MARMARA UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

DIGITAL LOGIC TERM PROJECT

REPORT

Number	First Name	Last Name
150119566	Müslim	Yılmaz
150119037	Ömer	Kibar
150119036	Serkan	Korkut

Brief Description:

We will design and implement a processor with several well-known instructions. When we are dealing with design and implementation, we will use our custom set of instruction set. First, we will decide on the instruction set. In the second part, we will visualize and design our processor using the Logisim software. After that we will add Control unit in existing Datapath. Lastly, we will design our processor on Verilog.

Each relevant phase will be discussed in detail below:

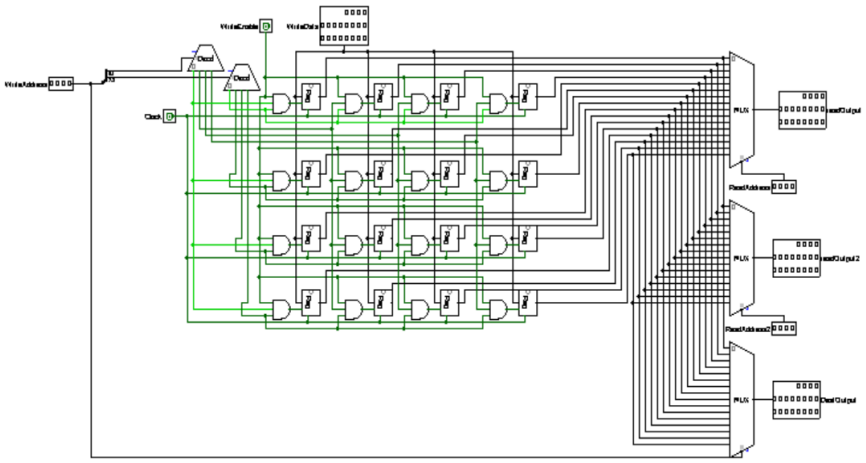
STEP 1 – ASSEMBLY PART

- In our processor we have several restrictions:
    - We will use 20 bits data width and 10 bits address.
    - We will implement given 17 different instructions.
- We decided how to store the given instructions in binary format in our processor. To distinguish given instructions, we set the first 4-5 (depends on the instruction) as opcodes and set a specific code for each instruction**.**
- We used registers as 4 bits. We tried to allocate bits to immediate values as much as possible.
- After deciding on the instruction set, we converted the given assembly instruction into hexadecimal form with using Python programming language.

INSTRUCTION SET ARCHITECTURE

	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	0	0	DEST				SRC1				SRC2				X	X	X
ADDI	0	0	0	0	1	DEST				SRC1				IMM						
SUB	0	0	0	1	0	DEST				SRC1				SRC2				X	X	X
SUBI	0	0	0	1	1	DEST				SRC1				IMM						
AND	0	0	1	0	0	DEST				SRC1				SRC2				X	X	X
ANDI	0	0	1	0	1	DEST				SRC1				IMM						
OR	0	0	1	1	0	DEST				SRC1				SRC2				X	X	X
ORI	0	0	1	1	1	DEST				SRC1				IMM						
XOR	0	1	0	0	0	DEST				SRC1				SRC2				X	X	X
XORI	0	1	0	0	1	DEST				SRC1				IMM						
LD	0	1	0	1	0	DEST				X	ADDR									
ST	0	1	1	0	0	SRC				X	ADDR									
JUMP	0	1	1	1	ADDR									X	X	X	X	X	X	X
PUSH	1	0	0	0	1	SRC				X	X	X	X	X	X	X	X	X	X	X
POP	1	0	0	1	1	DEST				X	X	X	X	X	X	X	X	X	X	X
BE	1	0	1	0	X	REG1				REG2				ADDR						
BNE	1	0	1	1	X	REG1				REG2				ADDR						

STEP 2 – COMPONENT DESIGN

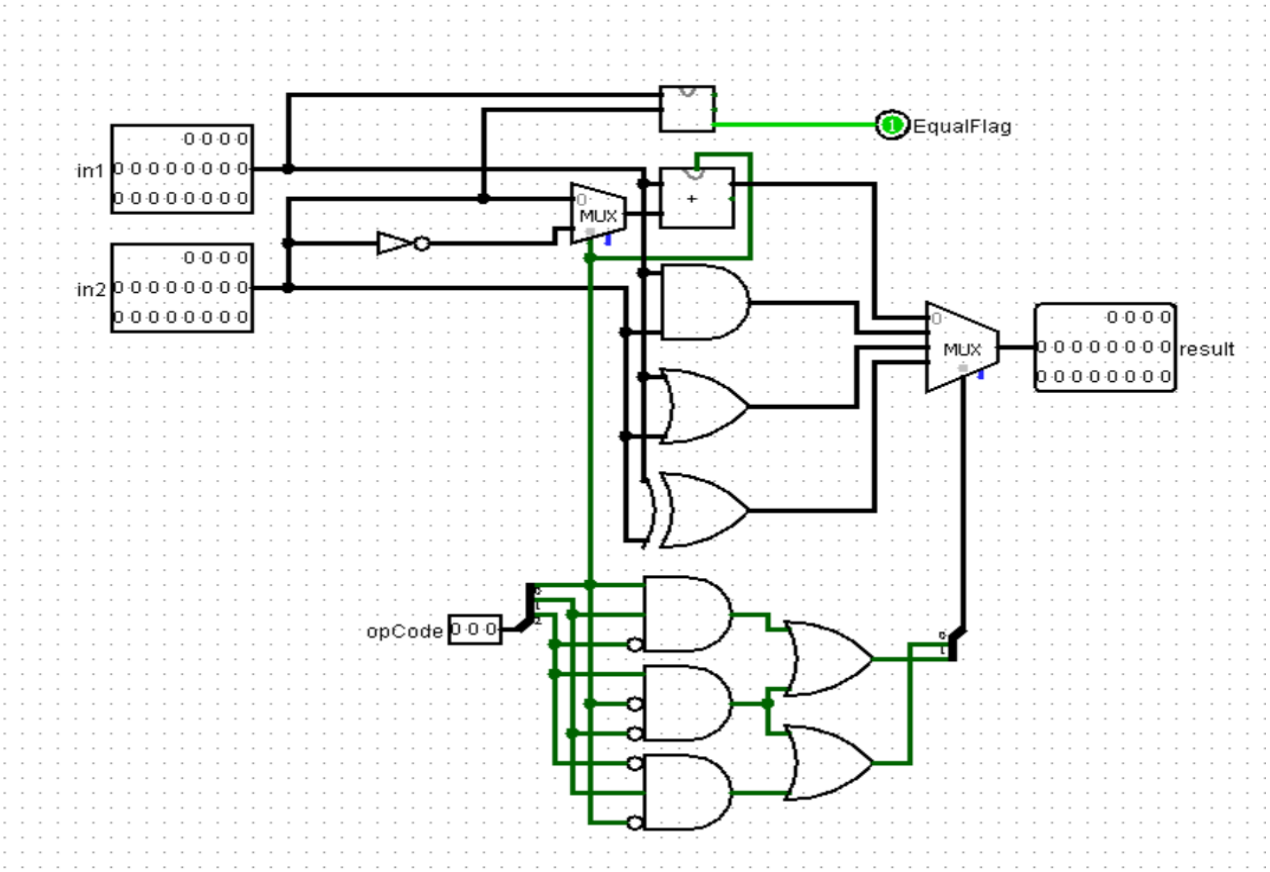


Register File

Register file has 16 registers each holding 20 bit data. It has 3 reading ports in the register file 2 of them are input to ALU and 1 of them is used for ST instruction. It writes writeData input

to the register in the address specified by writeAddress input when writeEnable input is 1.

Arithmetic Logic Unit



Inputs: 3 bit opcode, input1 20 bit, input2 20bit

Outputs: result 20 bit, EqualFlag

When the input values are same equal flag will be 1 this is used for branch instructions. We use last 3 bit of instruction's opcode as an opcode for ALU. Result is selected by opcode at multiplexer.

Other component we build in the logisim;

Full Adder, 4-Bit Adder, 20-Bit Adder, Full Comparator, 4-Bit Comparator, 20-Bit Comparator, 7-20 Sign extend, 10 Bit Adder, 4-Bit Register, 20-bit Register, D-Latch, D Flip-Flop, 10-Bit Register

STEP 3 – CONTROL UNIT

DATAPATH SIGNALS

PCWrite: Connected to load enable bit of program counter register. When active program counter value is updated.

JUMP: Connected to the multiplexer whose output goes to program counter data input. When active multiplexer output will be PC + jump value coming from instruction.

StackWrite: Connected to load enable bit of stack pointer register. When active stack pointer value is updated.

MemWrite: Connected to data memory store bit. When active data input of data memory will be written to the address specified from address input.

MemRead: Connected to load bit of data memory. When active data from the address input will be loaded to the output.

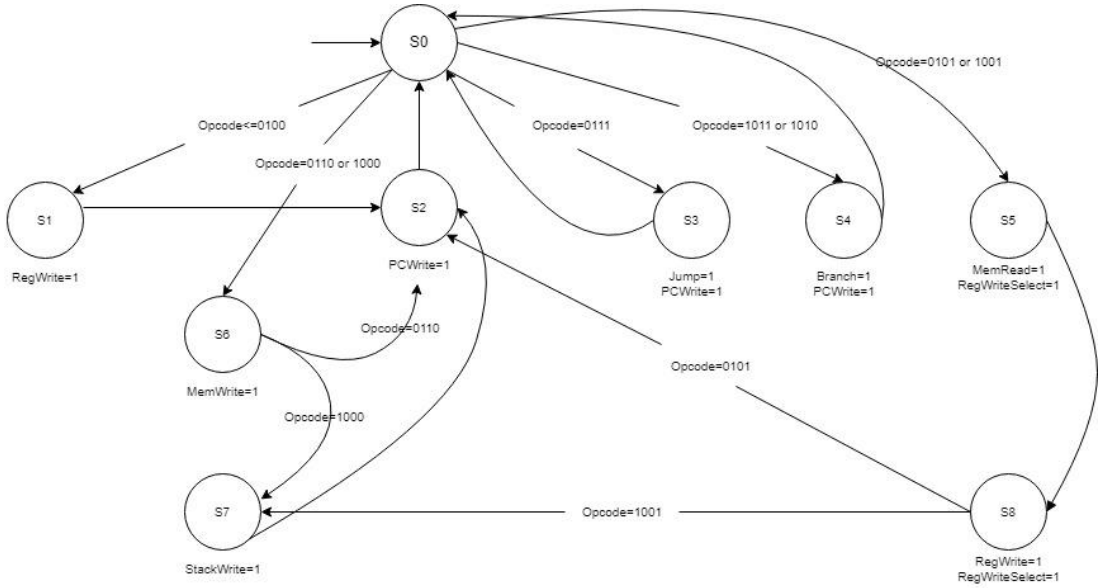
RegWrite: Connect to the load enable bit of register file. When active data input will be written to the destination address.

RegWriteSelect: Connected to multiplexer whose output connected to registerFile writeData input. When active data coming from memory will be selected by multiplexer.

Branch: When instruction is a branch instruction this signal should be generated by control unit according to equal flag coming from ALU multiplexer will select the next value of program counter.

For each instruction control unit needs to generate several signals to complete execution of that instruction and pass to the next instruction.

Diagram of FSM



State 0: Each instruction starts from state 0 to its execution. According to the opcode new state is determined. No signals are generated in this state.

State 1: ADD,ADDI,SUB,SUBI,XOR,XOR,AND,ANDI instructions goes to state 1 only. In this state control unit signals regWrite signal and output of the ALU will be written to the registerFile in the next clock cycle.

State 2: In this state control unit generates signal PCWrite. In the next clock cycle program counter will be incremented by one and control unit goes to state 0 again.

State 3: JUMP instruction will go in to state 3. In this state JUMP and PCWrite signals are generated by control unit. In the next clock cycle program counter value will be updated accordingly.

State 4: Branch instructions will go in to this state. In this state Branch and PCWrite signals are generated by control unit. In the next clock cycle program counter value will be updated accordingly.

State 5: POP and LD instructions will go in to this state. In this state MemRead and RegWriteSelect signals are generated by control unit. In the next clock cycle state will change to S8.

State 6: ST and PUSH instruction will go in to this state. MemWrite signal is generated by control unit in this state. So the data coming from registers will be stored in the memory in the next cycle.

State 7: In the last step of PUSH and POP instructions StackWrite signal should be generated. If it is a push instruction stack pointer will be decremented otherwise incremented in the next clock cycle.

State 8: RegWrite and RegWriteSelect signals are generated in this state. After MemRead signal is generated at the state 5, the data readed from memory will be written to the registers in the next clock cycle.