MERL
Accelerating Engineering Innovation

Usman Institute of Technology
عثمان انسٹیٹیوٹ آف ٹیکنالوجی

RV32I SINGLE CYCLE PROCESSOR

RISC-V Data Path

ABSTRACT

This project is based on the synchronous and single cycle logic that means on each rising edge clock impulse only single instruction is being executed providing same constant time to each instruction. The architecture on which the circuit is developed is RV32I of RISC-V enabling to understand the procedure performed at the back end of each supported instruction

AMMAR BIN AMIR

MERL-UIT (19B-004-EE)

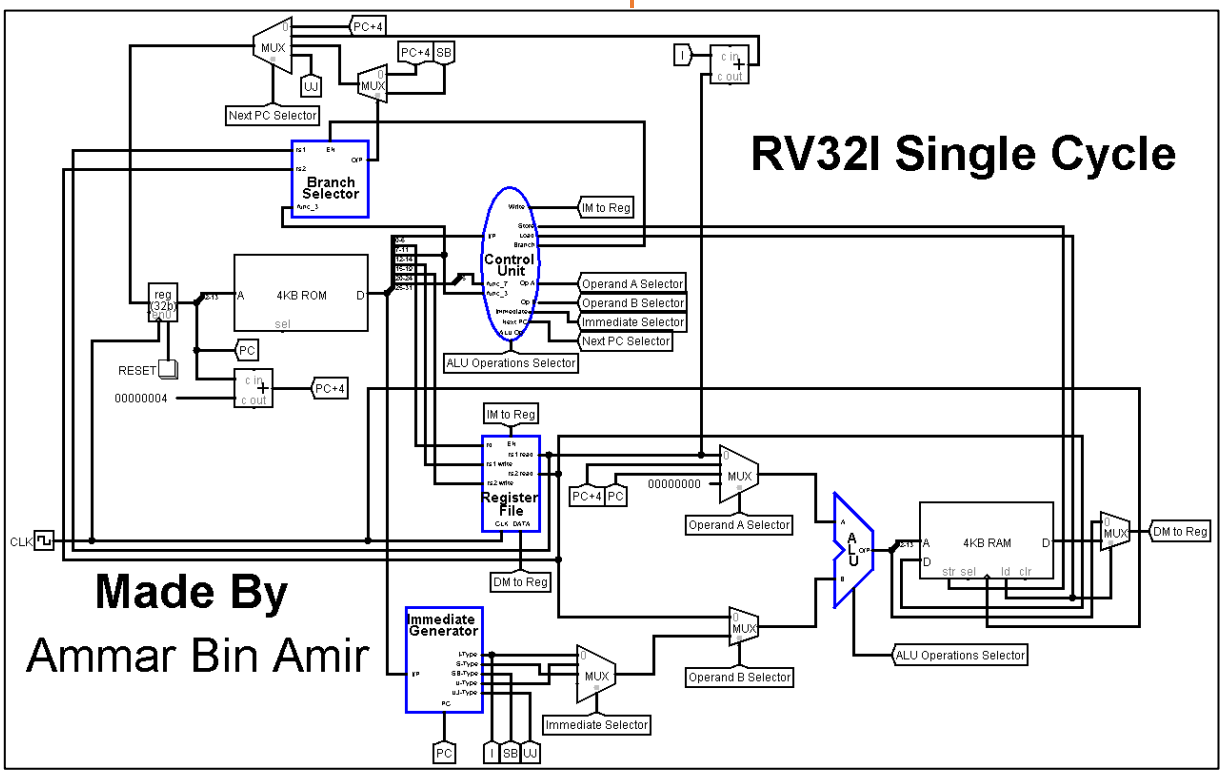

Table of Contents

Table of Figures

Introduction

RV32I Single Cycle CPU is based on 32 Bit RISC-V architecture. As the abbreviation itself describes that RISC (Reduced Instructions Set Computer) architecture provides an ease to the users to while comparing to the CISC (Complex Instructions Set Computer) architecture. The instructions studied in RV32I is being implemented on Logisim Software after getting hands on practice on Online Simulator Venus.

Components

The components used are as following:

- ## Program Counter (PC)
  The program counter tells the instruction memory about the next address whether it's the preceding one or a jump to another address.

- ## ROM (Instruction Memory)
  It stores the program i.e., number of instructions that are going to be implemented or executed.

- ## Control Unit
  It is divided into two sub-unit.

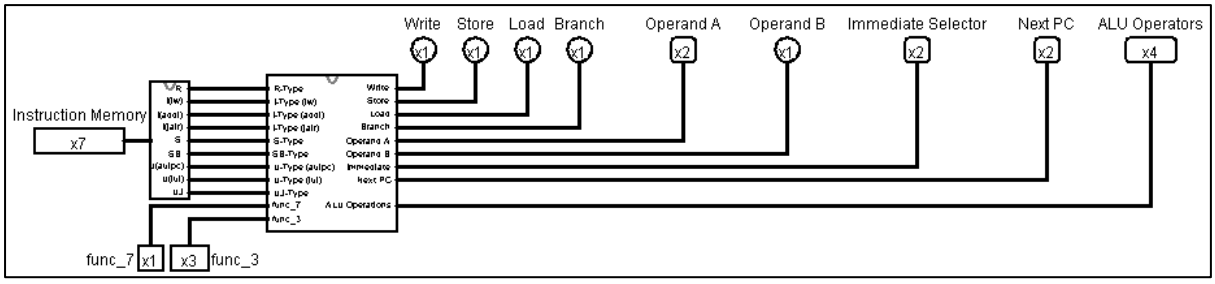

Figure 1: Control Unit Circuit

They are type decoder and control decoder. They are following:

- ## Type Decoder

This system differentiates the instruction into each type by the help of opcodes. This further helps in the circuit to perform functions required.

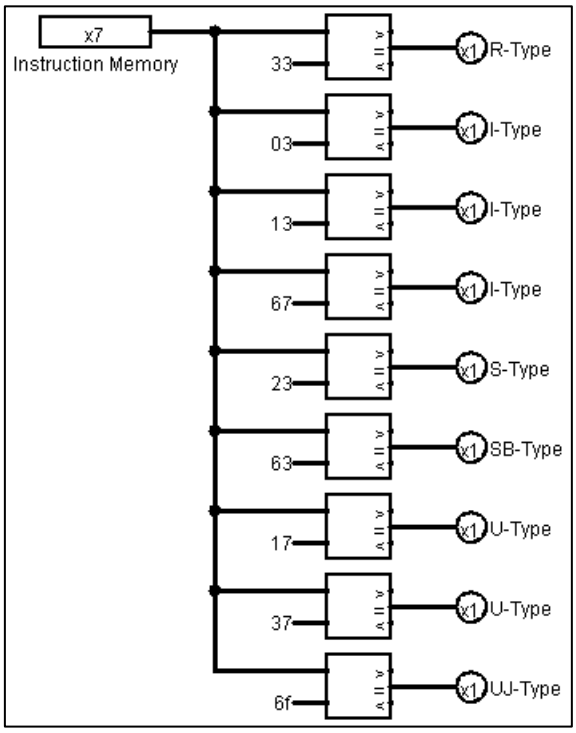

Figure 2: Type Decoder Circuit

- ## Control Decoder

This system is one of the main systems as it controls which part of the instruction is to be selected in the scenario when similar type of instruction is to be proceeded to get executed.

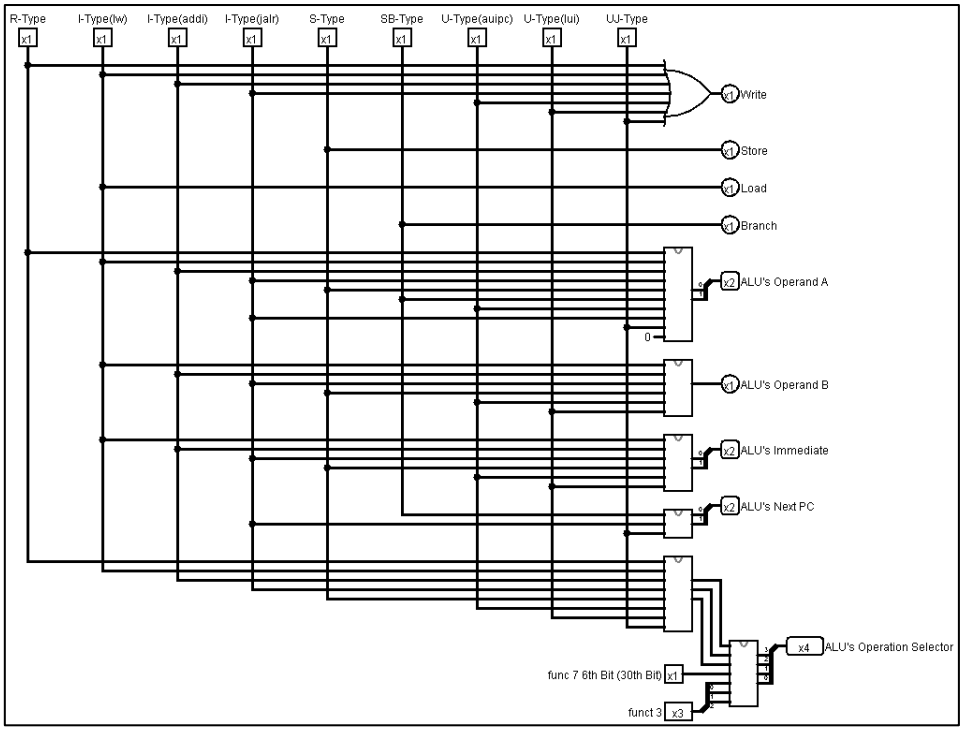

Figure 3: Control Decoder Circuit

- ## Register File

This 32-bit register file store the value respective to destination register, source register 1 and source register 2 from which it is made up off. On each rising edge, the value is being read and write regarding the instruction.

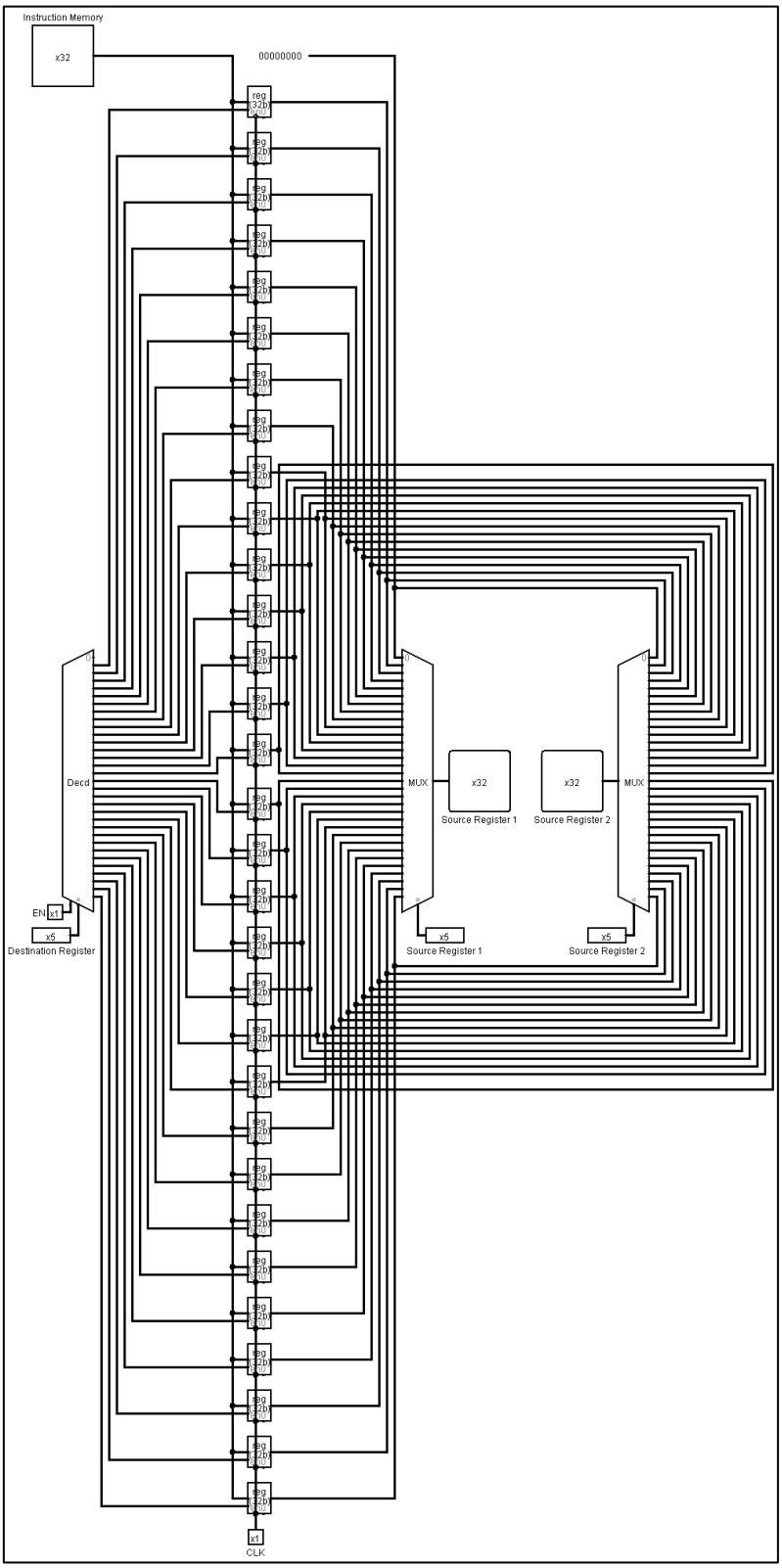

Figure 4: Register File

- ## ALU (Arithmetic Logical Unit)

  This system is also one of the main systems in the processor. It contains two operands that are operand A and operand B which perform addition to the bits as the type of instruction is executed. Its result holds great importance in the CPU. In its abstraction, all the supported instructions are being used accordingly with the help of components such as adder for addition, shifter for shifting, etc.

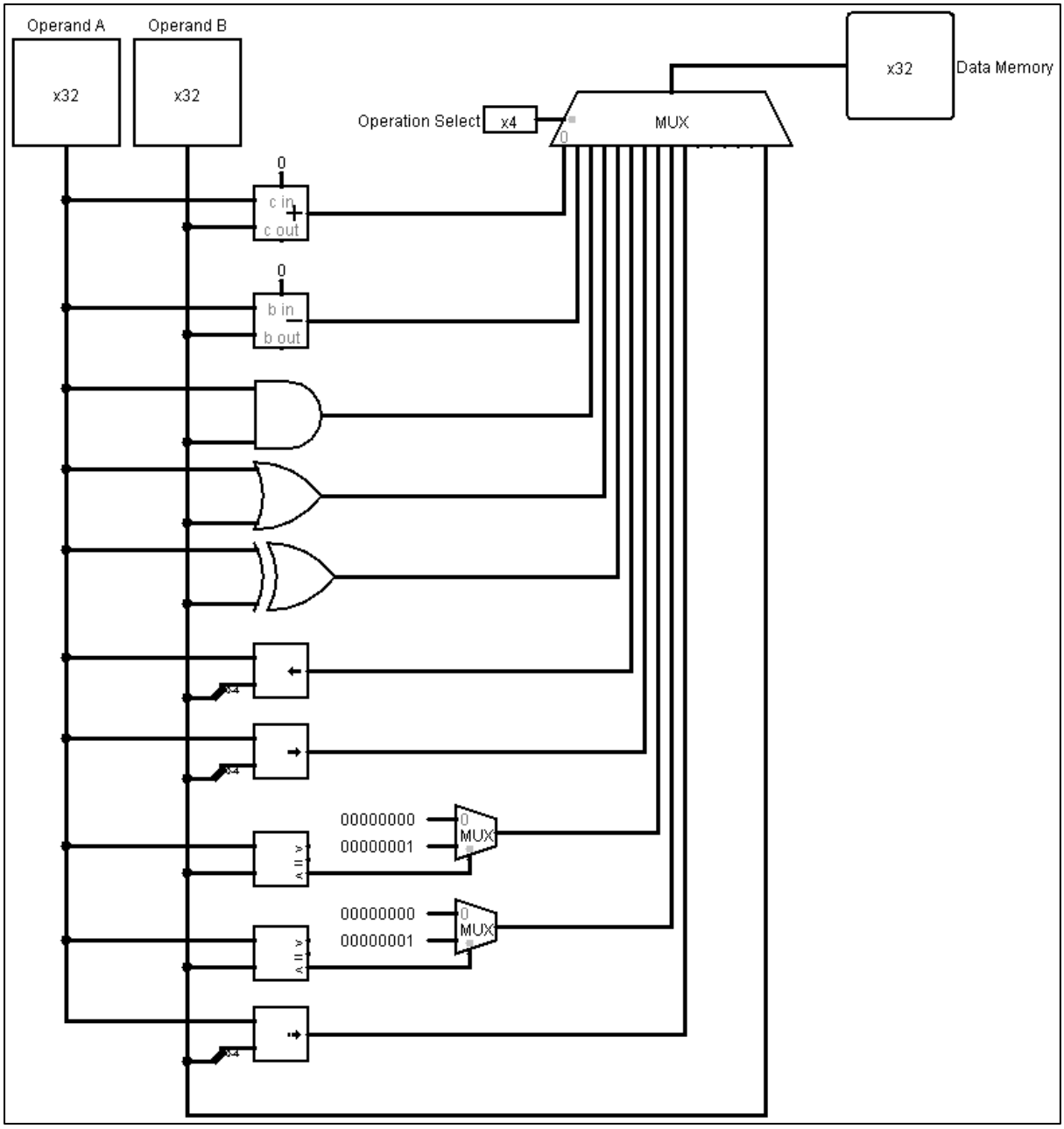

Figure 5: ALU Circuit

- ## Immediate Selector

This selector selects the immediate to be given input to the ALU. The RV32I core format possesses only two types of immediate either 12-bit or 20-bit.

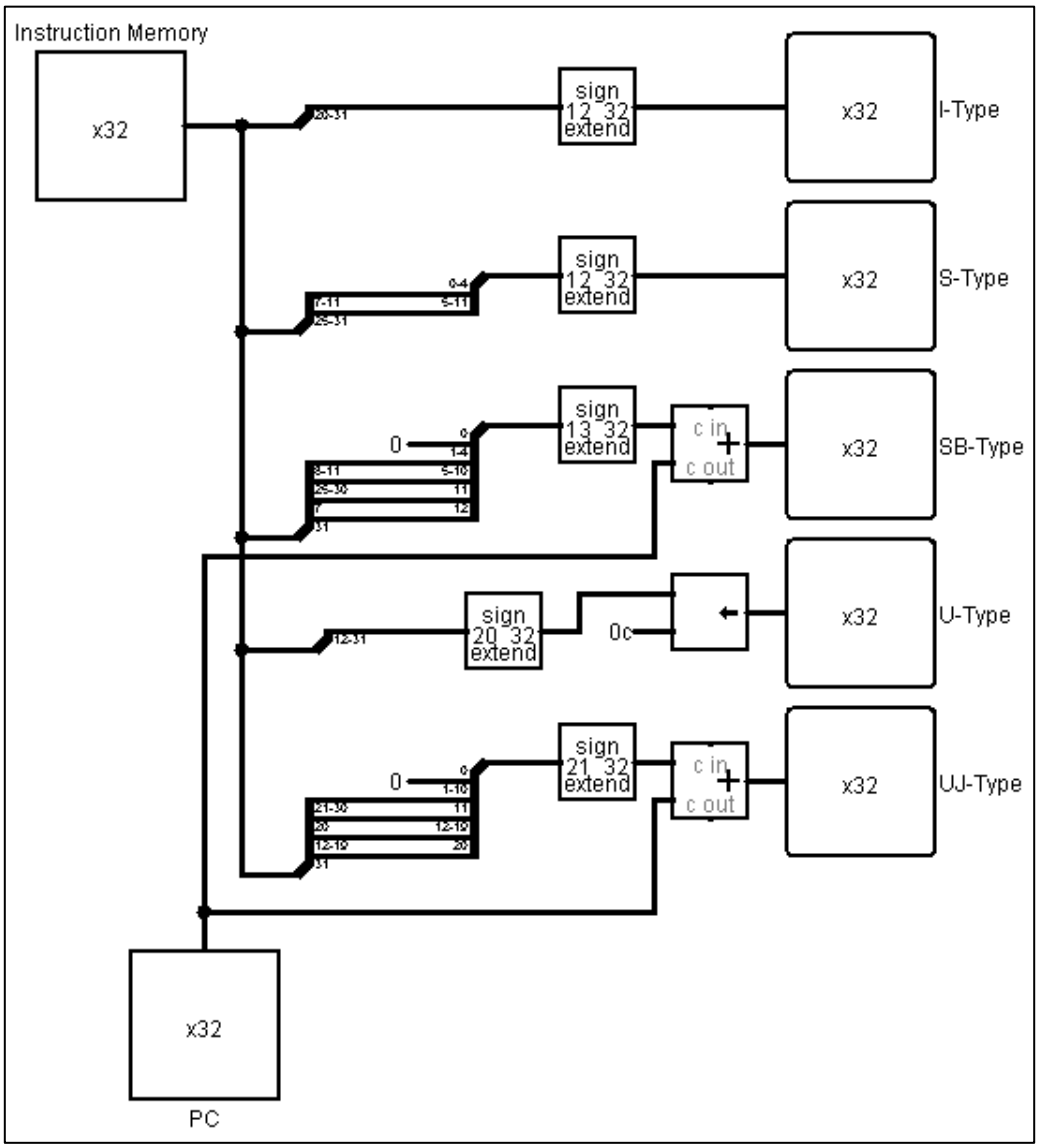

Figure 6: Immediate Generatior Circuit

- ## Branch Selector

  This selector determines the branch operations because the operations are on either true or false condition, that further initiates the instruction on which it has to be jumped.

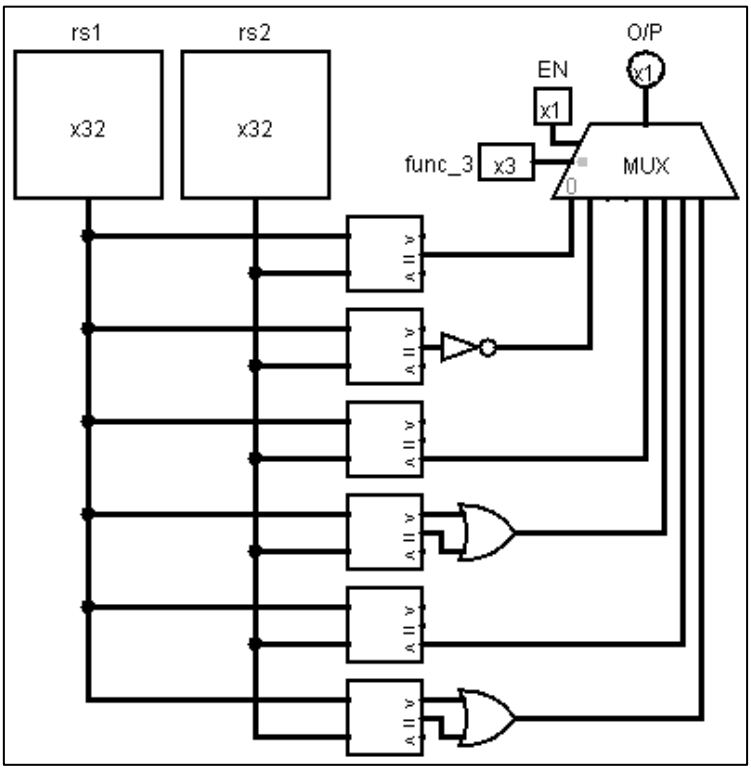

Figure 7: Branch Selector Circuit

- ## RAM (Data Memory)

  This is the data memory on which the store and load commands are executed. It stores data on its specific address.

Methodology

RV32I architecture is divided into six types. The core instruction format simplifies it in understanding the execution of each instruction respective of its type. The six types are as following:

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

Figure 8: Core Intructions Format Table of each Type

Each instruction is being understood clearly regarded to execution, formation and structure. Different examples are being practiced; codes are written with the combination of different instructions from each type on the assembly language. The codes contains both types of executions; from instruction memory to data memory and then to register and simply from instruction memory to register storing the data on specific register's addresses.

To understand the data path of the software, a circuit has been developed to give the clear concept of the abstraction behind the assembly language.

Implementation

The circuit is implemented using the concept of software. It is divided into main system and its sub-systems. They are as follows:

Main System

The first is to create a structure of all the components. According to the assembly language, the first component that is the initiator is the program counter which is change to next address on the offset of 4 for which an adder is attached. The PC has been input into the ROM of size 4Kbyte through 12-bit address bus eliminating first two bits providing the range of 2-13 bits. The output from the ROM is divided according to the core format that is transmitter through 32-bit address bus. The opcode, func_3 and func_7 is being transmitted into control unit whereas the remaining rd, rs1 and rs2 are transmitted into register file. Then, the ALU that is accepting two inputs operand A and operand B along with its operation selector performs addition and transmits the output into RAM that is of same size as ROM and the second input of RAM has been provided by the operand B that further transmits the output into register file. Last but not the least, the PC is getting an input selected between PC+4, Branch, U and UJ type that are jalr and jal respectively.

Sub-Systems

The sub-systems are following:

- ### Write
  According to the core format, there are two only two types out of six that doesn't contain destination register that are S and SB, so this enables the register file to accept the data to be written. It is implemented using OR gate.

- ### Branch
  This circuit is enabling the branch selector to be operated.

- ### Store
  This circuit is enabling the store pin in the RAM.

- ### Load
  This circuit is enabling the load pin in the RAM.

- ### Operand A Selector
  Basically, the operand A can be selected between rs1 from register file, current PC from U type, PC with the increment of 4 for jump commands and simply zero. This circuit is designed to select single output from these for which this is connected to the multiplexer that is connected with these four inputs providing only a single output to operand A at the instance.

- ### Operand B Selector
  This selects whether the operand B is to be selected from rs2 or from the immediate generator.

- ### Immediate Selector
  This circuit is designed to provide the inputs to the multiplexer connected after the immediate generator to enable the immediate from I, S and SB type to the operand B as all of them are of 12-bit.

- ### Next PC Selector
  This circuit is specially designed for the PC as there are few instructions on which the next instruction that is to be executed is not the preceding one but there is a skip of number of instructions and a jump that can be seen in SB, U and UJ types. This enables to select from these three either or simply provides PC+4 to the PC.

- ### ALU Operations Selector
  As it is previously mentioned that ALU is one of the main components in the circuit, therefore the importance of this component is described by its construction that this circuit accepts the func_3 and func_7 that is being

transmitted into control unit for this along with the personally assigned opcodes of 3 bits for the supported instructions. The output of this circuit is actually the input of the ALU operation select pin.

Results

After the implementation, the next step is to check or match whether the obtained results of the software and hardware are the same or not!

Verification

The verification process is not a hard nut to crack. In the software, after the code has been written, it is being executed stepwise from which the results are obtained in decimal system. In the hardware, we have to connect a 32-bit output from the pin or bus and the result is matched by converting the decimal values into binary system. The ease that we get is that if the concepts and views are clear then we can have more further check and balance connecting the outputs to any wire for more clarification.

Test Programs

There are several programs based on a specific type while some on a particular concept. One of them is discussed below:

Fibonacci Series

This program stores number '1' at zeroth address in data memory i.e. RAM, then according to the loop, the value is updated by an increment of one that means each time the next value is rewrite upon the previous value with addition of one on the same address.

The program is as follows:

```
addi x5,x0,0          # Initialization
loop:
addi x5,x5,1          # Increment
sw x5,0x0(x0)
lw x6,0x0(x0)
jal ra,loop           # Iteration
```

The outputs of software are following:

PC	Machine Code	Basic Code	Original Code
0x0	0x00000005	Invalid Instruction	Invalid Instruction
0x4	0x00128293	addi x5 x5 1	addi x5,x5,1 # Increment
0x8	0x00502023	sw x5 0(x0)	sw x5,0x0(x0)
0xc	0x00002303	lw x6 0(x0)	lw x6,0x0(x0)
0x10	0xFF5FF0EF	jal x1 -12	jal ra,loop # Iteration

Figure 9: Instruction Memory Output in Venus

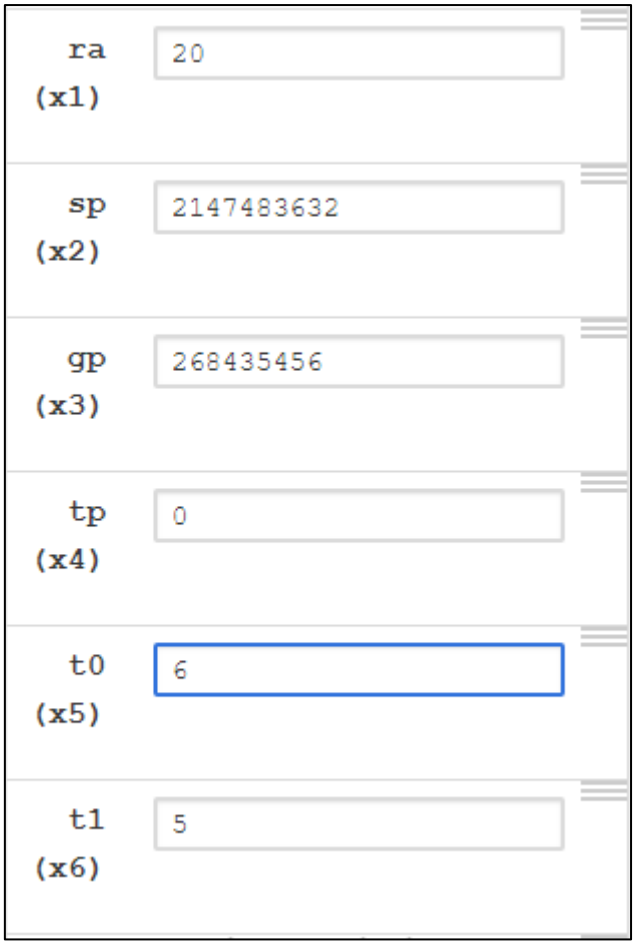

Figure 10: Register File Output in Venus

Address	+0	+1	+2	+3
0x00000018	0	0	0	0
0x00000014	0	0	0	0
0x00000010	-17	-16	95	-1
0x0000000C	3	35	0	0
0x00000008	35	32	80	0
0x00000004	-109	-126	18	0
0x00000000	5	0	0	0
----------	--	--	--	--

Figure 11: Data Memory Output in Venus

The outputs of hardware are following:

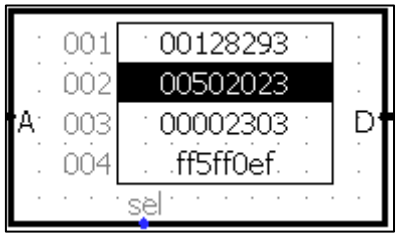

Figure 12: Instruction Memory Output in Logisim

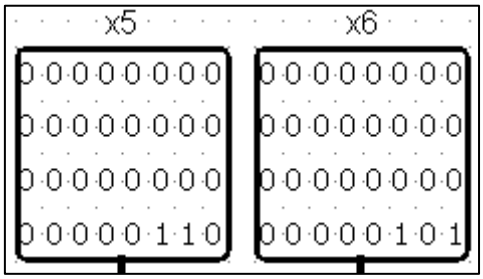

Figure 13: Register File Output in Logisim

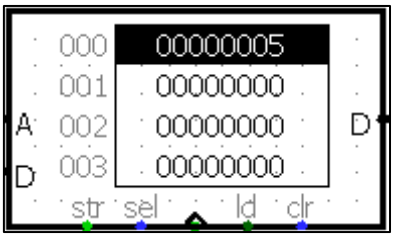

Figure 14: Data Memory Output in Logisim

Conclusion

It is concluded that, software-based instruction execution can be implemented on the hardware. It is also concluded that the synchronous logical circuits and single cycle processors are more efficient as compared to asynchronous logical circuits and pipeline processors. Last but not the least, RV32I architecture is easy to understand, and its implementation is quite feasible to get the knowledge about the working through Venus and Logisim.

Notes

- ✓ To get more information, check out the link below:
  https://github.com/Ammar-Bin-Amir/RISC-V
- ✓ The reason of eliminating the first two bits of the implementation on main system is because the first two bits are always constant that are zero. There are two feasibilities that we can get; firstly, by elimination the first two bits we have to cover these bits from the upper limits that are 12^{th} and 13^{th} bit from which we get more range or double the limit from the previous one, secondly the PC is counting by the offset of 4 whereas the elimination is going to provide the offset of 1 that helps in efficient execution.
- ✓ There are splitters used in the circuit for the buses.
- ✓ There are few multiplexers used that enables a single input to be selected to be transmitter further as an output.
- ✓ There are some pseudo instructions that provides ease in program execution. It is benefited in a way preventing us to type long instructions and are time efficient too.

References

[1] https://merledupk.org/img/core-img/logo.png

[2] https://www.uit.edu/assets/images/newUITLogo47.png