CSE315 DIGITAL LOGIC DESIGN TERM PROJECT REPORT

Instruction Set Architecture and Extra FSM Opcodes

In our project, we've started to declare opcodes from 0010 to enable fetch1 and fetch2 in our finite state machine design in Control Unit.

As seen below, ALU operations with registers have same opcode. In order to distinguish those instructions, we've inserted 2 additional bits at the end of the binary represantations. With the help of these two bits, ALU understands that which operations should've been done.

	[17:14](opcode)	[13:10]	[9:6]	[5:2]	[1:0]
AND	0010	DST	SRC1	SRC2	00
ANDI	0100	DST	SRC	Imm6	
ADD	0010	DST	SRC1	SRC2	01
ADDI	0011	DST	SRC	Imm6	
OR	0010	DST	SRC1	SRC2	10
ORI	0101	DST	SRC	Imm6	
XOR	0010	DST	SRC1	SRC2	11
XORI	0110	DST	SRC	Imm6	
ST	1001	SRC	ADDR		
LD	0111	DST	ADDR		
JUMP	1010	ADDR			
BEQ	1011	OP1	OP2	ADDR	
BGT	1100	OP1	OP2	ADDR	
BLT	1101	OP1	OP2	ADDR	
BGE	1110	OP1	OP2	ADDR	
BLE	1111	OP1	OP2	ADDR	

FETCH1	0000
FETCH2	0001
LOAD2	1000

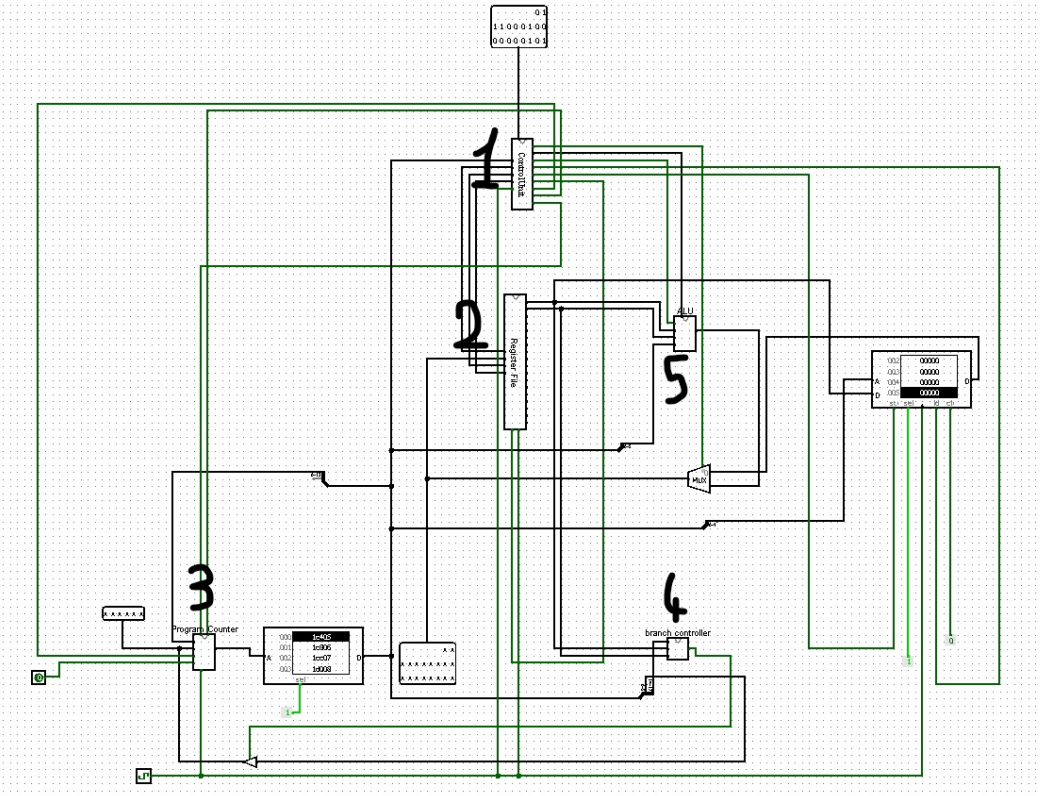
Inputs & Outputs

As seen below, our inputs have been converted to hexadecimal notation as an output by our Assembly Code written in C.

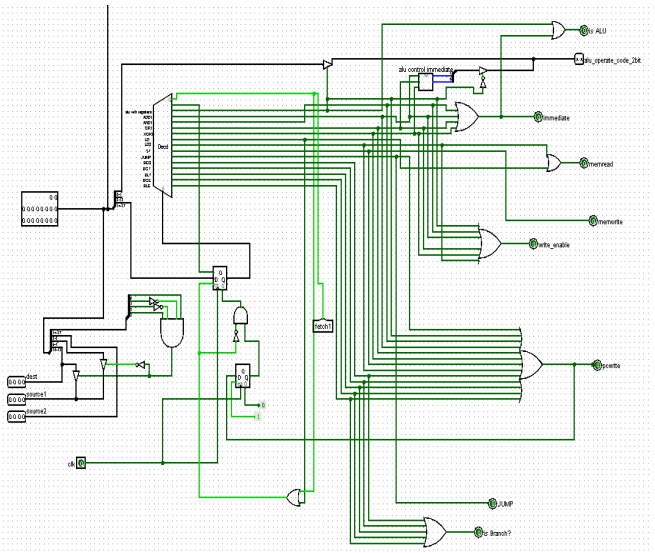
LD R1,5	v2.0 raw
LD R2,6	1C405
LD R3,7	1C806
LD R4,8	1CC07
LD R5,9	1D008
LD R6,10	1D409
LD R7,11	1D80A
ADD R8,R1,R2	1DC0B
ADDI R9,R3,12	0A049
AND R10,R3,R4	0E4CC
XOR R11,R5,R6	0A8D0
OR R12,R7,R8	0AD5B

Logisim Design

Our CPU design with external ROM and RAM components are given below. Each number indicates different component, which will be explained later.



1)Control Unit

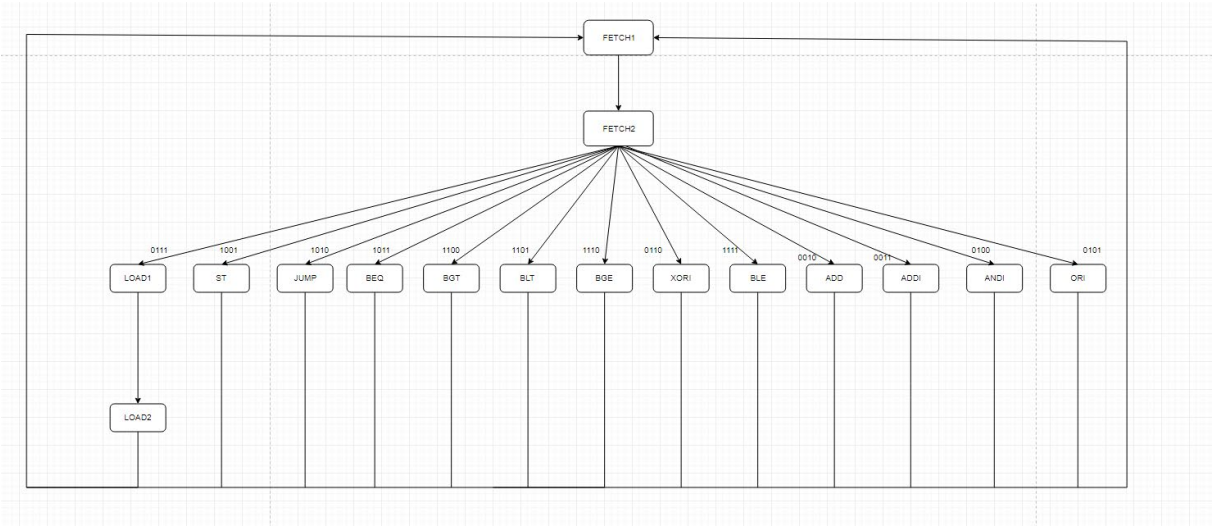


• Control unit works as a brain of our CPU. Each instruction directly goes to control unit to declare which operation must be executed. Every output signal exists for different functionality among different parts of components. For example, “write_enable” signal allows us to change Program Counter, “is Branch?” stands for activation of “Branch Controller” component.

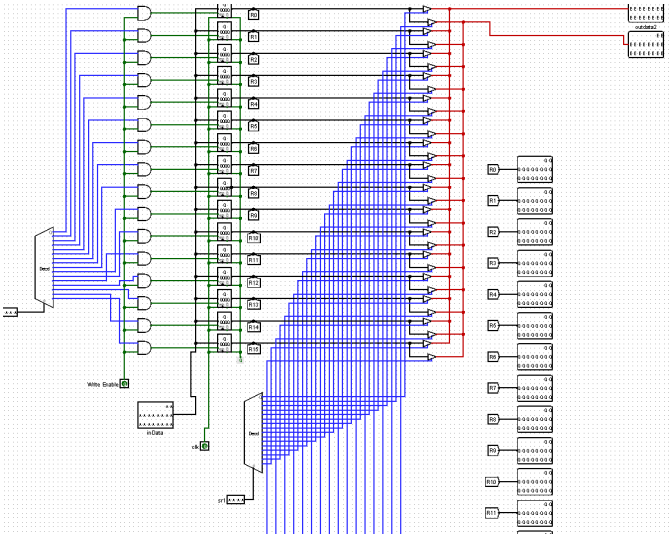
Finite State Machine

To prevent asynchronous operations, we’ve created a finite state machine. All of the instructions have 3 states except LOAD operation. As seen below, our finite state machine enables to do operations without any conflict.

(*P.S: Since the opcodes of ALU operations with registers are same, finite state machine only shows the ADD operation as a label*.)

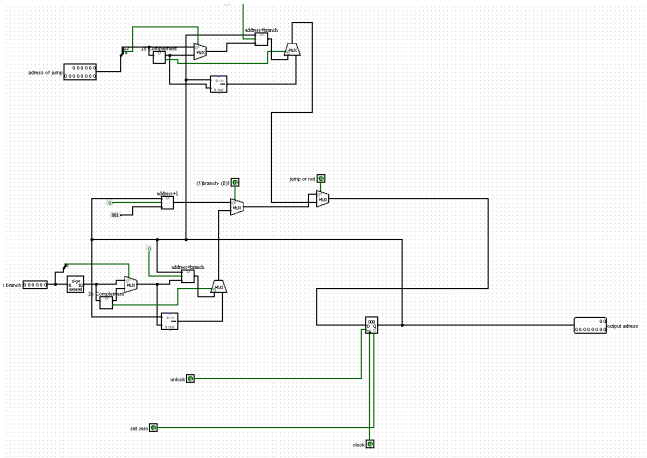


2)Register File



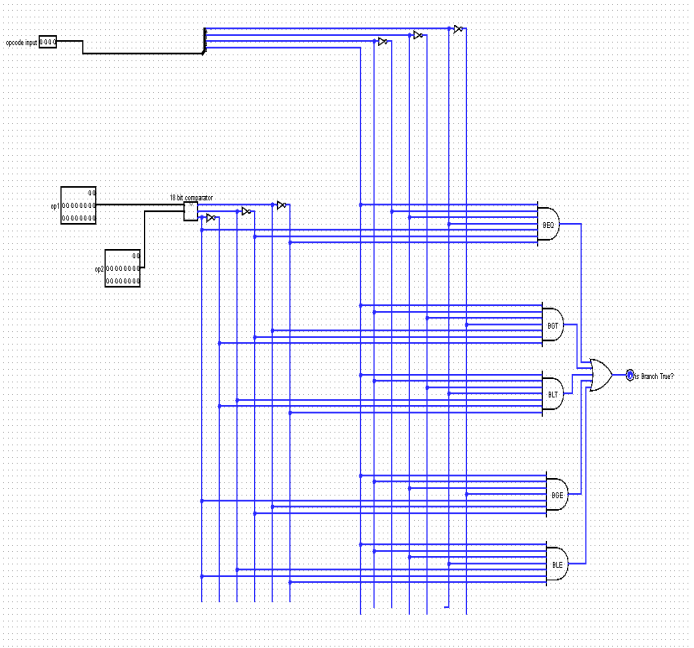
• Standard register file design of our CPU. There are 16 registers in this component which enable us to do various operations such as ADD,AND,OR,XOR,LOAD etc. Destination and Source Registers are declared according to outputs of Control Unit. Before doing ALU operations, LOAD operation must be executed to fill registers.

3)Program Counter



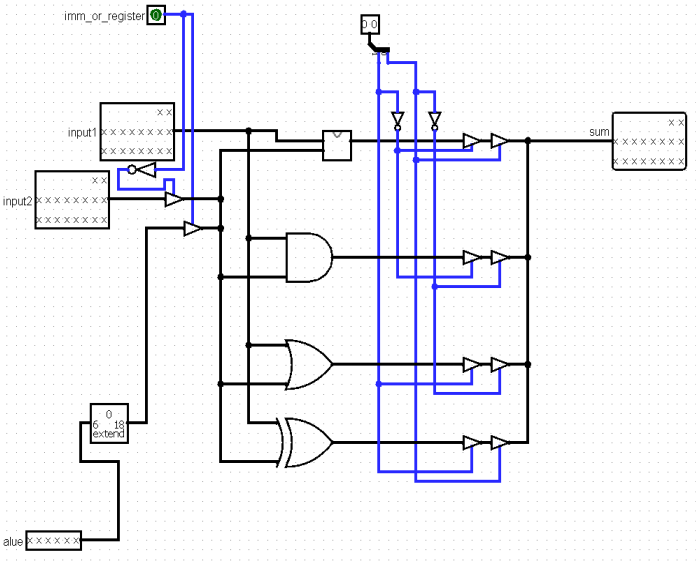
• If the given operation is ALU, our program counter increases pc by one, if it is BRANCH or JUMP operation, it jumps to the corresponding index (pc-relative) in our system. Program counter understands the instruction of current operation by its inputs.

4)Branch Controller



• Branch Controller only works if the current instruction is Branch operation (BEQ,BGT,BLT,BGE,BLE). Firstly without checking the opcode, it compares given two inputs from the angle of greaterness, equalness or lessness. After doing this operation it gives an output to the corresponding AND gate. Then, opcodes are enabling to give an output signal which is labeled as “is Branch True?”.

5)Arithmetic Logic Unit



• Firstly our ALU understands whether the given instruction is with Immediate Value or not. Then, according to the given instruction, it directs the signal to corresponding path and does the arithmetic logic. ALU uses registers as an input parameter, so before doing an ALU operation, registers should be filled with data.

A.Tunahan Cinsoy *Muhammed F. Eroğlu* *A. Enes Gündüz*
150117062 *150116022* *150116082*