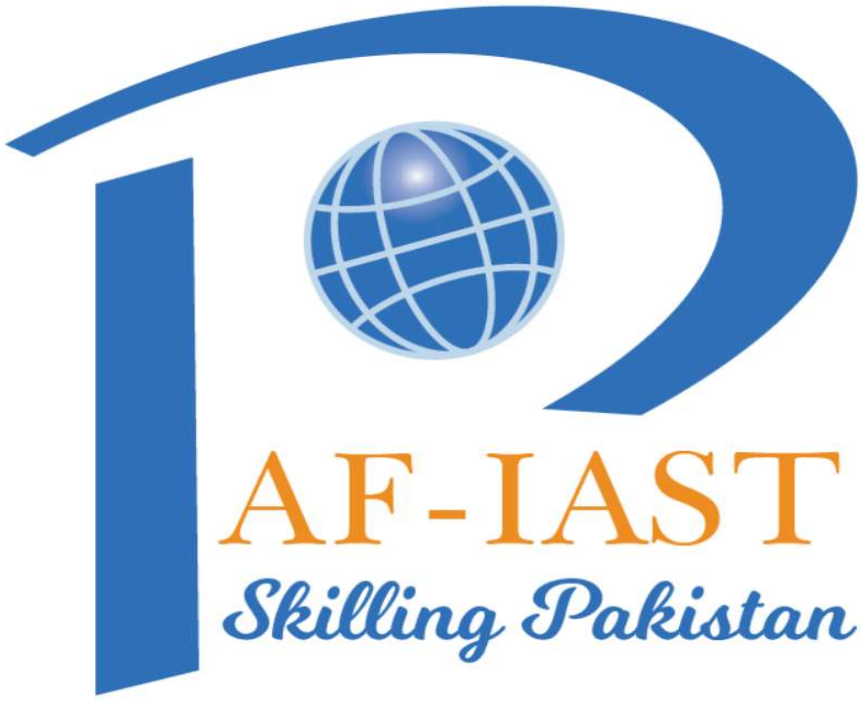
COMP-261 Computer Organization and Assembly Language Fall Semester 2021



Department: Computer Science

Project Title: Simple Microprocessor Design

By:

Dr Hashim Ali

Group Members:

Name	Participation
Saad Bin Khalid B20F0247CS010	Project report, Assembly to binary
M. Waseem Faiz B20F0286CS016	Control unit design, microprocessor
Abid Hussain B20F0437CS027	Microprocessor design Instruction to Assembly

Table of Contents:

1.1INTRODUCTION .....

1.1 OBJECTIVE 1
1.2 DELIVERABLES 1
1.3 SUBMISSION 1

2 DESIGN OF MICROPROCESSOR 3

2.1 INSTRUCTIONS 3
2.2 GENERIC IMPLEMENTATION 3
2.3 FUNCTIONAL UNITS 3

3 BUILDING BLOCKS OF MICROPROCESSOR 4

3.1 PROGRAM COUNTER 4
3.2 MEMORY 4

3.2.1 Instruction Memory 4

3.2.1.1 ROM Specifications 4

3.2.2 Data Memory 5

3.2.2.1 RAM Specifications 5
3.3 Register File 6
3.3.1 Register File Specifications 6
3.4 Arithmetic and Logical Units 6
3.4.1 ALU Specifications 7
4 INSTRUCTION FORMAT 8
4.1 Register to Register (R-Format) Instructions 8
4.2 Immediate (I-Format) Instructions 8
4.2.1 Arithmetic Instructions 8
4.2.2 Memory Reference Instructions 8
4.2.3 Branch Instructions 9
4.3 Jump (J-Format) Instructions 10
5 TYPES OF INSTRUCTIONS 11
6 DIGITAL CIRCUIT DESIGN 12
6.1 ALU Design (4-bit) 12
6.2 Register File (2^4x4) 14
6.3 Instruction Memory (2^4x16) 15
6.4 Data Memory (2^4x4) 15
6.5 Program Counter (4-bit) 16
6.6 Control Unit 17
7 LOGISIM 18

List of Figures:

Figure 1-1. Abstract Design of a Simple Single Cycle RISC Microprocessor 2
Figure 3-1. Block Diagram of Program Counter 4
Figure 3-2. Block Diagram of Instruction Memory 4
Figure 3-3. Block Diagram of Data Memory 5
Figure 3-4. Block Diagram of Register File 6
Figure 3-5. Block Diagram of 4-bit ALU 7

Figure 6-1. Digital Circuits of Half Adder and Full Adder 12

Figure 6-2. Digital Circuit of 1-bit ALU 12

Figure 6-3. Digital Circuit of 4-bit ALU 13

Figure 6-4. Digital Circuit of Register File 14

Figure 6-5. Digital Circuit of Instruction Memory 15

Figure 6-6. Digital Circuit of Data Memory 15

Figure 6-7. Digital Circuit of Program Counter 16

List of Tables:

Table 3-1. Specifications of Instruction Memory (ROM) 4

Table 3-2. Specifications of Data Memory (RAM) 5

Table 3-3. Description of Register File Input/output Pins 6

Table 3-4. Specifications of 4-bit ALU 7

Table 5-1. Types of Instructions (Examples) 11

Table 6-1. Verification of ALU Working 13

Table 6-2. Verification of Register File Working 14

Table 6-3. Verification of Data Memory Working 16

Table 6-4. Verification of Program Counter 16

Table 6-5. Control Signals of 4-bit Microprocessor 17

1 Introduction

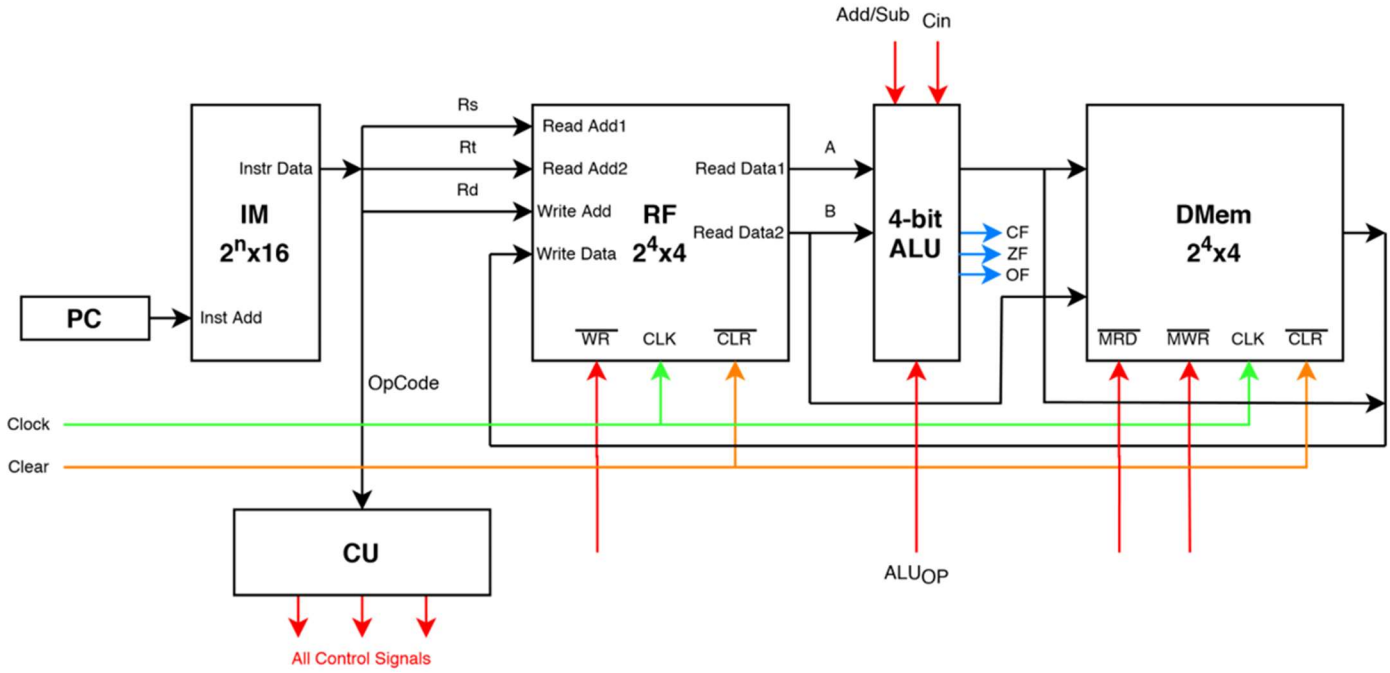
This document gives an introduction to a simple microprocessor architecture based on MIPS microprocessor. The goal of the project is to build a 4-bit processor at logic level and then simulate the processor at layout level. This document introduces the basic concepts of microprocessor architecture in the simplest possible way with a custom-defined instruction set. The design of the processor is very primitive, but already quite complex, as shown in Figure 1-1.

1.1 Objective

The objective and goal of this project is to Reproduce all basic building blocks of the RISC based microprocessor in order to design a complete data-path and control unit. Later to test the working of the processor, it is required to design a simple program to test the working of components as well as the microprocessor itself.

The tasks of the project are:

i. Design of building blocks of microprocessor, which consist of; ALU, Program Memory, Data Memory, Register File, Program Counter.
ii. Design of Instruction Formats, which includes: R-Format, I-Format, J-Format.
iii. Design of Data-path, connecting microprocessor blocks as per the designed instruction formats.
iv. Design of the Control Unit.
v. Construction of test program.



2 Design of Microprocessor

The design of simple processor architecture consists of:

2.1 Instructions

- Memory reference instructions: LW (Load Word), SW (Store Word).
- Arithmetic-Logical instructions: AND, OR, ADD, SUB, SLT (Set Less Than).
- Control flow instructions: BEQ (Branch equal), BNE (Branch not equal), J (Jump).

2.2 Generic Implementation

- Use Program Counter (PC) to supply instruction address.
- Get the instruction from the Memory (also called as, Instruction memory or Program memory).
- Read Registers from Register File.
- Use the instruction to decide exactly what to do.
- All instructions use the ALU after reading the registers.

2.3 Functional units

- Elements that operate on data values (*Combinational*) ○ Example: Instruction Memory (IM), Adder, ALU, Control Unit, Multiplexers etc.
- Elements that contain state (*Sequential*) ○ Examples: Data Memory (DMEM), Register File, Program Counter (PC).

3 Building Blocks of Microprocessor

3.1 Program Counter

A Program Counter (PC) is a 4-bit register and PC will start from 0000 to 1111.

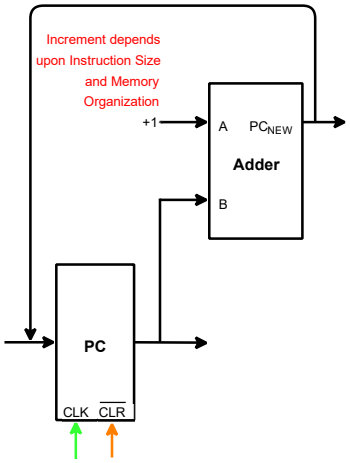


Figure 3-1. Block Diagram of Program Counter

3.2 Memory

There are two types of memories; ROM based Instruction Memory (IM) to store instructions and RAM based Data Memory (DMEM) to hold data.

3.2.1 Instruction Memory

- Instruction memory takes address from PC and supplies instruction data.

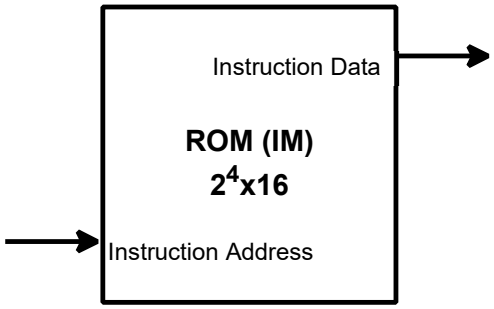


Figure 3-2. Block Diagram of Instruction Memory

3.2.1.1 ROM Specifications

Table 3-1. Specifications of Instruction Memory (ROM)

Type	Description
Size	***2^4x16*** Total memory addresses are 16. Each memory address can hold data of 16 bits (instruction size).
Type	**Description**
Input	!"#$%&'$()" +„%-## requires 4-bit address lines to access each memory address.
Output	!"#$%&'$()" ./$/ is of 16 bits and requires 16-bit data output lines to fetch each instruction.

3.2.2 Data Memory

- Data memory takes address and supply data for LW.
- Data memory takes address and data and write into the memory for SW.

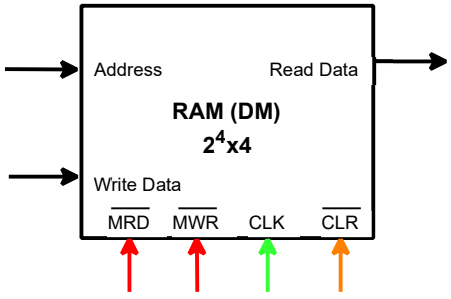


Figure 3-3. Block Diagram of Data Memory

3.2.2.1 RAM Specifications

Table 3-2. Specifications of Data Memory (RAM)

Type	Description

Size	***2^4x4*** Total memory addresses are 16. Each memory address can hold data of 4 bits (data size).
Control Inputs	201.222222 is an active low input; when 0 (reading mode). Data memory contents designated by the +, %-## input is put on the 1-/, . /$/ output. 20312222222 is an active low input; when 0 (writing mode). Data memory contents designated by the +, %-## input is replaced by the value on the 3% ($- . /$/ input. 245122222 is an active low input, when 0 all memory will be cleared.
Input	+, %-## requires 4-bit address lines to access any memory address. 3% ($-. /$/ requires 4-bit data lines to write on memory address specified by *Address* input.
Output	1-/, ./$/ is of 4 bits and requires 4-bit data output lines.

3.3 Register File

A Register File to include registers; 4 general-purpose registers of 2-bit each.

- It requires two operands and write a result back in register file.
- Sometimes part of operands comes from the instructions.
- Support of immediate class of instructions.

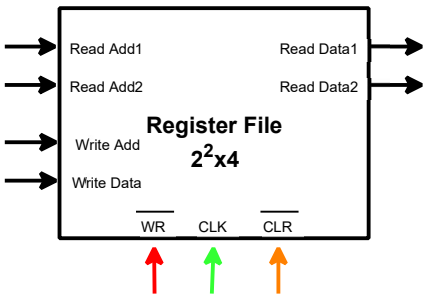


Figure 3-4. Block Diagram of Register File

3.3.1 Register File Specifications

Table 3-3. Description of Register File Input/output Pins

Type	Description

Size	***2^4x4*** Total number of registers are 4 and accessed using 2-bit address; e.g. 60 = 000, 61 = 001, …, 67 = 111. Each register can hold data of 4 bits.
Control Inputs	2312222 is an active low input; when 0 (writing mode) and when 1 (reading mode). 245122222 is an active low input; when 0 all registers data will be cleared.
Input	1-/,+„;, 1-/,+„<, 3%($-+„ requires 2-bit address lines to access specific registers. If 2312222 = ; and 45= = ;, then based on the 1-/,+„;, 1-/,+„< addresses, the register file will generate outputs. If 2312222 = > and 45= = >, then based on the 3%($-+„ address the data present on 3%($-./$/ lines will be written on the specific register.
Output	3%($-./$/;, 3%($-./$/< are 4-bit data output lines.

3.4 Arithmetic and Logical Units

An ALU will be composed of following units:

- Logical Unit: AND, OR.
- Arithmetic Unit: ADD, SUB.
- Comparator: SLT.

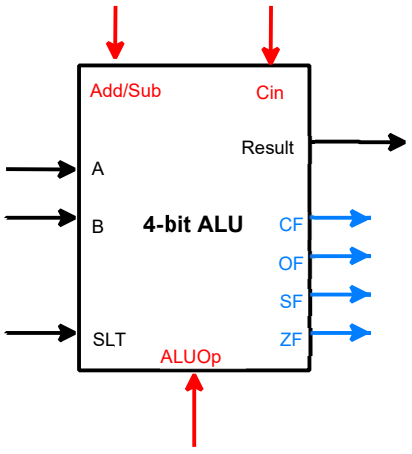


Figure 3-5. Block Diagram of 4-bit ALU

3.4.1 ALU Specifications

Table 3-4. Specifications of 4-bit ALU

Type	Description
Input	+ and ? are 4 bits each. @5A is a single bit.
Output	1-#&B$ is of 4 bits.
Control Signals	+„/@&D and 4(" are single bit. +5EFG is 2 bits select line of 4x1 MUX, where: 00 – Connects AND gate output to Result 01 – Connects OR gate output to Result 10 – Connects FA to Result 11 – Connects SLT to Result
Status Flags	Each flag is of 1 bit each. 4H – Carry Flag FH – Overflow Flag @H – Sign Flag IH – Zero Flag

4 Instruction Format

The simple microprocessor uses 16-bit instructions that are stored in the Instruction Memory.

There are three types of instruction formats:

4.1 Register to Register (R-Format) Instructions

- Instructions ○ AND, OR, ADD, SUB, SLT
- Process ○ Read source operands from Registers ○ Execute operation (AND, ADD etc.) in ALU ○ Write result back to Registers

Unused	Opcode	Rs	Rt	Rd	—
15:12	11:8	7:6	5:4	3:2	1:0

4.2 Immediate (I-Format) Instructions

4.2.1 Arithmetic Instructions

Instructions: ANDi, ORi, ADDi, SUBi, SLTi

Process:

- Read first source operand from Registers
- Read second source operand from Instruction
- Execute operation (Or, ADD etc.) in ALU
- Write result back to Registers

Unused	OpCode	Rs	Rt	Immediate Data (IMD)
15:12	11:8	7:6	5:4	3:0

4.2.2 Memory Reference

Instructions Instruction: 53 (Load Word)

Instruction

Process:

- Read first source operand from Registers
- Read second source operand from Instruction
- Execute operation (+.) in ALU to calculate DMEM address
- Read word from Data Memory
- Write result back to Registers

Unused	OpCode	Rs	Rt	Immediate Address (IMA)
15:12	11:8	7:6	5:4	3:0

Instruction: @3 (Store Word) Instruction

Process:

- Read first source operand from Registers
- Read second source operand from Instruction
- Read data from Registers
- Execute operation (+..) in ALU to calculate DMEM address
- Write word to Data Memory

Unused	OpCode	Rs	Rt	Immediate Address (IMA)
15:12	11:8	7:6	5:4	3:0

4.2.3 Branch Instructions

Instructions: ?JK, ?LJ

Process:

- Read source operands from Registers
- Execute operation (@E?) in ALU to identify IH
- For BEQ: If IH = ;, then M4 + ; + FOO#-$ otherwise M4 + ;
- For BNE: If IH = >, then M4 + ; + FOO#-$ otherwise M4 + ;

Unused	OpCode	Rs	Rt	Immediate Offset
15:12	11:8	7:6	5:4	3:0

4.3 Jump (J-Format) Instructions

Instruction: P

Process:

• Load new PC with address specified within instruction: M4 = +„%-##.

Unused	OpCode	Unused	Address
15:12	11:8	7:4	3:0

5 Types of Instructions

Data Operations:

Arithmetic (ADD, SUB, ADDi, SUBi)

Logical (AND, OR, ANDi, ORi)

Data Transfer:

- Load (LW) – Memory to Register
- Store (SW) – Register to Memory **Sequencing:**
- Branch (Conditional, e.g., <, >, ==)
- Jump (Conditional, e.g., j)

Assume that all registers initially store the numeric value 0, and Rs, Rt, Rd can refer to any register within Register File. The microprocessor you have to design will have the following types of instructions:

Table 5-1. Types of Instructions (Examples)

Function	OpCode	Type	Instruction	Operation
AND	0000	R	AND Rd, Rs, Rt	Rd = Rs & Rt
OR	0001	R	OR Rd, Rs, Rt	Rd = Rs | Rt
ADD	0010	R	ADD Rd, Rs, Rt	Rd = Rs + Rt
SUB	0011	R	SUB Rd, Rs, Rt	Rd = Rs - Rt
ANDi	0100	I	ANDi Rt, Rs, IMD	Rt = Rs & IMD
ORi	0101	I	OR Rt, Rs, IMD	Rt = Rs | IMD
ADDi	0110	I	ADD Rt, Rs, IMD	Rt = Rs + IMD
SUBi	0111	I	SUB Rt, Rs, IMD	Rt = Rs - IMD
SLT	1000	R	SLT Rd, Rs, Rt	if (Rs < Rt) then Rd=1 else Rd=0
SLTi	1001	I	SLT Rt, Rs, IMD	if (Rs < IMD) then Rt=1 else Rt=0
BEQ	1010	I	BEQ Rt, Rs, Offset	if (Rs == Rt) then goto PC+1+ Offset
BNE	1011	I	BNE Rt, Rs, Offset	if (Rs != Rt) then goto PC+1+ Offset
J	1100	J	J Address	Goto PC = Address
Unused	1101	--	--	--

LW	1110	I	LW Rt, IMA(Rs)	Rt = DMEM[Rs + IMA]
SW	1111	I	SW Rt, IMA(Rs)	DMEM[Rs + IMA] = Rt

6 Digital Circuit Design

6.1 ALU Design (4-bit)

The core of the processor - all the actual computations are performed here. As shown in the instruction set, operations such as addition, subtraction and logical operations are all done in this unit. Also, the output of the ALU is used as the address for certain memory related operations. The block diagram of 4-bit ALU is mentioned above in the Figure 3-5.

An hierarchical approach is used to design the 4-bit ALU, starting from gates to construct Half Adder (HA), and then using two HAs to design Full Adder (FA), as shown in Figure 6-1, and then arithmetic unit (for addition and subtraction operations) and lastly logical component (of AND and OR gate operations) is integrated to design 1-bit ALU, as shown in Figure 6-2. By using 4 1-bit ALUs, 4-bit ALU is designed, as shown in Figure 6-3.

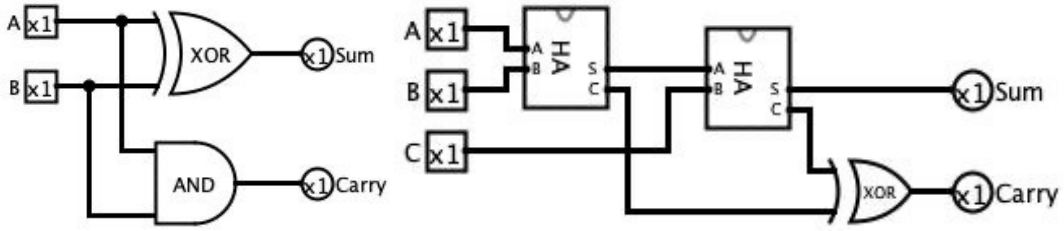


Figure 6-1. Digital Circuits of Half Adder and Full Adder

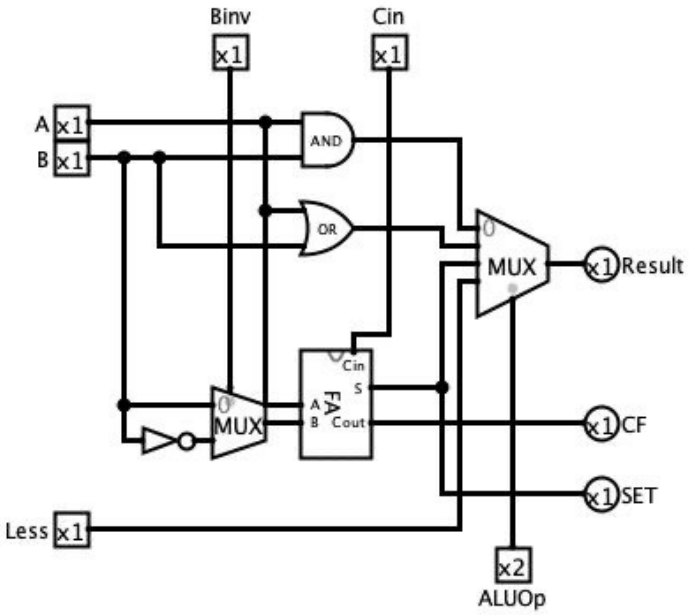
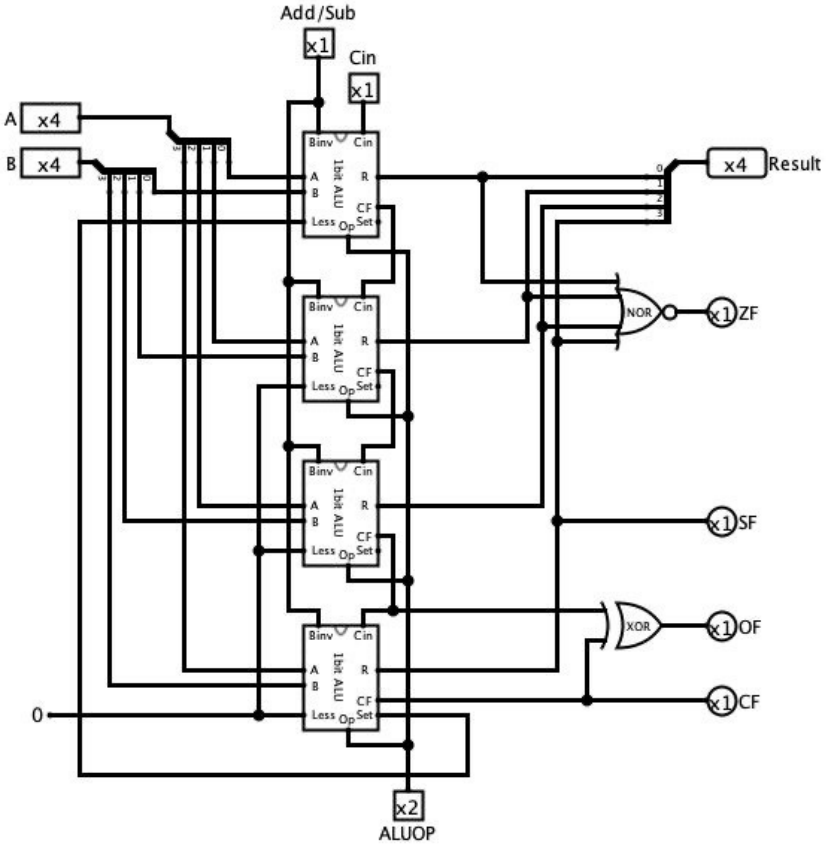
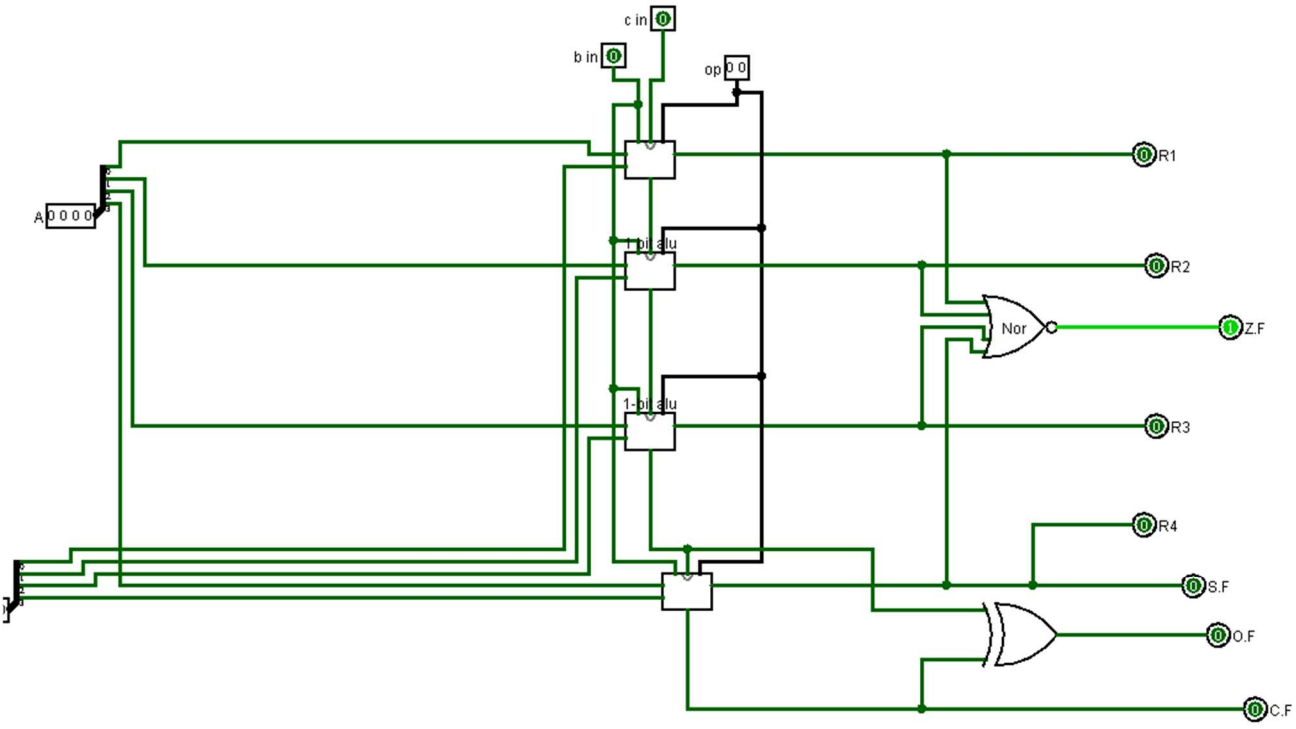


Figure 6-2. Digital Circuit of 1-bit ALU



ALU Design (4-bit)



Conclusions:

In the construction of 4-bit ALU first we made half adder. With the help of Half Adder, we made Full Adder. In Full Adder we used AND, OR and XOR gates.

Then we implemented these concepts to make 1 Bit ALU, in which we used MUX in addition with previous Concepts. We have verified the table below after constructing the 4 Bit ALU.

Table 6-1. Verification of ALU Working

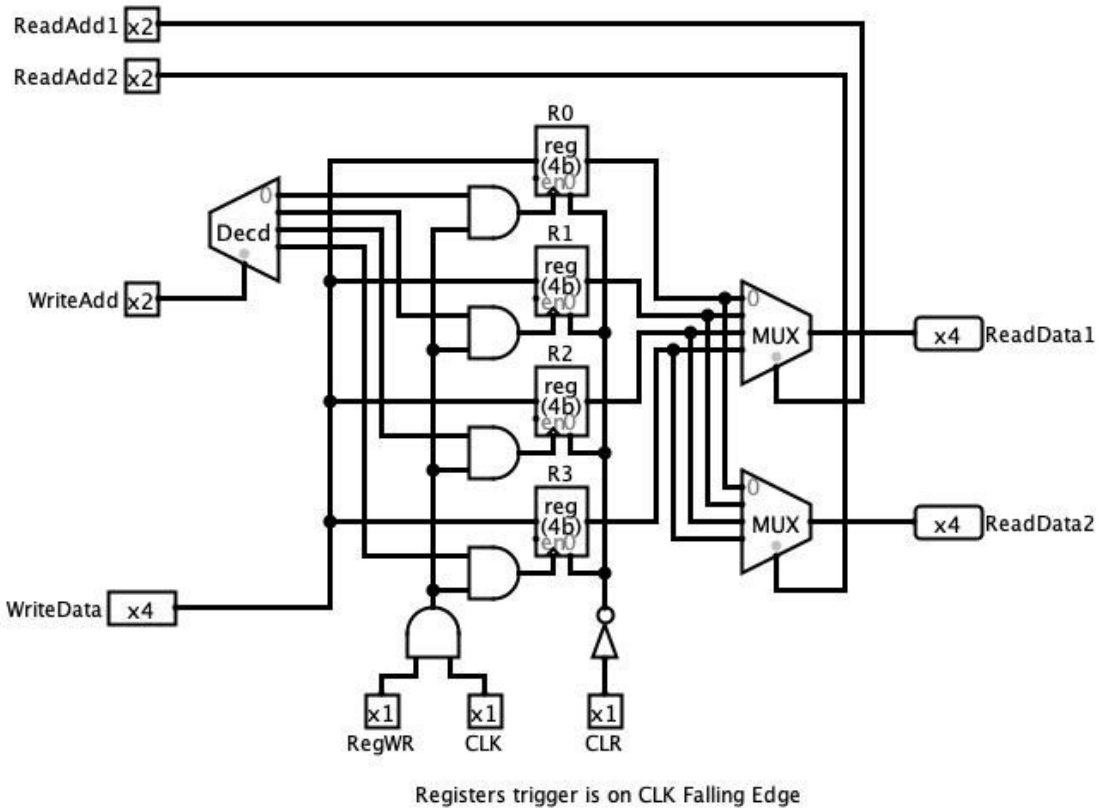
Operation	Inputs		Control Inputs				Output	Status Flags			
	A	B	ALUOp [A1A0]		Add/Sub (as Binv)	Cin	Result	CF	OF	SF	ZF
AND	1010	0110	0	0	x	x	0010	x	x	x	x
OR	0010	1000	0	1	x	x	1010	x	x	x	x
ADD	0011 (+3)	0011 (+3)	1	0	0	0	0110 (+6)	0	0	0	0

ADD	1101 (-3)	0111 (+7)	1	0	0	0	0100 (+4)	1	0	0	0
SUB	1100 (-4)	0001 (+1)	1	0	1	1	1011 (-5)	1	0	1	0
SUB	0111 (+7)	0111 (+7)	1	0	1	1	0000 (0)	1	0	0	**1**
SUB	1000 (-8)	0001 (+1)	1	0	1	1	0111 (x)	1	**1**	0	0
SLT	0011 (+3)	0010 (+2)	1	1	1	1	If A<B, then 0 0000	1	0	0	1
SLT	0011 (+3)	0110 (+6)	1	1	1	1	If B<A, then 1 0001	0	0	0	0

Rebuild the similar ALU in Logisim and verify it's working.

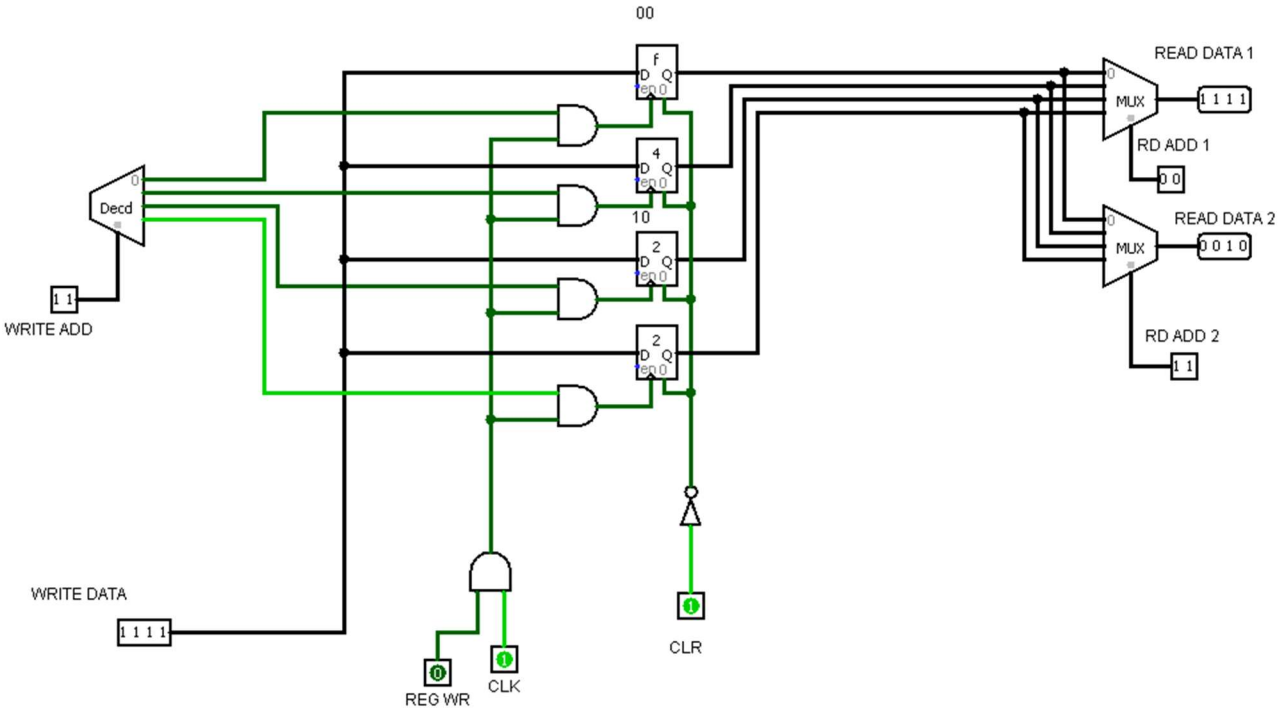
6.2 Register File (2^4x4)

Consider the block diagram of Register File (RF) as mentioned above in the Figure 3-4. The RF is a set of 4 registers: named as R0, R1, R2, R3, each storing a 4-bit value. There are 2 output values 6QRSTRUR1 and 6QRSTRUR2, whose values are selected based on the instruction fields: 6W (6QRSXSS1) and 6U (6QRSXSS2), as in the instruction definitions given above by giving control signal (6QYZ6 = 1). There is one port by which data can be written into by one of the register 6S (Z[\UQXSS), but make sure to provide a control signal (6QYZ6 = 0) to control whether or not to update the value. The clear signal]222^2262 is to clear the contents of RF.



Registers trigger is on CLK Falling Edge

Register File:



In Logisim, D-Flip Flops or Register component (*trigger is on CLK Falling Edge*) can be used to create Register File, as shown in the Figure 6-4.

Rebuild the similar Register File in Logisim and verify it's working.

Table 6-2. Verification of Register File Working

Operation	Inputs				Control Inputs			Outputs	
	ReadAdd1	ReadAdd2	WriteAdd	WriteData	RegWR	CLK	CLR	ReadData1	ReadData2
Clear	xx	xx	xx	xxxx	x	x	0	xxxx	xxxx
Operation	Inputs				Control Inputs			Outputs	
	ReadAdd1	ReadAdd2	WriteAdd	WriteData	RegWR	CLK	CLR	ReadData1	ReadData2
Write R0	xx	xx	00	0010	0	↓	1	xxxx	xxxx
Write R3	xx	xx	11	0100	0	↓	1	xxxx	xxxx
Read R0,R3	00	11	xx	xxxx	1	↑	1	0010	0100

Conclusions:

In this part we made a register file by using the d flip flaps and registers component decoder and multiple MUX and create a register file .

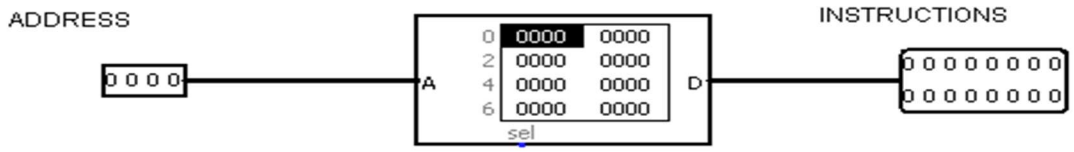
And also verify the truth table

We constructed Register File by Using Four 4 Bit counters

6.3 Instruction Memory (2^4x16)

This can be a combinational unit - it takes just the address bus (4-bit value) as input and gives out a 16-bit value that is the instruction to be decoded. The address is provided by the Program Counter (PC). The Block diagram of Instruction Memory is mentioned above in the Figure 3-2.

Instruction Memory:

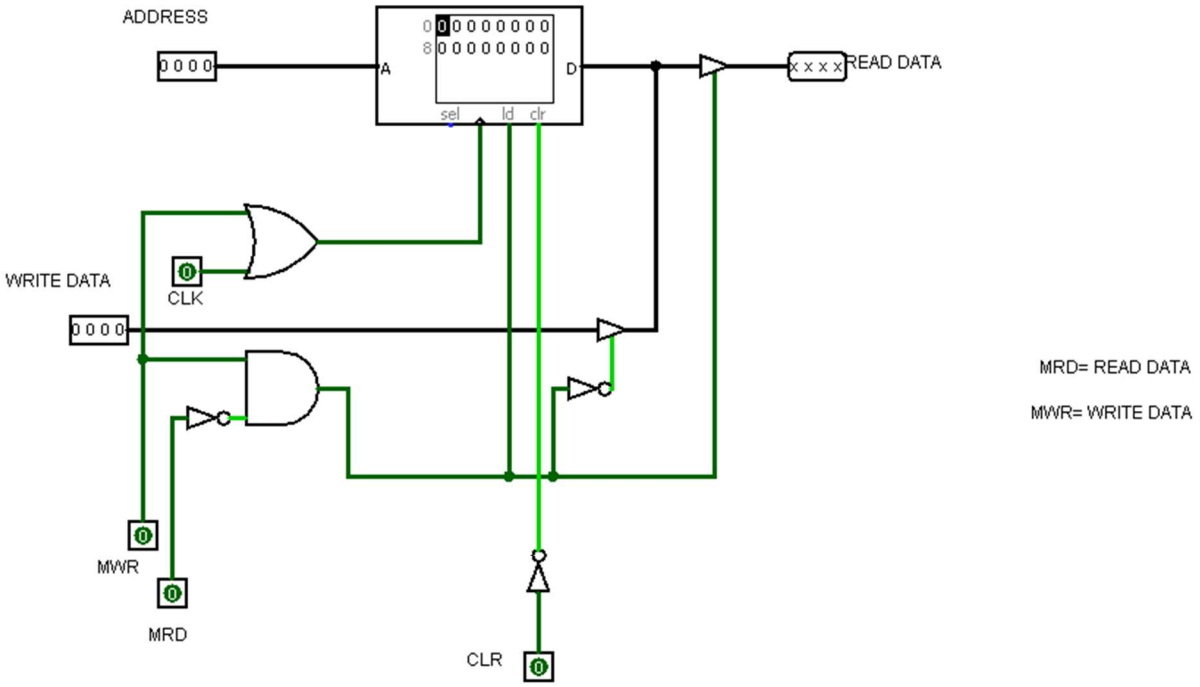


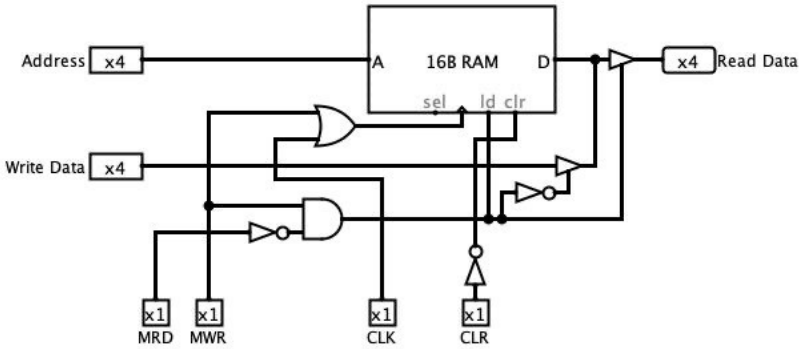
Build the Program Memory using ROM component in Logisim, as shown in Figure 6-5 and verify it's working.

6.4 Data Memory (2^4x4)

This needs to be a sequential / clocked unit. At any given cycle, you are either reading from it or writing to it. In case of a LW or SW instruction, the address is either immediate or the output of the ALU. The data output of the DMEM will always go into the Register File, where it gets stored into a register selected by Rd. The Data Memory circuit diagram is shown in Figure 6-6.

Data memory:





Rebuild the similar Data Memory in Logisim using RAM component and verify it's working.

Table 6-3. Verification of Data Memory Working

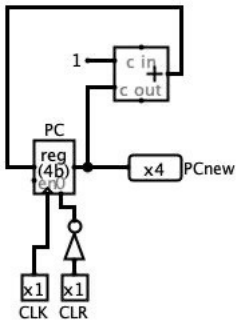
Operation	Inputs		Control Inputs				Output
	!""#$%%	&#'($)*(*	+---,--)--	-+--&,-----	./0	-./,-----	,$*")*(*

Clear	**xxxx**	**xxxx**	**x**	**x**	**x**	**0**	**xxxx**
Write DMEM[5]	**0101**	**1111**	**1**	**0**	**↑**	**1**	**xxxx**
Read DMEM[5]	**0101**	**xxxx**	**0**	**1**	**↑**	**1**	**1111**

6.5 Program Counter (4-bit)

Program Counter is a 4-bit register. Under normal operations, will always increment by 1 on every clock cycle, to access the next instruction.

The Program Counter circuit diagram is shown in Figure 6-7.



Program Counter (4-bit)

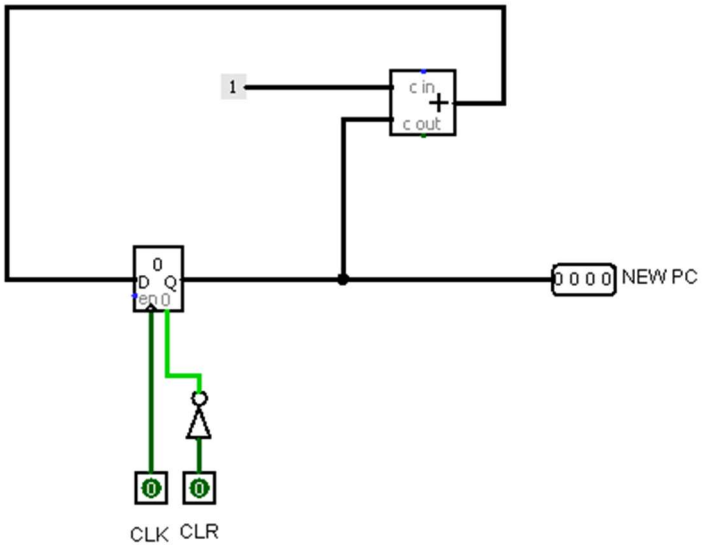


Figure 6-7. Digital Circuit of Program Counter

Rebuild the similar Program Counter in Logisim and verify it's working.

Table 6-4. Verification of Program Counter

Operation	Control Inputs		Output
	./0	·/,------	1.2$3
Clear / Reset	**x**	**0**	**0000**
At each CLK pulse: Increment	↑	**1**	**0001** **0010** **…** **1111** **0000** **0001** **…**

6.6 Control Unit

The unit that actually makes the entire processor work as expected. The input to this is the instruction word, and the output is a set of control signals that decide, for example, whether the register file is to be updated, what operation is to be done by the ALU, whether memory read or write is required etc.

One way to implement this is to make it a combinational block that will generate the following signals:

Control Unit:

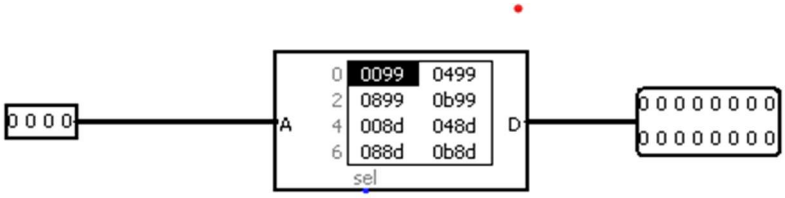
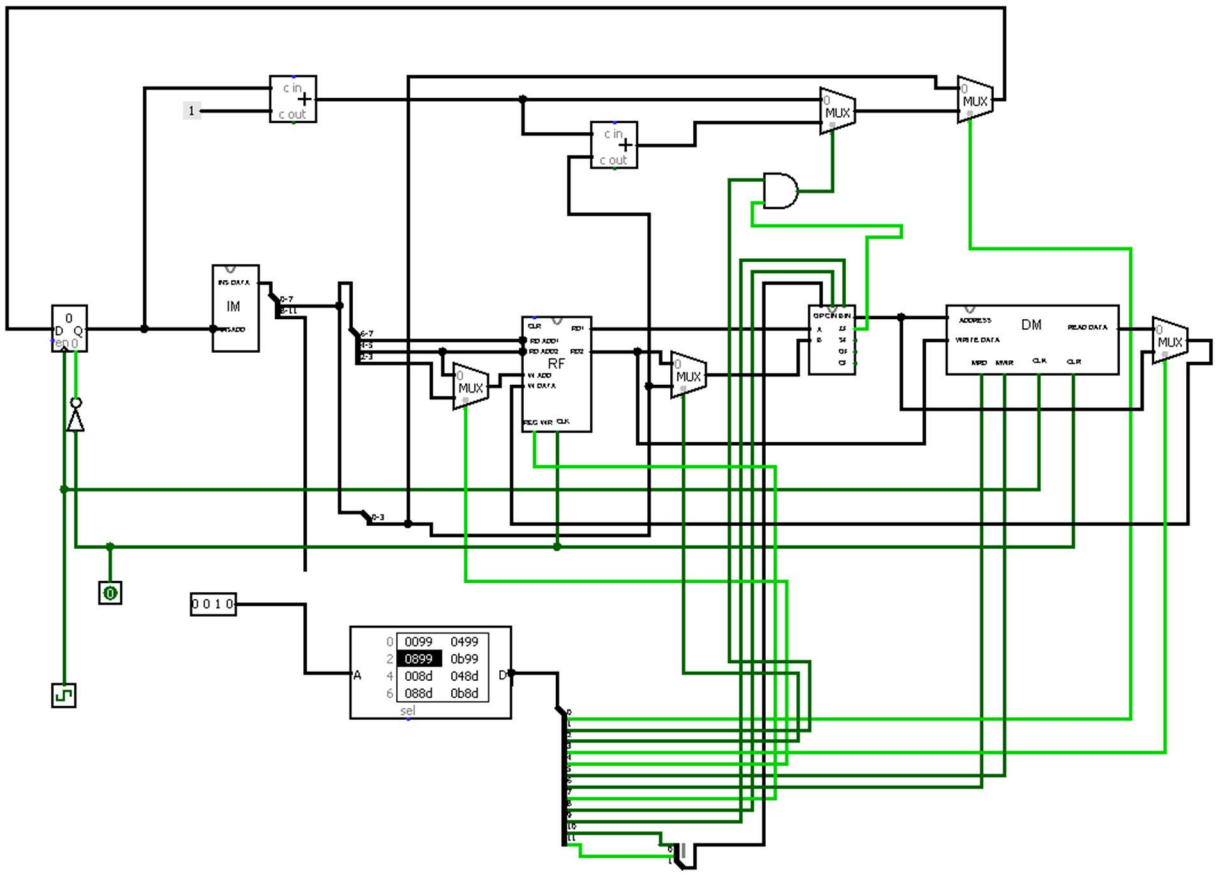


Table of Control signal Microprocessor:

Function	Type	Input	Outputs										
		OpCode	A1A0	Add/Sub	Cin	Reg WR	Mrd	mwr	RegDst (SW)	MemtoReg (LW)	ALUSrc (IMM)	Branch	Jump
AND	R	0000	00	0	0	0	1	1	1	1	0	0	1
OR	R	0001	01	0	0	0	1	1	1	1	0	0	1
ADD	R	0010	10	0	0	0	1	1	1	1	0	0	1
SUB	R	0011	10	1	1	0	1	1	1	1	0	0	1
ANDi	I	0100	00	0	0	0	1	1	0	1	1	0	1
ORi	I	0101	01	0	0	0	1	1	0	1	1	0	1
ADDi	I	0110	10	0	0	0	1	1	0	1	1	0	1
SUBi	I	0111	10	1	1	0	1	1	0	1	1	0	1
SLT	R	1000	11	1	1	0	1	1	1	1	0	0	1
SLTi	I	1001	11	1	1	0	1	1	0	1	1	0	1
BEQ	I	1010	10	1	1	1	1	1	0	0	0	1	1
BNE	I	1011	10	1	1	1	1	1	0	0	0	1	1

J	J	1100	00	0	0	1	1	1	0	0	0	0	0
Unused	--	1101	XX	X	X								
LW	I	1110	10	0	0	0	0	1	0	0	1	0	1
SW	I	1111	10	0	0	1	1	0	0	1	1	0	1

Simple Microprocessor Design :



Assembly Instruction:

11 A ← DMEM [0] ← -1

12 B ← DMEM [2] ← 2

13 C ← DMEM [1] ← 1

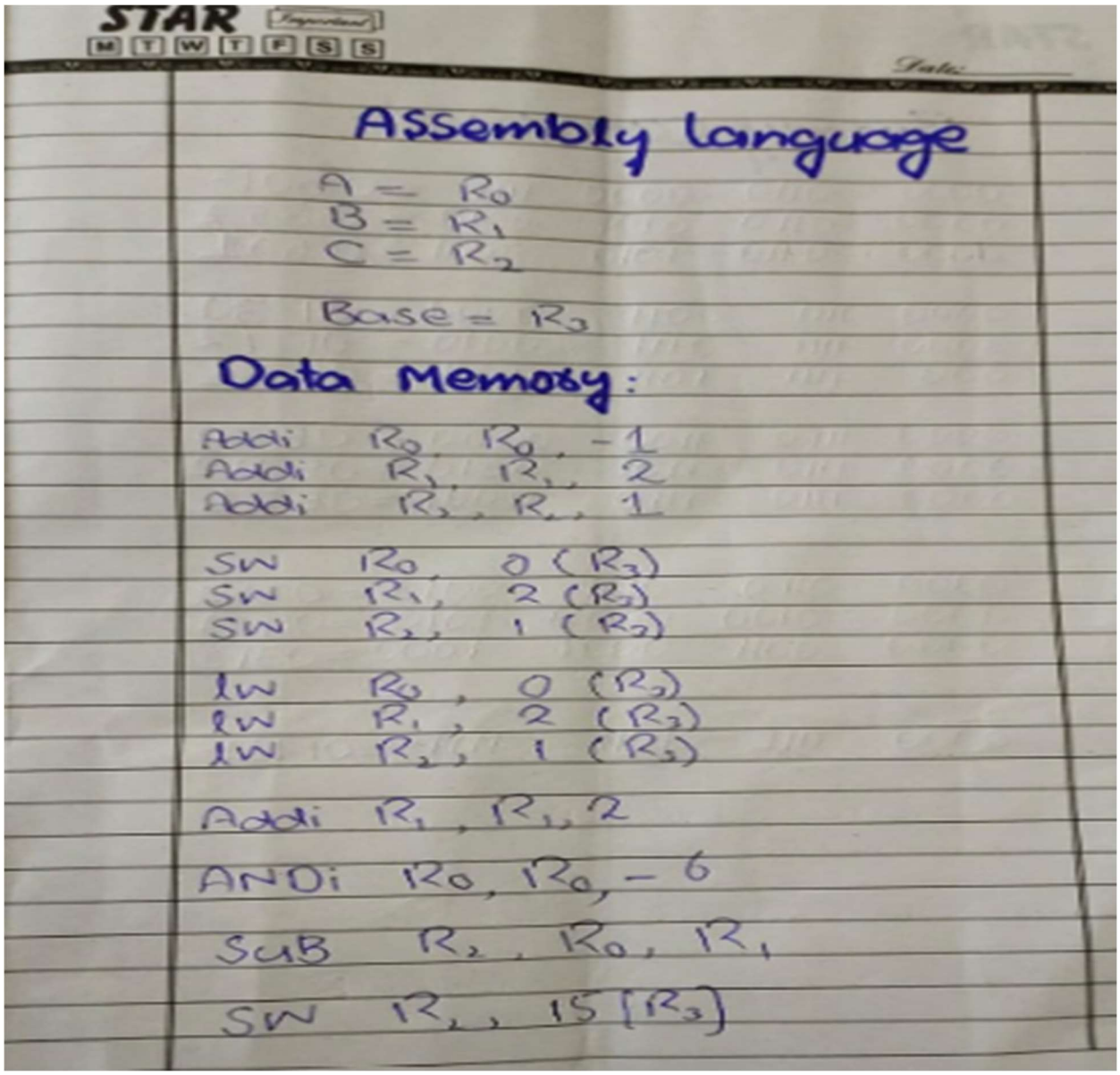
14 B ← B + 2

15 A ← A & -6

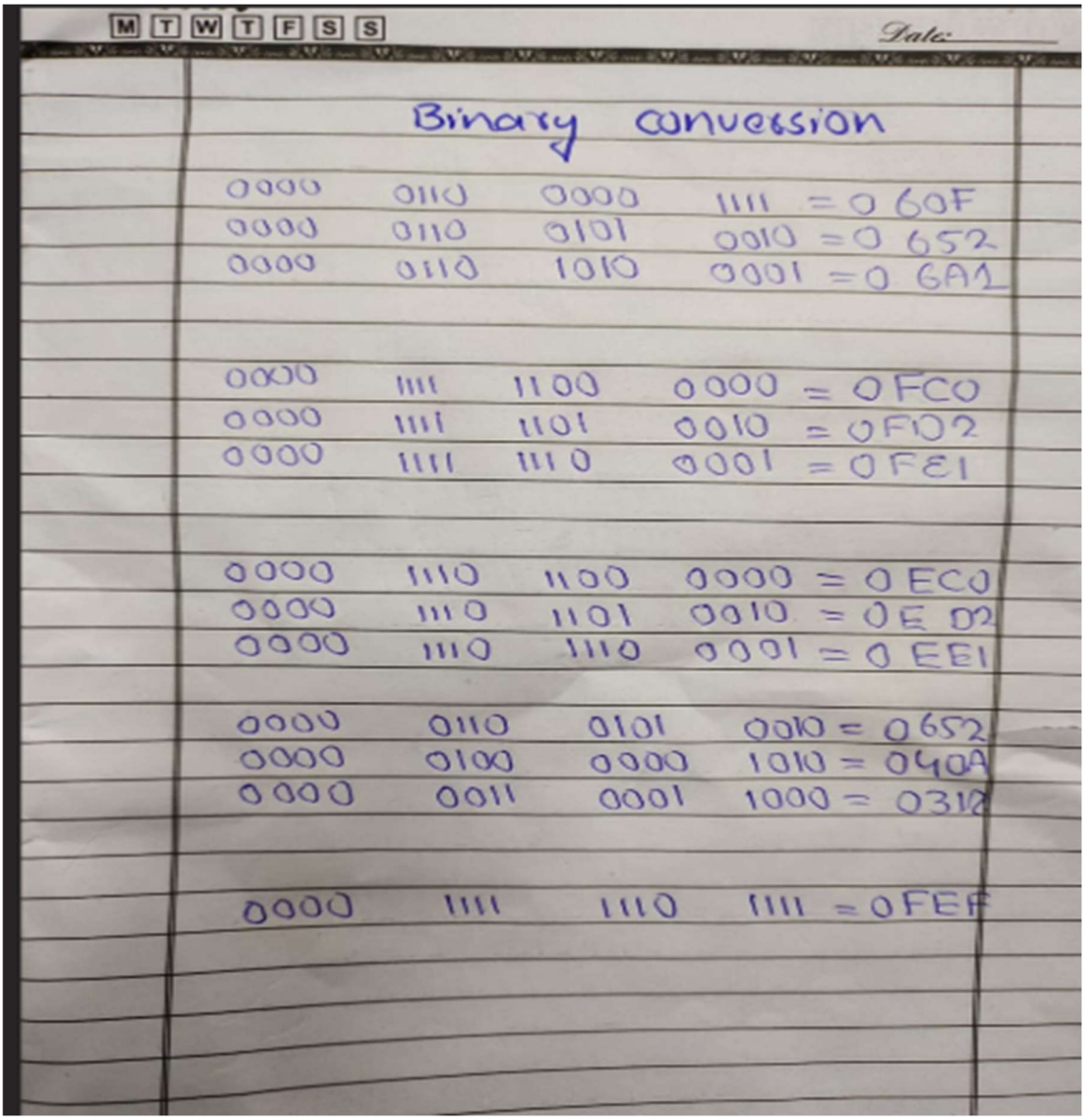
16 C ← A – B

17 DMEM [15] ← C

Binary Instruction:



Assembly code:



Conclusions

In this project we construct a simple microprocessor by using the given instruction as we used the the instruction memory , data memory ,register file ,4 bit ALU and also the MUX , PC counter and some gate to make first the basic things to made a control unit we used the ROM types controller in we made the truth table correct and then we are able to verifying that all component which we used in this project

as we at first step to verifying our table and then move on the unit that actually makes the entire processor work as expected.

The input to this is the instruction word, and the output is a set of control signals that decide. Then we combine all the component to make a simple microprocessor. After that we our task is about Assembly code convert into the binary or hgza decimal form to perform the all vale in which the microprocessor are preformed perfectly. So we put our assembly into the binary or hagza decimal into the instruction memory .and given table into the he data memory to run the processor.

Finally, we make our microprocessor correct according to the given condition, so our microprocessor preform all the operation correctly. We learn so many things to do this project.
