Bangladesh University of Engineering and Technology

Course No: CSE 306

Computer Architecture Sessional

Experiment 3
4-bit MIPS Design, Simulation, and Implementation

Submitted By:

Lab Group-06

Section: B2

Group Members:

ID: 1805098

ID: 1805103

ID: 1805112

ID: 1805113

ID: 1805115

ID: 1805117

Department of CSE, BUET

Date of Performance: 12th August 2022

Date of Submission: 17th August 2022

1 Introduction

The processor refers to a digital circuit that performs operations on some external data source, usually memory or other data stream. It can do basic arithmetic and logical I/O operations specified by the instructions in a particular computer program. Among many different types of processor design implementation, MIPS (Microprocessor without Interlocked Pipelined Stages) is a Reduced Instruction Set Computer (RISC) Instruction Set Architecture (ISA).

The objective of this assignment is threefold. Firstly, we had to design a simplified 4-bit processor that implements a given reduced version of the MIPS instruction set. Then, we had to simulate the workflow in a preferred software (Logisim). Finally, a hardware implementation is to be done according to the already performed design and software simulation.

The address bus of the designed processor supports 8 bit, while the data bus is of 4 bits. The arithmetic logic unit (ALU) is of 4 bit, hence the processor can support 4 bit MIPS instruction set. Apart from the ALU, other major components of the system include: Register File, Instruction Memory, Data Memory, and Control Unit. 5 temporary 4-bit registers were designed in the Register File module to carry out the internal operations on the given input assembly code. The Control Unit has been micro-programmed, and the control signals associated with the ALU operations were stored in a separate ROM. Instruction Memory and Data Memory modules also use separate ROMs to operate. All these major components were properly connected among themselves using sufficient multiplexers, adder units, basic logic gates and wires.

Each MIPS instructions took 1 clock cycle to be executed, and the clock cycle length was long enough so that the longest instruction on the MIPS instruction set could be executed in a single clock cycle.

2 Instruction Set

The opcodes of the instruction have been set between 0 to 15 based on the sequence of instruction ID provided: AJHIKPLNCDEFGMBO. The MIPS instructions were 16-bit long with 4 formats: R, S, I and J.

Opcode	Instruction ID	Instruction Type	Instruction	Category
0 (0000)	A	Arithmetic	add	R
1 (0001)	J	Logic	srl	S
2 (0010)	H	Logic	ori	I

3 (0011)	I	Logic	sll	S
4 (0100)	K	Logic	nor	R
5 (0101)	P	Control	j	J
6 (0110)	L	Memory	lw	I
7 (0111)	N	Control	beq	I
8 (1000)	C	Arithmetic	sub	R
9 (1001)	D	Arithmetic	subi	I
10 (1010)	E	Logic	and	R
11 (1011)	F	Logic	andi	I
12 (1100)	G	Logic	or	R
13 (1101)	M	Memory	sw	I
14 (1110)	B	Arithmetic	addi	I
15 (1111)	O	Control	bneq	I

2.1 Opcode For ALU

ALU Opcode	Action/Function
000	Addition
001	Subtraction
010	And
011	Or
100	Nor
101	Shift Left
110	Shift Right

2.2 Control Bits

The control bits were arranged in the serail following RegWrite, ALUSrc, MemWrite, 3 bits of ALUOp, MemToReg, MemRead, Bneq, Beq, Jump, RegDst respectively.

Opcode	**Control Bits**
0 (0000)	100000000001
1 (0001)	110110000000
2 (0010)	110011000000
3 (0011)	110101000000
4 (0100)	100100000001
5 (0101)	000000000010
6 (0110)	110000110000
7 (0111)	000001000100
8 (1000)	100001000001
9 (1001)	110001000000
10 (1010)	100010000001
11 (1011)	110010000000
12 (1100)	100011000001
13 (1101)	011000000000
14 (1110)	110000000000
15 (1111)	000001001000

3 Circuit Diagram

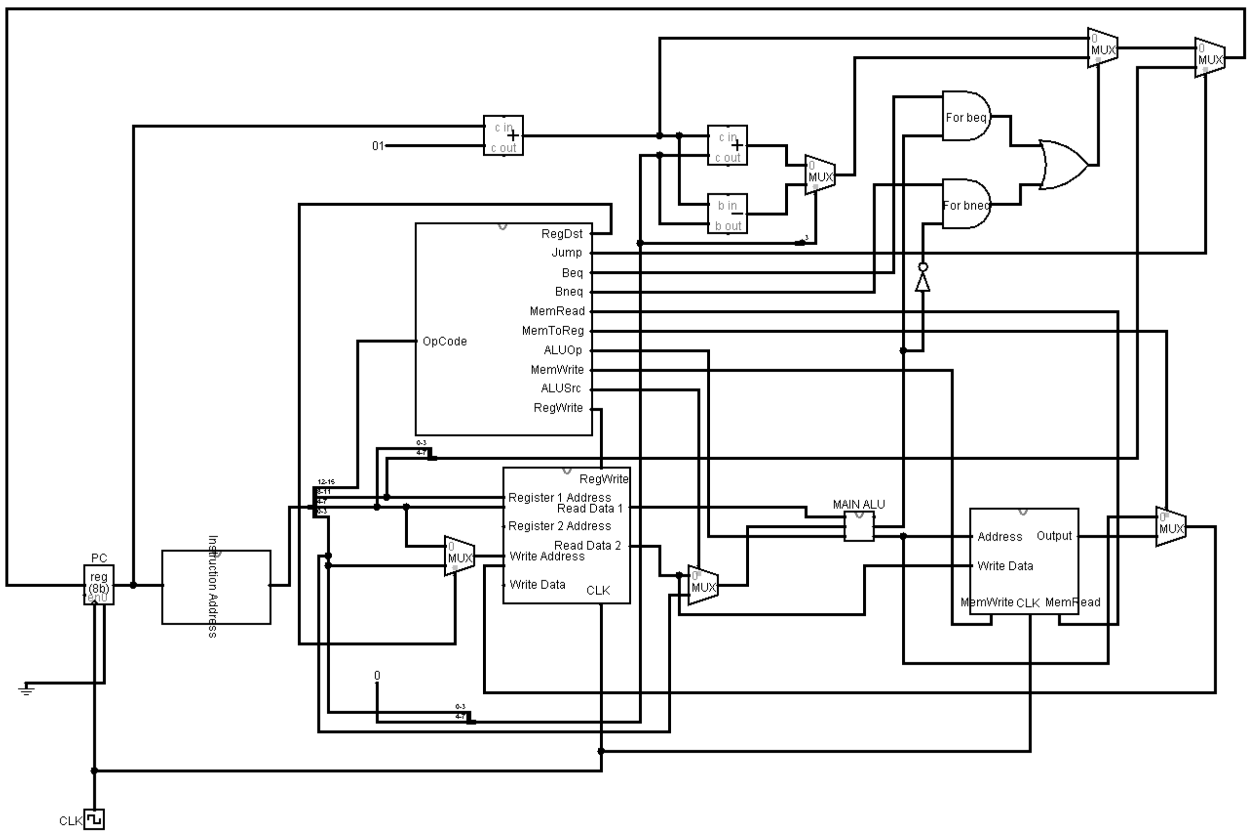


Figure 1: Complete Block Diagram of a 4-bit MIPS Processor

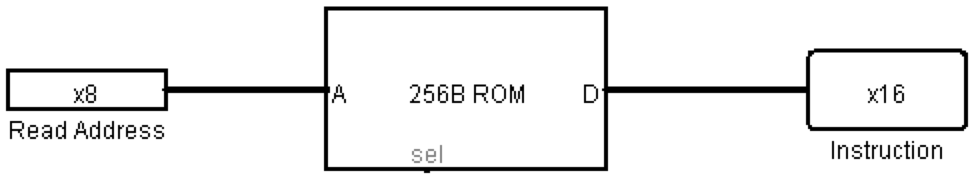


Figure 2: Instruction Memory

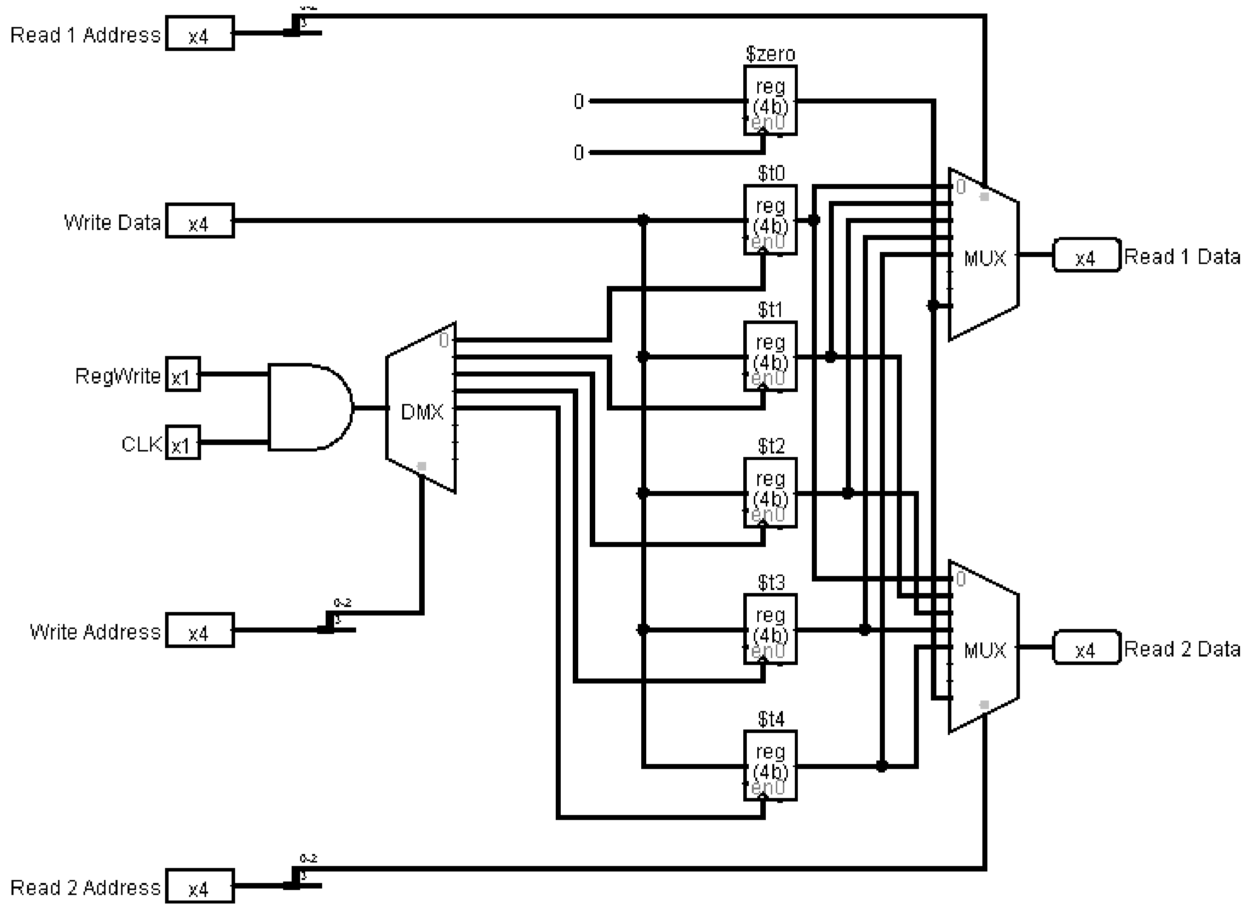


Figure 3: Registers

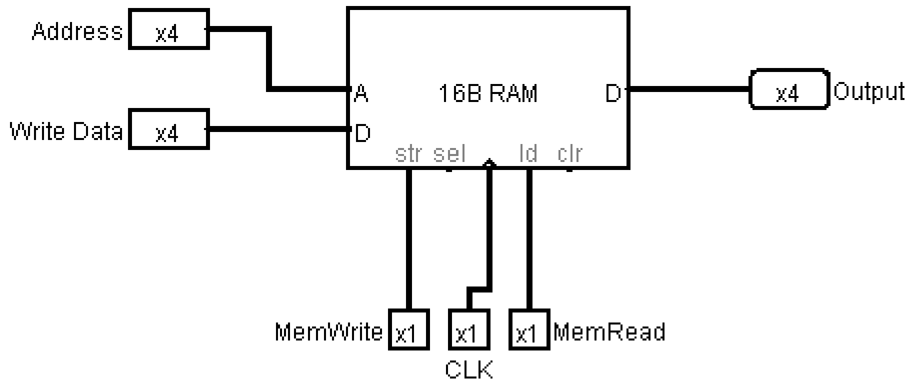


Figure 4: Data Memory

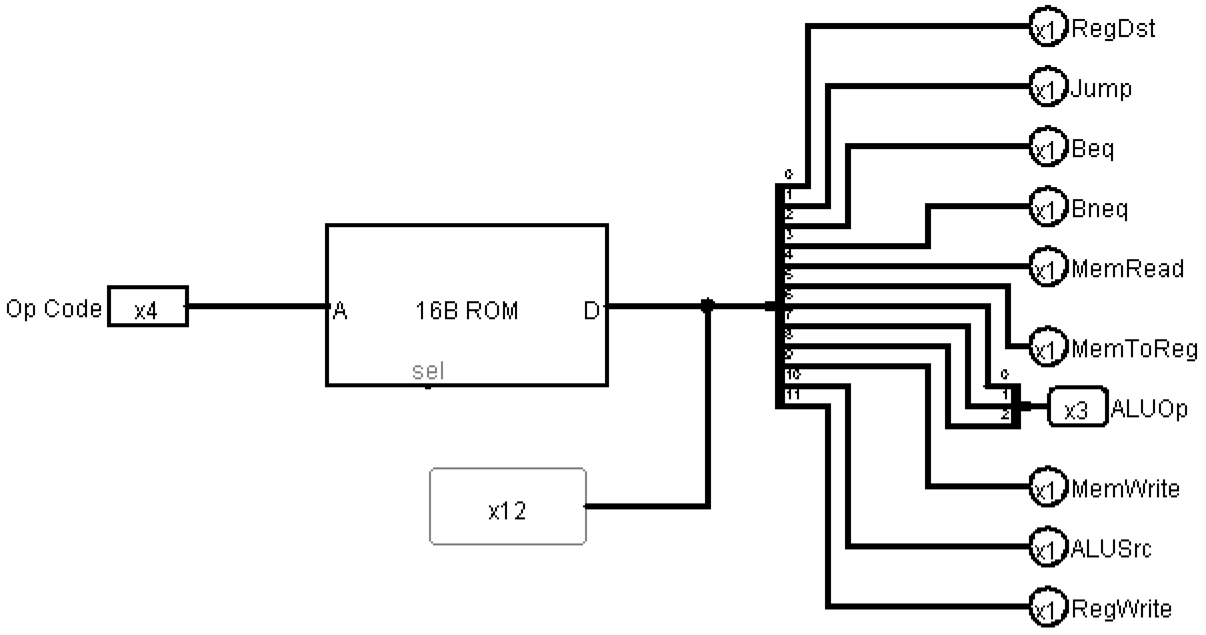


Figure 5: Control Unit

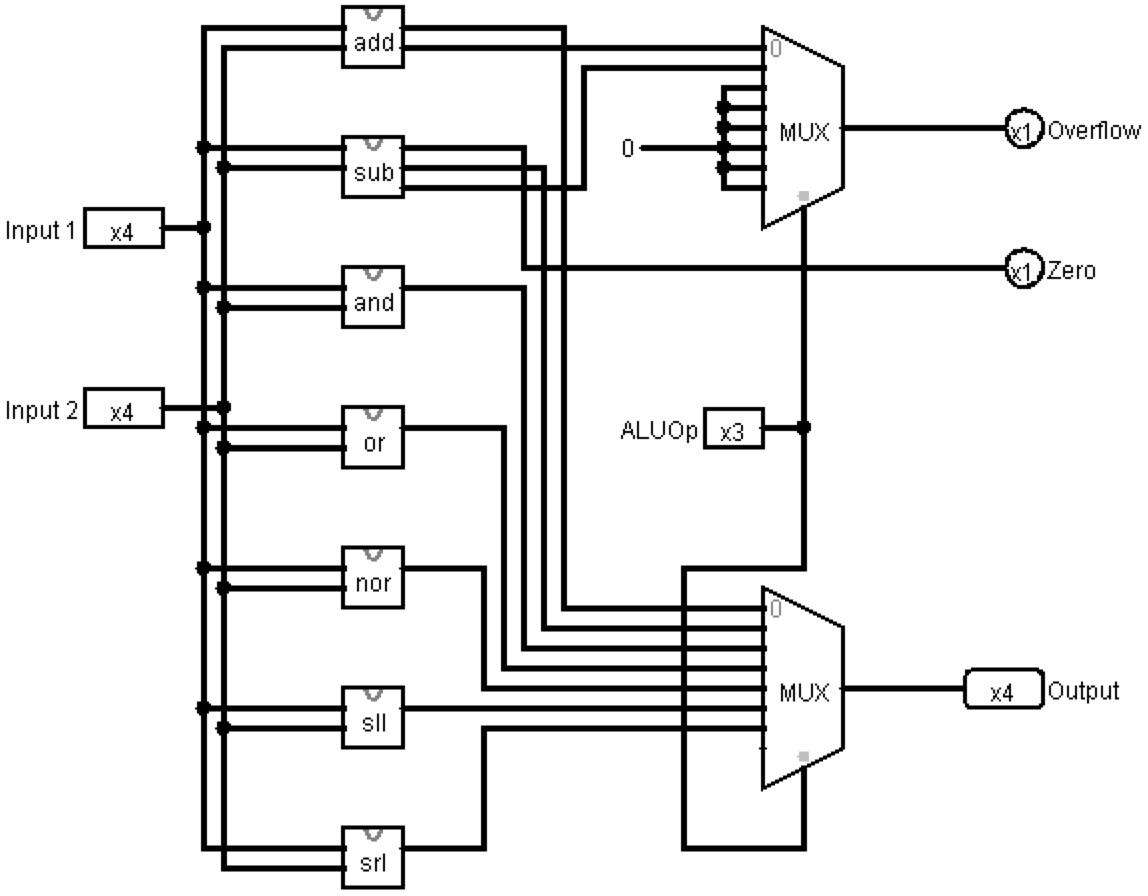


Figure 6: 4-bit ALU

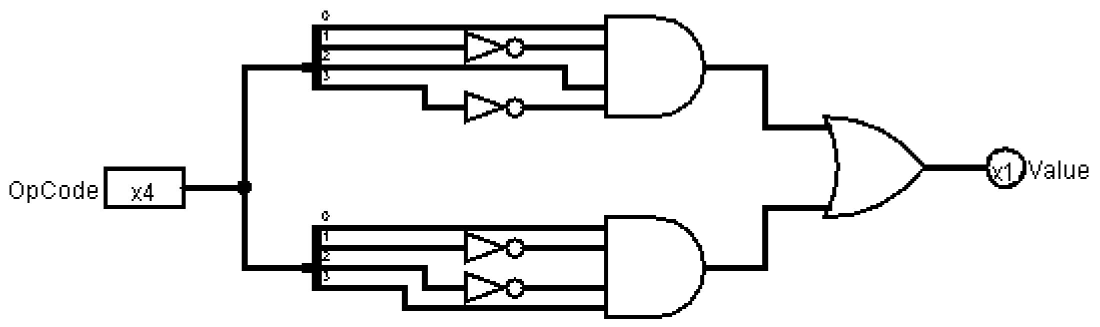
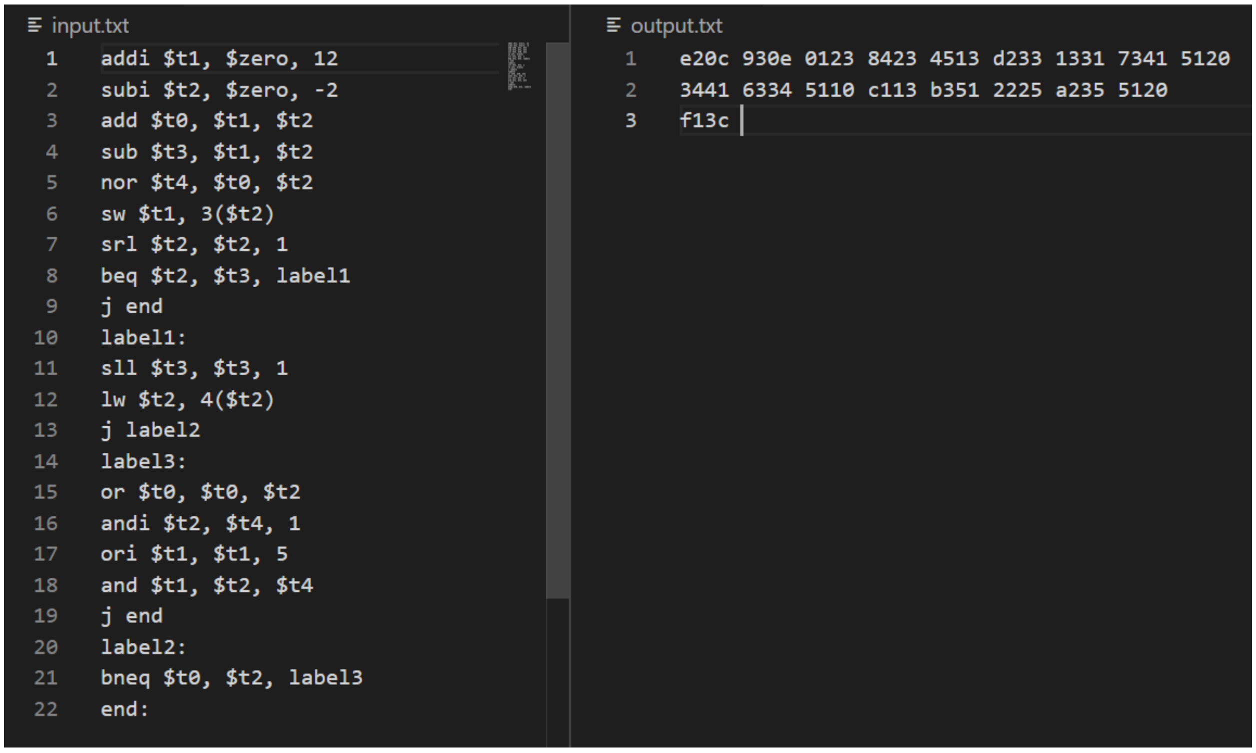


Figure 7: Shift Unit

4. Program Execution Flow

We used a helper program written in Java to convert MIPS instruction set input file into corresponding machine code output file. The hex machine code then was manually loaded into the Insruction Memory segment of the processor. In each clock pulse, the next instruction was executed through the data path and control flow.



```
addi $t1, $zero, 12
subi $t2, $zero, -2
add $t0, $t1, $t2
sub $t3, $t1, $t2
nor $t4, $t0, $t2
sw $t1, 3($t2)
srl $t2, $t2, 1
beq $t2, $t3, label1
j end
label1:
sll $t3, $t3, 1
lw $t2, 4($t2)
j label2
label3:
or $t0, $t0, $t2
andi $t2, $t4, 1
ori $t1, $t1, 5
and $t1, $t2, $t4
j end
label2:
bneq $t0, $t2, label3
end:
```

```
e20c 930e 0123 8423 4513 d233 1331 7341 5120
3441 6334 5110 c113 b351 2225 a235 5120
f13c
```

Figure 8: MIPS to Machine Hex Code Conversion

Register and Data Memory values were updated according to the instruction in each cycle. Instructions were fetched from the Instruction Memory during each falling edge of a clock pulse, and the flow continued through the data path and combinational logic we designed. In contrast, Register and Data Memory values were updated during the rising edge of a clock pulse.

5. IC Count

The list shows the logic gates and modules we used in the software implementation and the probable IC chips we will be using for the hardware part.

Logic Gate/Module	Count (Software)	IC	Count (Hardware)
AND	6	7408	
OR	5	7432	
NOT	5	7404	
XOR	2		
NOR	1		
Adder	6	7483	
Subtractor	1		
Negator	1		
Right Shifter	1		
Left Shifter	1		
Register	7	ATMega32A	
MUX	13	74151, 74157	
DEMUX	1		
ROM	2	ATMega32A	
RAM	1	ATMega32A	

6. Discussion

In this assignment, a 4-bit MIPS processor was designed using Logisim version 2.7.1. The main components of the processor were designed as per the specification provided. We used a ROM as the Instruction Memory. A separate ROM was used for the Control Unit and the control signal bits were mapped against the opcode of the individual instructions. Moreover, a RAM was used to implement the Data Memory, as both data read and write are necessary in this part. We used the built-in ROM and RAM modules available inside Logisim for the software part.

The 4-bit ALU was designed using Multiplexors and basic logic gates available in Logisim. Built-in shifter module was used for implementing shift operations. 6 temporary registers were designed used built-in register module. A separate register was kept for the PC. PC was incremented by 1 in case of normal flow and the address was fed to the instruction memory.

Using minimal number of logic gates and modules was a priority while designing the processor. Although we used built-in modules and logic gates for the software simulation part, we are to use IC counterparts of the gates, and ATMega32A microcontrollers to implement the major components of the processor instead. We will try to minimize those ICs as well.

Suitable labels and captions was provided with the software simulation circuit files. Finally, the circuit was tested several times with sample inputs to make sure it works without any sort of error.