CPU Lab 2

Brian Beilby, 3524

CSC137-02, Computer Organization, Spring 2023

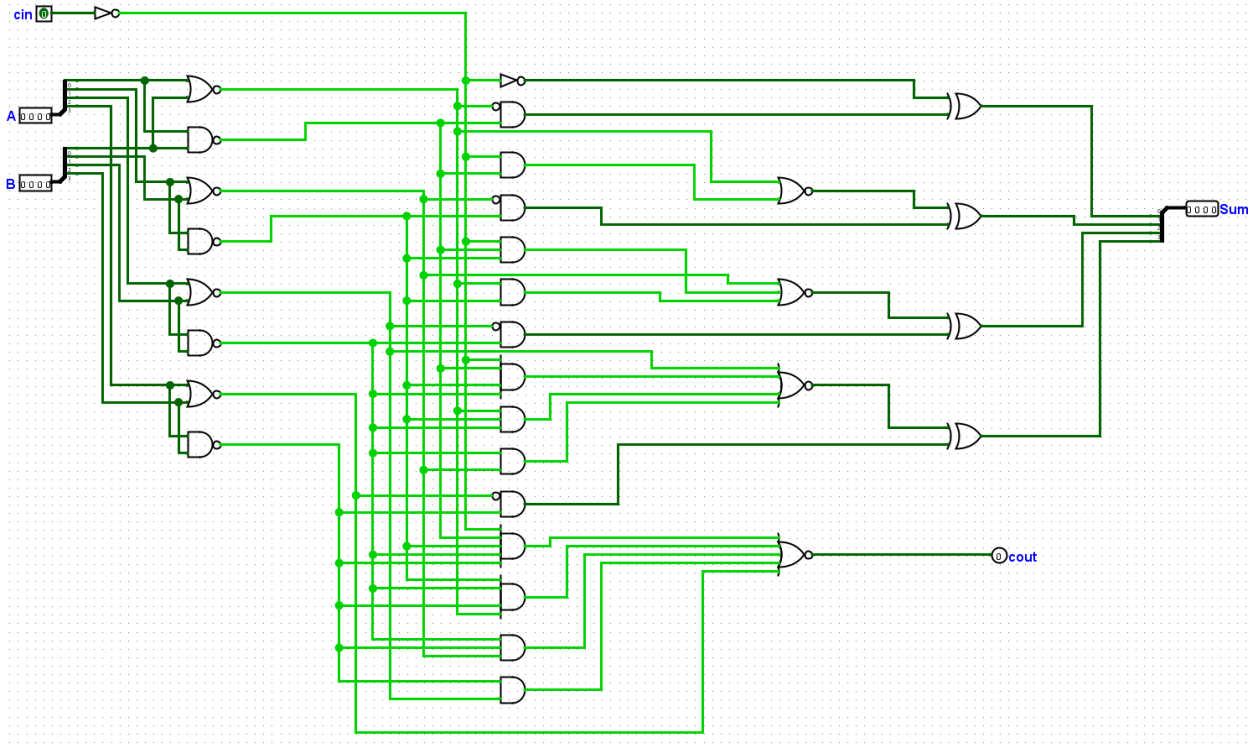
05/09/2023

Introduction

In this lab, I built a simple MISC CPU supporting 13 different instructions. The components of the CPU include an 8k x 12 instruction memory, a 13-bit program counter, a control unit taking a 12-bit input and outputting the corresponding control signals, a 4x8 register file with 3 read ports, an 8-bit ALU using two 4-bit CLA's, a 256 x 8 data memory, and displays mapped to the ports F0 and F1 of data memory. The CPU functions similarly to the CPU I designed in the previous lab. The assembled code is loaded into the instruction memory. The current instruction is loaded into an instruction register temporarily holding its value. The controller, then, generates the proper control signals for the given instruction. If required, the ALU then performs the operation of the instruction given any necessary operands from the register file. Then, the result of the ALU is written to the destination register of the register file specified in the instruction. For branching instructions, the specified target address is simply routed back into the program counter so that the instruction at that address would be executed next.

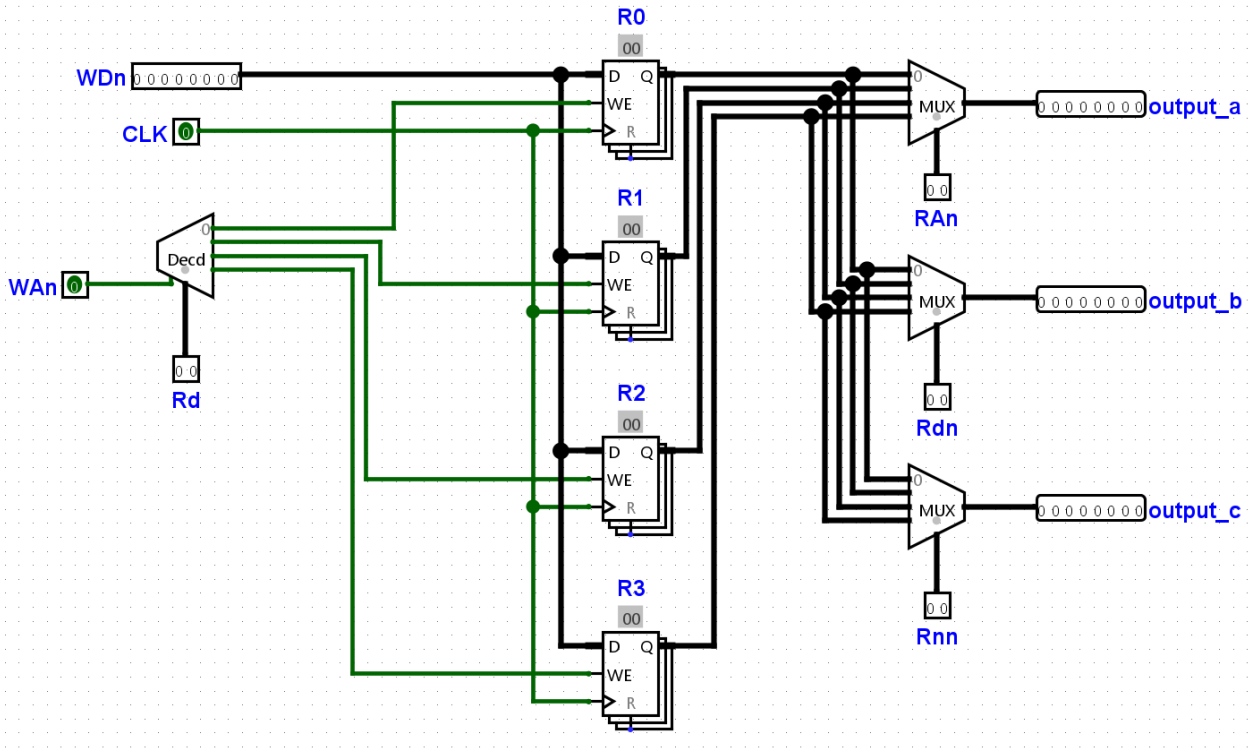
Subcircuit Designs

I. FourBitCLA



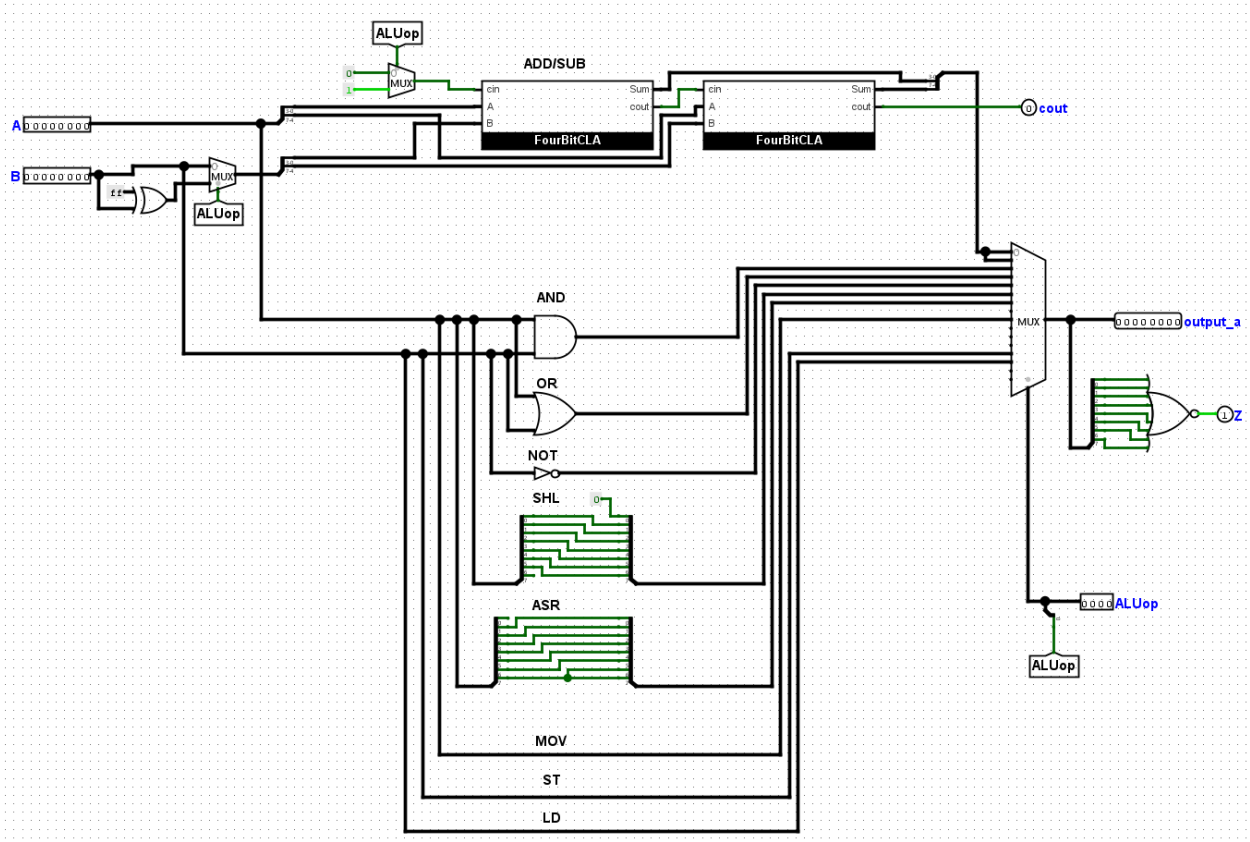
This subcircuit implements a 4-bit carry look-ahead adder. It takes 3 inputs: The two 4-bit values to be summed as well as the 1-bit carry-in input. It has 2 outputs: The 4-bit sum as well as the 1-bit carry-out output. It generates carry signals for each bit of the sum based on the two inputs as well as the carry-in input.

II. Register File



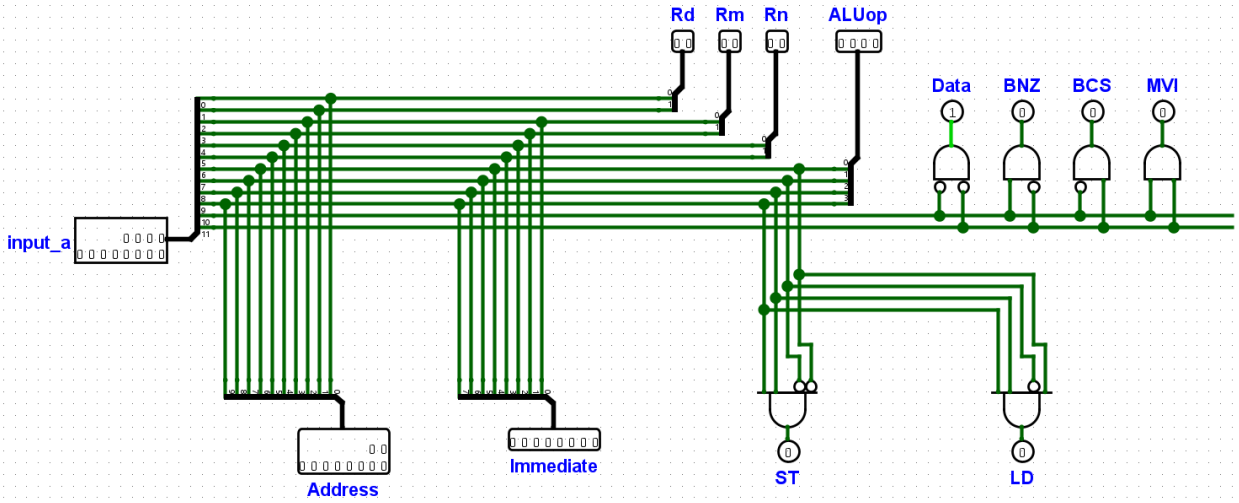
This subcircuit is for my 4x8 register file. The WDn input specifies the 8-bit value that would be written into a specified register. The CLK input is simply the clock input to keep all components of the circuit synchronized. The WAn input is a 1-bit signal that goes into the enable of a decoder determining which register should be written to. If this value is 0, no register would be written to because the write enable would be 0. The Rd input is a 2-bit input going into the select input of the decoder specifying which register to write to. The last 3 inputs are for the 3 different read ports. They each go into the select input of a multiplexer specifying which register to read from. The 3 outputs of this subcircuit are simply the 8-bit values read from each read port.

III. ALU



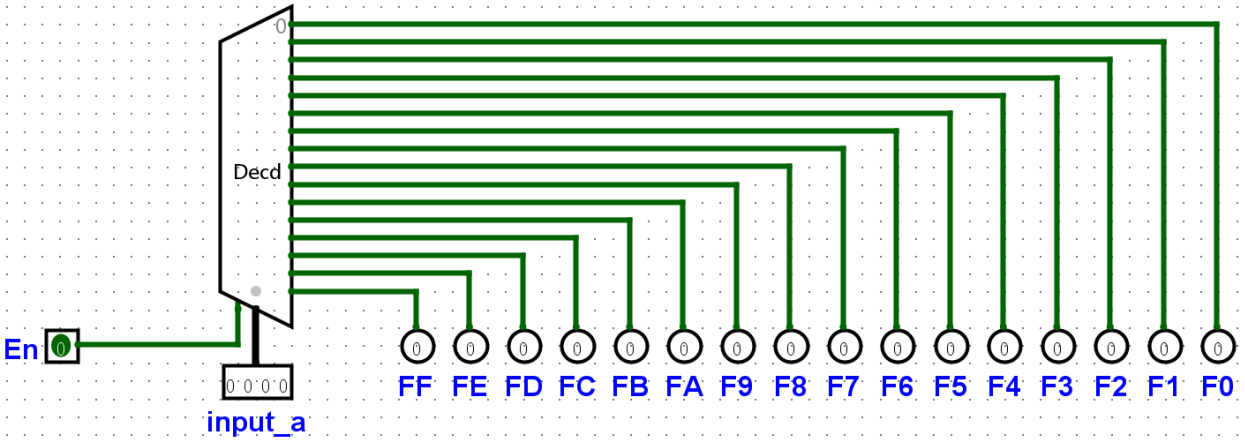
This subcircuit is for my 8-bit ALU. The A input is an 8-bit input representing the Rn component of an instruction. The B input is an 8-bit input representing the Rm component of an instruction. To perform addition and subtraction, two instances of my 4-bit CLA are used. To perform addition, the CLA's are used normally with a carry-in of 0. To perform subtraction, the value of B is inverted using an XOR gate and a carry-in of 1 is used. The AND, OR, and NOT operations are straightforward. The inputs are simply fed into the corresponding gate for the instruction. The SHL operation is implemented using two splitters to shift the bits to the left by 1 discarding the most-significant bit. The ASR operation is also implemented using two splitters to shift the bits to the right by 1 and preserving the sign of the input. For the MOV, ST, and LD operations, no logic is implemented because none is needed. The input is copied to the output of the ALU. The ALUop input is a 4-bit input specifying the opcode of the instruction which is fed into the select input of a multiplexer determining which operation to output. The cout output is simply a 1-bit output that would be 1 if an addition/subtraction resulted in a carry-out. The output_a output is an 8-bit output representing the result of the operation. The Z output is a 1-bit output that would be 1 if the operation resulted with 0.

IV. Controller



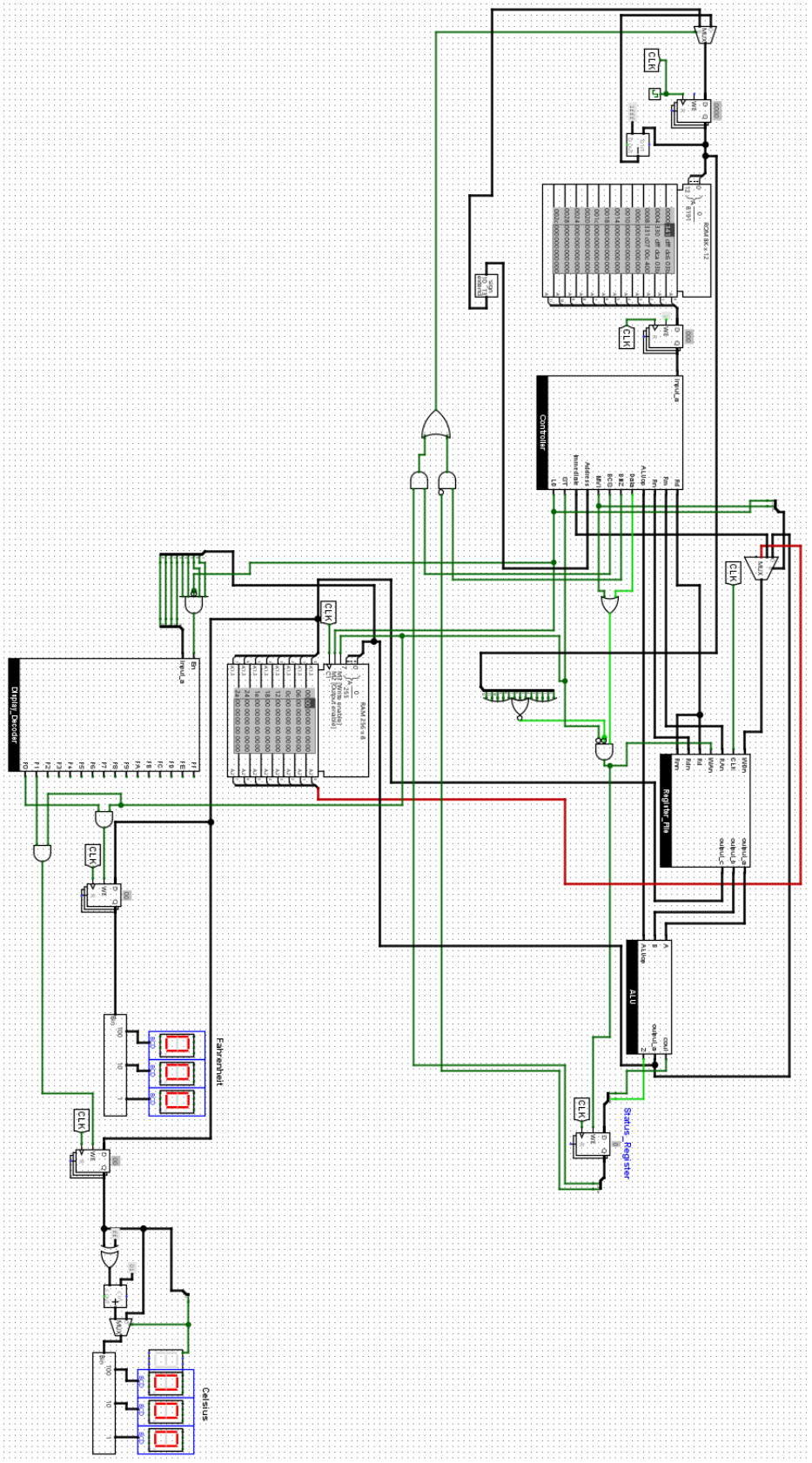
This subcircuit generates control signals decoding the components of the instruction. Its only input is the 12-bit instruction. Each bit of the input is split into single bits using a splitter. 3 register outputs: Rd, Rm, and Rn simply hold the values of the registers specified in the instruction. The ALUop, Address, and Immediate outputs similarly extract the corresponding value from the 12-bit instruction input. The other 6 outputs are 1-bit signals that decode the opcode of the instruction to determine what type of instruction it is. If it is a "Data" instruction, the opcode would be 00 so that output would be 1. Similarly, the other 1-bit outputs are 1 if the opcode specifies that instruction.

V. Display Decoder



This subcircuit is for my Display Decoder. When I got to implementing the displays and memory-mapped I/O is when I really started struggling to understand. This circuit simply takes a 4-bit input and decodes it, determining which of the 16 I/O addresses it corresponds to. The 16 outputs are 1-bit signals corresponding to each of the 16 I/O addresses.

VI. The Main Circuit



This is the main circuit that implements all of my subcircuits to create a fully functioning CPU. The image is rotated in order to make it larger. Hopefully, this isn't an inconvenience. For each instruction in Instruction Memory, the controller decodes it and generates control signals for it. The data is then fed into the ALU, if necessary, which executes the operation of the instruction and reads and writes to and from the register file of the CPU. The Data Memory write enable is high if it is an ST instruction and its output enable is high if it is an LD instruction. Like I mentioned earlier, implementing the displays was a real challenge for me. I could not figure out how to implement a binary to BCD algorithm in a circuit so, unfortunately, I had to use the binary to BCD converters which I know we weren't supposed to use. The Fahrenheit display is mapped to the I/O port F0 and the Celsius display is mapped to the I/O port F1.

Assembly Programming

I. Part A: Lookup Table

```
loop:    ld r1 r0        ;Load the Celsius Value into r1
         mvi r3 127      ;add up to the value of F0
         mvi r2 113
         add r3 r3 r2
         st r0 r3        ;Display the Fahrenheit value
         mvi r3 127      ;add up to the value of F1
         mvi r2 114
         add r3 r3 r2
         st r1 r3        ;Display the Celsius value
         mvi r3 1        ;Increment the Fahrenheit value
         add r0 r0 r3
         bnz 0
```

In this assembly program, I use the Data Memory as a lookup table holding all of the celsius values. The program simply increments through the Fahrenheit values from 0 to 240 (since 16 addresses are I/O ports), and loads the corresponding Celsius value from memory displaying both values on their displays.

Running it in the CPU:

As can be seen from the image in Appendix II, the CPU properly executes the program. The Simulator is paused at the Fahrenheit value of 8 which is -13 in Celsius.

II. Part B: Conversion Formula

I was not able to get this assembly program working to convert the Fahrenheit values to Celsius without using a lookup table. Unfortunately, due to the many Final exams that I need to study for right now, I did not have the time to look into this program much. However, this is what I have so far:

```
        mvi r0 0
        mvi r1 32
convert: mvi r3 127
        mvi r2 113
        add r3 r3 r2
        st r0 r3
        mvi r2 114
        add r3 r3 r2
        st r1 r3
        mov r2 r0
        mvi r3 32
        sub r2 r2 r3
        mvi r3 5
        shl r2
        add r2 r2 r3
        asr r2
        mov r1 r2
        mvi r3 1
        add r0 r0 r3
        mvi r0 -128
        bnz 2
```

Conclusion

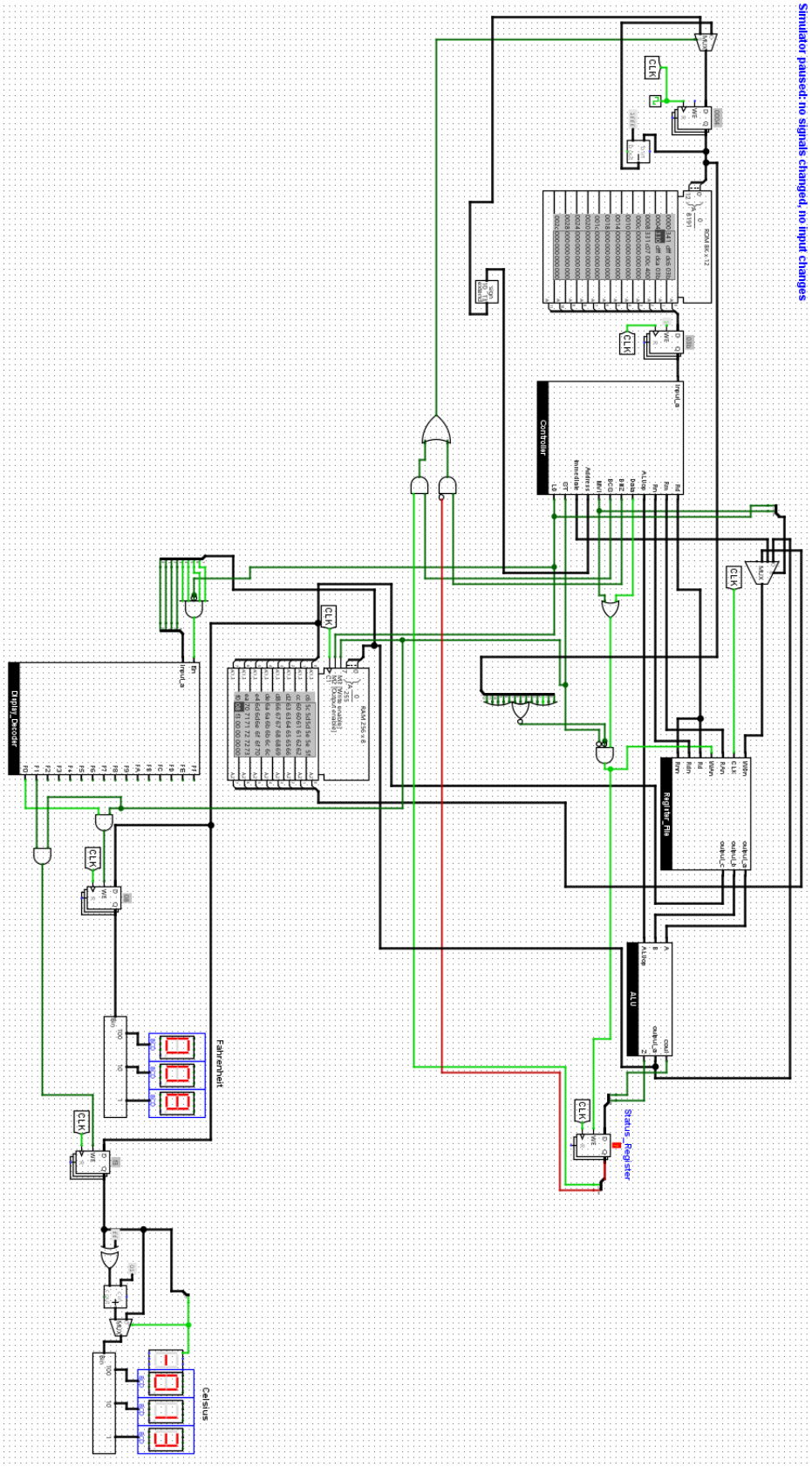
In this lab, I built a fully functioning CPU as well as a Fahrenheit to Celsius conversion program in assembly to test the CPU. Overall, this lab was extremely beneficial to me and I learned so much from it. Not only did I learn the basics of how a CPU functions, I learned how the individual components of a CPU operate such as an ALU and built them in a circuit. Prior to this course, I had no experience with circuitry at all so this experience has really helped me build a good understanding of how computers work at a very low level. Although I wasn't able to figure out the displays and the binary to BCD algorithm, I still built a CPU that I am extremely proud of.

Appendix

I. Assembler Source For tablef2c.s:

```
loop:   ld r1 r0        ;Load the Celsius Value into r1
        mvi r3 127      ;add up to the value of F0
        mvi r2 113
        add r3 r3 r2
        st r0 r3        ;Display the Fahrenheit value
        mvi r3 127      ;add up to the value of F1
        mvi r2 114
        add r3 r3 r2
        st r1 r3        ;Display the Celsius value
        mvi r3 1        ;Increment the Fahrenheit value
        add r0 r0 r3
        bnz 0
```

II. Runtime Output of The Simulator For tablef2c.s:



III. Assembler Source For formulaf2c.s:

```
        mvi r0 0
        mvi r1 32
convert: mvi r3 127
        mvi r2 113
        add r3 r3 r2
        st r0 r3
        mvi r2 114
        add r3 r3 r2
        st r1 r3
        mov r2 r0
        mvi r3 32
        sub r2 r2 r3
        mvi r3 5
        shl r2
        add r2 r2 r3
        asr r2
        mov r1 r2
        mvi r3 1
        add r0 r0 r3
        mvi r0 -128
        bnz 2
```