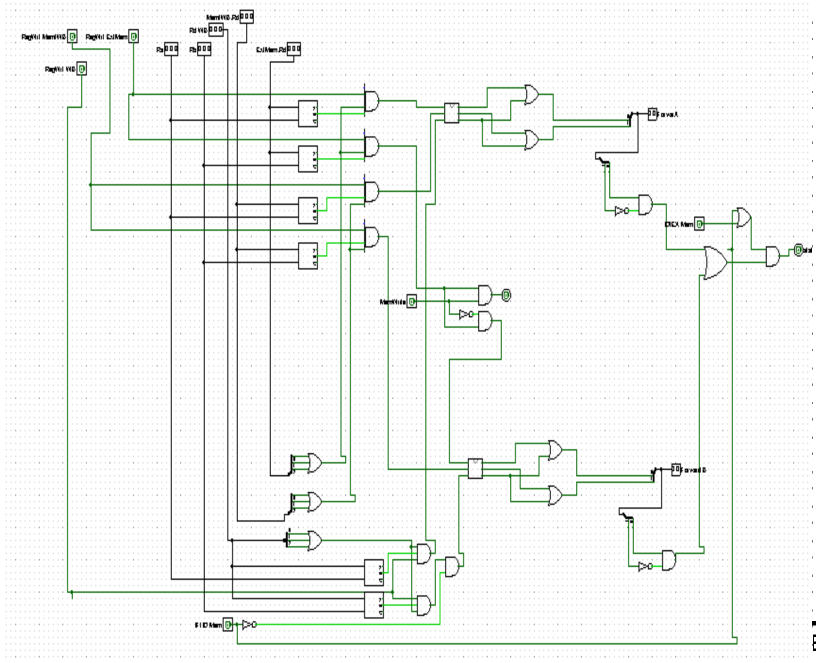


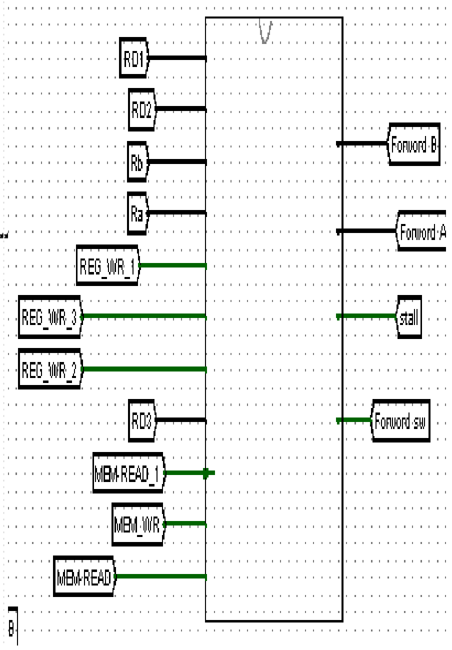


Forwording and Stall Unit :

The Forwarding unit selects the correct ALU inputs for the EX stage :

- If there is no hazard, the ALU's operands will come from the register file .
- If there is a hazard, the operands will come from either the EX/MEM or MEM/WB pipeline registers instead. The ALU sources will be selected by two new multiplexers, with control signals named ForwardA and ForwardB.





For Data Hazards:

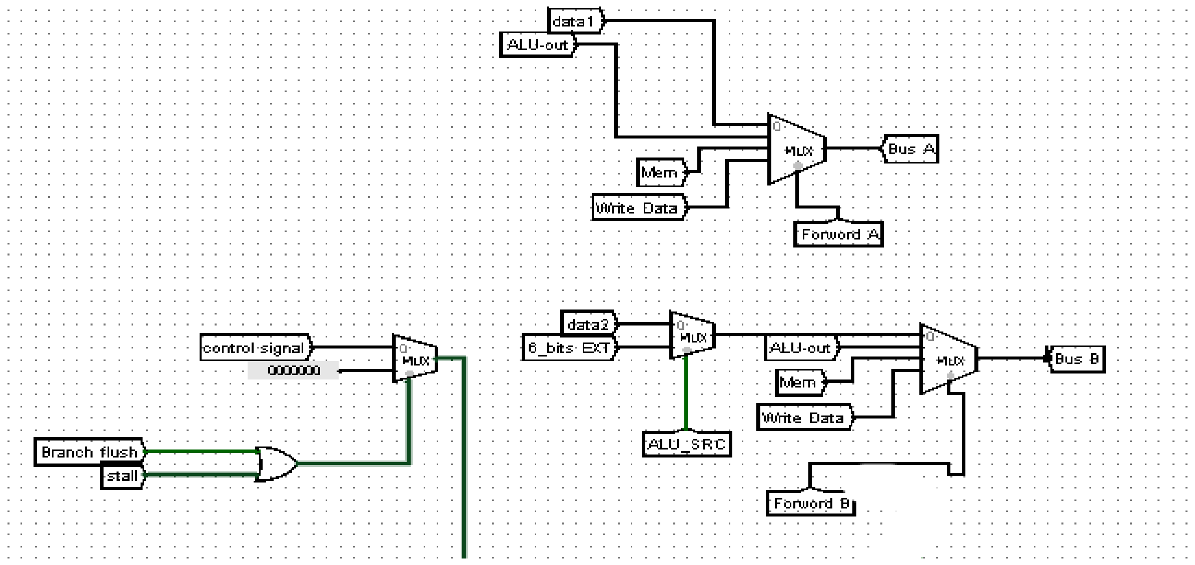
The data hazard means that an instruction is dependent on a previous instruction output that has still not have been written in the register file Rd. For example, adding a value into register one, then adding two registers (register one and two) into a third register, notice that register three's result is dependent on register's one value. However, to solve this problem we forward the data from the desired next stage into BusA or BusB. The data is forwarded by a mux having four inputs:

1- Zero: for normal flow.

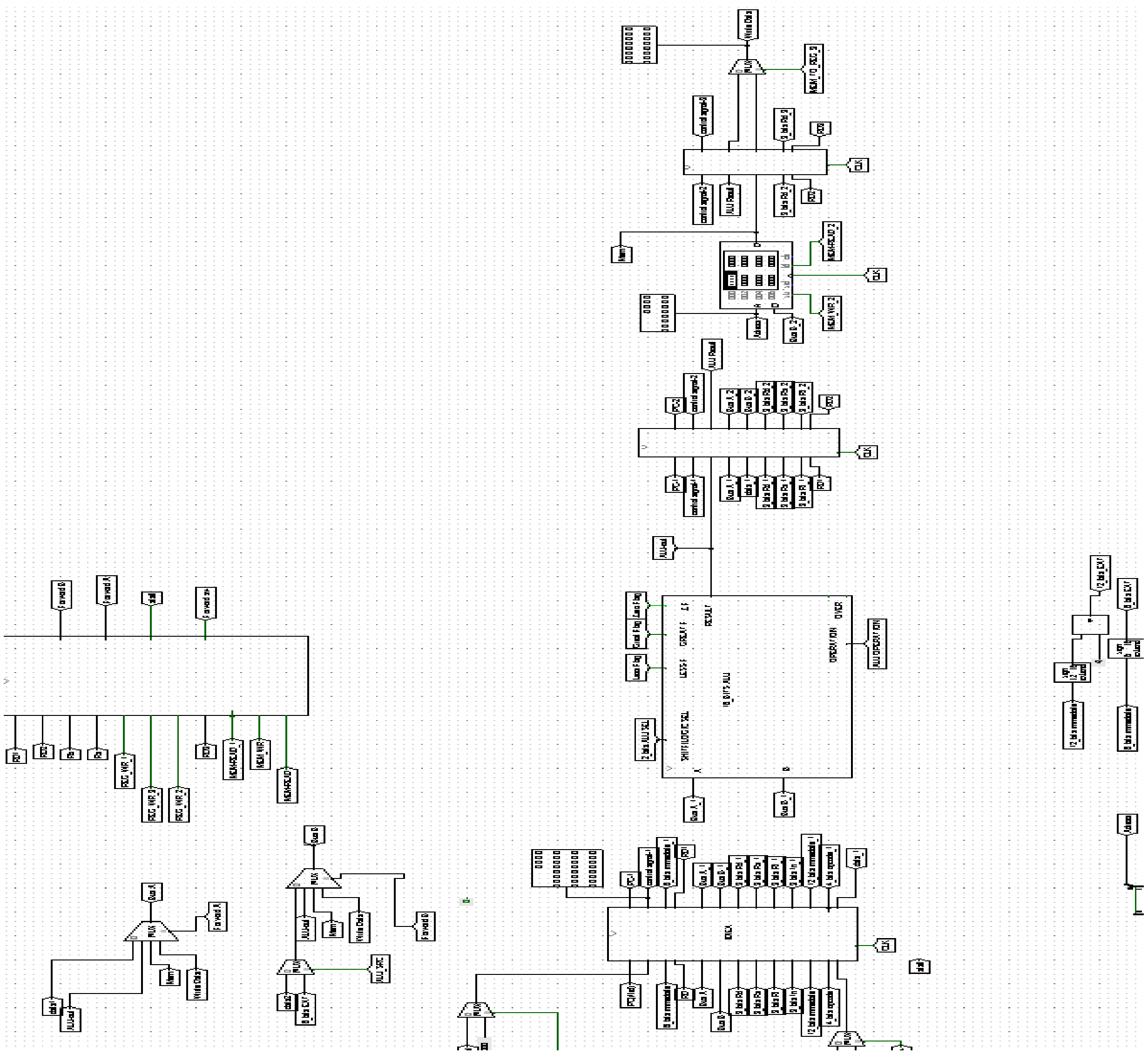
2- One: for execution dependency, either (ALU).

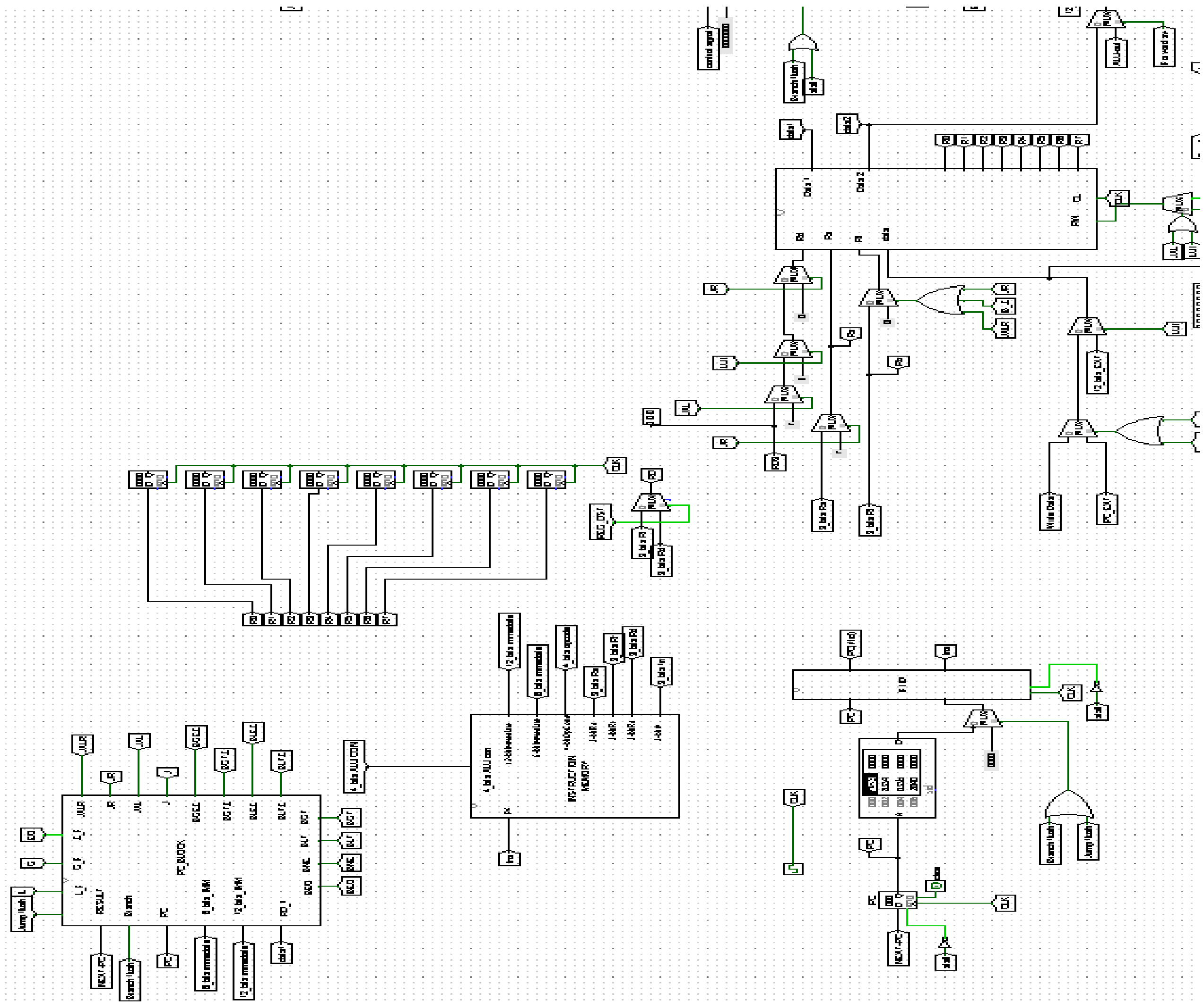
3- Two: for memory dependency.

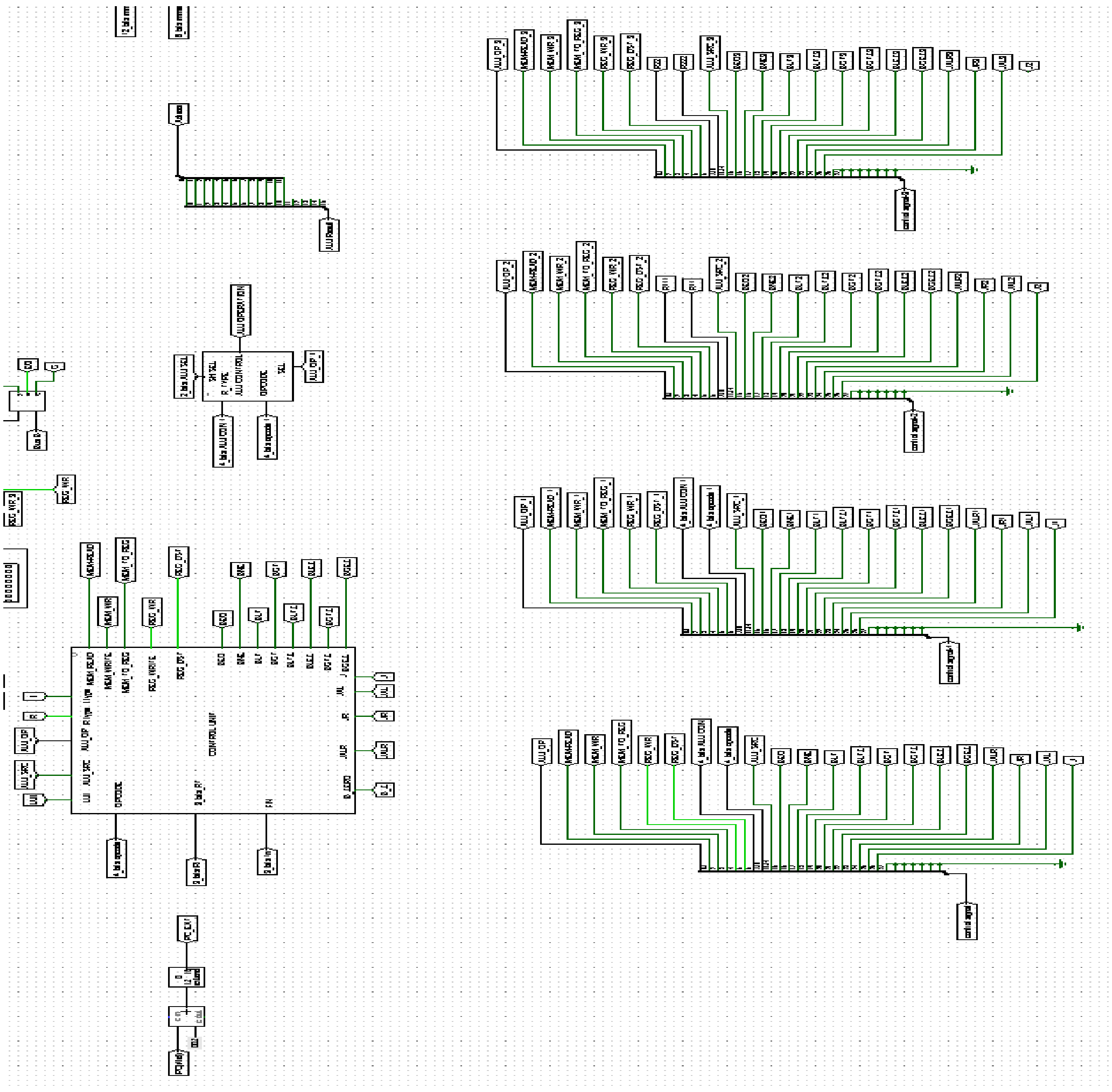
4- Three: for write back dependency.



❖ Putting all together


Write Data
CLK
MUX
RESULT
ALU Result



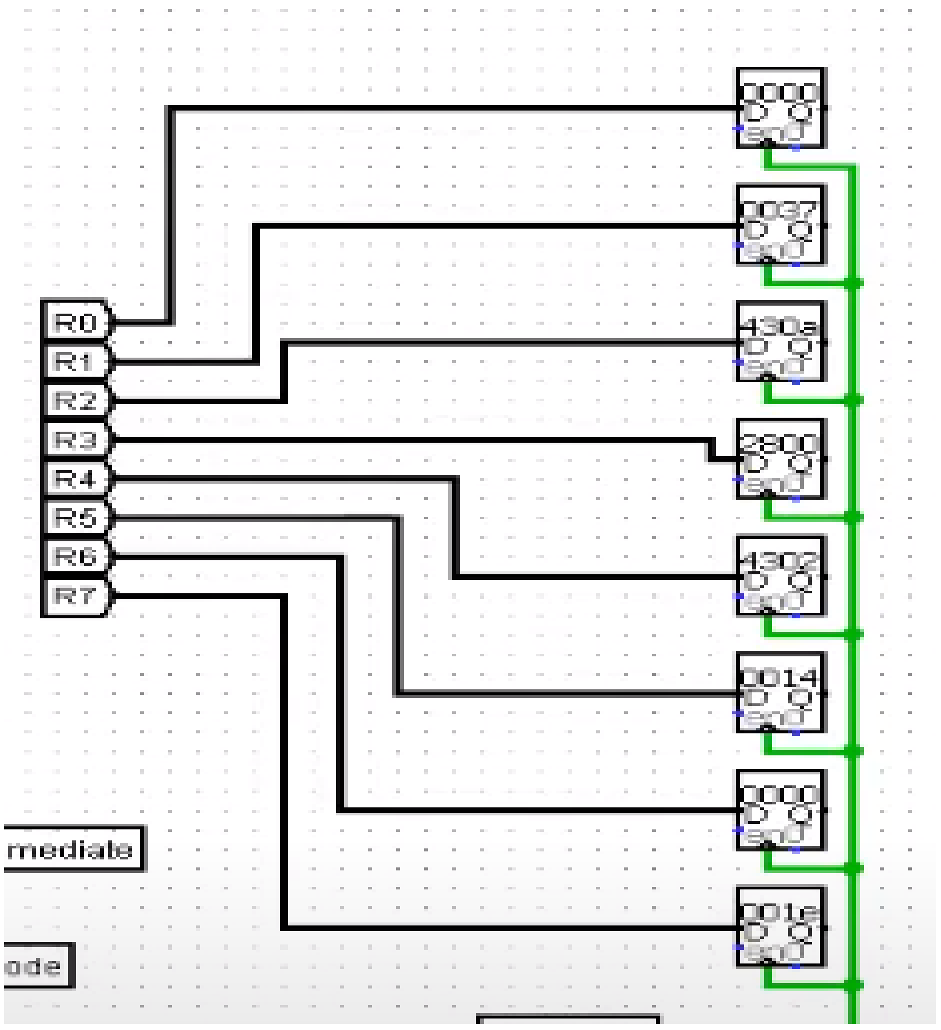


List all the instructions that do not run properly:

- The Forwording Unit doesn’t Work and we had to change it.
- There was a problem in load word delay, where the stall detection was at injecting an instruction into the MEM stage.

Final Test:

The Values in Register after applying the test code are



3. *Team Work* :

- Hamad's part:
  - Alu circuit.
  - Alu control.
  - EX/MEM stage.
- Mariam's part:
  - Control Unit.
  - Register File.
  - IF/ID stage.
  - ID/EX stage.
  - MEM/WB stage.
  - Forward Unit.
- Heba's part:
  - Instruction Memory.
  - Data Memory.
  - Write the Documentation.
- The Team:
  - Implement The Single Cycle DataPath.
  - Final Testing.