CSE 3015 DIGITAL LOGIC DESIGN TERM PROJECT

Uğur Alp Yavuz – 150115054

Taylan Rojen Döğer - 150117055

PHASE 1 – Assembler

In the first step of the project we had to create an assembler. To do this we designed an instruction set architecture. Then we wrote an assembler that will convert the instructions contained in this ISA into machine code that the CPU can understand. We preferred java language to write the assembler.

Here is our ISA:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
AND	0	0	0	0	DST	DST	DST	SR1	SR1	SR1	0	0	0	SR2	SR2	SR2
ADD	0	0	0	1	DST	DST	DST	SR1	SR1	SR1	0	0	0	SR2	SR2	SR2
ANDI	0	0	1	0	DST	DST	DST	SR1	SR1	SR1	Imm	Imm	Imm	Imm	Imm	Imm
ADDI	0	0	1	1	DST	DST	DST	SR1	SR1	SR1	Imm	Imm	Imm	Imm	Imm	Imm
LD	0	1	0	0	DST	DST	DST	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR
ST	0	1	0	1	SR1	SR1	SR1	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR
CMP	0	1	1	0	0	0	0	OP1	OP1	OP1	0	0	0	OP2	OP2	OP2
JMP	0	1	1	1	0	0	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR
JE	1	0	0	0	0	0	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR
JA	1	0	0	1	0	0	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR
JB	1	0	1	0	0	0	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR
JBE	1	0	1	1	0	0	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR
JAE	1	1	0	0	0	0	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR	ADDR

In our assembler we get instructions from a .txt file then convert them to hex codes and print them out a .txt file.

Since we have a total of 13 instructions, the allocation of 4 bits for each opcode will suffice.

Here are some code parts from our assembler:

toBinary Method:

```
public static void toBinary(String[] numbers) throws IOException {
    String b;
    String opTyp;
    String[] last = new String[numbers.length];

    // System.out.println(numbers[2]);
    for (int j = 0; j < numbers.length; j++) {
        // System.out.println(numbers[j]);
        opTyp = numbers[j].replaceAll("[,]", "");
        String[] opType = opTyp.split("\\s");
        // System.out.println(opType[0]);
        b = numbers[j].replaceAll("[^-0-9]", " ").trim();
        String[] splitStr = b.split("\\s+");
        // System.out.println(splitStr[]);

        for (int k = 0; k < splitStr.length; k++) {
            int number = Integer.parseInt(splitStr[k]);
            if (number < 0) {
                last[k] = Integer.toBinaryString(number).substring(20);
            } else {
                last[k] = Integer.toBinaryString(number);
            }
            // System.out.println(last[k]);
        }
        instructions(last, opType[0]);
    }
}
```

instructions method:

```
public static void instructions(String[] binaryValue, String opType) throws IOException {
    int count = 0;
    String newString;
    String err = "????";

    // System.out.println(binaryValue);
    if (opType.toUpperCase().equals("AND")) {
        count++;
        // System.out.println(opType);
        for (int k = 0; k < count; k++) {
            newString = "0000" + String.format("%3s", binaryValue[k]).replace(' ', '0')
                    + String.format("%3s", binaryValue[k + 1]).replace(' ', '0') + "000"
                    + String.format("%3s", binaryValue[k + 2]).replace(' ', '0');
            System.out.println(newString);
            toHexadecimal(newString);

        }

    }
```

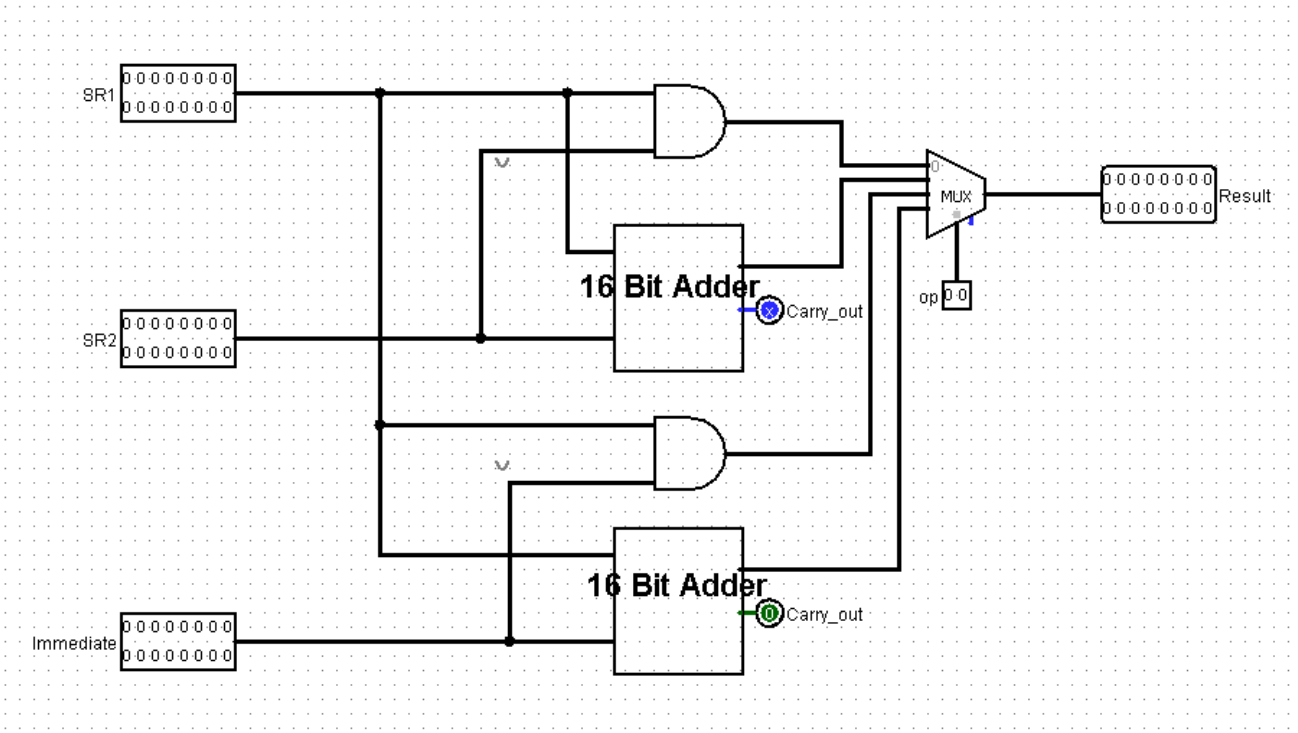
toHexadecimal method:

```
else if (opType.toUpperCase().equals("ADD")) {
    count++;
    // System.out.println(opType);
    for (int k = 0; k < count; k++) {
        newString = "0001" + String.format("%3s", binaryValue[k]).replace(' ', '0')
                + String.format("%3s", binaryValue[k + 1]).replace(' ', '0') + "000"
                + String.format("%3s", binaryValue[k + 2]).replace(' ', '0');
//       System.out.println(newString);
        toHexadecimal(newString);
    }

} else if (opType.toUpperCase().equals("LD")) {
    count++;
    // System.out.println(opType);
    for (int k = 0; k < count; k++) {
        newString = "0100" + String.format("%3s", binaryValue[k]).replace(' ', '0')
                + String.format("%9s", binaryValue[k + 1]).replace(' ', '0');
    //   System.out.println(newString);
        toHexadecimal(newString);
    }

}

else if (opType.toUpperCase().equals("ADDI")) {
    count++;
    // System.out.println(opType);
    for (int k = 0; k < count; k++) {
        newString = "0011" + String.format("%3s", binaryValue[k]).replace(' ', '0')
                + String.format("%3s", binaryValue[k + 1]).replace(' ', '0')
                + String.format("%6s", binaryValue[k + 2]).replace(' ', '0');
//       System.out.println(newString);
        toHexadecimal(newString);
    }

}
```

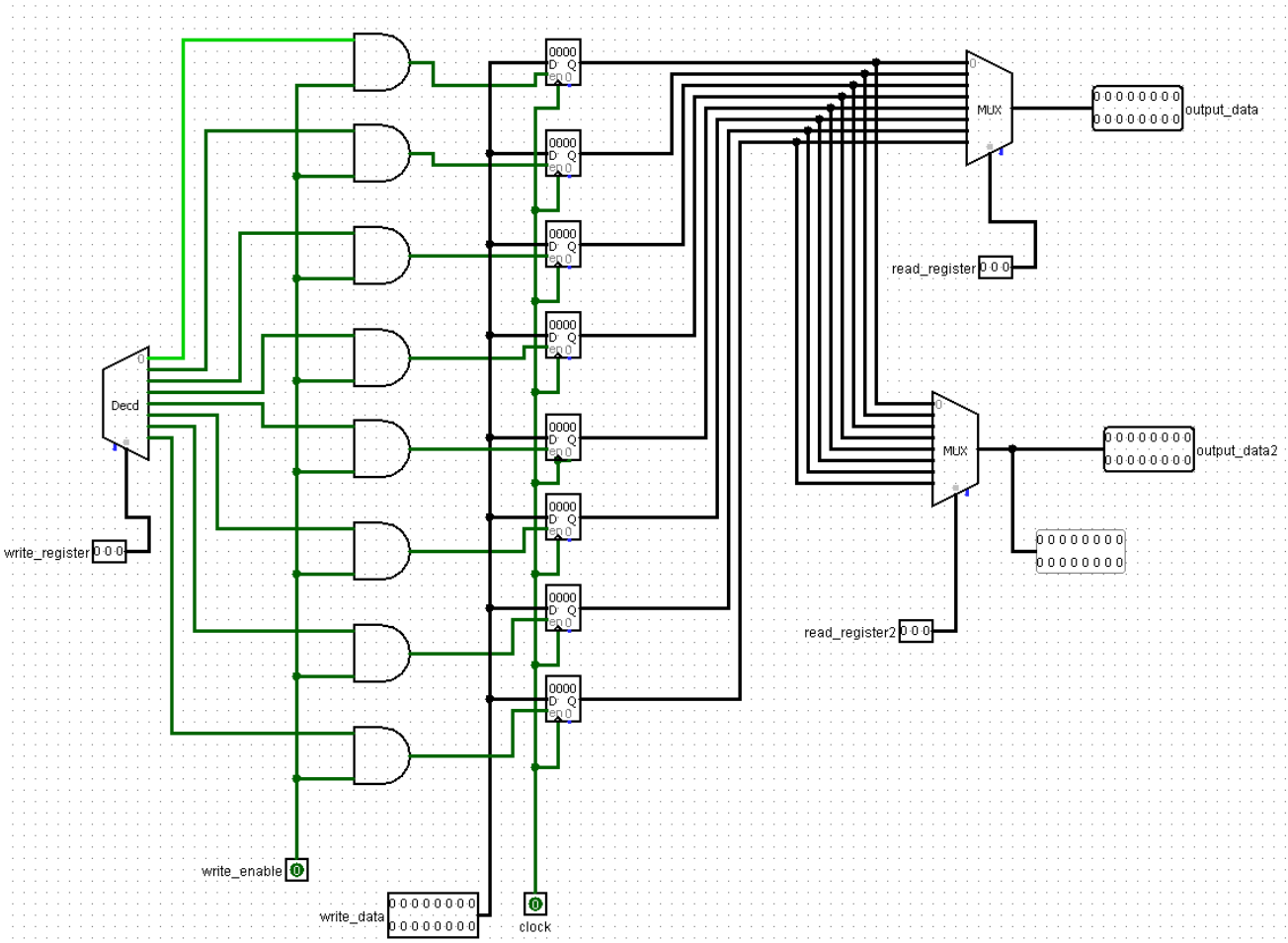
PHASE 2 – Logisim Component Design

In this phase of the project, we are supposed to design our components on the datapath except Control Unit.

Arithmetic Logic Unit:

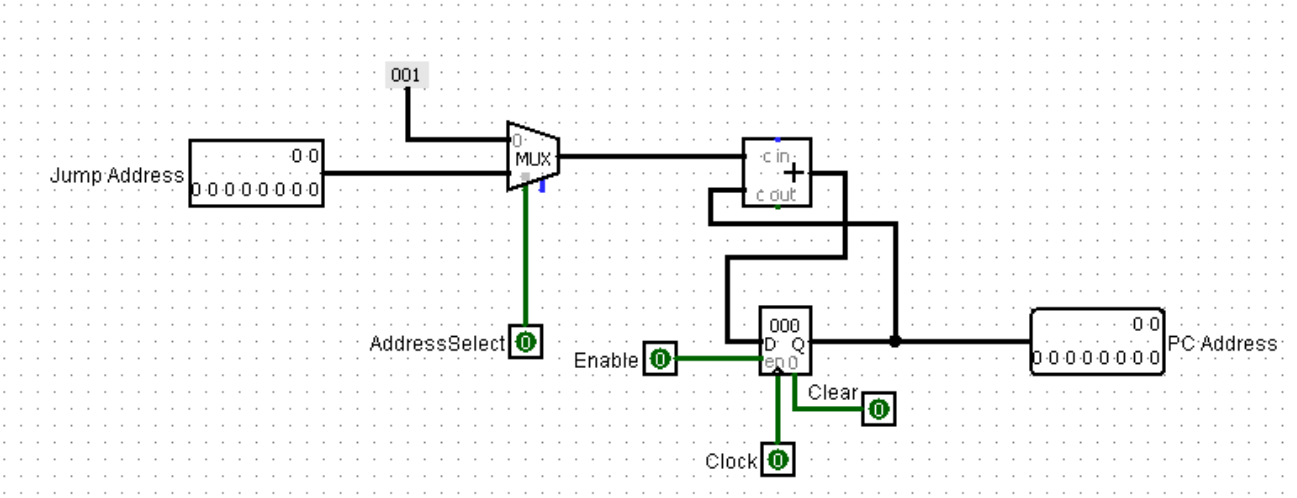


Register File:



(Additionally we created our own 16 bit registers, it can be found in Project.circ)

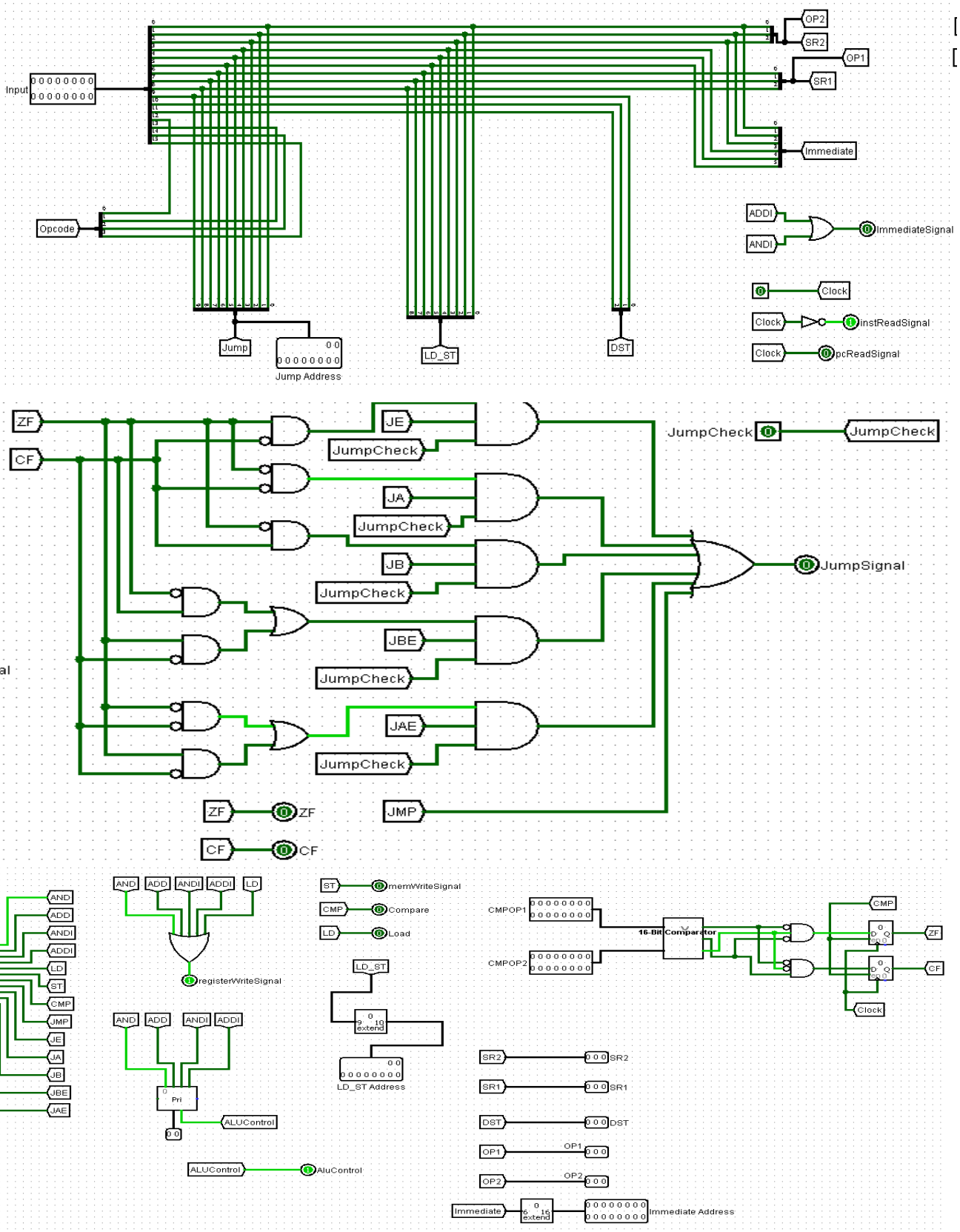
Pc Register:



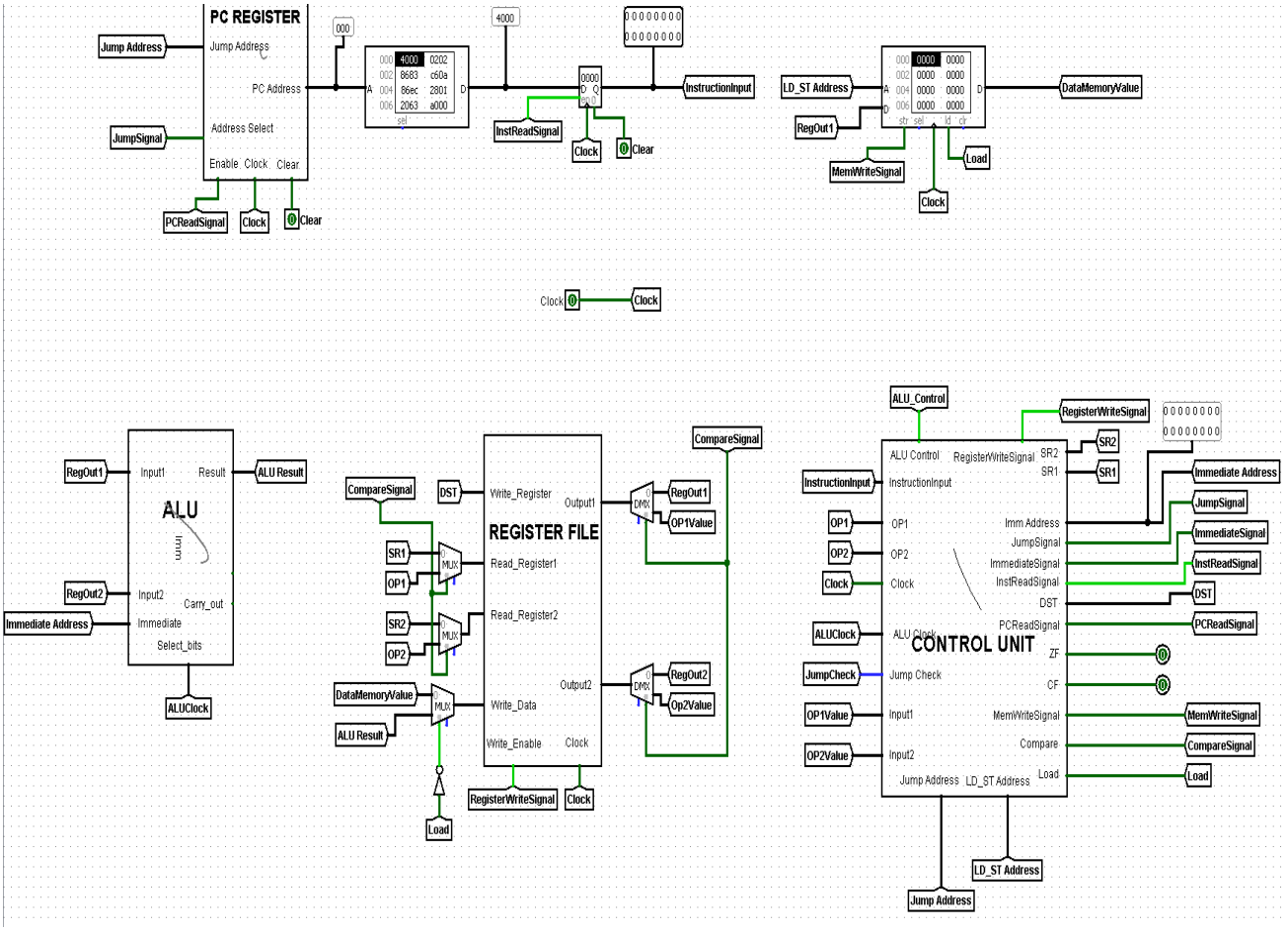
PHASE 3 – Logisim Design with Control Unit

In this phase of the project, we are supposed to design our Control Unit.

Control Unit:



CPU:


PC REGISTER
Jump Address
PC Address
Address Select
Enable Clock Clear
JumpSignal
PCReadSignal
Clock
Clear
InstReadSignal
InstructionInput
LD_ST Address
DataMemoryValue
RegOut1
MemWriteSignal
Load
ALU
RegOut1
RegOut2
Immediate Address
Input1
Input2
Immediate
Result
Carry_out
Select_bits
ALU Result
ALUClock
REGISTER FILE
CompareSignal
DST
Write_Register
Read_Register1
Read_Register2
Write_Data
Write_Enable
Output1
Output2
OP1Value
Op2Value
SR1
SR2
OP1
OP2
DataMemoryValue
RegisterWriteSignal
CONTROL UNIT
ALU_Control
ALU Control
InstructionInput
Clock
ALUClock
JumpCheck
Jump Check
OP1Value
OP2Value
Input1
Input2
Jump Address
LD_ST Address
RegisterWriteSignal
Imm Address
JumpSignal
ImmediateSignal
InstReadSignal
DST
PCReadSignal
ZF
CF
MemWriteSignal
Compare
Load
Immediate Address
JumpSignal
ImmediateSignal
InstReadSignal
DST
PCReadSignal
CompareSignal