CSE 315 - DIGITAL LOGIC DESIGN
TERM PROJECT

-

REPORT

GÖKSEL TOKUR - 150116049

ERTUĞRUL SADIÇ - 150116061

MUSTAFA FURKAN ARAS - 150116019

MARMARA
UNIVERSITY

Project

We designed a basic CPU which has the 18 bit data size, 16 registers and instructions AND, ANDI, ADD, ADDI, OR, ORI, XOR, XORI, LD, ST, JUMP, BEQ, BGT, BLT, BGE, BLE.

- ADD instruction will add two register, and store result into another register.Structure of instruction is:

ADD DST,SRC1,SRC2 where SRC1 and SRC2 are source registers, and DST is destination register for the operation.

- ADDI instruction will add a register value and immediate value, and store the result into another register.Form of instruction is:

ADDI DST,SRC1,IMM where SRC1 is a register, DST is destination register and IMM is immediate value. IMM size will be max available size on your processors design.

- JUMP instruction will set the Program Counter(PC will hold current instruction's address) to the given value in the instruction.

JUMP ADDR where ADDR will be in PC relative mode. ADDR will be offset and it can be negative.

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

LD DST,ADDR where DST is a register to load and ADDR is a address in max available bit size. Upper bits of ADDR will be zero extended.

- ST instruction will store value from a register to Data Memory.

ST SRC,ADDR where SRC is a register to fetch data and ADDR is a Data Memory address to store content of the register. Upper bits of ADDR will be zero extended.

- Branch instruction will compare two operands, then will jump to the address according to this comparison. Your instruction set architecture must reserve 3 bit for branch instructions named as n,z,p which are negative,zero and positive.

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

BLT OP1,OP2,ADDR will compare registers OP1 and OP2, if OP1 is less than OP2, PC will be set to ADDR(PC-relative). Instructions n,z,p binary values will be 1,0,0.

BGT OP1,OP2,ADDR will compare registers OP1 and OP2, if OP1 is greater than OP2, PC will be set to ADDR(PC-relative). Instructions n,z,p binary values will be 0,0,1.

BLE OP1,OP2,ADDR will compare registers OP1 and OP2, if OP1 is less than or equal to OP2, PC will be set to ADDR(PC-relative). Instructions n,z,p binary values will be 1,1,0.

BGE OP1,OP2,ADDR will compare registers OP1 and OP2, if OP1 is greater than or equal to OP2, PC will be set to ADDR(PC-relative). Instructions n,z,p binary values will be 0,1,1.

Instruction Set Architecture and Assembler

We first design an instruction set architecture and adjust data bits and address bits for each instruction (AND, ANDI, ADD, ADDI, OR, ORI, XOR, XORI, LD, ST, JUMP, BEQ, BGT, BLT, BGE, BLE).

Then we wrote 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.

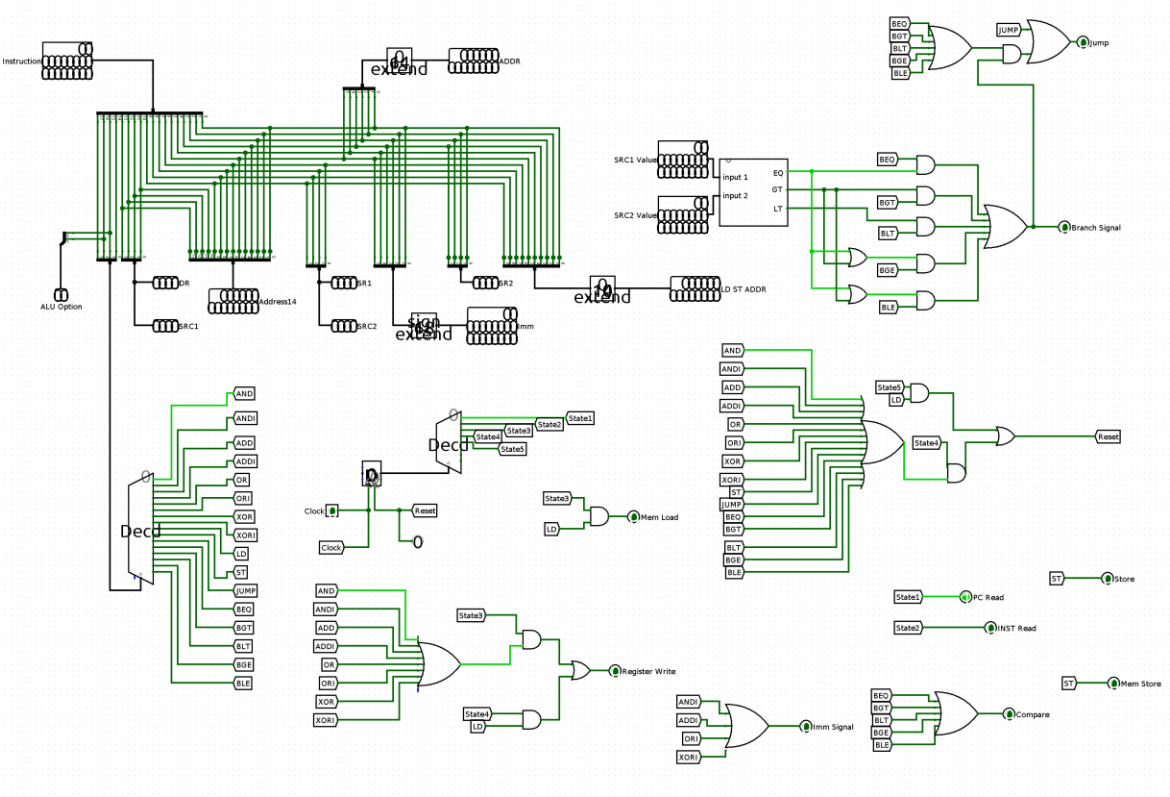
Instruction Set Architecture:

	Opcode[17:14]	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AND	0000	DR				SR1				0	0	SR2			
ANDI	0001	DR				SR1				Imm6 (two's complement)					
ADD	0010	DR				SR1				0	0	SR2			
ADDI	0011	DR				SR1				Imm6 (two's complement)					
OR	0100	DR				SR1				0	0	SR2			
ORI	0101	DR				SR1				Imm6 (two's complement)					
XOR	0110	DR				SR1				0	0	SR2			
XORI	0111	DR				SR1				Imm6 (two's complement)					
LD	1000	DR				Address10									
ST	1001	SR				Address10									
JUMP	1010	Address14													
BEQ	1011	SR1				SR2				ADDR			n	z	p
BGT	1100	SR1				SR2				ADDR			n	z	p
BLT	1101	SR1				SR2				ADDR			n	z	p
BGE	1110	SR1				SR2				ADDR			n	z	p
BLE	1111	SR1				SR2				ADDR			n	z	p

We have 16 instructions, 4 bit allocation is enough to indicate each instruction. Imm value will be represented as two's complement and we allocated 6 bits for it.

Logisim

1) Control Unit

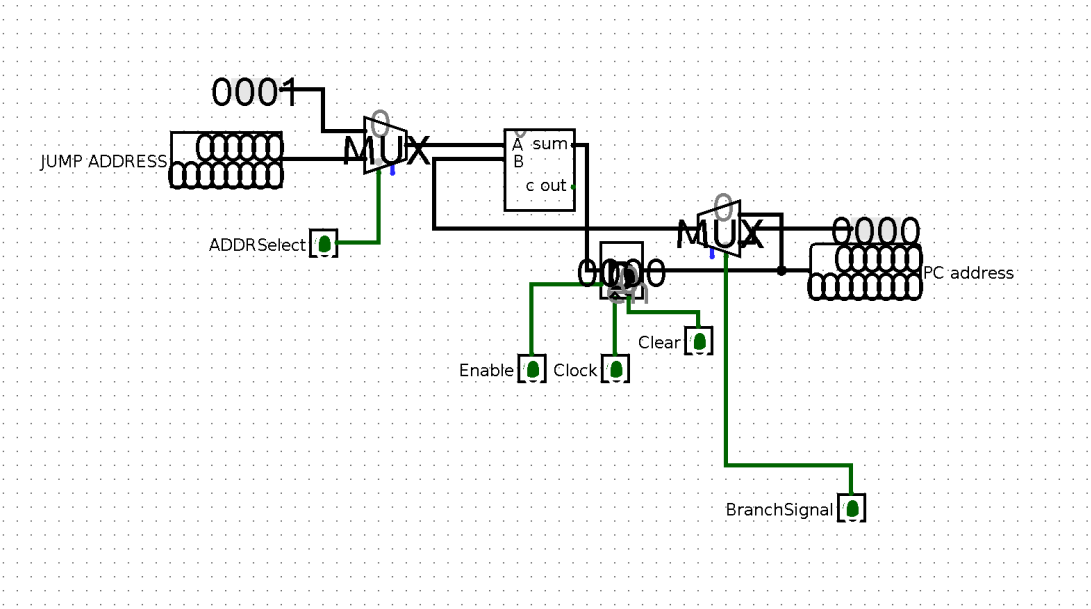


Control unit is the unit that we handle all the signals that other components need. Instruction is sperated bit by bit compatible with instruction set architecture. Finite state machine is developed by a counter. For every clock cycle, counter's value increase and states change.

- For state 1 we generate PC Read signal to enable PC Register.
- For state 2 we generate INST Read signal to enable instruction register.
- For state 3 there are different types of signals that is generated for each instruction set.
- For AND, ANDI, ADD, ADDI, OR, ORI, XOR, XORI Register Write signal is generated.
- Also, for ANDI, ADDI, ORI, XORI operations imm signal and is generated.
- For LD instruction Mem Load signal generated.
- For ST Mem Store signal generated.
- For BEQ, BGT, BLT, BGE, BLE Compare signal is generated.
- Compare signal takes values from register and it compares the values. From those values branch signal is generated. Also for JUMP, instruction, jump signal is generated.

- For AND, ANDI, ADD, ADDI, OR, ORI, XOR, XORI, ST, JUMP, BEQ, BGT, BLT, BGE on state 4 resets the counter. For LD, on state 5 resets the counter.

2) PC Register



Program Counter Register is designed to store PC and to handle operations like increase and JUMP to the destination address.

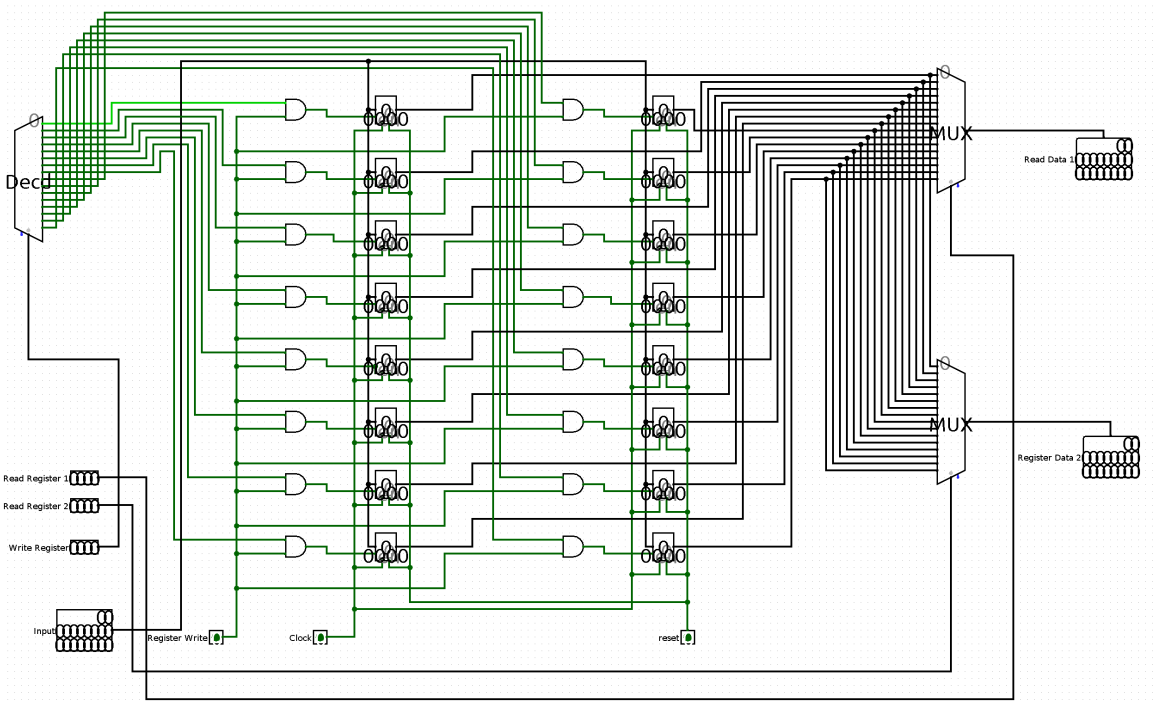
If ADDR select signal is 1, JUMP ADDRESS will be added to the PC value, if it is 0 PC will be increased by 1.

If BranchSignal is 1, PC address will be set to JUMP ADDRESS that comes from instruction address space. If BranchSignal is 0, current PC address will be sent to the adder.

Clear signal is to clear PC register's stored value.

Enable signal is to activate PC register.

3) Register File

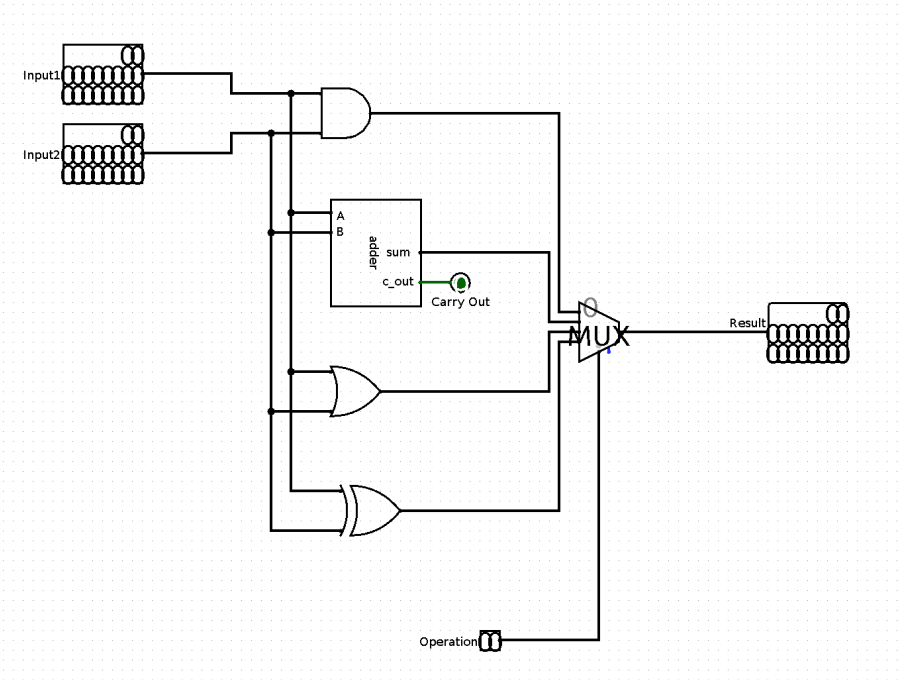


The register file contains sixteen registers, referred to by number as R0 through R15. 18 bit data are stored in each register. All of the registers are general purpose in that they may be freely used by any of the instructions that can write to the register file. Decoder is used to select register which the data will be stored and multiplexers to read values inside the registers.

Register Write signal to enable registers, if it is 1. Data can be written to the destination register. If it is 0 register file just read the data bits of the register.

If clear signal is enabled all registers' data bits will be cleared.

4) 18-Bit Arithmetic Logic Unit



Arithmetic logic unit handles arithmetic operations, in our case ADD, ADDI, AND, ANDI, OR, ORI, XOR and XORI.

ALU has 2 inputs input 1 and input 2 which is directed by the values read from Register File or immediate part of the instruction. Operation signal decides the operation to be done in ALU.

If Operation signal is 00 operation is AND,

If Operation signal is 01 operation is ADD,

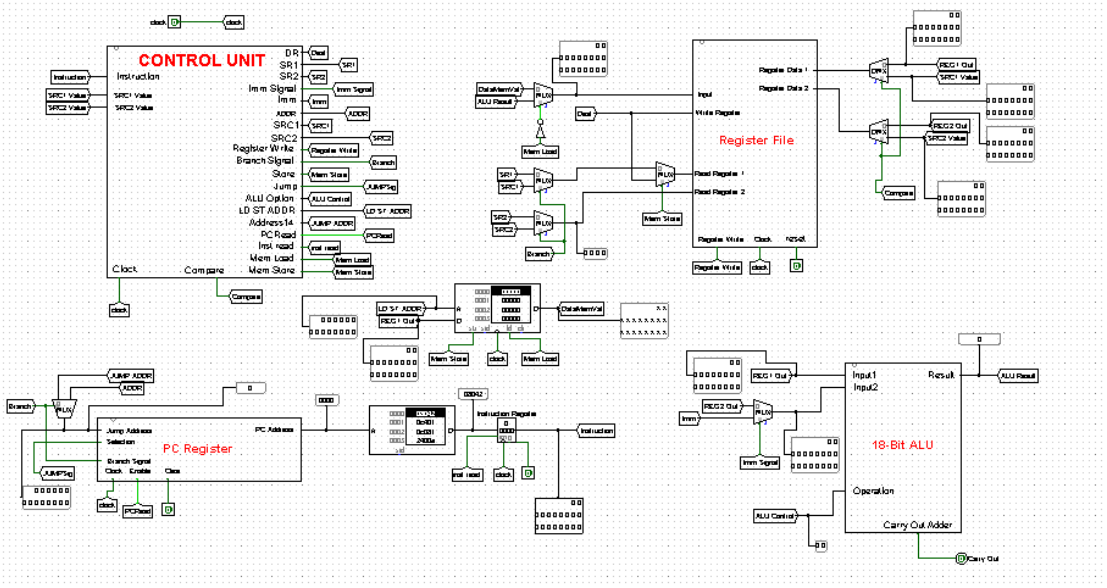
If Operation signal is 10 operation is OR,

If Operation signal is 11 operation is XOR.

5) Adder

We used our own adder that performs addition of numbers in ALU. It consists of full adders and a half adder. The adder which adds two inputs and produces two outputs. Inputs are A and B. The output carry is designated as C-OUT and the normal output is designated as S which is SUM.

6) CPU

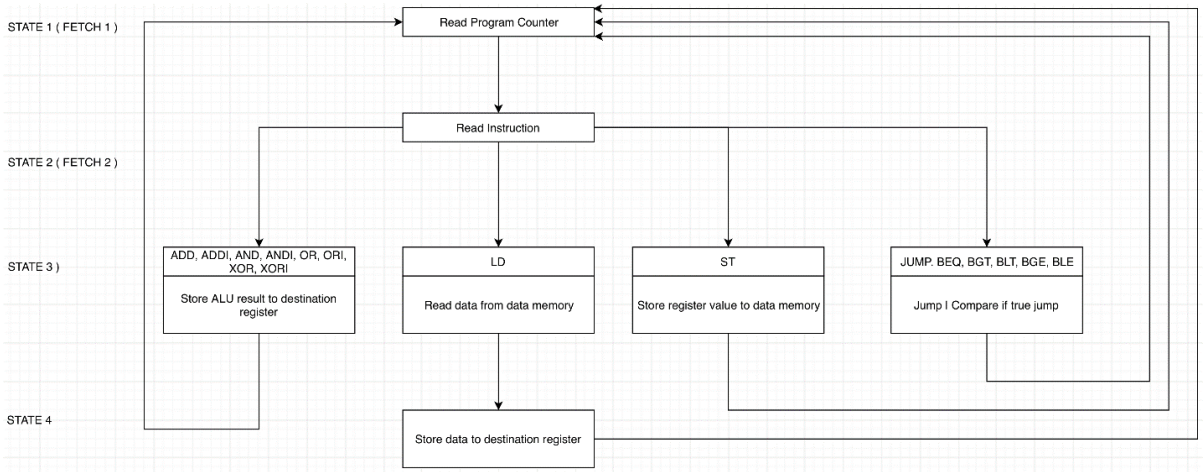


All signals that come from the control unit are sent to the correct locations in the CPU with tunnels and CPU consist of 6 main components. These are Control Unit, Register File, PC Register, 18-bit Arithmetic Logic Unit, Instruction Memory and Data Memory.

Besides above, some necessary logic operations are performed on the CPU.

Finite State Machine

Finite state machine of our processor is defined below;



LD instruction consists of 4 states,

1. Read PC value
2. Fetch instruction from Instruction Memory with address given in PC.
3. Read Data Memory with given address in instruction.
4. Write content from Data Memory to destination register.

All of other instructions consist of 3 different steps,

1. Read PC value
2. Read Instruction
3. Process

Verilog

Verilog part of project consist of 5 part; Arithmetic Logic Unit (ALU), Register File, Program Counter Register, Control Unit and CPU.

Arithmetic logic unit handles arithmetic operations, in our case ADD, ADDI, AND, ANDI, OR, ORI, XOR and XORI.

The register file contains sixteen registers and 18 bit data are stored in each register.

Program Counter Register is designed to store PC and to handle operations like increase and JUMP to the destination address.

Control unit parse incoming instruction and setting signals according to this instruction.

We could not complete this part entirely, we just code and compiled main structures of components.