REPORT TITLE:

"RISC-V (RV32I) SINGLE CYCLE"

MADE BY:
SYEDA RAFIA FIZZA NAVEED

INSTRUCTED BY:
ZEESHAN RAFIQUE

❖ ***TABLE OF CONTENTS:***

"RISC-V (RV32I) SINGLE CYCLE"

❖ *INTRODUCTION:*

Over the past decade, the open-source software community has expanded rapidly.The RISC-V Foundation is one notable exception, bringing open-source concepts to the hardware world, aiming to develop an open microprocessor architecture standard that everyone can use without royalty.RISC-V is an open source software standard instruction set architecture (ISA) that has originally designed to support computer architecture research and education.

RISC-V is a universal ISA. It is designed to work well with existing software stacks and languages, to suit all sizes of processor, from the smallest micro-controllers to the largest supercomputers, and to allow extensive specialization for different workloads. To achieve these goals the ISA is designed with modularity and simplicity in mind, so it can be efficiently implemented for all microarchitectures and fabrication technologies.

RISC-V has 32-bits general purpose registers (N=32). Register x0 cannot be written and stores the value 0. Although the other registers can be used to write/read data in a general way, by convention some are used for specific purposes.RISC-V is a three operand machine, in which two source registers and one destination register.

❖ *REQUIRED EQUIPMENTS:*

- Register File
- ALU
- Control Unit
- Control Decode
- Type Decode
- Immediate Generation
- Program Counter/ Instruction Pointer -Memory
- Instruction Memory (ROM)
- Data Memory (RAM)
- Branch Circuit

❖ *METHODOLOGY:*

The methodology that is going to discuss next is based on following a sequence of stages that facilitate the design process. As we know, every processor has 5 stages. These stages are: 1) Fetch ; 2) Decode; 3) Execute; 4) Memory; 5) Write Back.

1) FETCH:

Fetch stage includes a program counter (PC), Instruction Memory and Adder. Program counter or Instruction pointer basically holds the address of the current Instruction. Then, the next part is Instruction memory. Instruction memory holds different instructions than instructions which are executed. Also, the adder part is used to increment the value of the program counter or instruction pointer. So, the program counter points to the next instruction which was fetched from an instruction memory.

2) DECODE:

Decode stage includes immediate generation, control unit and register file. Control unit has its inputs coming from instruction memory in the form of bits from 0 to 6. These bits are basically an opcode of an instruction. In RISC-V (RV32I) we have six types of instructions which are: R-type, I-type, S-type, SB-type, U-type, UJ-type.

	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

The opcode control unit has its signal output. These signals control different mux which are used in the datapath of logisim. The second part is immediate generation in which we have two inputs, first input comes from instruction memory and second input comes from the program counter. Immediate generation basically adds two inputs then sign extend and made compatible for ALU. So, the ALU will be able to compute. Now, we have a 32-bit register file. In this we have 32 registers from x0 to x31. In the register file, rs1 and rs2 are the source register which are coming from instruction memory. And also have to read and write data in a register file.

3) EXECUTE:

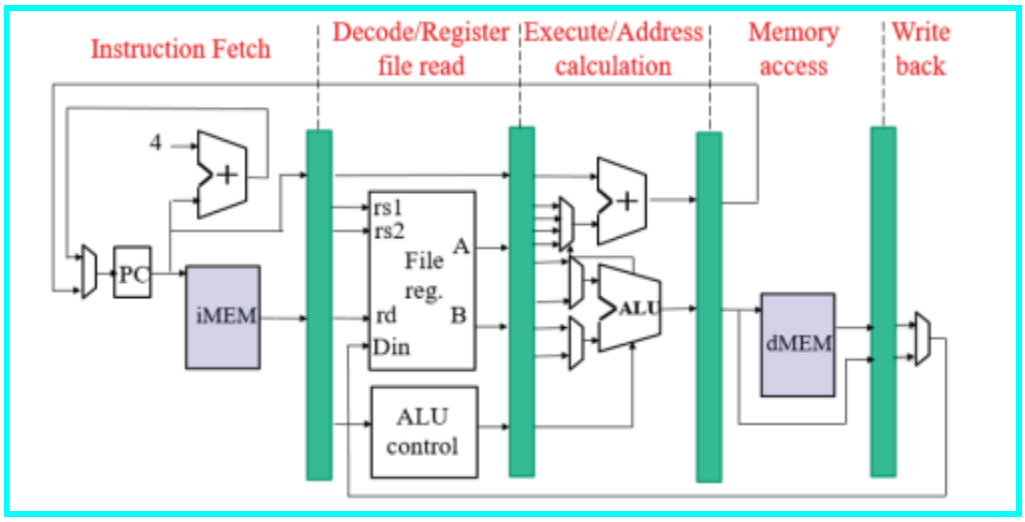
Execute stages include ALU control and ALU. In ALU, basically it consists of three inputs, func3 and func7 are coming from instruction memory. ALU-OP which is used to control signals from control units. Which signals combine and generate 5 bits ALU opcodes which basically test which operation ALU will perform. ALU performs different operations depending upon which instruction is executed. If we have, add instruction and subtract instruction from instruction memory then we will compute the result and the result will be marked in ALU result output. And, If we have branch instruction and branch is true, so the ALU result branch is high.

4) MEMORY:

If Instruction memory has a load and store instruction or wants to load any data from memory and store it, use data memory. Data memory signal is controlled by a control unit, which shows as a mem-to-reg and mem-to-write signal.

5) WRITE BACK:

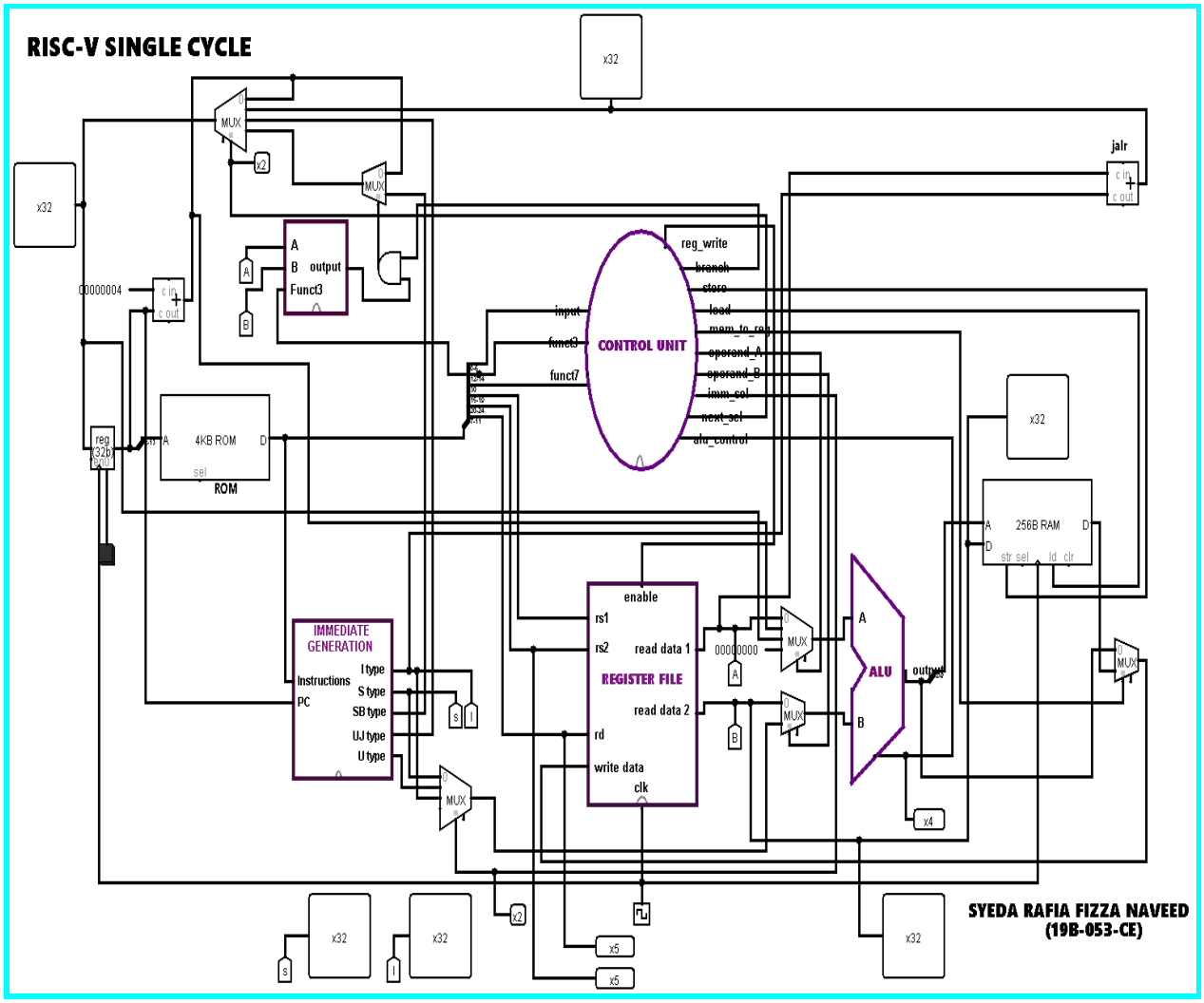
The final stage is writing back. To write any ALU result to load data from memory use the write back stage.



❖ *IMPLEMENTATION:*

Construct the simple single cycle RISC-V (RV32I) processor on logisim,by using 32 bit register file, control unit, immediate generation, ALU and use a Logisim ROM for the instruction memory and a Logisim RAM for the data memory.Firstly, create the 32 bit register file,this register file takes 5-bit address to select one of the 32 registers and write data to it using register enable wire.Secondly, make 32 Bit ALU with 4-bit having operations of ADD,SUB,AND,OR,XOR,slt,sltu,srl,sll,sra,DB which selects which operation to perform according to the instruction.Then make a control unit with the combination of type decode and control decode. Also create additional subcircuits, such as branches, jalr and the sign extension and shift 2, to keep your main circuit manageable and connected these components using wires and tunnel.And then provided register file uses the clock of logisim, also need to connect the PC and Data Memory (RAM) to the clock.

❖ *CIRCUIT DIAGRAM:*



❖ ***RESULTS:***

● ***VERIFICATION:***

As the simulation verification is very low-level, requiring manipulation of machine code and access to arbitrary memory locations, only system programming languages could be used. I have assessed C, C++ and Rust, and eventually decided to use C++.

C++ is the only choice remaining. I have used C++ for a few years already, and I consider myself reasonably experienced in C++, so choosing C++ is less risky than a new language.

C++'s high-level paradigms can make algorithm implementations easier and more readable, while it also has low-level abilities such as using C libraries, performing signal handling, and accessing arbitrary memory. Overall I believe C++ is the best fit for the project.

● ***TEST PROGRAMS:***

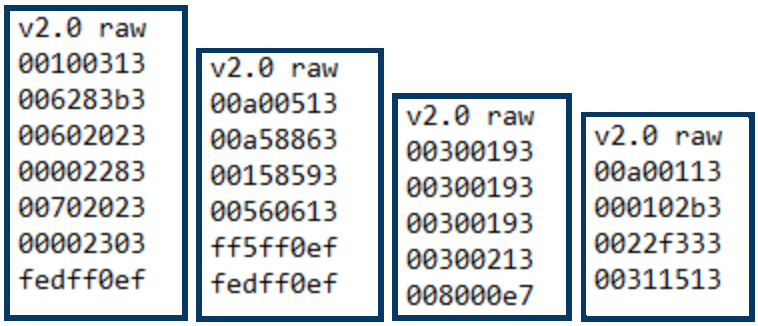
```
# FABONACCI SERIES PROGRAM CODE
addi x6,x0,1
up:
    add x7,x5,x6
    sw x6,0x0(x0)
    lw x5,0x0(x0)
    sw x7,0x0(x0)
    lw x6,0x0(x0)
    jal up
```

```
# PROGRAM CODE: TABLE OF 5
main:
addi x10,x0,10
loop:
beq x11,x10,exit
addi x11,x11,1
addi x12,x12,5
jal loop
exit:
jal main
```

```
addi x3,x0,3
addi x3,x0,3
addi x3,x0,3
addi x4,x0,3
jalr ra,x0,8
```

```
addi x2,x0,10
add x5,x2,x0
and x6,x5,x2
slli x10,x2,3
```

- ***HEX CODES:***



❖ *CONCLUSION:*

A 32-bit RISC-V (RV32I) instruction set, was designed and simulated. RISC-V now provides many interesting new possibilities when designing open cores, as it is open-source and completely free to use. A methodology for the design of RISC-V (RV32I) processors has been described. The objective of this methodology is to facilitate the development of processors by applying a modular design strategy.

❖ *FUTURE WORK:*

As RISC-V keeps changing rapidly, it would be interesting to see what features are implemented, and how RISC-V is able to compete with the more traditional ISAs. The RISC-V (RV32I) can also be expanded upon, implementing more instructions.

❖ *REFERENCES:*

- David A. Patterson and John L.Hennessy: "Computer Organization and Design" (1st ed.). Morgan Kaufmann,1994.
- "The RISC-V Instruction Set Manual. Volume I: User Level ISA.
- ISA MANUAL
- RISC V BOOK