DIGITAL DESIGN PROJECT REPORT

150117007 – Edanur Öztürk **150117044 – Sueda Bilen**

150119841 – Zehra Kuru **150517059 – Özge Saltan**

About the Project

In this project we are expected to design a CPU which supports a set of instructions, 8 registers with 16-bit data width. During the design process of the multi-cycle processor, we followed the same strategy to design our circuit as we learned in the lectures. The program counter is updated when an appropriate instruction is fetched for the instruction memory. The instruction is processed in control unit and the output for the given inputs is generated and if needed written in the data memory. We also can use the data memory to get the values to perform our operations. Control unit is the key elements of our circuit as it helps us to work on the instructions. In order to test whether our design works in a correct way, we began with implementing the code to work as an assembler. Then, according to this assembly code, we generated the hexadecimal form of the instructions to give them as an input to the instruction memory part of the Logisim. In addition to these inputs, we gave the data memory inputs, which we wanted to work on during the test, to the Logisim.

Instructions

AND SRC1, SRC2, DST makes and operation between SRC1 and SRC2 and stores into the DST.

ADD SRC1, SRC2, DST adds SRC1 and SRC2 and stores into the DST.

ANDI SRC1, DST, IMM6 makes and operation between SRC1 and IMM6 and stores into the DST.

ADDI SRC1, DST, IMM6 adds SRC1 and SRC2 and stores into the DST.

CMP OP1, OP2 compares OP1 and OP2.Sets ZF and CF values.

JUMP ADDRESS12 sets PC with given value.

JE ADDRESS12 sets PC with the given value if ZF= 1 and CF= 0

JA ADDRESS12 sets PC with the given value if ZF= 0 and CF= 0

JB ADDRESS12 sets PC with the given value if ZF= 0 and CF= 1

JAE ADDRESS12 sets PC with the given value if CF= 0

JBE ADDRESS12 sets PC with the given value if ZF= 1 or CF=1

LD DST, ADDRESS9 will load value from Data Memory to DST register.

ST SRC, ADDRESS9 will store a value from SRC register to Data Memory.

Assembler

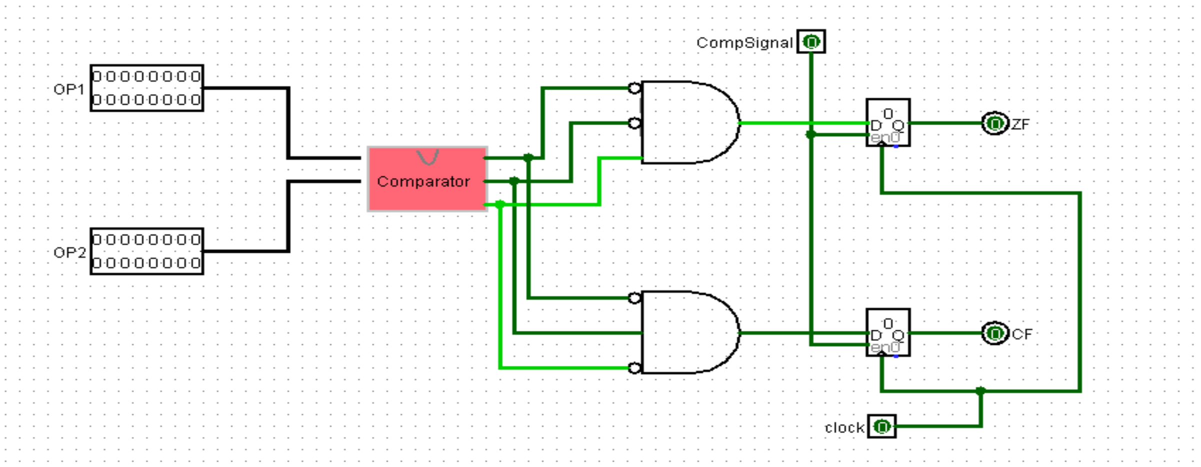
We coded our assembler in Java. Assembler converts assembler code to binary firstly, then it converts binary to hexadecimal. This hexadecimal output is the input of our CPU. We designed assembler according to our instruction set architecture.

We used 4 bits for opcodes and designed our instruction set architecture as 1 to 13 respectively. In other words, we used different opcodes for each instruction and we also implemented our assembler for this order at first.

When we started to work on control unit on Logisim, we saw that this instruction set architecture is difficult to implement fetches. Because of this problem, we changed our instruction set architecture and also assembler.

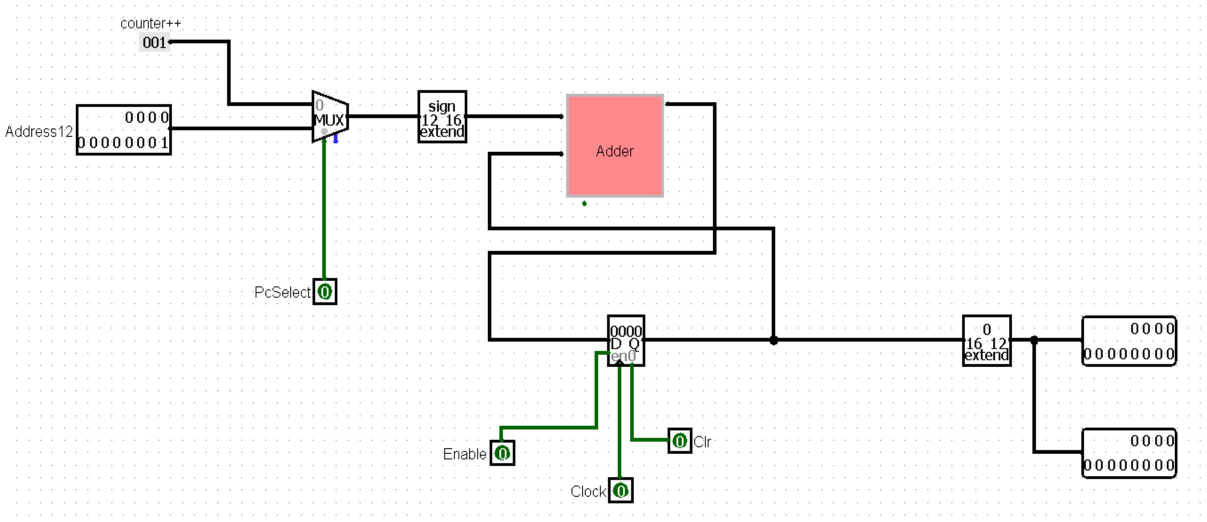
COMPInstruction

This data component works for CMP instruction. CMP is in charge of deciding zero flag and carry flag. Two operands are connected to comparator; the first connection of comparator checks whether it is greater or not, the second connection checks whether it is less than or not and lastly the third one checks the equality. If first operand is greater than the second one, then none of these flags are turned on. If the first operand is less than the second operand, then only carry flag is turned on. If they are equal, then only zero flag is turned on.



Program Counter

Program counter is responsible of keeping value of the next instruction. When one instruction is done, program counter will increase the pc value by one if it's not a jump related instruction. If the instruction is a kind of jump then PcSelect signal will come in to the pc as 1 and pc value will be addressing that the instruction shows add previous pc value.

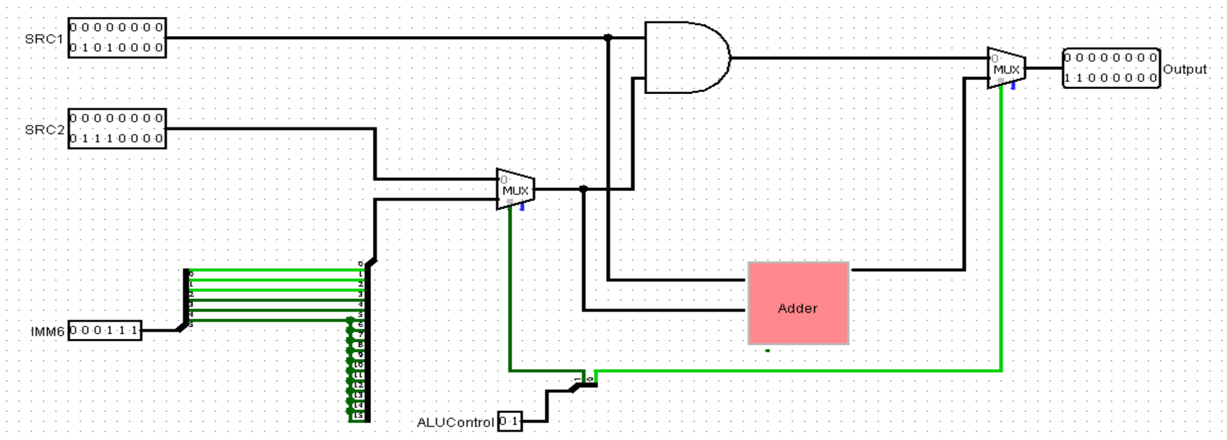


Register File

Register file has input WriteData, WriteRegister, ReadReg1, ReadReg2, writeReg, clock and clear. There are 8 registers that each has 16 bits data-width. WriteData include 16 bits data input. Clock signal connected with all register. Clear signal is cleared the value of registers. WriteRegister is the signal that 3 bits input to select register which to be loaded. ReadReg1 & ReadReg2 are the 3 bits signal that decides which register to read. writeReg is the signal that enables the registers to be activated so that they can be write operation. Register file has output Data1 and Data2. These outputs generate the output based on the ReadReg1&ReadReg2 selected. When writeReg is 1, WriteData loaded in the correct register. When writeReg is 0, the registers keep their values.

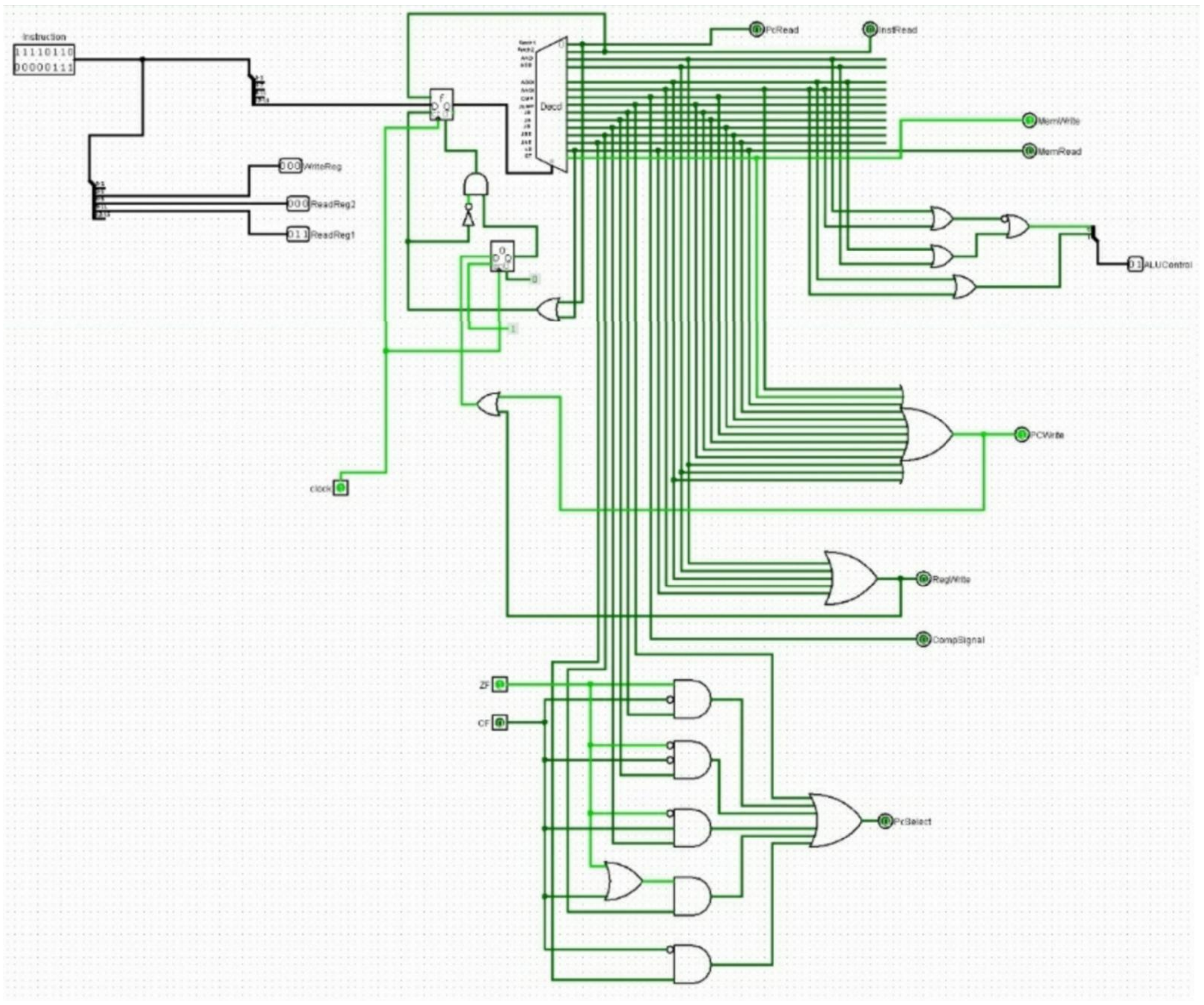
ALU (Arithmetic Logic Unit)

Arithmetic logic unit handles arithmetic operations, in our case ADD, ADDI, AND, and ANDI. ALU has 3 inputs SRC1, SRC2 and IMM6 which is read by SRC1 and SRC2 from Register File and immediate value from instruction memory. ALUControl signal decides to perform the 4 operations on the ALU. ALUControl signal has 2 bits. ALUControl[1] controls the SRC2 or IMM6. ALUControl[0] controls the AND or ADD. If ALUControl signal is 00, SR1 and SRC2 is AND. If it is 01, SRC1 and SRC2 is ADD. If it is 10, SRC1 and IMM6 is ANDI. If it is 11, SRC1 and IMM6 is ADDI.



Control Unit

Control Unit uses decoder to choose signals for data path. If counter value is 0, fetch1 will occur and PcRead signal will be 1. If counter value is 1, fetch2 will occur and InstRead signal will be 1. If the instruction is an ADD, AND, ADDI or ANDI operation then ALUControl signal will be sent to the ALU. If the instruction is a CMP operation CmpSignal will be 1. If the instruction is a JUMP-Relative operation PCSelect signal will be 1 according to flag values. If the instruction is a LD operation MemRead signal will be 1. If the instruction is a ST operation MemLoad signal will be 1.



FINITE STATE MACHINE

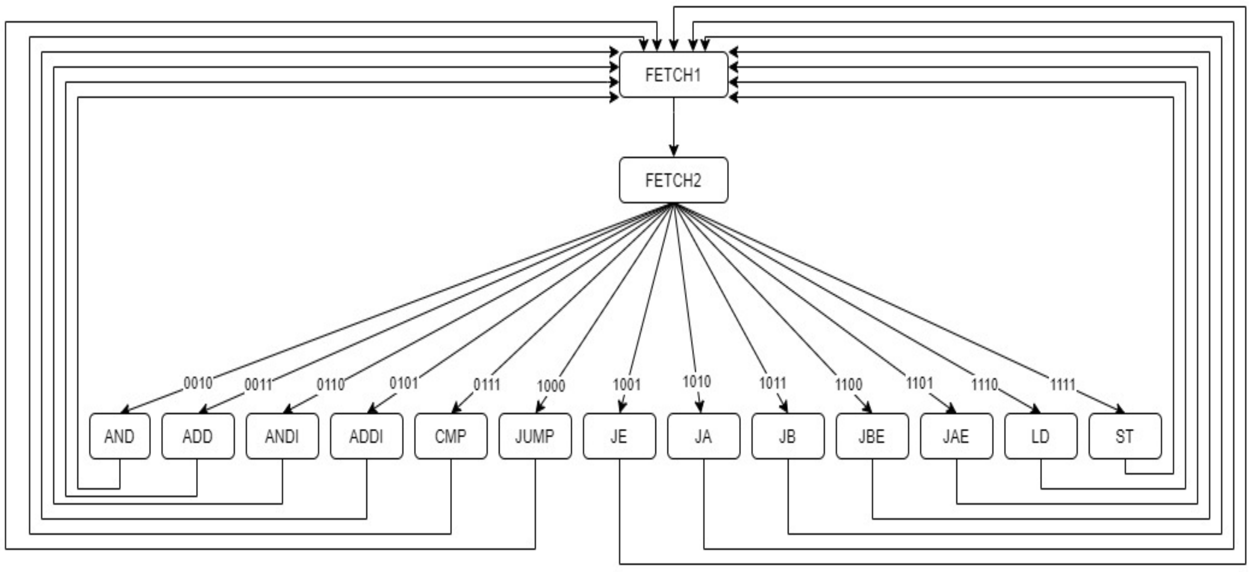


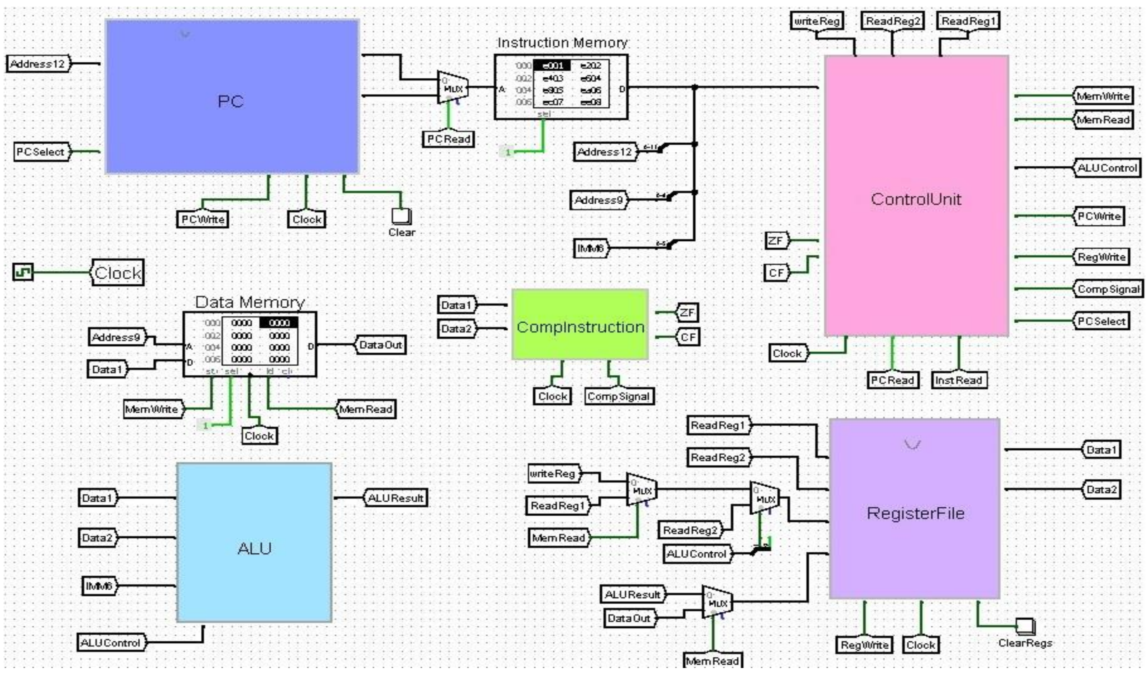
Figure 1-Finite State Machine Diagram

Inputs								Outputs													
Opcode				CurrentState				PcRead	InstRead	MemWrite	MemRead	ALUControl		PCWrite	RegWrite	CompSignal	PcSelect	NextState			
[3]	[2]	[1]	[0]	[3]	[2]	[1]	[0]	[0]	[0]	[0]	[0]	[1]	[0]	[0]	[0]	[0]	[0]	[3]	[2]	[1]	[0]
x	x	x	x	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0					0	1	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	1					0	1	0	0	0	0	0	0	0	0	0	0	1	1
0	1	1	0					0	1	0	0	0	0	0	0	0	0	0	1	1	0
0	1	0	1					0	1	0	0	0	0	0	0	0	0	0	1	0	1
0	1	1	1					0	1	0	0	0	0	0	0	0	0	0	1	1	1
1	0	0	0					0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	1
1	0	1	0					0	1	0	0	0	0	0	0	0	0	1	0	1	0
1	0	1	1					0	1	0	0	0	0	0	0	0	0	1	0	1	1
1	1	0	0					0	1	0	0	0	0	0	0	0	0	1	1	0	0
1	1	0	1					0	1	0	0	0	0	0	0	0	0	1	1	0	1
1	1	1	0					0	1	0	0	0	0	0	0	0	0	1	1	1	0
1	1	1	1					0	1	0	0	0	0	0	0	0	0	1	1	1	1
x	x	x	x	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
x	x	x	x	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
x	x	x	x	0	1	1	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0
x	x	x	x	0	1	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0
x	x	x	x	0	1	1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0
x	x	x	x	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
x	x	x	x	1	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0
x	x	x	x	1	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
x	x	x	x	1	0	1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0
x	x	x	x	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
x	x	x	x	1	1	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0
x	x	x	x	1	1	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0
x	x	x	x	1	1	1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0

Figure 2-Finite State Machine Truth Table

CPU

Our CPU consists of 7 main components. These are **Control Unit, ALU, Register File, PC, Data Memory, Instruction Memory, CompInstruction.** We wanted to make a clear, understandable and readable CPU so that we used tunnels for transmitting the signals. Instruction Memory reads and extracts the loaded input, after that Control Unit decides which components to execute the instruction then send signals to them.



PC
Address12
PCSelect
PCWrite
Clock
Clear
PCRead
Instruction Memory
Address12
Address9
IMM6
writeReg
ReadReg2
ReadReg1
ControlUnit
MemWrite
MemRead
ALUControl
PCWrite
RegWrite
CompSignal
PCSelect
ZF
CF
Clock
PCRead
InstRead
Clock
Data Memory
Address9
Data1
DataOut
MemWrite
MemRead
Clock
Data1
Data2
CompInstruction
ZF
CF
Clock
CompSignal
ALU
Data1
Data2
IMM6
ALUControl
ALUResult
ReadReg1
ReadReg2
writeReg
ReadReg1
MemRead
ReadReg2
ALUControl
ALUResult
DataOut
MemRead
RegisterFile
Data1
Data2
RegWrite
Clock
ClearRegs