MARMARA UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING

Digital Logic Design – CSE3015
Term Project

Name:	Student Number:
Hasan Fatih Başar	***150118015***
Bahadır Alacan	***150118042***
Emir Said Haliloğlu	***150119039***
Boran Kanat	***150119051***

General Info:

General Info: In this project, we are assigned to make an 18-bit CPU that supports instruction set: (AND, OR, ADD, LD, ST, ANDI, ORI, ADDI, XOR, XORI, JUMP, BEQ, BGT, BLT, BGE, BLE). Our CPU has 18-bit address width and 18-bit data width, and it has 16 registers.

Instructions:

ADD: This instruction adds two register, and store result into destination register. ADD, AND, OR, XOR have same form.

ADDI: This instruction adds a register value and immediate value and store the result into destination register. ADDI, ANDI, ORI, XORI have same form.

JUMP: JUMP instruction set the Program Counter to the given value in the instruction and Address in PC relative mode. ADDR is offset, and it can be negative.

LD: LD instruction load a value from Data Memory to any register.

ST: ST instruction store value from a register to Data Memory.

-Branch instruction compare two operands, then will jump to the address according to this comparison.

BEQ: Compare registers OP1 and OP2 if they are equal, PC set to ADDR(PC-relative). Instructions n,z,p binary values will be 0,1,0.

BLT: Compare op1 and op2. If op1 is less than op2 then PC set to ADDR(PC-relative). Instructions n,z,p binary values will be 1,0,0.

BGE: Compare op1 and op2. If op1 is greater than or equal to op2 then PC set to ADDR(PC-relative). Instructions n,z,p binary values will be 0,1,1.

BLE: Compare op1 and op2. If op1 is less than or equal to op2, PC set to ADDR(PC-relative). Instructions n,z,p binary values will be 1,1,0.

BGT: Compare op1 and op2. If op1 is greater than op2, PC set to ADDR(PC-relative). Instructions n,z,p binary values will be 0,0,1.

PART1:

In this part we designed our ISA table. We use 4 bits for opcode for 16 instructions. We use 6 bits for immediate values, 10 bits for address values in LD and ST, 3 bits address for branch instructions. Then we write an assembler with JAVA to produce machine code input to give the processor. Assembler input is a code sequence of assembly language. Assembler will convert given mnemonics to the binary codes.

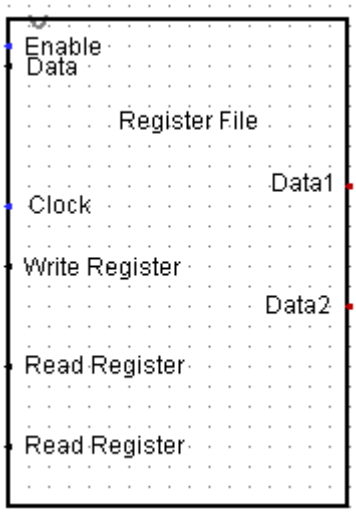
ISA:

	Opcode[17:14]	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AND	0000	DR				SR1				SR2				0	0
ANDI	0001	DR				SR1				imm6					
ADD	0010	DR				SR1				SR2				0	0
ADDI	0011	DR				SR1				imm6					
OR	0100	DR				SR1				SR2				0	0
ORI	0101	DR				SR1				imm6					
XOR	0110	DR				SR1				SR2				0	0
XORI	0111	DR				SR1				imm6					
LD	1000	DR				ADDRES10									
ST	1001	SR				ADDRES10									
JUMP	1010	ADDRES14													
BEQ	1011	SR1				SR2				ADDR			0	1	0
BLT	1100	SR1				SR2				ADDR			1	0	0
BGE	1101	SR1				SR2				ADDR			0	1	1
BLE	1110	SR1				SR2				ADDR			1	1	0
BGT	1111	SR1				SR2				ADDR			0	0	1

PART2:

We have designed our components.

Register File:



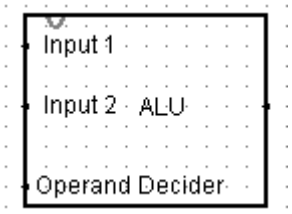
We have 16 registers in register file. 18 bits data stored in each register. All the registers are general purpose in that they may be freely used by any of the instructions that can write to the register file. Register File has 6 inputs.
- **Data** has 18 bits data width and stored into register.
- **Write Register** is 4-bit input to select register which to be loaded. It connected to decoder for specify which register is loaded.
- **Read Registers** are 4-bit input to select which register to be read. It connected to decoder for specify which register is read.
-Register Write signal is connected to **Enable** input, which is used in LD, AND, ANDI etc. instructions.

Adder:
We design adder for add two input which has 18-bit data width. Adder use half adder and full adder. We have also designed them.

Comparator:
We design comparator for compare two input which has 18-bit data width. Comparator use half comparator and full comparator. We have also designed them.

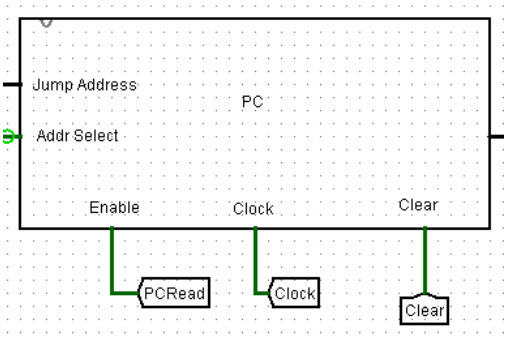
ALU:

ALU is used for arithmetic operations such as AND, ADD, OR, XOR, ANDI, ADDI, ORI, XORI. ALU has 3 inputs.
- **Operand decider** is 2 bits and used for specifying which operation is used. It connects to multiplexer. Opcode's first and second bit is used in operand decider.
-**Input** has 18-bit data width. It takes data from register file, or it can be immediate value.



PC:

Program Counter's duty is keeping track of the instructions. It keeps track with two methods: Incrementing by default or jumping according to instruction. Program Counter will jump to an address if the instruction has a jump command or with branch instructions. Otherwise, it will increment the address by one. It has 5 inputs.



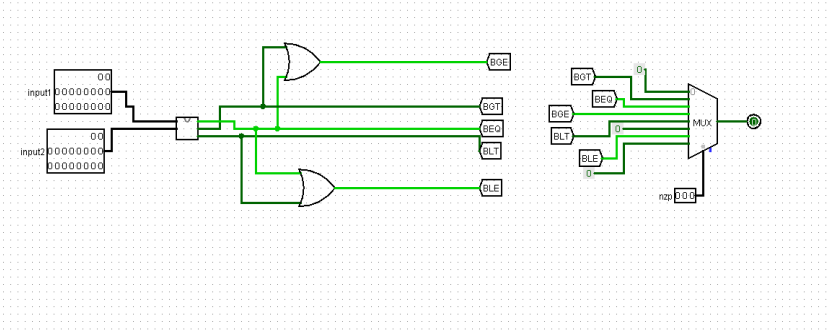
-**Jump Address** is used for JUMP and Branch operations it jumps to given address.

-**Addr Select** is used for specifying which value is used for next PC-value. If it is 0 it increments PC by 1, otherwise it uses Jump Address.

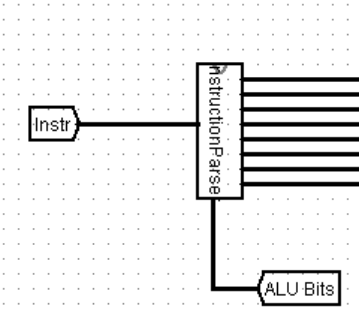
Branch Operator:

Our branch operator comprehends which branch operation we should use from the instruction by nzp values. If the operation's conditions are met it sends a signal that allows pc to jump. It has 3 inputs:

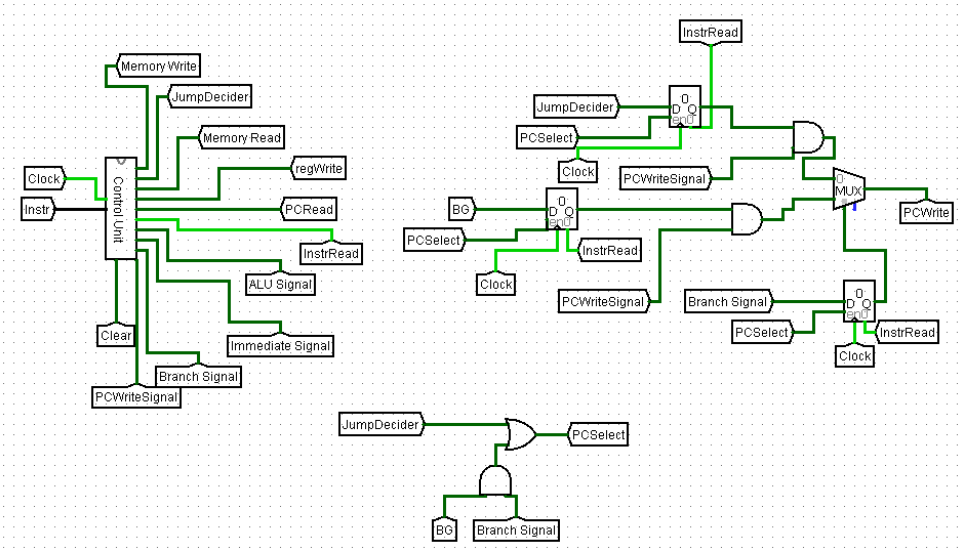
Source 1, Source 2, and nzp value.



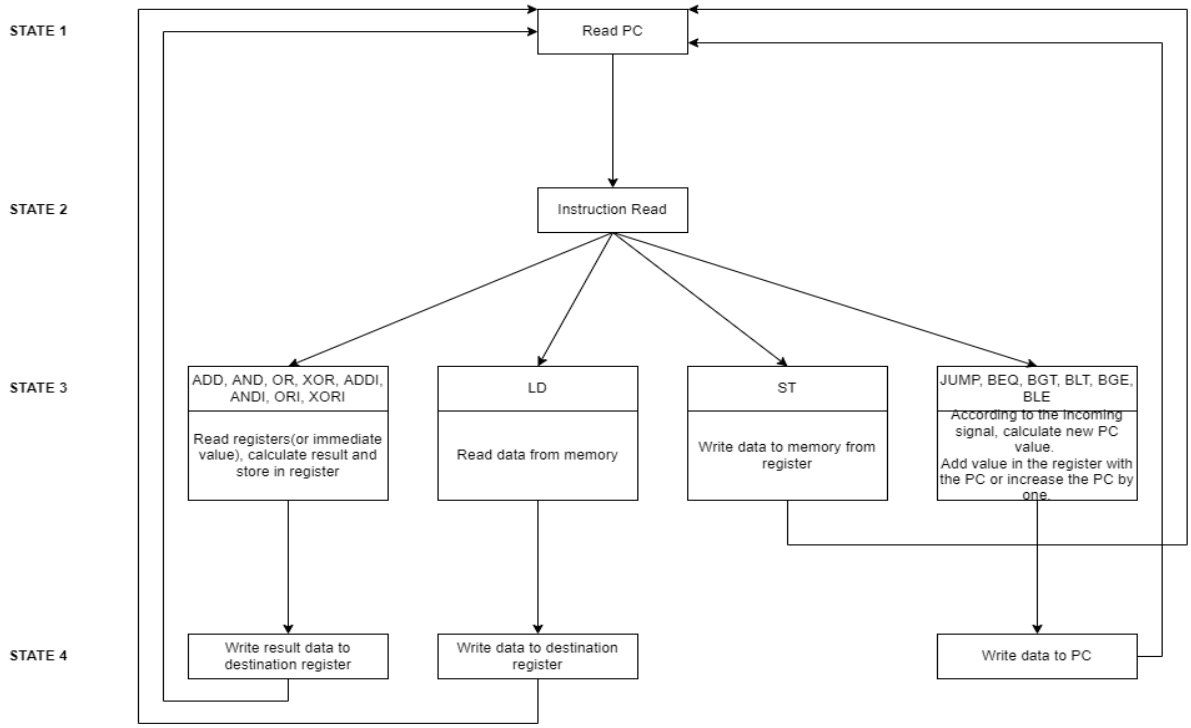
Instruction Parser: It parses instruction for take the requiered parts of instructions.



Control Unit:
Control Unit decode opcodes and generates signals for data paths.



Finite State Machine:
We implemented an FSM to control states of the control unit.



SIGNALS:

PCRead: Enable program counter. (State1)
InstrRead: Enable instruction register to read. (State2)
Memory Write: Enable memory to write data. (State3)
Memory Read: Enable memory to read data from memory. (State3)
ALU Signal: Enable register which connected with Register File and store result in this register. (State3)

Immediate Signal: Enable register which connected with Register File and store result in this register. (State3)
These signals connected to mux for specify which data is used.
JumpDecider: Enable **PCSelect** signal. (State3)
BranchSignal: Enable **PCSelect** signal if branch operation returns 1. And it specifies which data is used in JumpAddress. (State3)
PCSelect: Select data which is connected by Program Counter. (State3)
RegWrite: Enable register file to write data to register. (State4)
PCWrite: If conditions are true PCWrite enable, and PC adds the value from jump address or branch instruction address. (State 4)

MAIN

First it reads PCValue and find the instruction value (State1). At second state it reads instruction (State2).
Then it started to make processes for the instructions.

LD: Memory Read signal is enabled and read data from memory (State3). Then **RegWrite** signal is enabled, and it writes data to destination Register (State4).

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

AND,ADD,OR,XOR: 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 (State3). Then **RegWrite** signal is enabled, and it writes data to destination Register (State4).

ANDI,ADDI,ORI,XORI: Immediate signal is enabled and ALU makes the operations. ALU use operand decider bits for which operation perform. Immediate value has extended by its sign bit. It enables register which connected with Register File and store result in this register (State3). Then **RegWrite** signal is enabled, and it writes data to destination Register (State4).

Immediate Signal and Alu Signal connected to mux to specify which data load to register file.

JUMP: PCSelect signal is enabled and select data to next PC-value (State3). **PCWrite** signal is enabled and write the new PC-Value (State4).

BEQ,BGT,BLT,BLE,BGE: If branch operations returns true (branch signal component's output is 1) **PCSelect** signal is enabled and select data to next PC-value (State3). **PCWrite** signal is enabled and write the new PC-Value (State4).