**MARMARA UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING**

****

**Digital Logic Design**

**CSE 3015 – Term Project Report**

Group members for the project and student numbers:

**150119811 Abdulah Puskar**

**150120056 Halil Olcay**

**150118048 Mustafa Yanar**

**150119639 Erdem Pehlivanlar**

**150118073 Serkan Koç**

In our project we were expected to design and implement a processor which supports following instruction set: (SUB,ADD, AND, OR, XOR, ADDI, SUBI, ANDI, ORI, XORI, LD, ST, JUMP, PUSH, POP, BE, BNE).

Processor has 10 bits address width and 20 bits data width. Processor has 5 parts as it is suggested in the instructions. The 5 parts are as follows:

**Register File** that holds register values and signal to write into any register. There are 16 registers in processor.

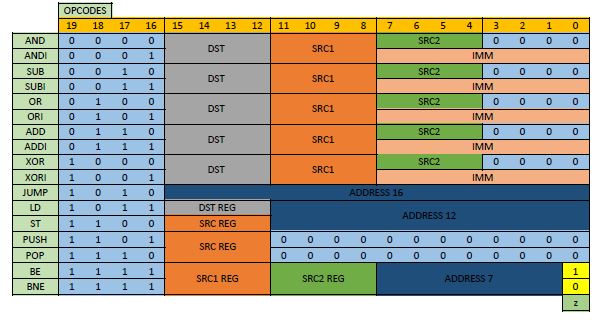
**Instruction Memory** is a read-only memory and its instructions are stored in this component. It has conditions, so for example: If the current instruction is not one of the JUMP, BE, BNE; the next instruction is fetched and executed consecutively from this memory.

**Data Memory** is a read-write memory which will store data. Program is able to read data from data memory, and also store data to this memory. Data Memory also has 10 bits address width, and 20 bits data width.

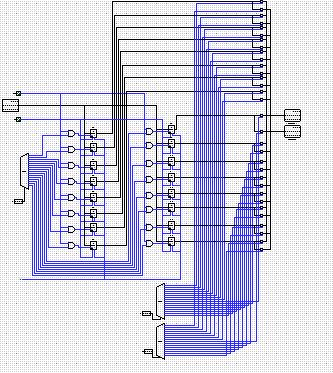
**Control Unit** will produce proper signals to all data path components. For example if the instruction is ST, control unit should produce memWrite signal which will allow Data Memory component to write data value on its data input to the address on its address input.

**Arithmetic Logic Unit (ALU)** computes arithmetic operations ADD, SUB, AND, OR, XOR, ADDI, SUBI, ANDI, ORI, XORI. Operands are fetched from register + register or register+ immediate value. Result will be stored to the Register File. Control unit should produce proper signals to ALU according to instruction operation code.

Here you can see our ISA table designed. We have used 4 bits for opcode for 17 instructions. We have used 8 bits for immediate values, 12 bits for address values in LD and ST, 16 bits address for JUMP, and 7 bits address for branch instructions. Assembler input is a code sequence of assembly language, and it will convert given mnemonics to the binary codes.

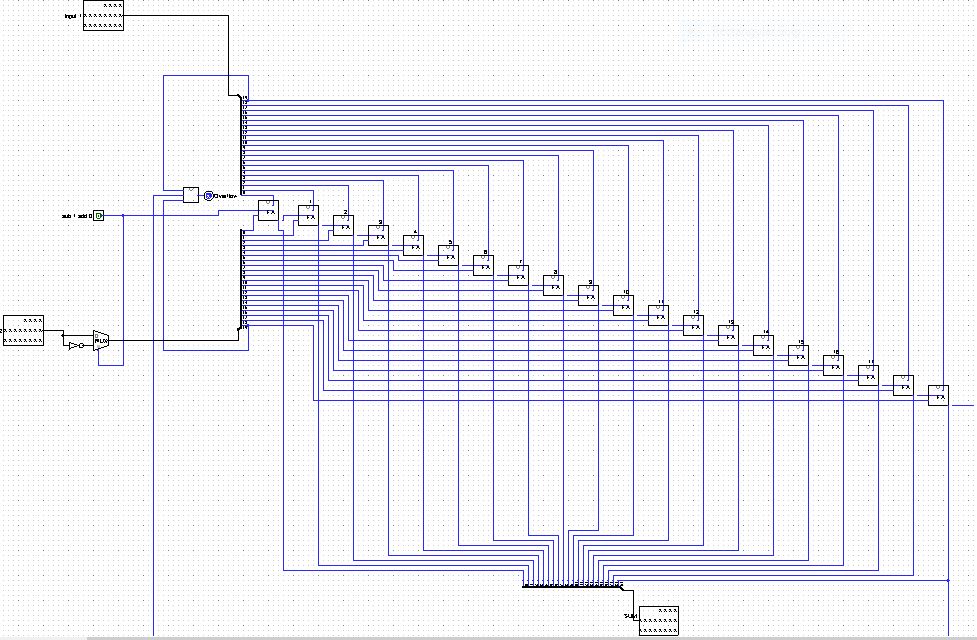
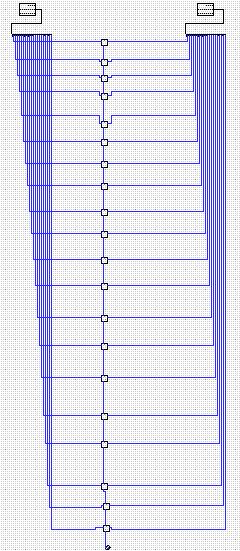


Register File

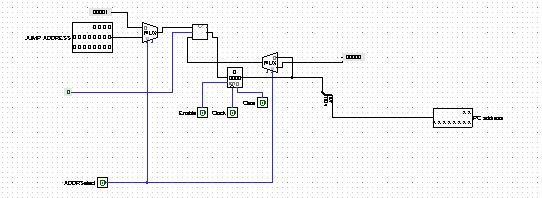
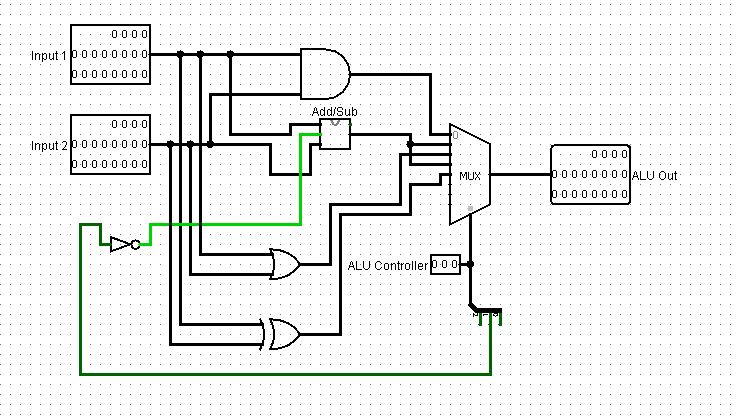


Adder and Comparator

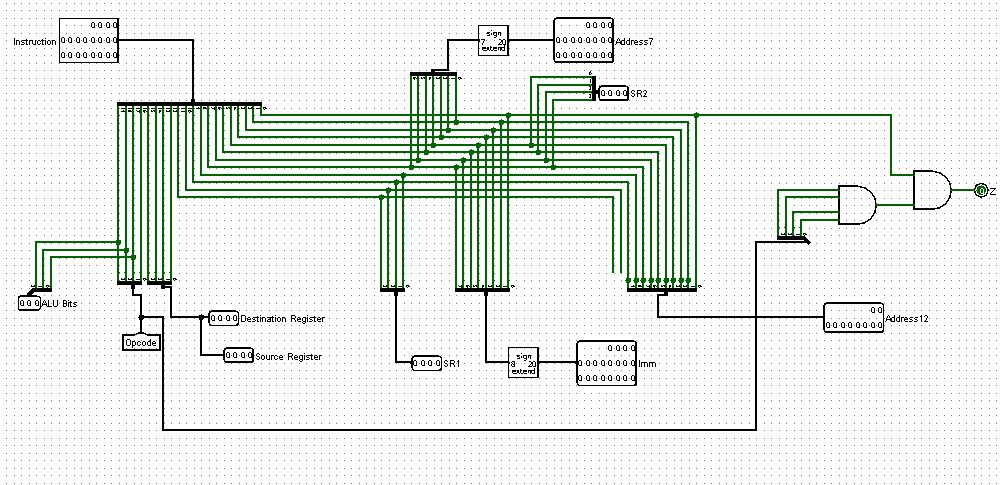
We have designed 20 bits Adder and Comparator. Also, we have designed half and full adder and comparator. You can find overflow detector in our design as well.



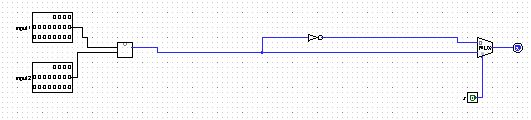
ALU Program Counter

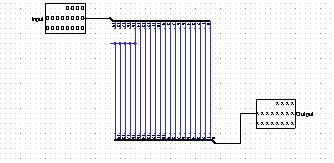


Instruction Parser

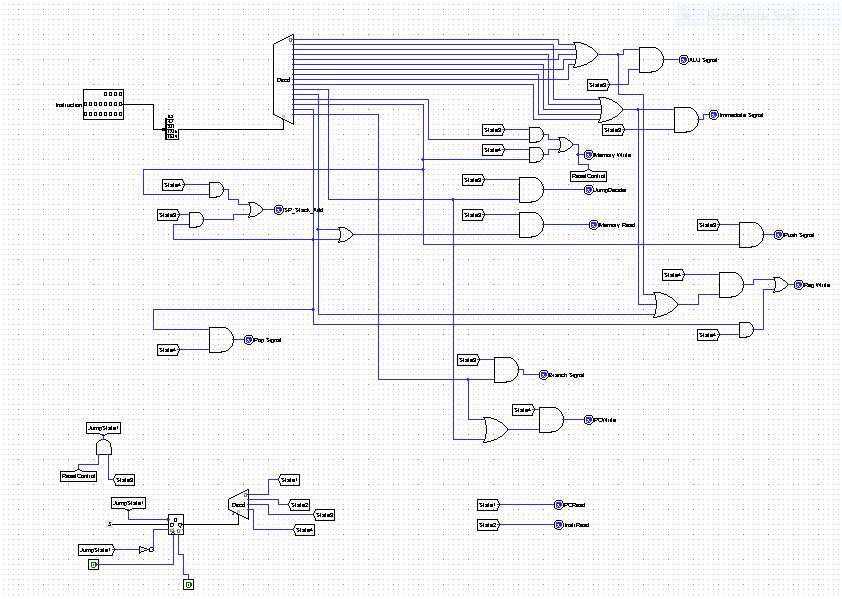


Sign extended value and Branch signal



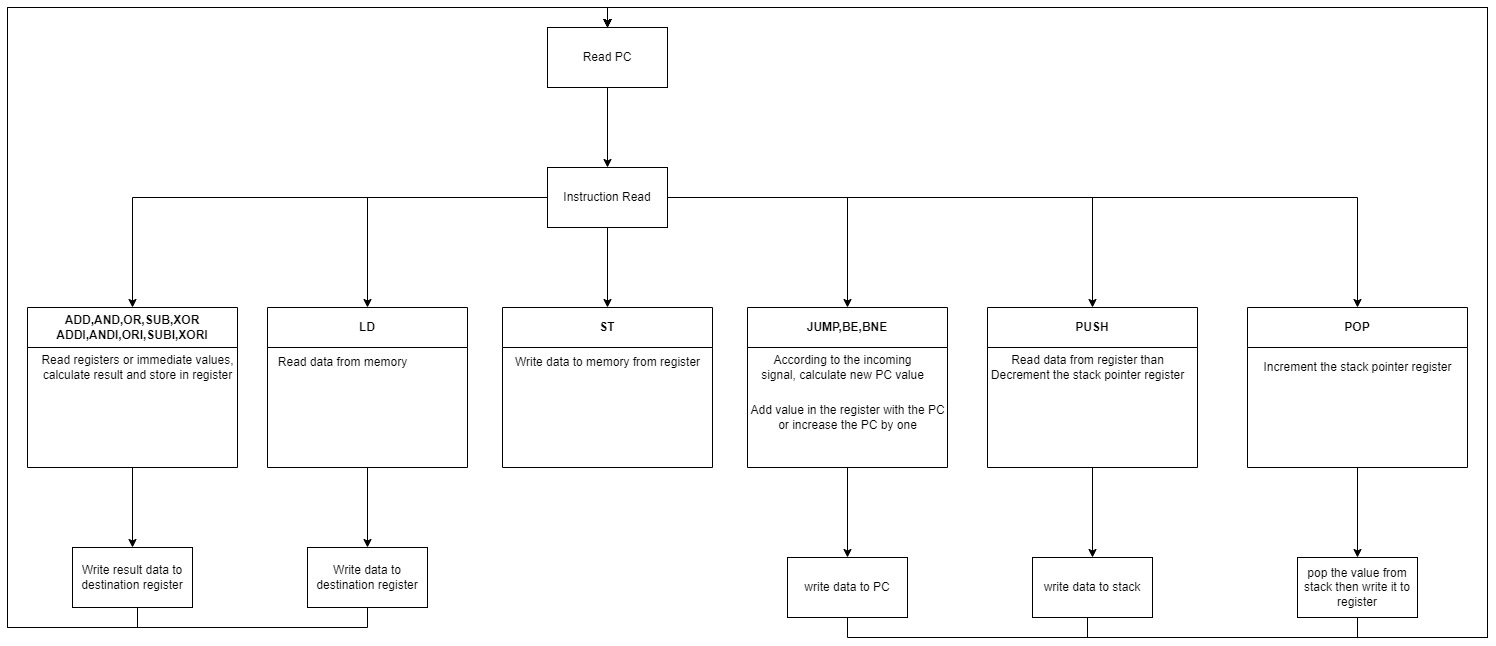


Control Unit

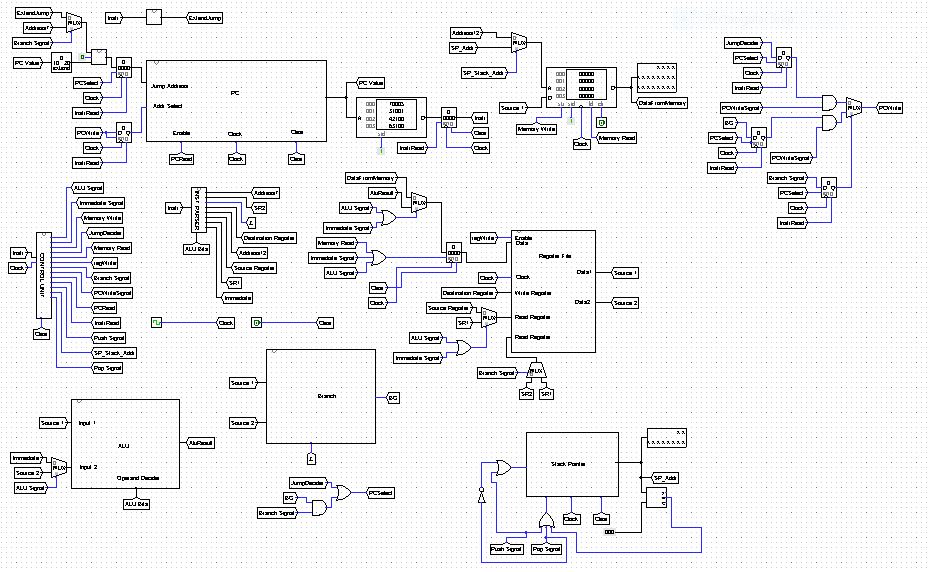


Finite State Machine:

We implemented the FSM in order to control states of the Control Unit, in the picture below you can see it more clearly, it is consisted of 4 states which enable handling easier



CPU



Signals:

PCRead: Enable program counter (State 1)

InstrRead: Enable instruction register to read. (State 2)

Memory Write: Enable memory to write data (State 3)

Memory Read: Enable memory to read data from memory. (State 3)

ALU Signal: Enable register which connected with Register File and store result in this register. (State 3)

Immediate Signal: Enable register which connected with Register File and store result in this register. (State 3)

These signals connected to mux for specify which data is used.

Jump Decider: Enable PCSelect signal (State 3)

Branch Signal: Enable PCSelect signal if branch operation returns 1 and it specifies which data is used in JumpAddress (State 3)

PCSelect: Select data which is connected by Program Counter (State 3)

RegWrite: Enable register file to write data to register. (State 4)

PCWrite: If conditions are true PCWrite enable and PC adds the value from jump address or branch instruction address. (State 4)

SPStackAddr: This signal arranges the address of the memory currently running on the ram. (Depends on the PUSH or POP State 3 or State 4)

Push Signal: Decrements the stack pointer register value.

Pop Signal: Increments the stack pointer register value.

Operation Flow

First it reads PCValue and find the instruction value(State 1). At second state it reads instruction (State2). Then it starts to make processes for the instructions.

LD: Memory Read is enabled and read data from memory(State3). Then RegWrite signal is enabled and it writes data to destination Register (State 4)

St: Memory Write signal is enabled and write data to memory from given register.

AND,ADD,OR,XOR,SUB: ALU signal is enabled and ALU makes the operations. Alu use operand decider bits for which operation perform. It enables register which connected with Register File and store result in this register (State 3). Then RegWrite signal is enabled and it writes data to destination Register (State 4)

ANDI,ADDI,ORI,XORI,SUBI: Immediate Sginal is enabled and ALU makes the operations. ALU use operand decider bits for which operation perfom. Immediate value has extended by its sign bit. It enables register which connected with Register File and store result in this register(State 3). Then RegWrite is enabled and it writes data to destination Register (State 4).

JUMP: PCSelect signal is enabled and select data to next PC value(State 3). PCWrite signal is enabled and write the new PC value.(State4)

BE,BNE: If branch operations returns true PCSelect signal is enabled and select data to next PC value(State 3). PCWrite signal is enabled and write the new PC Value(State4)

PUSH: First we activate Push Signal (State 3) and then this decrements the stack pointer value by one. Then SPStackAdd works (State 4) and points the value in the stack. Memory write also enables and writes the value to the memory from register file.

POP: Firstly SP Stack Add is enabled and this points the address in the stack(State 3). Then POP Signal is enabled and it increments the sp value by one. Also RegWrite is enabled and writes the popped value to register. (State 4)