**CSE 3015 PROJECT REPORT**

Sena Altıntaş – 150118007

Emine Çığ – 150118012

Mehmet Akif Akkaya – 150118041

**1.DEFINATION OF THE PROJECT**

In this project, our goal is to design a basic processor that perfoms some instructions. The processor we designed has an 18 bit adresss size, 18 bits data size and 16 registers. Data memory is10 bits address width, and 18 bits data width.

**1.A Instructions**

The **ADD** instruction works with 3 registers. It sums the data in the SRC1 register and the data in the SRC2 register and writes it to the DEST register.

ADD DST,SRC1,SRC2 : SRC2 + SRC1 -> DEST

The **ADDI** instruction works with 2 registers and one immediate value . It sums the data in the SRC1 register and the IMM and writes it to the DEST register.

ADDI DST,SRC1,IMM: IMM + SRC1 -> DEST

The **AND** instruction works with 3 registers. It performs the “and” operation with the data in the SRC1 register and the data in the SRC2 register and writes the result to the DEST register.

AND DST,SRC1,SRC2 : SRC2 & SRC1 -> DEST

The **ANDI** instruction works with 2 registers. It performs the “and” operation with the data in the SRC1 register and IMM.Then writes the result to the DEST register.

ANDI DST,SRC1,IMM : IMM & SRC1 -> DEST

The **OR** instruction works with 3 registers. It performs the “or” operation with the data in the SRC1 register and the data in the SRC2 register and writes the result to the DEST register.

OR DST,SRC1,SRC2 : SRC2 || SRC1 -> DEST

The **ORI** instruction works with 2 registers. It performs the “or” operation with the data in the SRC1 register and IMM.Then writes the result to the DEST register.

ORI DST,SRC1,IMM : IMM || SRC1 -> DEST

The X**OR** instruction works with 3 registers. It performs the “xor” operation with the data in the SRC1 register and the data in the SRC2 register and writes the result to the DEST register.

XOR DST,SRC1,SRC2 : SRC2 xor SRC1 -> DEST

The X**ORI** instruction works with 2 registers. It performs the “xor” operation with the data in the SRC1 register and IMM.Then writes the result to the DEST register.

XORI DST,SRC1,IMM : IMM xor SRC1 -> DEST

The **JUMP** instruction gets the signed ADDR value that PC- relative address. By adding this Addr value to the PC value, it obtains the new PC value.

JUMP ADDR : PC + ADDR ->PC

The **LD** instruction loads the data in the address value in the data memory that matches the ADDR value to the DST register.

LD DST,ADDR

The **ST** instruction stores the data in the SRC register at the address in the data memory that matches the ADDR.

ST SRC,ADDR

**BEQ, BLT,BGT,BLE,BGE** instructions take register number named OP1 and OP2 and ADDR address value. According to the OP1 and OP2 comparison result, it jumps to the address that matches the ADDR.

In the **BEQ** instruction, the equality of the data in the OP1 and OP2 directory is checked and set as nzp 010.

BEQ OP1,OP2,ADDR

In the **BLT** instruction, it is checked that the data in the OP1 register is smaller than the data in the OP2 register and nzp is set to 100.

BLT OP1,OP2,ADDR

In the **BGT** instruction, it is checked that the data in the OP1 register is greater than the data in the OP2 register and nzp is set to 001.

BGT OP1,OP2,ADDR

In the **BLE** instruction, it is checked that the data in the OP1 register is smaller or equal than the data in the OP2 register and nzp is set to 110.

BLE OP1,OP2,ADDR

In the **BGE** instruction, it is checked that the data in the OP1 register is greater or equal than the data in the OP2 register and nzp is set to 011.

BGE OP1,OP2,ADDR

**2. ASSEMBLER (ITERATION 1)**

We created the instruction set architecture (ISA) according to the instructions supported by our process.

First of all, since there are 18 instructions, we have determined a 4-bit opcode area to separate them.

The first 3 bits of the opcode field ([16:14]) are reserved for nzp for branch instructions. For example, when the nzp value for blt is 100, the first three bits of the opcode ([16:14]) are 100. Since the next field OP1([13:10]) and OP2([9:6]) are also registers, four bits each are allocated to them. Remaining 6-bit space([5:0] allocated to ADDR.

Since there are 16 registers, it is sufficient to allocate a 4-bit field to the registers. For this reason, DEST ([13:10]) and SRC1([9:6]) four-bit fields are reserved for ALU operations. The remaining 6-bit space is given for IMM or SRC2.

For LD and ST instructions, the remaining space ([9:0]) is given to ADDR after the 4 bits after the opcode ([13:10]) are allocated to the register.

For the jump instruction, after the 4-bit opcode area is reserved, the remaining 14-bit area [13:0] is allocated to ADDR.

Table

Description automatically generated

Figure 1: ISA

After determining the area to be allocated according to the instruction, each input value given to be used as input in the processor, firstly we will create was converted into binary according to ISA using Java. Then the output file was created by converting the binary values to hex values. While converting decimal to binary, depending on the situation, two's complement or unsigned converter is used. Additionally for each different instruction its own special function is created.

**3. LOGISIM COMPONENT DESIGN (ITERATION 2)**

**Text

Description automatically generated**

*Figure 2 Component Menu*

**3.1 Register File**

The Register File holds data for 16 register on each register components , which outputs the data in the selected registers. In the register file, the input data is used for the incoming data and the destreg input, which takes the value of which register will store the input data.

Enable signal called registerWrite is used. When this signal is 1, the processor can collect data in register. In addition, clock and clear signals are also given as inputs.

In addition, the decoder was used to select the register that stores the given data, and the multiplexer that takes the register number as the select bit and is used to select the data in the registers is used.

**3.2. Adder**

In this iteration, the half adder is used to add the 0th bits. The full adder was used to collect the later bits. Adder, which takes 2 18-bit inputs, was added using half adder and full adder. Carry in and carry out were taken into account while doing this.

**3.3. ALU**

In order for the data in the incoming registers to give results according to the instruction; Various operations such as AND, OR, ADD, XOR have been created within the ALU.

**3.4. Comparator**

In order to compare the data in the OP1 and OP2 registers in the branch instructions, a Comparator was created, which takes 2 18-bit inputs and outputs the nzp value. When creating this comparator, a 1-bit comparator was created first, which performs 1-bit comparisons, and then a 3-bit comparator that performs 3-bit comparisons using 1-bit comparator.

**3.5. 6-18 bit extender (IMM)**

6-bit space is reserved in ISA for Immediate (IMM). In order to use this value with the data in the SRC2 register, a 6-18 bit extender was created that extends to 18 bits. Two's complement has been taken into account.

**3.6. 14-18 bit extender (jump-ADDR)**

14 bits are reserved in Isa for ADDR in Jump instruction. In order to use the address value here, a 14-18 bit extender was created. Two's complement has been taken into account.

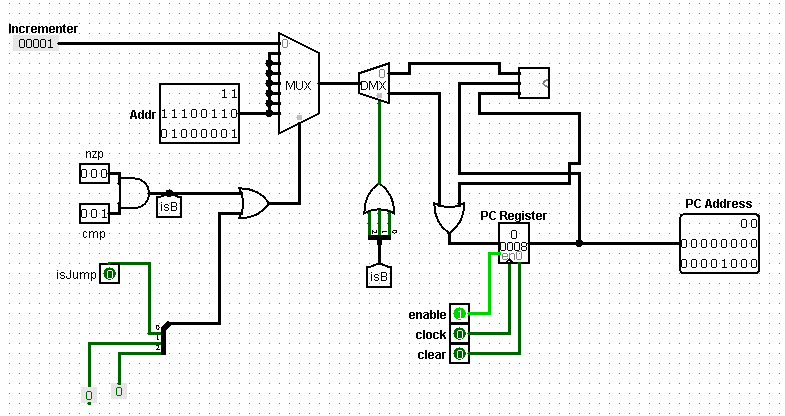
**3.7. Unsigned Extender (6-18)**

A 6-bit field is reserved for the ADDR value in the branch instructions. In order to use this address value, 6-18 bit unsigned extender has been created.

**3.8 PC Controller**

PC controller circuit changes and store current PC value on a register depending on the isJump and isBr signals . Circuit first determines operation type whether gets Addr or incrementer value , then checks PC is added with this value or is assigned directly.

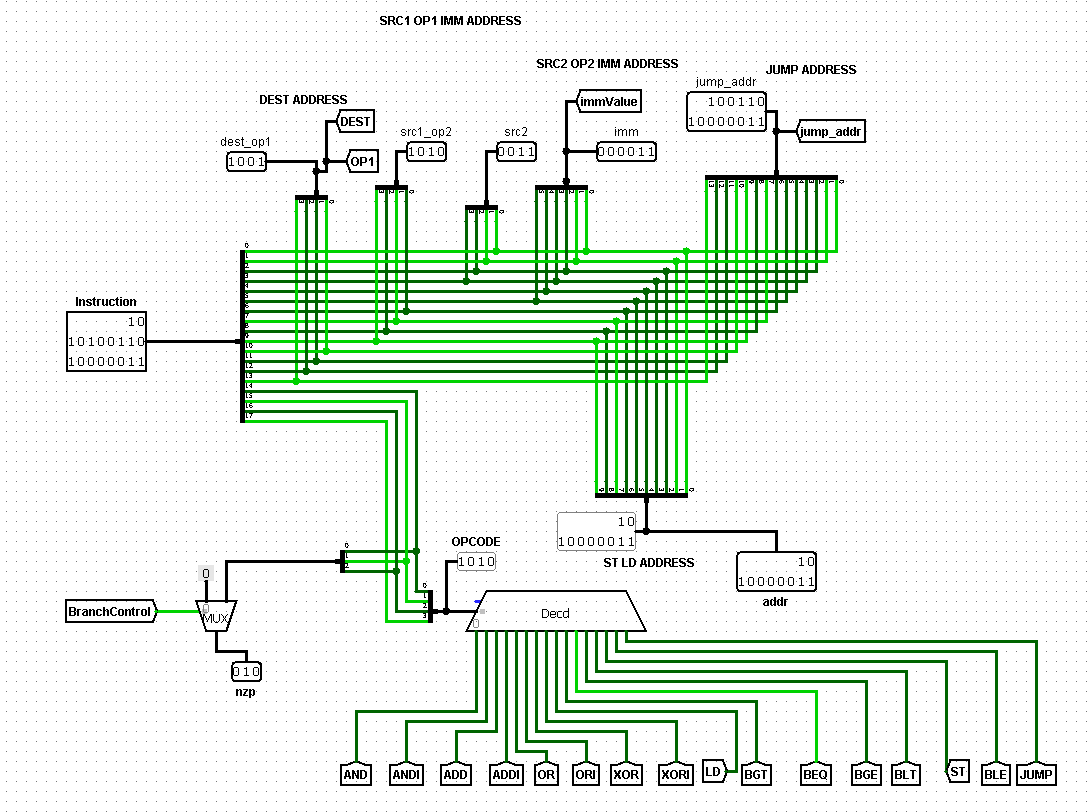
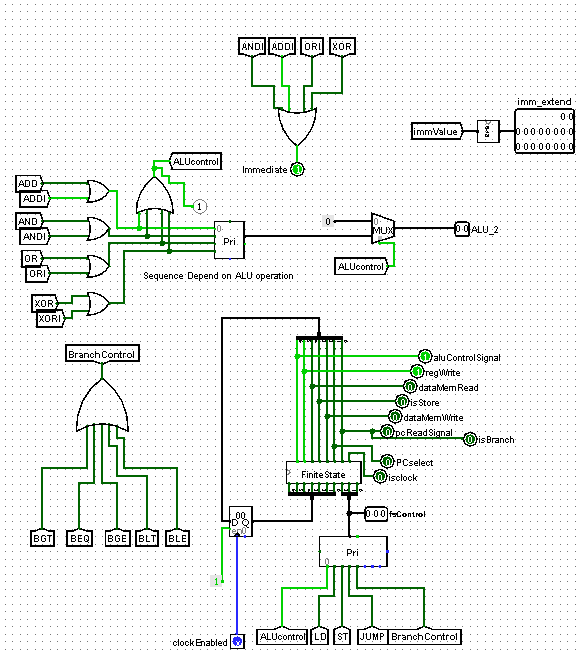
If there is addr operation it can be a jump or branch operation so it has to be determined addr is added to the pc or directly jumps to the new addr value.Also, branch operation if it is not satisfied with the npz then pc is incremented to the next value with incrementer as 1.

****

**3.9 Control Unit**

The Control unit gets current instruction binary sequence and seperated it into on sequenctial parts.Circuit include some small controllers and signals .These signals are using on processor part to perform instraction operation.For each instruction we have conrol signal .

Firstly circuit gets current instruction opcode, nzp ,dest,src,imm values , addr values for appropriate instruction.Then gives value by using an decoder with select bit as opcode for current instruction tunned depend on the ISA file opcode values.

** **

*Firgure Control Unit : Instruction Code Splitter Part*

*Firgure Control Unit : Usage Of Finite State Machine and getting Immediate ,Alu signals*

Immediate Signal gets value when we have one of ANDI,ADDI,ORI,XOR operations.We use this signal to select src2 or imm value as one of opertion register value.

ALU\_2 Signal get encoded 2 bit value for represented current Alu operation type.If operation is not an ALU operation then gets 00 to remove third state .

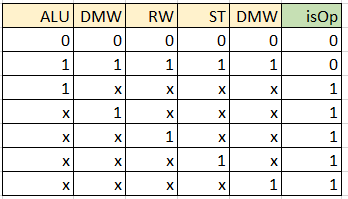
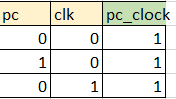
Imm\_Extend Signal, 6 bit immValue is extended to 18 bit using 6-18 signed bit extender to use Alu operations .

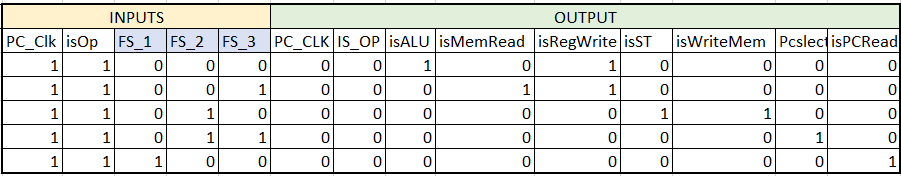
Circiut assign value to some of tunnels for current operation form type due to all have same operation form on the same type .BranchControl gets value if current operation are one of BGT,BEQ,BGE,BLT,BLE . ALUcontrol get value if operation are one of ADD,ADDI,AND,ANDI,OR,ORI,XOR,XORI .

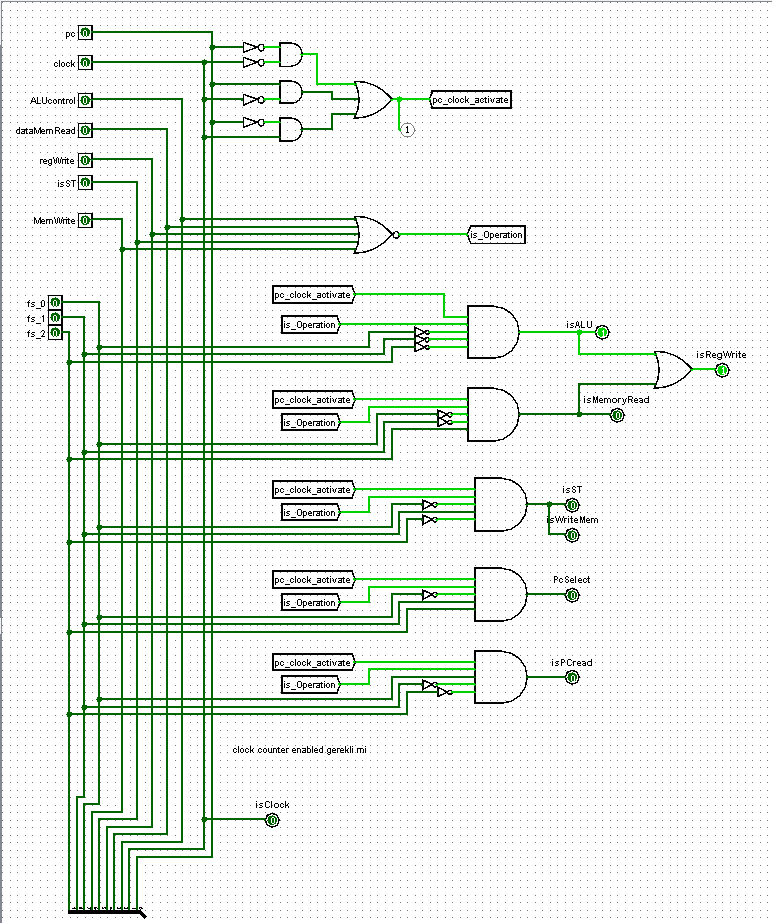
Finite State Machine is seperate circuit from ControlUnit .It gets control tunnel signals for different operation types, then encode this operations as fsControl then gives apropriate signals depend on the operation type .Also use a FSregister to store 10 signal value for each operation

**3.10 FSM**

The finite State Machine controller is separated from control untit to handle signal processing. Signal processing mainly depends on the fsControl bit input sequence encoded depend on the operation type form. Using this fsControl input sequence appropriate signals get values .Below input and output tables are shown. Because of the more understandable, separated tables included.For example if we have an ST operation, fsControl bit sequence 010 isRegWrite anf isST signals gets value 1.

** **

****

****

**3.11 Processor**

On the Processor, all components are connected. There is two memory which is instruction memory is 18-bit data and address size, dataMemory is 10-bit address, and 18-bit data size. On the clock rising edges, instruction binary code gets from Insmem as an input of ControlUnit circuit. From ControlUnit, obtained needed signal and data values to handle operation. Using appropriate signal values small circuit on the left side determines two register addresses as operation register as an input of RegisterFile later. If we have an ALU operation ALU circuit is used to get proper inputs then send the result to the register file again to store on the operation register address after checking operation include immediate value or not by imm signal. PC Controller gets proper inputs to determine the next PC register value then gives to the InstMem gets proper instruction sequence.

For example we have below instruction ,

ADDI R14,R14,1

ALU , RW(register write),IMM signals gets 1 .Operation perform RegisterFile holds value .Then PC circiut increment pc register value 1.



