32-BIT SINGLE CYCLE MIPS CPU

Huy Duong Eiman Nour Uyen Le Wilson Pinales

April 23, 2020

1. **Design and Implementation**

**A. description of the datapath**

In this project, we designed a basic MIPS CPU with 16 instructions including 6 R-Type, 5 I-Type, 3 J-Type. The datapath will start with a PC which will load the instructions into instruction memory and receive the instructions. After that, it loads into the register file and reads the source used for the instruction.

For R-type, the values go from the registers and pass through the ALU to compute the instruction operation and then store. For I-type, the operand is an immediate value different from the R-Type’s instruction that it is from the second register. With both R and I types, they write the output of the ALU into the destination register. But for the sw (store word) this instruction will save into data memory. In J-type instructions, after the output of the ALU is computed, it is sent back to the PC at the point the PC control unit sends the selector for the mux to accept the branched or jump PC address. The control units will take control of which path it takes depending on the instruction and give the suitable control signal to the destination they need to go.

**B. Components**

* Register File
* PC
* Instruction Memory
* Data Memory
* ALU
* ALU Control
* PC Control
* Main Control Unit

**C. Control and Implementation**

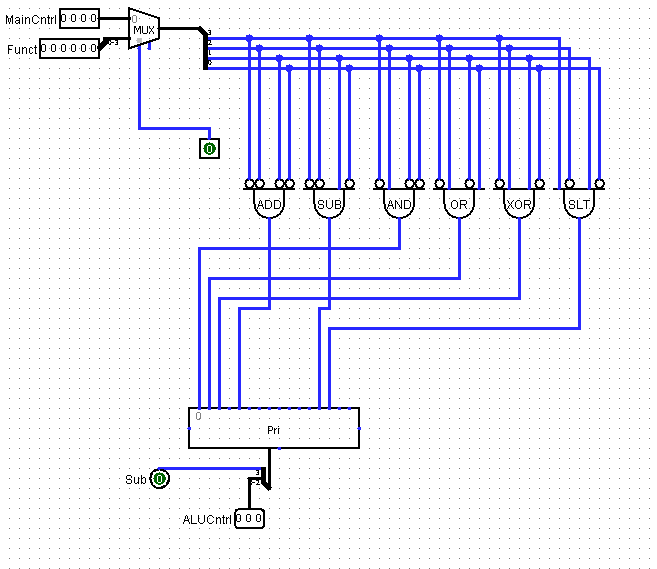
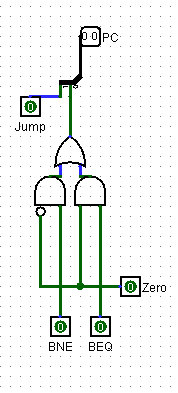
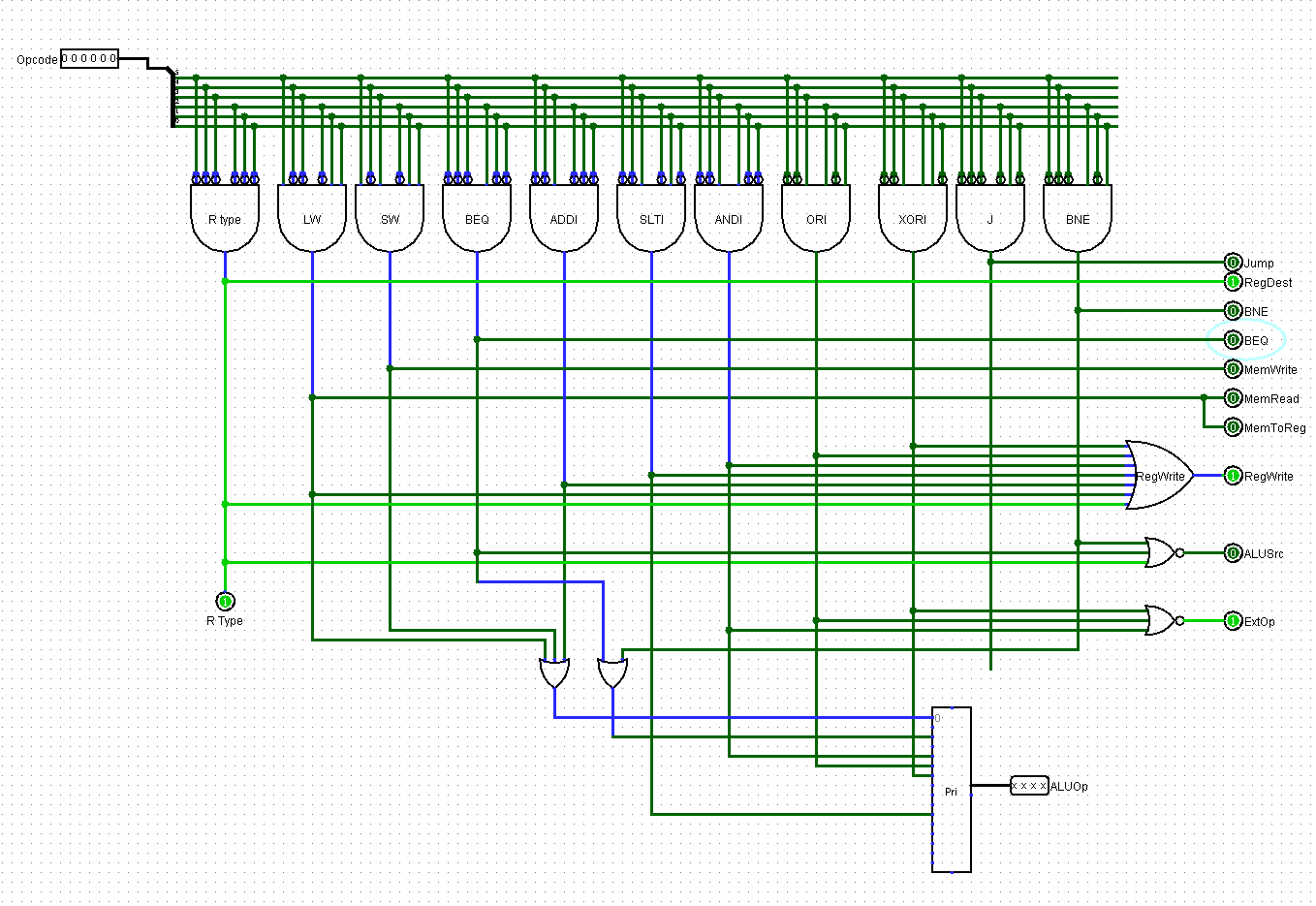
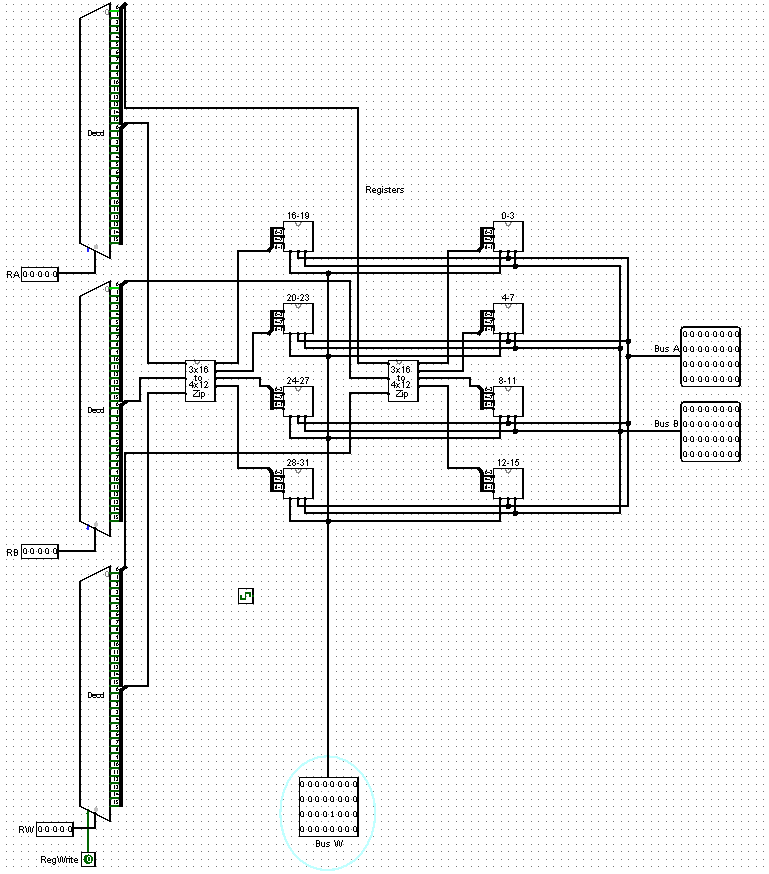
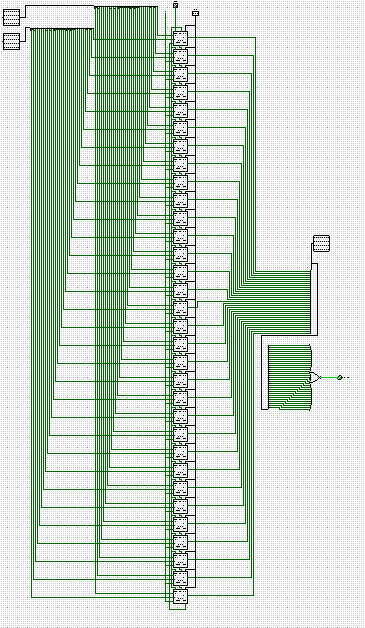
To control the CPU, there are three units; Main Control Unit, ALU control, and PC Control.

The Main Control Unit manages to interpret the opcode from the existing instruction and enable all of the relevant signals in order to work as expected. Some of the Main Control Unit signals are supplied to the PC Control unit and the ALU unit.

The ALU Control Unit takes the signals ALUSrc and ALUOp alternatively if the instruction handles immediate mode instructions, it will recover the opcode from the func bits in the original instruction recovered from instruction memory from the Main Control Unit, after that, according to those signals, recovers two data pieces A and B, where A is continually from a register, and B could be from a register or from the immediate value rooted in the instruction as selected by ALUSrc, then based on the ALUOp value the ALU Operation will be selected and performed.

The PC Control unit manages to get the next address the PC register should save for the next cycle, which will dictate which instruction in instruction memory that is removed and fed into the remaining components. The PC Control Unit selects whether the PC will increment, jump, or branch. By getting the signals of Jump, BNE, and BEQ, and computes from it a selection value to activate a channel in its multiplexer.

**D. Drawings and Designs**

* **ALU Control  
  **
* **PC Controller  
    
  **
* **Main Controller  
  **
* **Register  
    
    
  **
* **ALU  
  **

**E. Control table**

1. **Main control**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Jump** | **RD** | **BNE** | **BEQ** | **MW** | **MR** | **MTR** | **RW** | **Src** | **ExtOp** | **ALUOp** |
| **add** | **X** | **1** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **0** | **0000** |
| **sub** | **X** | **1** | **X** | **X** | **X** | **X** | **X** | **1** | **1** | **0** | **0010** |
| **and** | **X** | **1** | **X** | **X** | **X** | **X** | **X** | **1** | **1** | **0** | **0100** |
| **or** | **X** | **1** | **X** | **X** | **X** | **X** | **X** | **1** | **1** | **0** | **0101** |
| **xor** | **X** | **1** | **X** | **X** | **X** | **X** | **X** | **1** | **1** | **0** | **0110** |
| **slt** | **X** | **1** | **X** | **X** | **X** | **X** | **X** | **1** | **1** | **0** | **0101** |
| **addi** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **1** | **0** | **1** | **0000** |
| **slti** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **1** | **0** | **1** | **1010** |
| **andi** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **1** | **0** | **0** | **0100** |
| **ori** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **1** | **0** | **0** | **0101** |
| **xori** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **1** | **0** | **0** | **0110** |
| **lw** | **X** | **X** | **X** | **X** | **X** | **1** | **1** | **1** | **0** | **0** | **0000** |
| **sw** | **X** | **X** | **X** | **X** | **1** | **X** | **X** | **X** | **0** | **0** | **0000** |
| **beq** | **X** | **X** | **X** | **1** | **X** | **X** | **X** | **X** | **1** | **0** | **0010** |
| **bne** | **X** | **X** | **1** | **X** | **X** | **X** | **X** | **X** | **1** | **0** | **0010** |
| **j** | **1** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **0** | **0** | **X** |

**2. ALU control**

|  |  |  |
| --- | --- | --- |
| **Function** | **ALU Op Code** | **Sub Flag** |
| **and** | **000** | **0** |
| **or** | **001** | **0** |
| **xor** | **010** | **0** |
| **add** | **100** | **0** |
| **sub** | **100** | **1** |
| **slt** | **101** | **1** |

1. **PC control**

|  |  |  |  |
| --- | --- | --- | --- |
| **Op** | **Op Code** | **Jump** | **BNE or BEQ** |
| **PC++** | **00** | **0** | **0** |
| **Branch** | **01** | **0** | **1** |
| **Jump** | **10** | **1** | **0** |

**F. Logic equation**

Jump = J

RegDest = R-Type

BNE = BNE

BEQ = BEQ

MemWrite = SW

MemRead= LW

MemToReg = LW

RegWrite = R-Type + LW + ADDI + SLTI + ANDI + ORI + XORI

ALUSrc = ~(BNE + BEQ + R-Type)

ExtOp = ~(XORI + ORI + ANDI)

**2. Simulation and Testing**

**a . Carry out the simulation of the processor developed using Logisim**

**VIDEO HERE.**

**b. Describe the test programs that you used to test your design, describing the program, its inputs, and its expected output.**

We used four diﬀerent programs. The ﬁrst two programs check the eight instructions: and, or, sub, lw, sw, slt, beq, add. We included hex text ﬁles with preloaded memory ﬁles that work with logisim. Thanks to the website: <https://cs.nyu.edu/courses/fall14/CSCI-UA.0436-001/MIPS_Test_Programs.html>. Third and fourth programs we loaded them into MARS then created the hex text ﬁle and loaded it to Logisim. Third program tested instructions like: addi, xor, bne, andi, slti, xori, ori.

The output for the first program should be 0x37 on the first memory byte. The second program should have 2 hex outputs in the data memory, 1 output as a result of the OR function and 1 for the AND function.

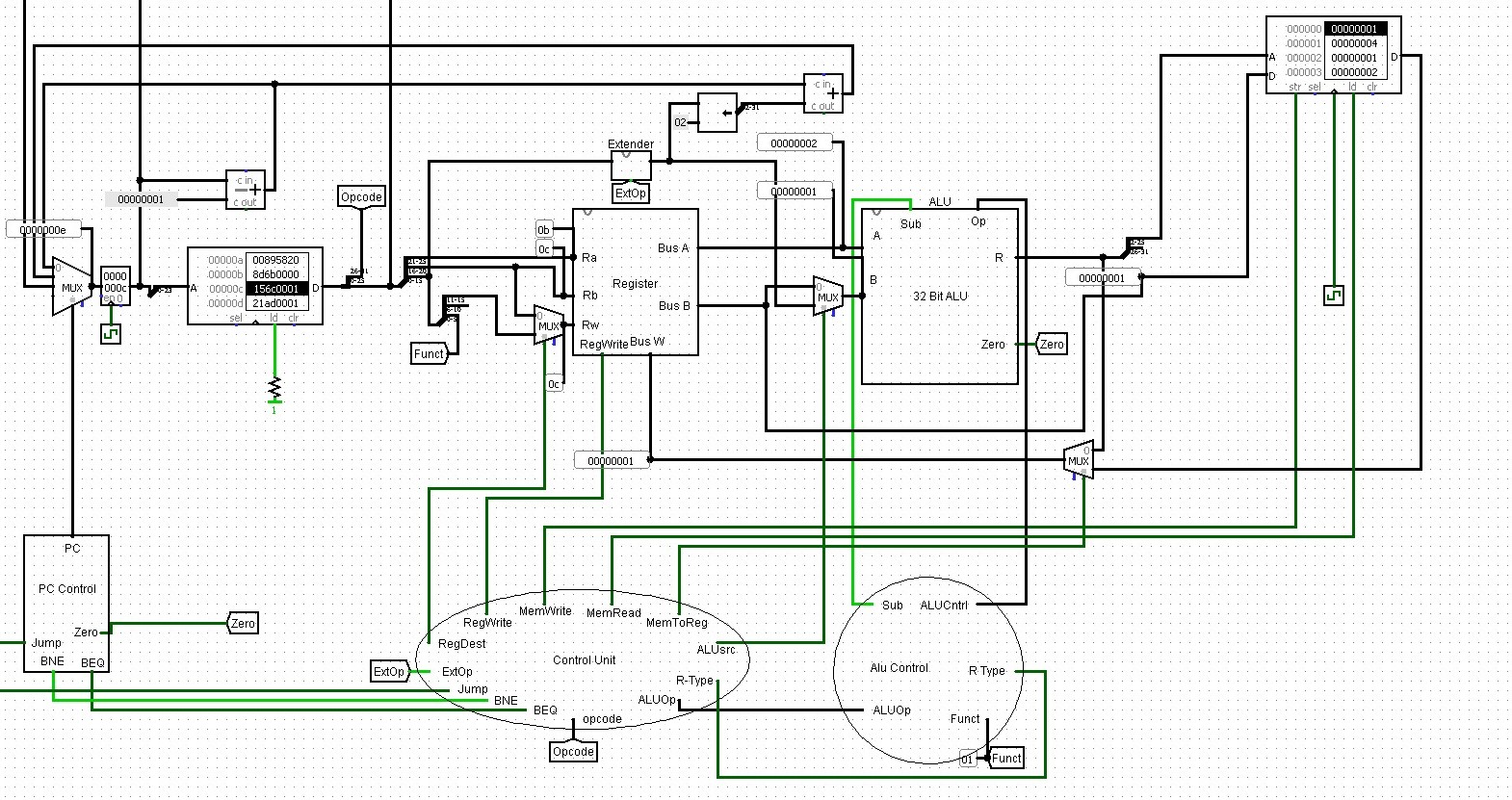
The fourth program was an example given in homework assignment #2, which had an array of elements(the array we used had 25 numbers instead of 5000) and the integer that repeated the most was stored in register $v1, register $v0 contained how many times it repeated. It ensured the registers were working as intended and it writes the values into the data memory, 7(the repetitions) and 1(the value) should be the expected results in that order.

All the instructions worked without errors. We tested if the instructions worked, then we made sure the branch instructions do not branch when they should not with the last program. We tried to find any way our design would yield incorrect results, or present strange behavior, but we couldn’t make that happen. Our video will show you testing of the programs, and they will be included in the zip folder with this submission.

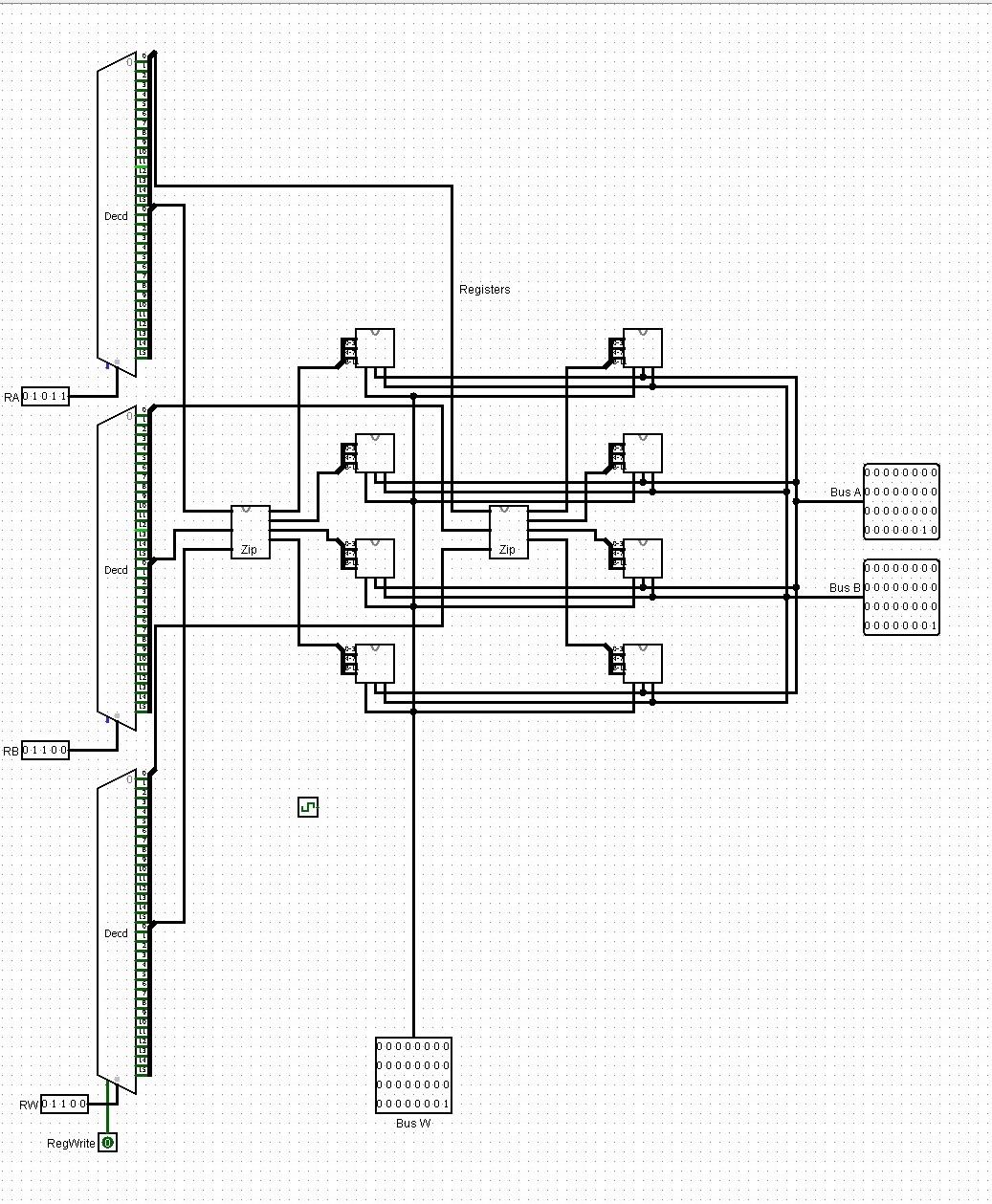
**c. List all the instructions that were tested and work correctly and instructions that do not run properly.**

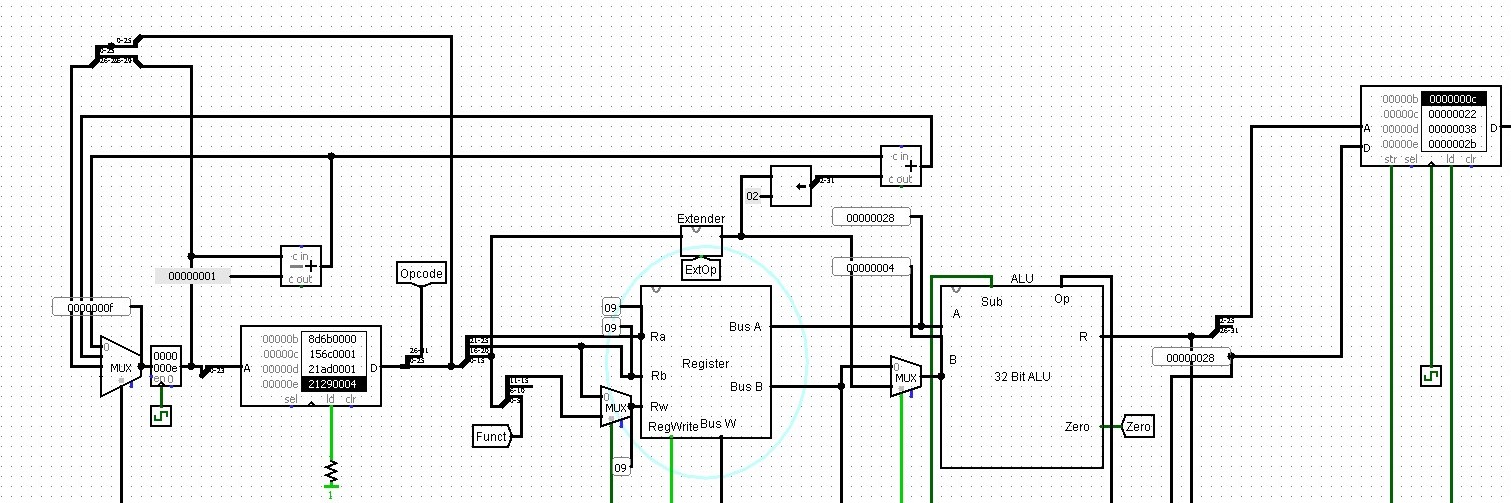
All our tests show that all the instructions have been implemented and operated properly. The tests which are shown in the video, and the auxiliary tests.

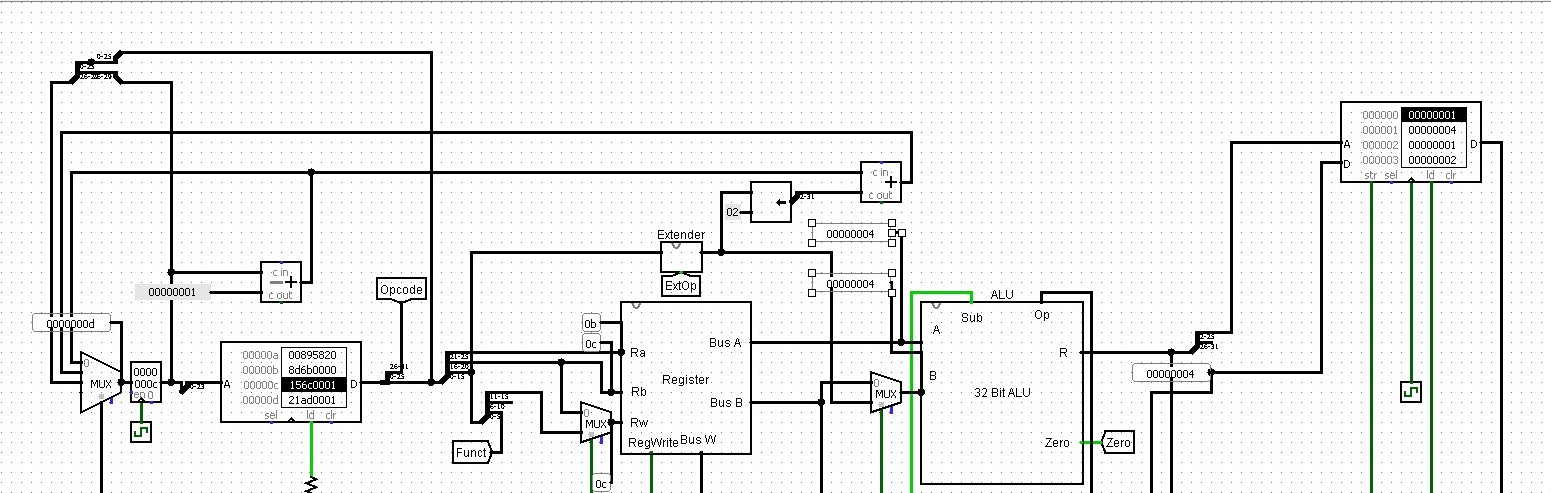
**d. Provide snapshots of the Simulator window with your test program loaded and showing the simulation output results**



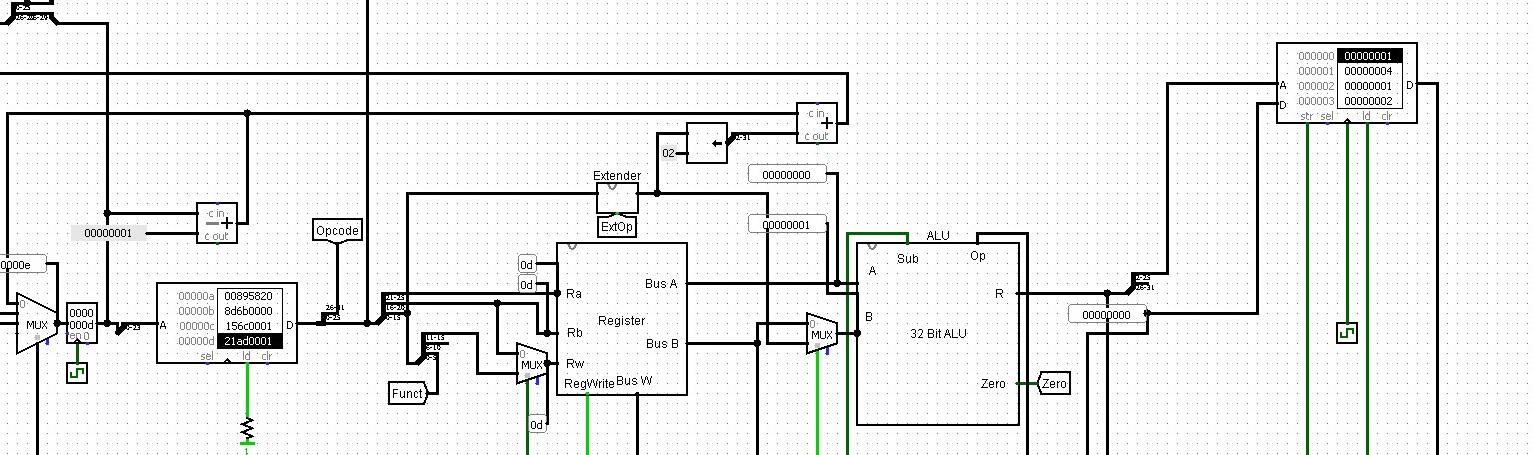
The instruction is testing if the bne is working properly, if the values stored in registers $t3 and $t4 are the same it should skip the next instruction, otherwise it sets the pc to the next instruction.

This shows the values of the registers when the bne instruction is called. And it skips the instructions.





It doesn’t branch when values from the registers are the same as intended.

****

The pc is set to the next instruction that addi 1 on the register $t5, that at this point is 0.

**3. Teamwork**

Our team has been performing most of the design process remotely through Google doc, before we start the project, we did a lot of research, watched many videos about the project and shared the logisim design ﬁles between each other in a peer to peer repository. Throughout the early month of April, most of the basic design of the process was heavily researched and discussed remotely. Unfortunately, due to the coronavirus pandemic, we have to stay at home and not be allowed to go to school, and various scheduling issues arise in each individual member’s time. We decided to schedule time for the project and make sure that we do not miss the due date. The development of the ALU circuit module was developed by both Huy and Uyen, and the Register ﬁle was developed by Eiman and Wilson. The full debugging process for both components were delayed until both components were brought up to basic feature parity to be tested together. But for the most part, individually in the teams work for the components were split up into circuit designer and circuit debugger roles. One person would handle designing the circuit and explaining the schematic to the others, while the partner would go through and attempt to get the circuit to work while relaying any issues arising from the circuit, while the others ﬁxed any issues that arose. Once Part 1 Of The Processor Was Complete, in order to complete part 2, the team decided to video call meetings on google hangouts to discuss part 2, in order to fully collaborate on what parts of the processor data path should be implemented. The ﬁrst phase of the part 2 meetings were implementing the remainder of the components, namely the memory, control units, and the PC register logic. Huy was able to quickly get the memory and PC register up and running. From there the control unit work was split up between the ALU Control developed by Uyen and Eiman, and the PC control and the main control unit was developed by Huy and Winson. From there Control unit debugging was handled by Uyen, and Processor testing was handled by Eiman, and the was tested by Wilson.