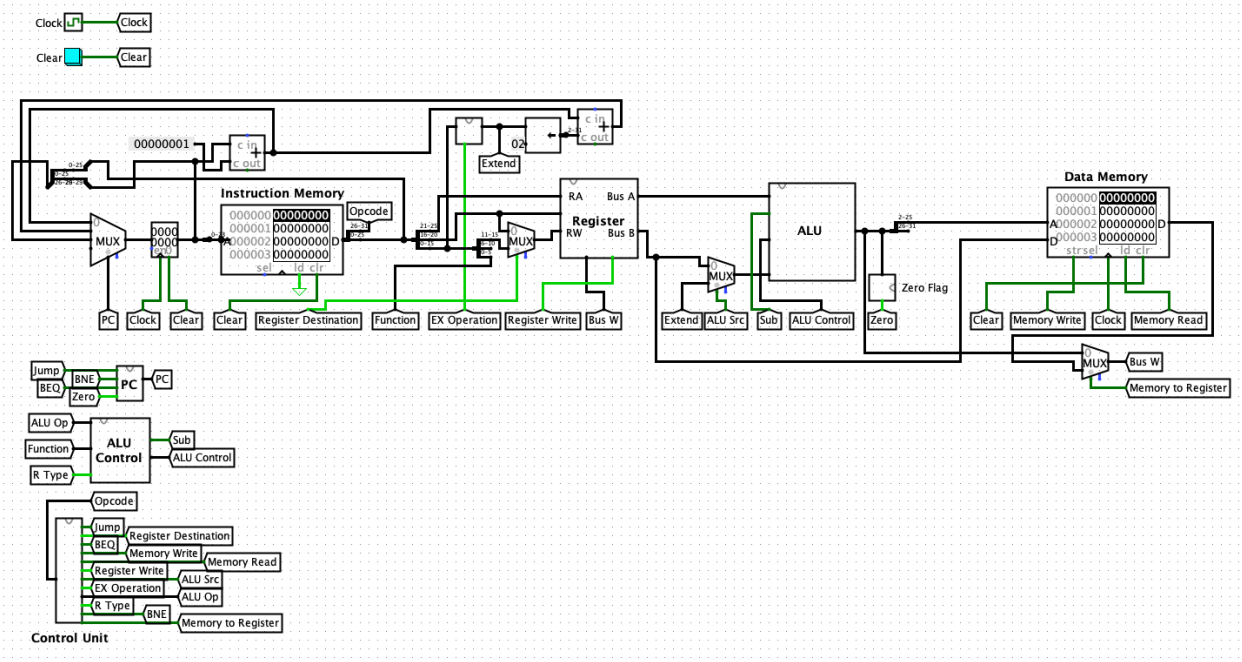
MIPS 32-bit CPU Design
May 1st, 2019
Computer Architecture

Design Overview:

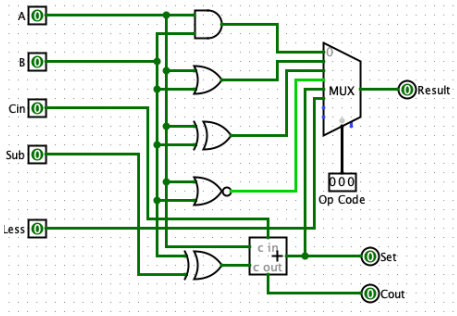


For our final project, we were to make a single cycle 32-bit processor on Logisim. Our tasks involved: modeling the designed 32x32-bit register file as one single module in Logisim, designing and modeling a 32-bit ALU to perform all the arithmetic, shift and logic operations required by our data path in Logisim, design and model a datapath for a single-cycle CPU in Logisim, apply the needed values for the control signals needed for the execution of each instruction, design and model the control unit for the designed data path in Logisim, model the single cycle CPU design in Logisim by combining the datapath and control units, then finally testing each task and the correct functionality of the designed CPU by storing all the implemented instructions in the instruction memory and verifying the correct execution of each instruction.

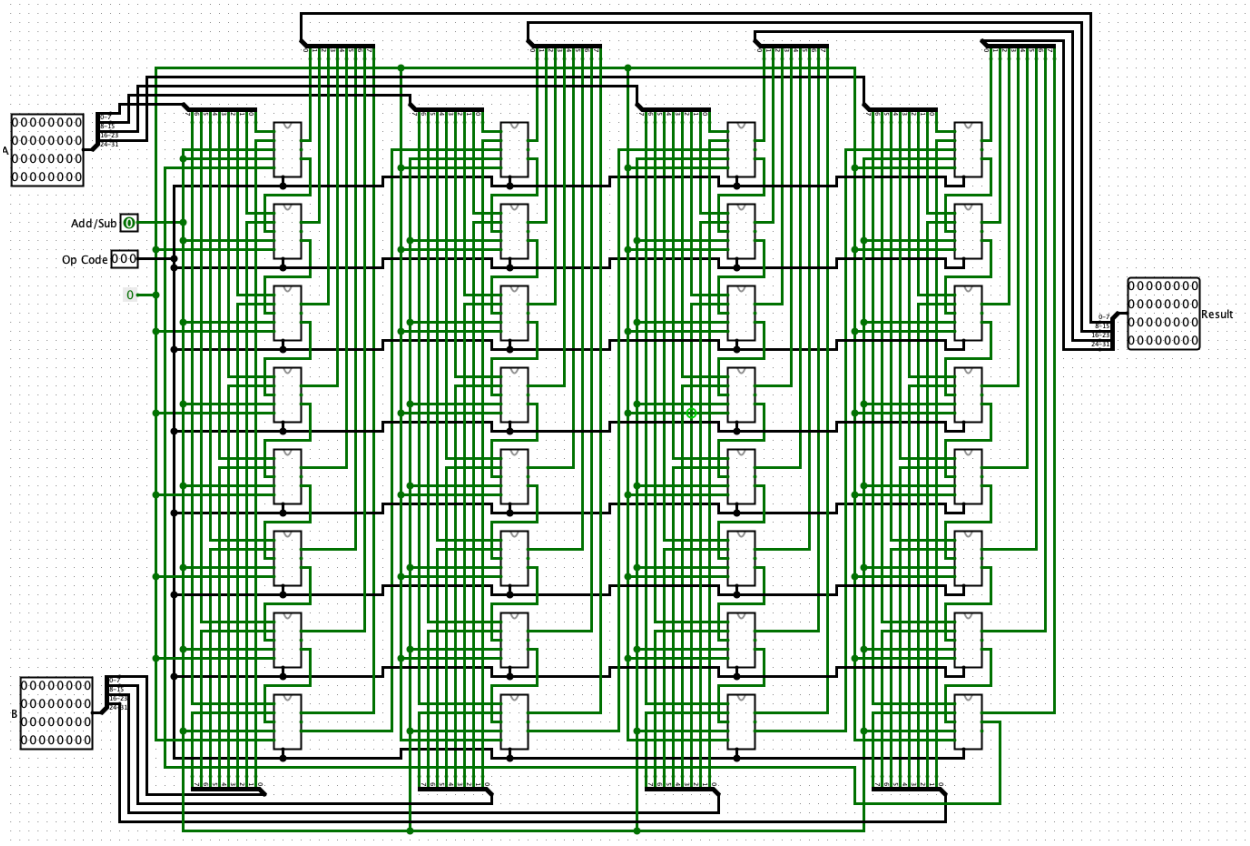
Design and Implementation:

ALU:

The ALU was implemented first by creating a 1-bit ALU, and then creating a 32-bit ALU from the 1-bit ALU.



1-bit ALU

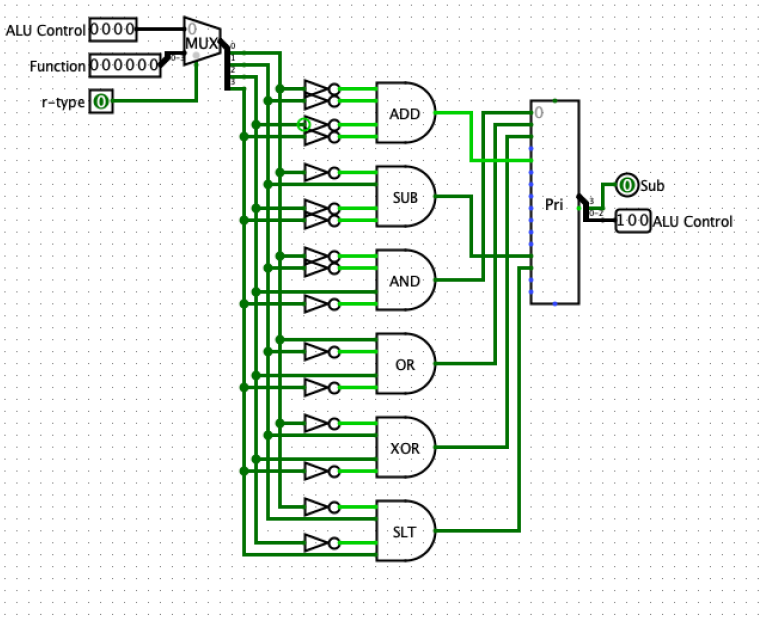


32-bit ALU

As can be seen from above, the ALU has 2 inputs: A and B. Each are 32 bits in length. The other inputs are Add/Sub, a 1-bit control, and the Op Code, a 3-bit input. Add/Sub

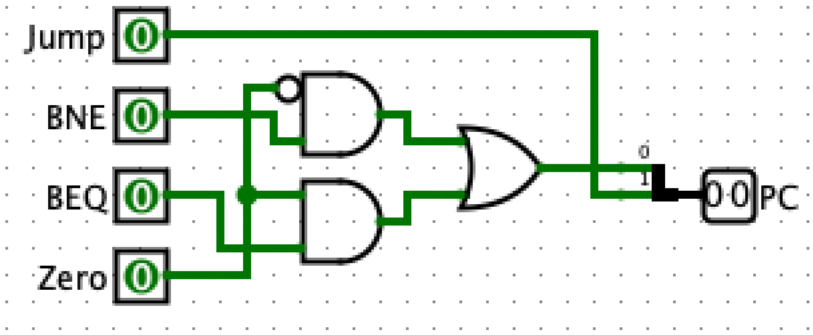
is responsible for switching the ALU from addition to subtraction. The Op Code indicates the operation that is to be preformed by the CPU. This is originally derived by the control unit.

ALU Control:



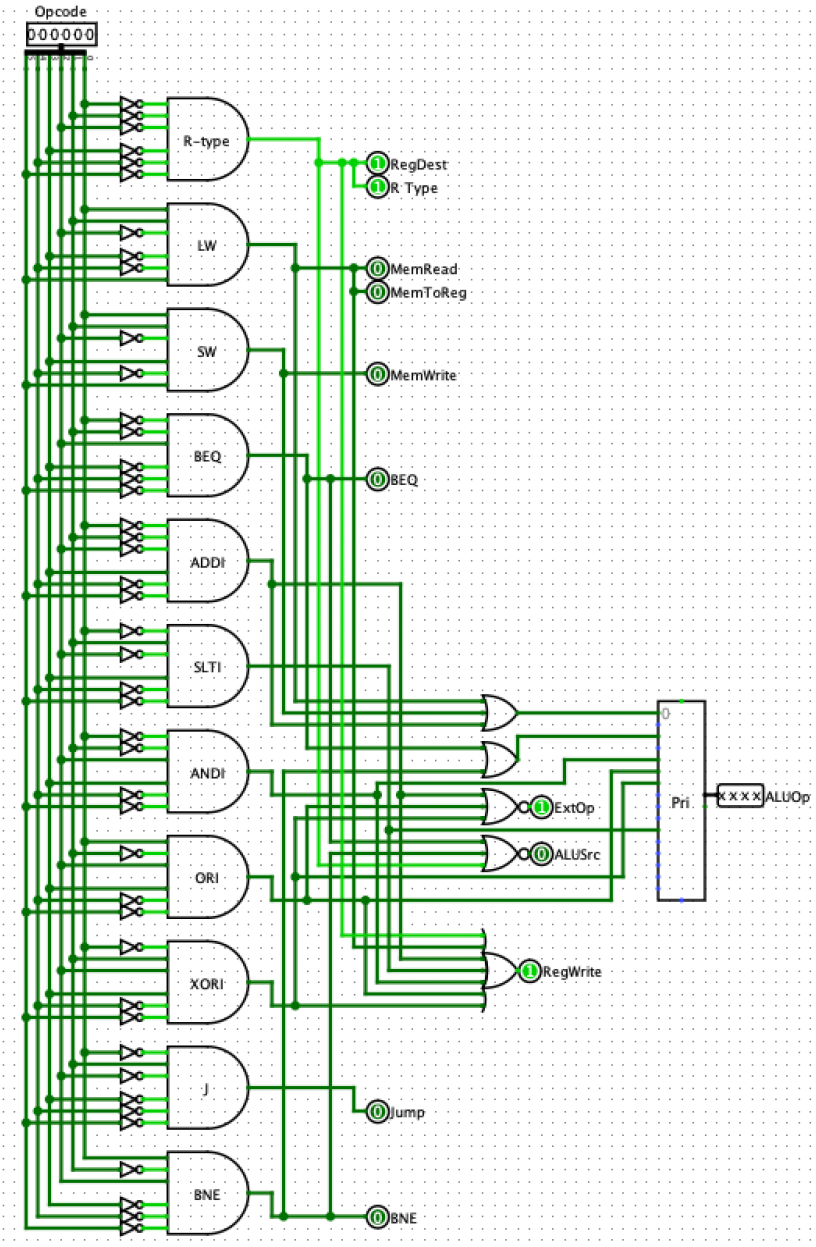
The ALU control is where the derived ALU control input comes from. The inputs here are a 4-bit ALU Control, a 6-bit function control and a r-type indicator. These inputs are passed through an array of AND gates which outputs to an encoder to filter out the function that is to be preformed. The ALU control is this indication.

PC Control:



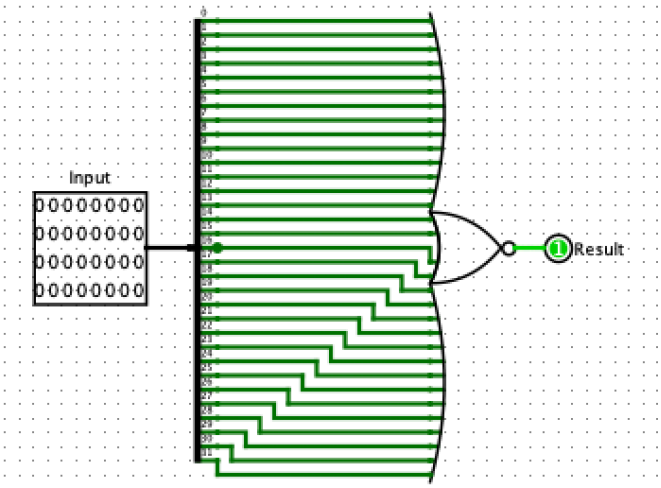
The PC Control is a simple circuit but it is responsible for outputting four types of instructions: Jump, BNE, BEQ and Zero. The PC register services as the selection control for a multiplexer which is fed directly into the Instruction Memory for the CPU.

Control Unit:



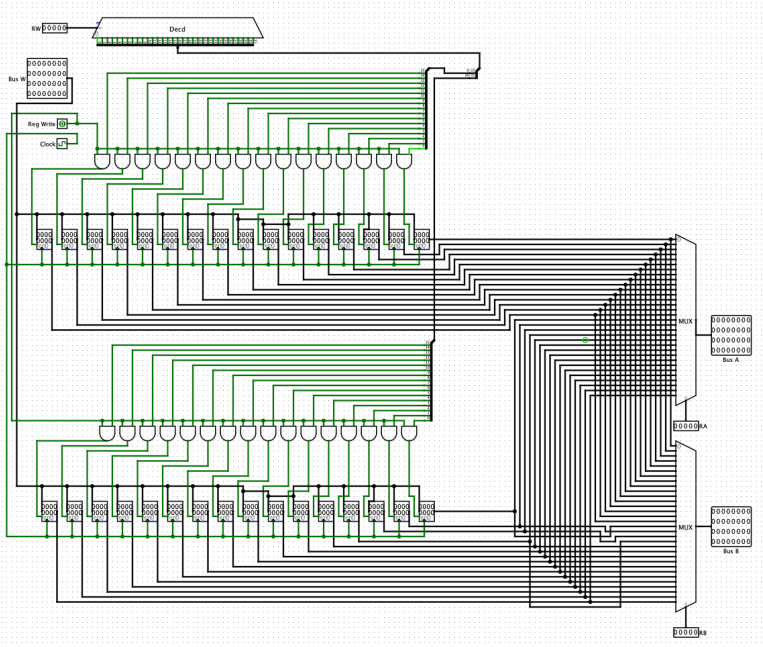
The control unit is responsible for taking in input from the Op Code, a 6-bit input, and selecting functions for the CPU to implement. The Control unit disperses inputs at all stages of the CPU clock. The Opcode is taken from the data out port of the instruction memory.

Zero-flag:



The zero-flag circuit simply outputs 1 if the output is 0.

Register:

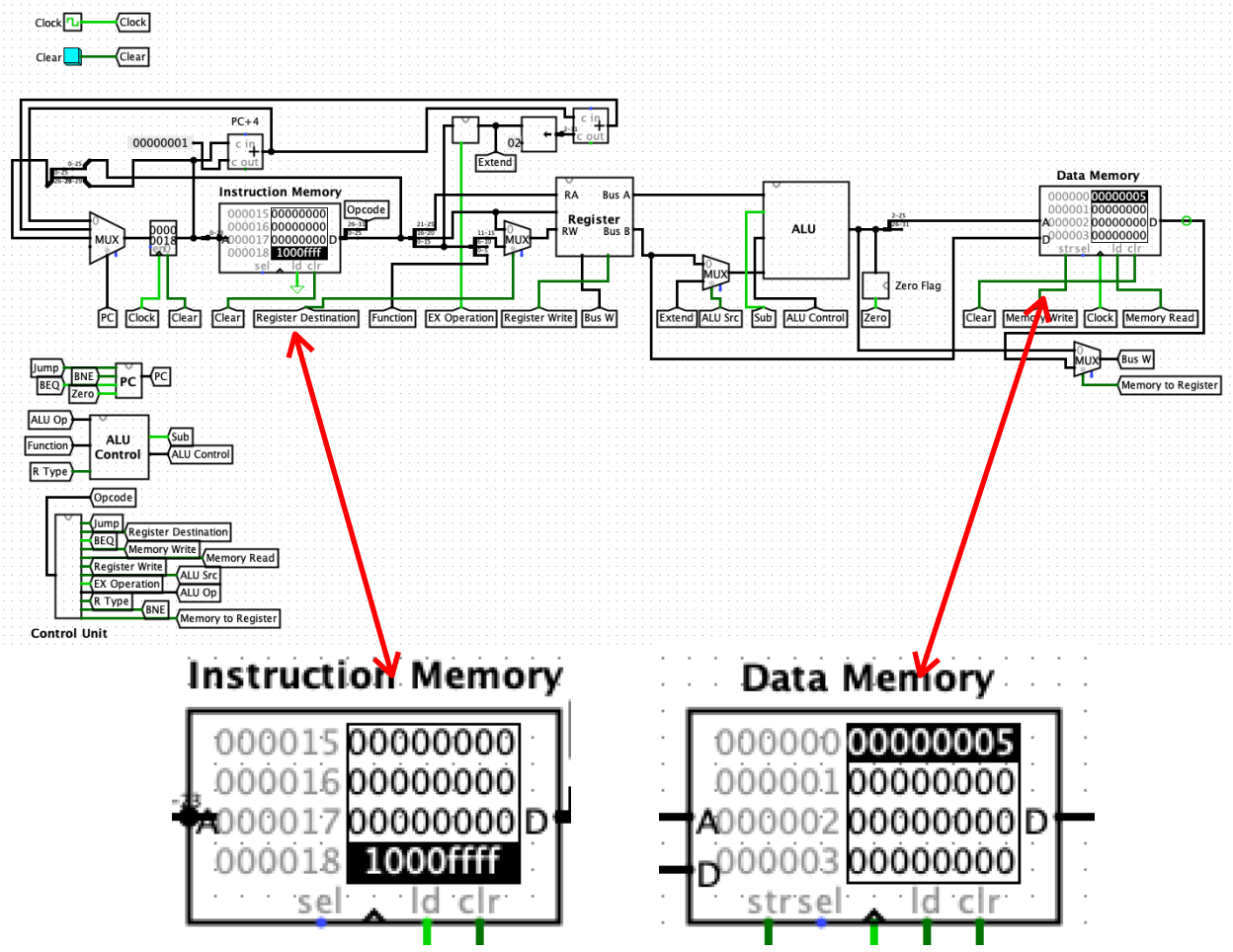


The register file above represents the registers in the CPU implementation. Its inputs include RW, Bus W, RA, Register Write, clock and RB while its outputs include Bus A and Bus B. Bus A and Bus B both serve as output busses for reading two registers. The register file behaves as a combinational block. Once RA or RB have valid data the content of the read register will on the output busses.

Simulation and Testing:

Using MIPS assembly language, and a MIPS to HEX converter from Bucknell University, machine code was generated to test the functionality of the MIPS CPU. The first test program written initializes the registers, and simply adds 5 and stores it in register 4. Below is the assembly code and the matching machine code.

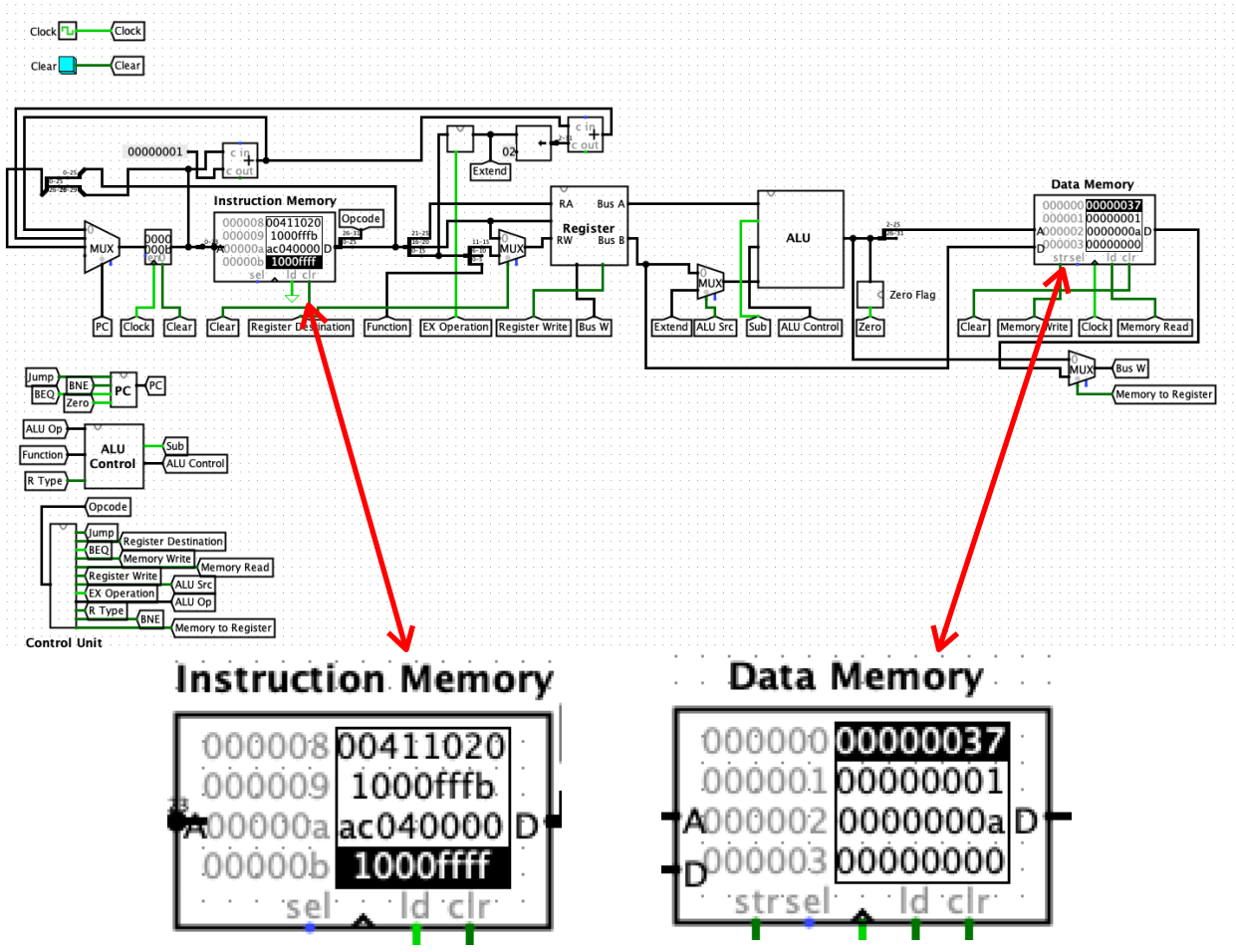
Assembly code:	Machine Code:
`sub r0,r0,r0`	`00000022`
`lw  r1,0(r0)`	`8c010000`
`lw  r2,4(r0)`	`8c020004`
`lw  r3,8(r0)`	`8c030008`
`addi r4, r4, 0x5`	`20840005`
`sw  r4,0(r0)`	`ac040000`
`beq r0,r0,-1`	`1000ffff`



When the above program is finished running, it ends on the last instruction with the value 0x5 in the data memory, as expected.

Utilizing the same methods as above, and test files provided by New York University, we can test more complex code on the CPU for test 2. Below is the assembly and machine code for a program that calculates the sum from 0 to 10 inclusively and stores that value. The expected value is 0x37 or 55 in decimal. Below are those results.

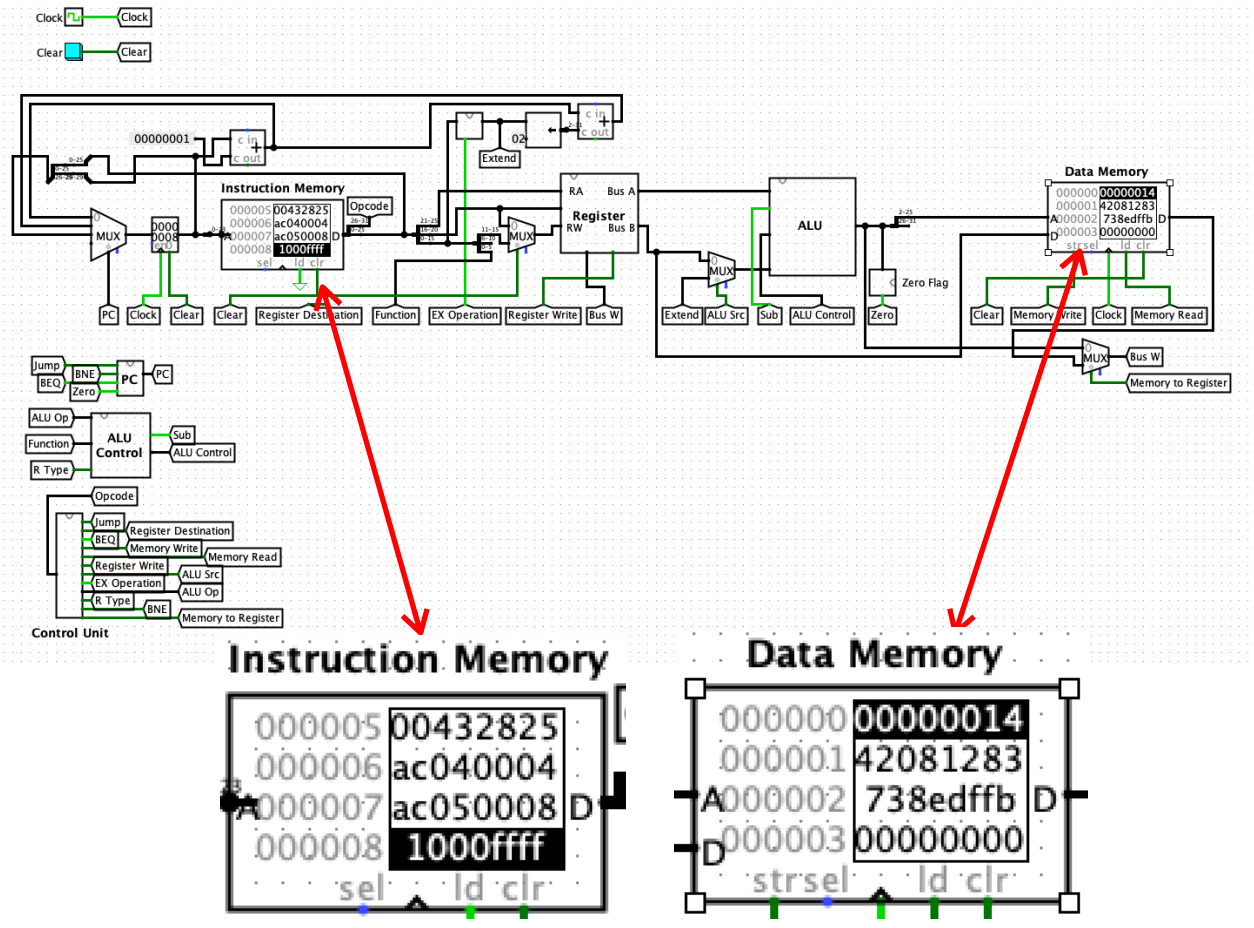
Assembly code:	Machine Code:
`sub r0,r0,r0`	`00000022`
`lw r1,0(r0)`	`8c010000`
`lw r2,4(r0)`	`8c020004`
`lw r3,8(r0)`	`8c030008`
`sub r4,r4,r4`	`00842022`
`add r4,r2,r4`	`00822020`
`slt r5,r2,r3`	`0043282a`
`beq r5,r0,2`	`10a00002`
`add r2,r1,r2`	`00411020`
`beq r0,r0,-5`	`1000fffb`
`sw r4,0(r0)`	`ac040000`
`beq r0,r0,-1`	`1000ffff`



On the page above, we can see the results from the instruction memory and the data memory. The instruction memory has stopped at the last instruction and the first register initialized has the final result: 0x37, or 55 decimal.

Below is the final test conducted with test files. This program tests the AND and OR operations of the CPU utilizing offsets from the base register. The program does the following:

- computes A AND B and places the result in memory location 4 (1)
- computes A OR B and places the result in memory location 8 (1)



As can be seen from the results above, the instruction memory has finished executing its instructions as it is remaining on the last instruction. The resulting registers can be seen in the data memory above with their corresponding values.

Team Work

Devin and Heeral spent most of their time creating the Registers and Memory files and Jessica helped primarily with writing past reports. Myself, Benjamin, created the ALU and all of the single-cycle MIPS CPU circuits. In addition I created the test programs, found other test material, created the data path, and modified all circuits to work together in the CPU. I feel that this group could have done a better job sharing the workload, making themselves available and communicating better. This document has been available for two days before this report was due, so all members have had an opportunity to voice their opinions on team work.

Acknowledgments

1. New York University - Department of Computer Science
https://cs.nyu.edu/courses/fall14/CSCI-UA.0436-001/MIPS_Test_Programs.html
Test programs 1 & 2

2. Bucknell University
https://www.eg.bucknell.edu/~csci320/mips_web/
Instruction to HEX converter