



RISC-V Processor Implementation

-Gaurav Kulkarni

INTRODUCTION

RISC-V is an open-source Instruction Set Architecture (ISA).

Open-source ISAs can lower the hardware costs and enable scalability by allowing community to include custom instructions that can simplify software.

ISAs comes on two broad categories, Complex Instruction Set Computers (CISC) and Reduced Instruction Set Computers (RISC).

CISC instruction can perform many complex tasks while single RISC instruction can perform only single simple tasks.

RISC-V is much simpler instruction set specifically designed to simplify CPU design.

The instruction set followed here is the RV32I

Von Neumann architecture

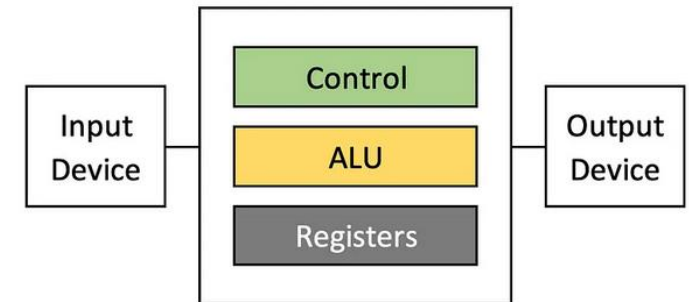
Arithmetic and Logic Unit (ALU)

that performs mathematical operations like add, subtract as well as logical operations to check if two numbers are equal, less than or greater than and so on.

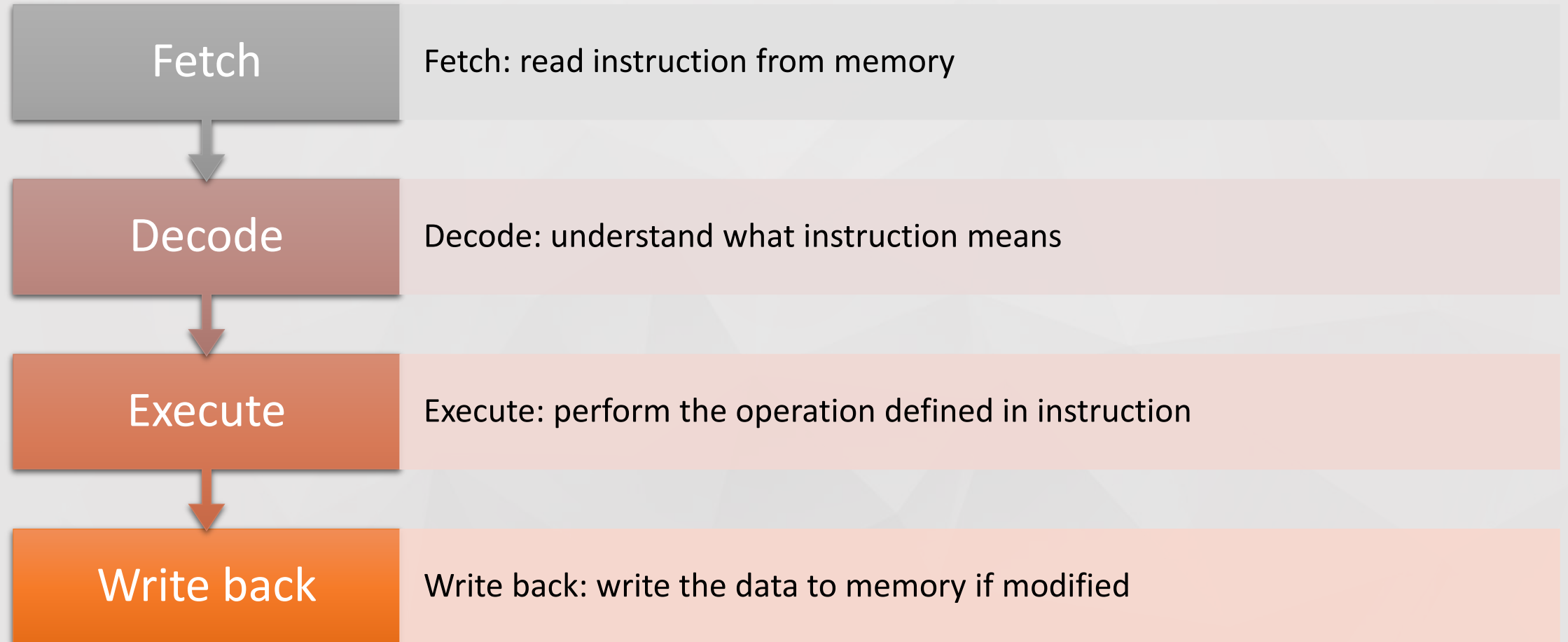
Memory management unit (MMU)

that stores instructions and data based on which the program operates. Input and output devices (IO) that reads instructions and stores the updates back to Memory.

Control unit (CU) that directs and orchestrates execution of the instruction and program including data flows. Control unit governs single cycle or multi-cycle pipeline and normally associated with performance of the CPU.

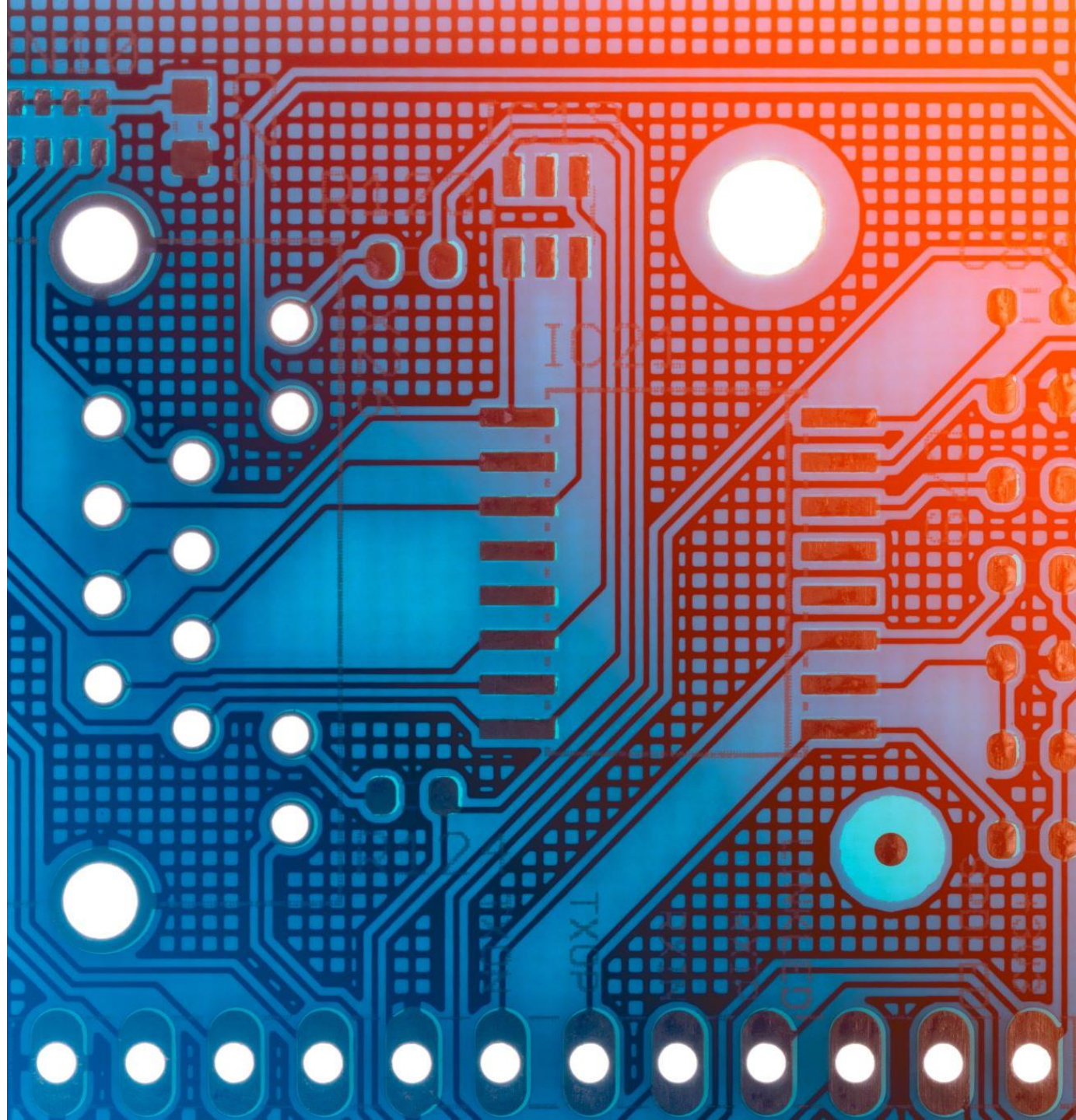


Implementation Steps of RISC-V



Blocks in a RISC-V processor

- Program Counter
- Instruction Set
- Instruction Decoder
- Control Enable
- ALU
- Register File
- Data Memory
- MUXs




Program Counter

- The output of the program counter is the address or index of the instruction set.
- After every execution of an instruction, the program counter increments the address to execute the next instruction.
- In case of branch instructions, the program counter may have to increment by a specific value which is determined by the instruction.
- On activating reset, the program counter goes to its initial value.

A large orange circle is positioned on the left side of the slide, partially cut off by the edge.

Instruction set

- The instruction set contains a list of all the instructions to be executed.
 - The program counter gives the address of the instruction to be decoded.
 - A block RAM has been used and it has 1 clock cycle latency for read and write operations
- 
- A series of four yellow dashed line segments are arranged in a curved, upward-sloping pattern in the bottom right corner of the slide.

Instruction decoder

- The instruction decoder takes the instruction from the instruction set and decodes it based on the opcode.
- The opcode is the least significant 7 bits of the 32 bit instruction.
- Accordingly, the instruction is split into various parts and sent to other parts of the processor.
- This is operated on the negative edge of the clock pulse.

Register File and Data Memory

- The register file and data memory are both RAMs.
- The register file has 2 read ports and 1 write ports.
- The data memory has 1 read port and 1 write port.
- They have read enable and write enable pins, controlled by the instruction decoder.
- Block RAMs are used for each.

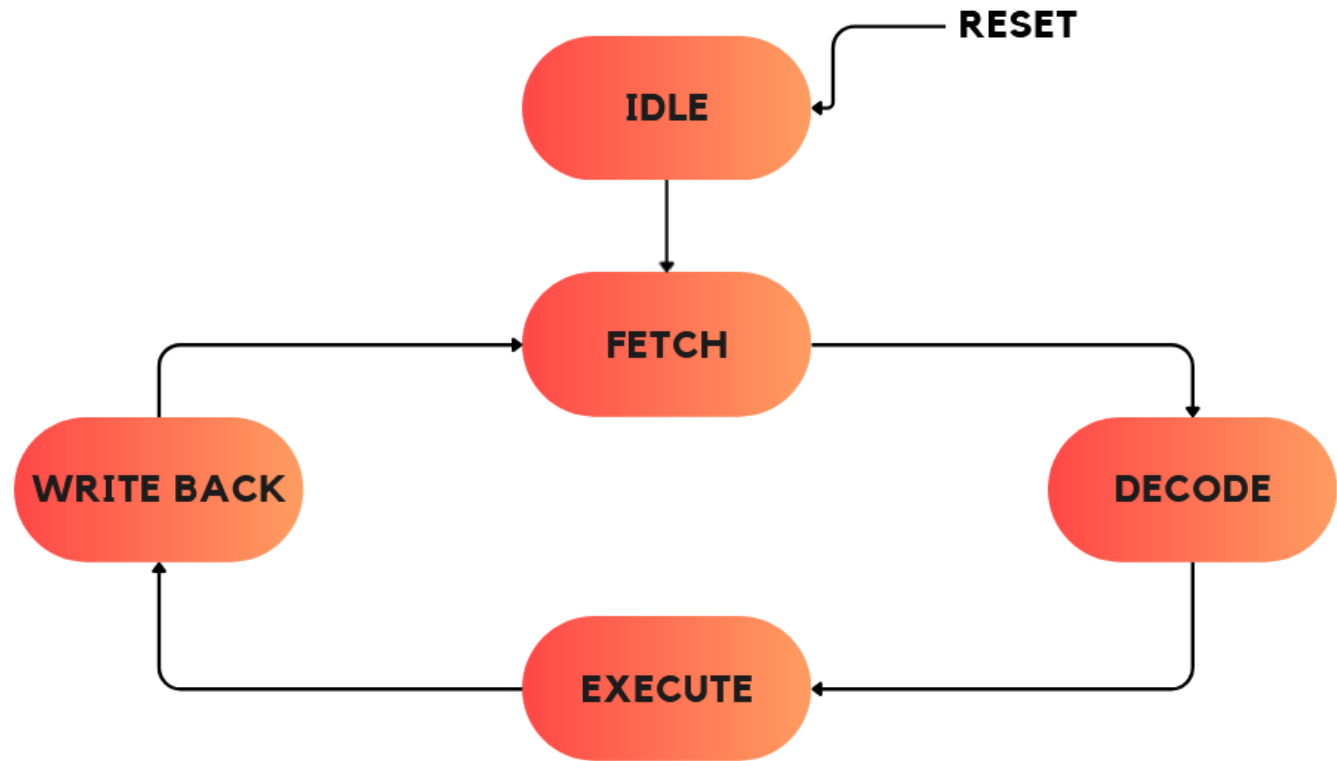
Arithmetic Logic unit

- The ALU performs the required operation based on the instructions sent from the instruction decoder.
- The instruction decoder determines exactly which operation needs to be executed.
- One of the inputs comes directly from the register file, while the other input may come from instruction or from the register file.

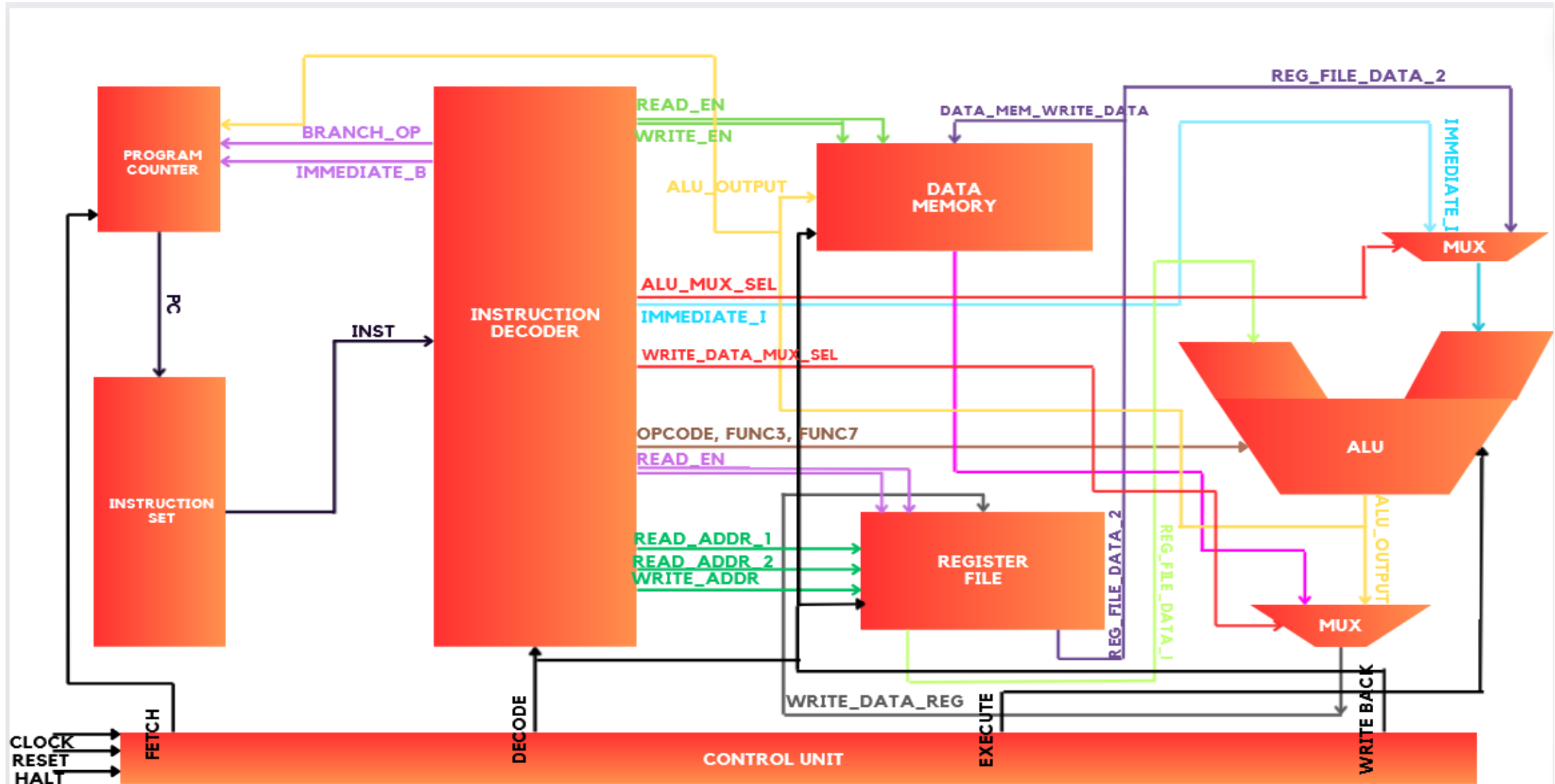
Control Enable

- The control unit is the backbone of the RISC-V processor.
- The control unit is responsible for enabling the required block during the respective cycles.
- The control states are as follows
 1. Fetch: Program Counter, Instruction set
 2. Decode: Instruction Decoder, Register file, Data memory
 3. Execute: ALU
 4. Write Back: Register File, Data memory

State Machine



Block Diagram



Executable Operations

- Integer Register-Register operations (R-type)
- Integer Register-Immediate operation (I-type)
- Load (I-type) and Store (S-type) instructions
- Conditional Branch instructions



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
funct 7							rs 2					rs 1					funct3			rd				Opcode							

R-type Operation

- The instruction contains the addresses of three registers in the register file i.e. rs1, rs2 and rd.
- Values from registers rs1 and rs2 are sent to the ALU.
- Arithmetic operation is performed on them based on the data in funct3 and funct7.
- The result is written in the register with address rd of the register file.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Imm												rs 1			funct3			rd				Opcode									

I-type Operations

- The instruction contains the addresses of two registers of the register file i.e rs1 and rd.
- The instruction also contains a 12-bit immediate value which is sign extended and sent to ALU along with the value from rs1.
- ALU performs arithmetic operation based on the data in funct3.
- The result is written in the register file at address rd.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Imm												rs 1			funct3			rd				Opcode									

Load instruction

- The instruction contains the addresses of two registers in the register file i.e rs1 and rd.
- The instructions also contains a 12-bit immediate value which is sent to the ALU along with the value in register rs1.
- The result is the address of a register in the data memory.
- The value from the data memory is loaded into the register file in the register with address rd.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm[11:5]							rs 2					rs 1					funct3			imm[4:0]					Opcode						


Store Instruction

- The instruction contains the addresses of two registers of the register file called rs1 and rs2.
- The ALU performs arithmetic operations on the data in rs1 and the immediate value in the instruction, based on the funct3.
- The result is the address of a register in the data memory.
- The data from register rs2 of register file is written in the data memory.

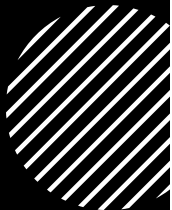

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
imm[12,10:5]						rs 2						rs 1						funct3	imm[4:1,11]						Opcode						

Branch Instruction

- The instruction contains the addresses of two registers of the register file called rs1 and rs2.
- The ALU performs operations on the data stored in rs1 and rs2 based on the value of funct3.
- If the ALU results satisfies a given condition, the program counter is increased by the immediate value given in the instruction.



Design and Implementation Flow (FPGA)



RTL design – Writing Verilog Code as per Specs



Simulation – Writing testbench for Sanity Test and debugging issue



Synthesis – Synthesis for the FPGA and Checking on Design optimization



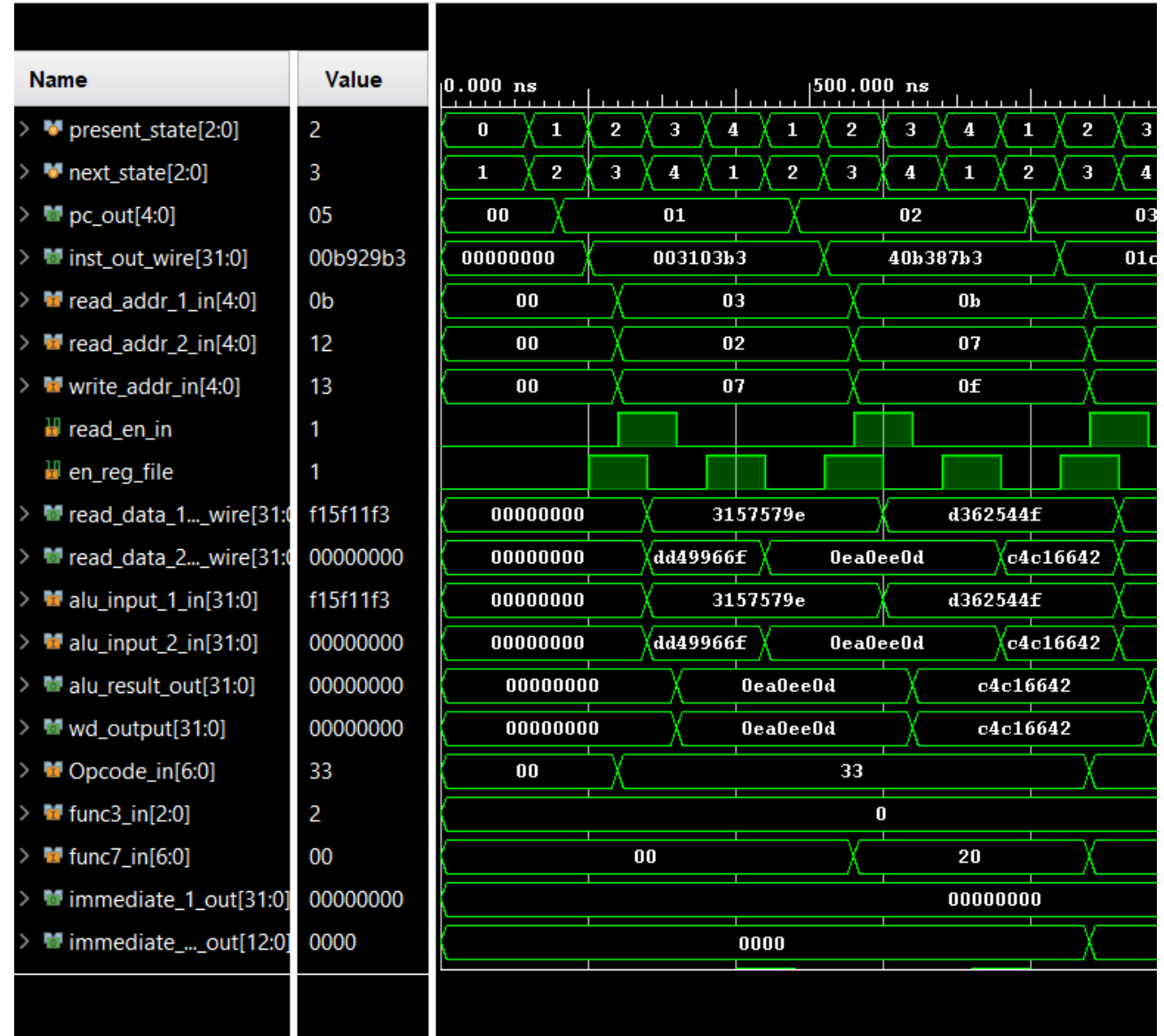
Implementation – FPGA utilization, Timing analysis, I/O layout, etc.



Bitstream generation and hardware testing, Validation

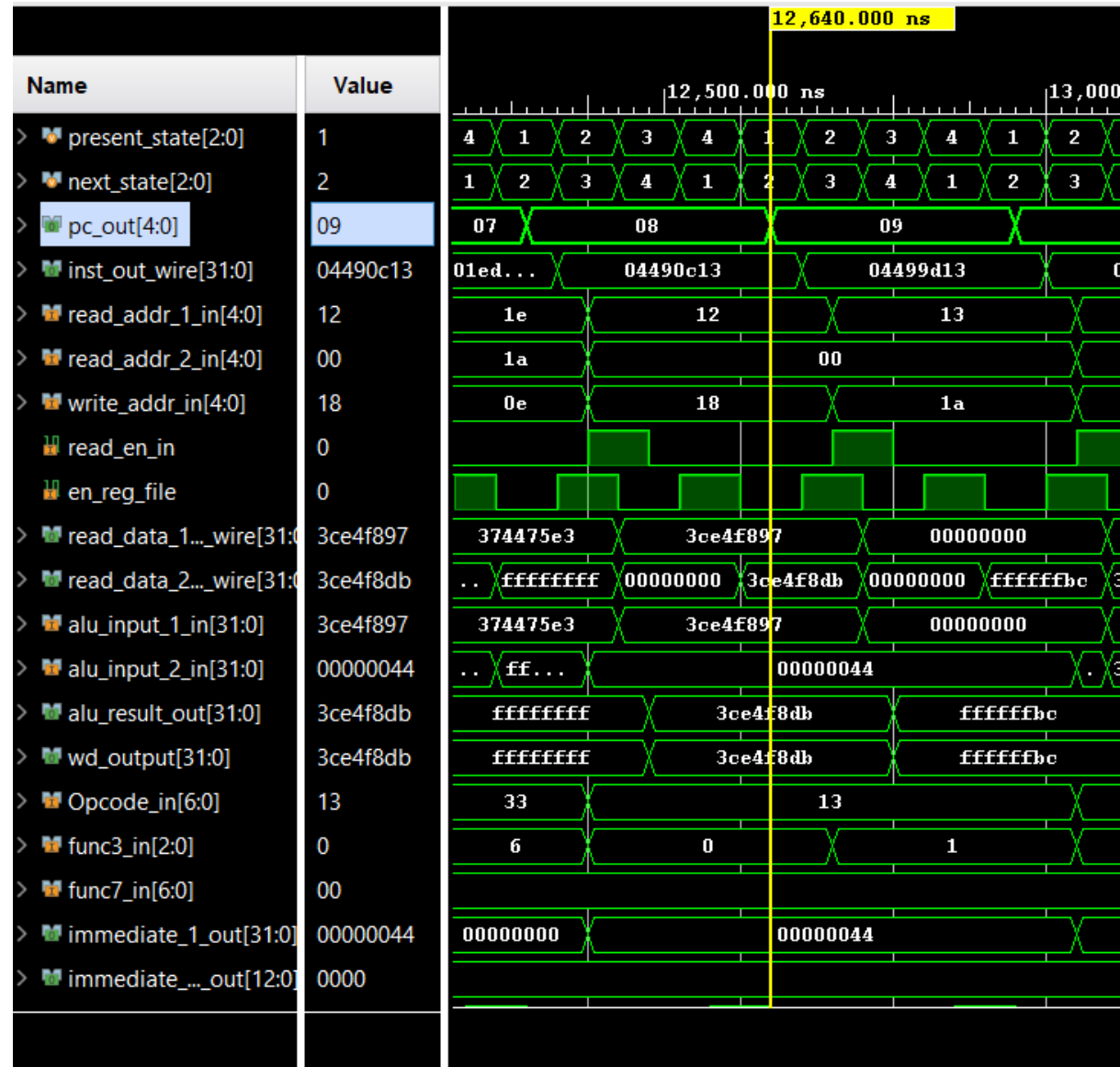
Waveforms

Register Arithmetic
operation
Opcode – 33(hex)



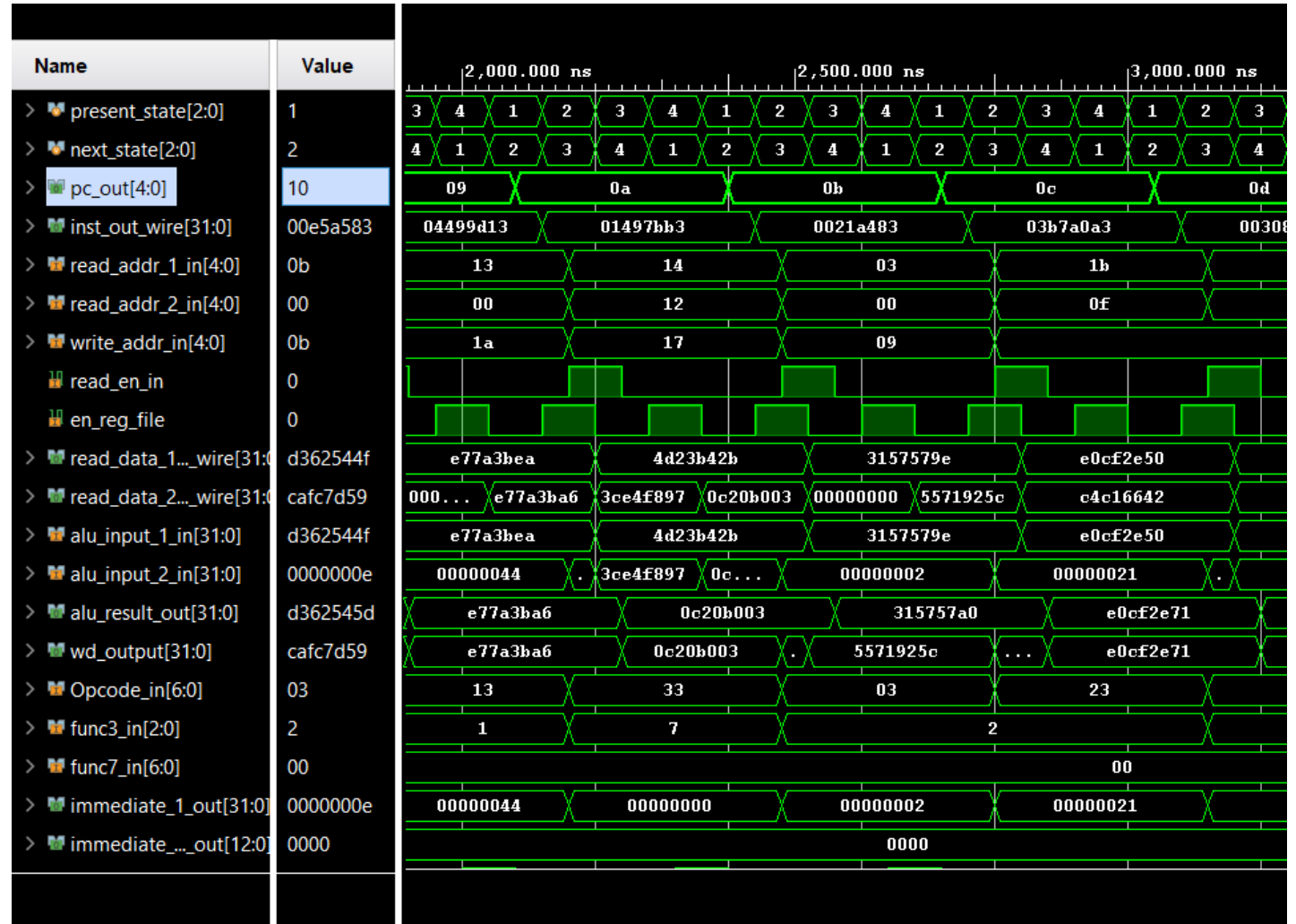
Waveforms

Register Immediate
operation
Opcode – 13(hex)



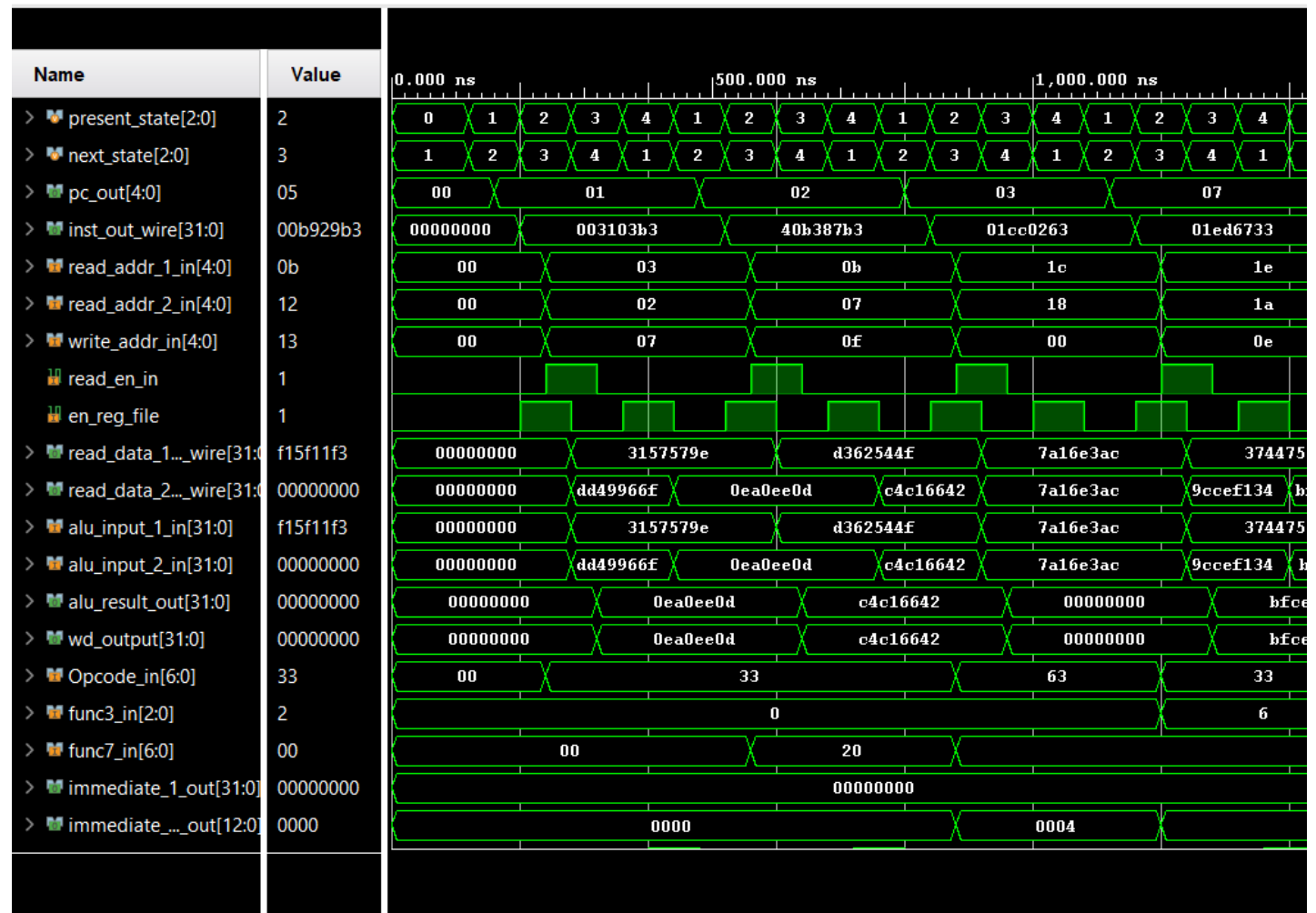
Waveforms

Load and store operations:
Opcodes – 03(hex)
and 23(hex)
respectively

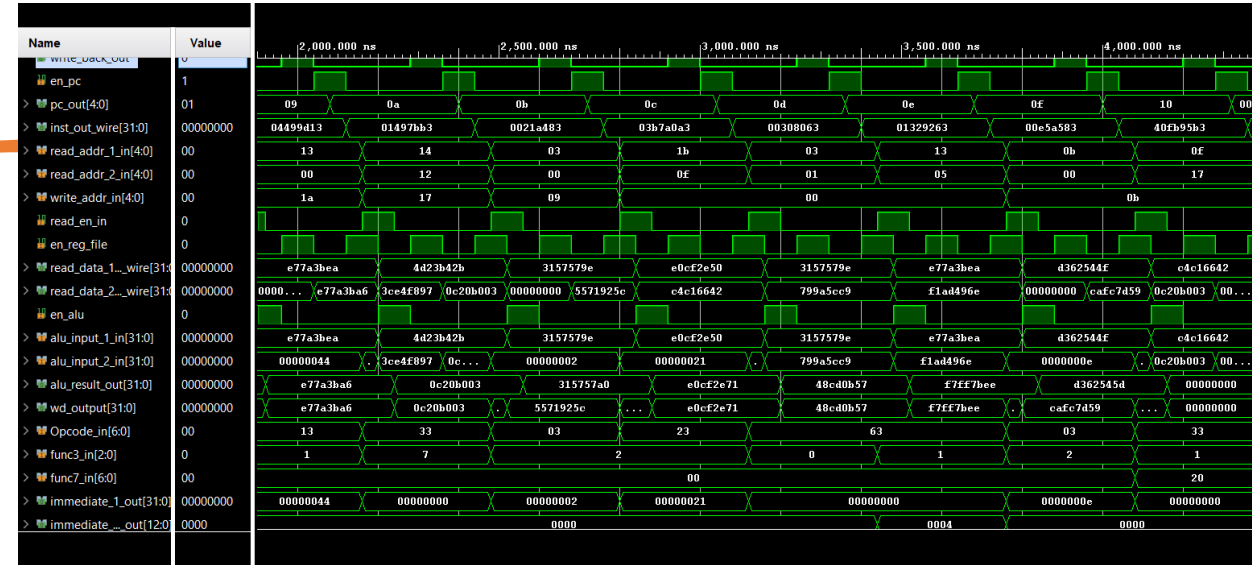
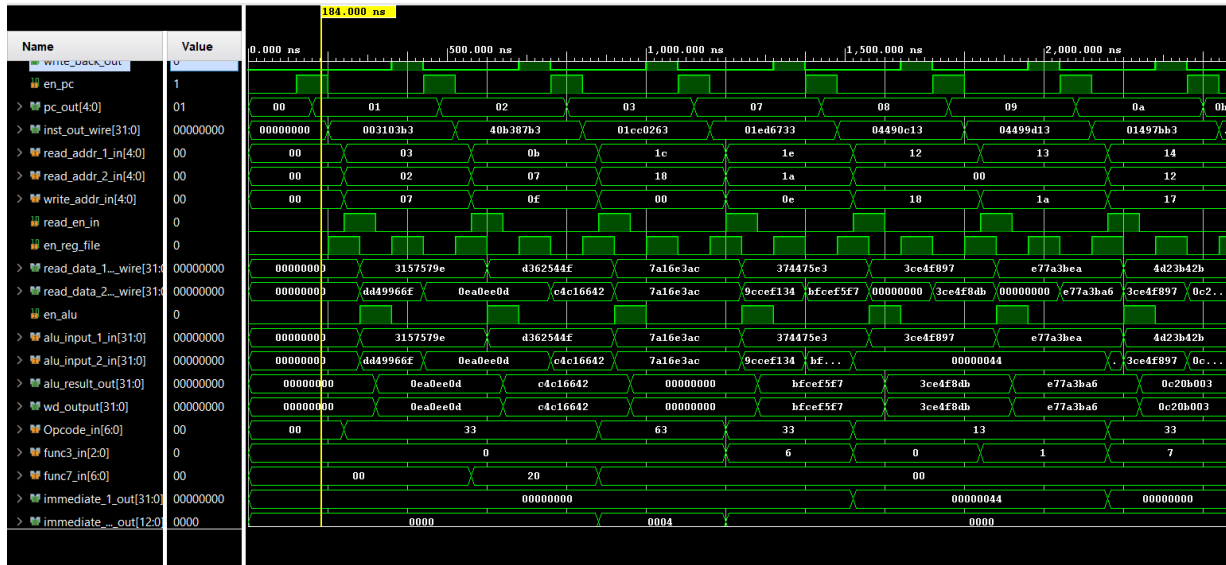


Waveforms

Branch operation
Opcode – 63(hex)

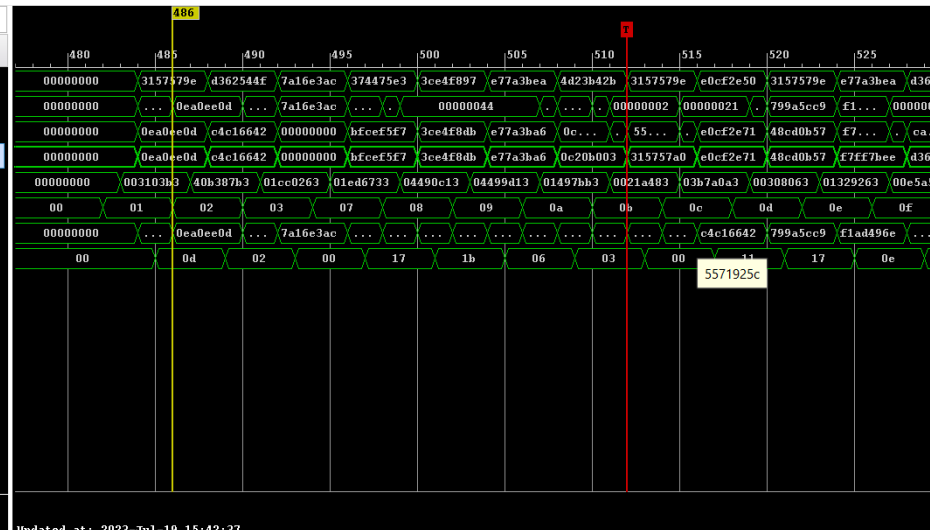


Simulation and hardware results

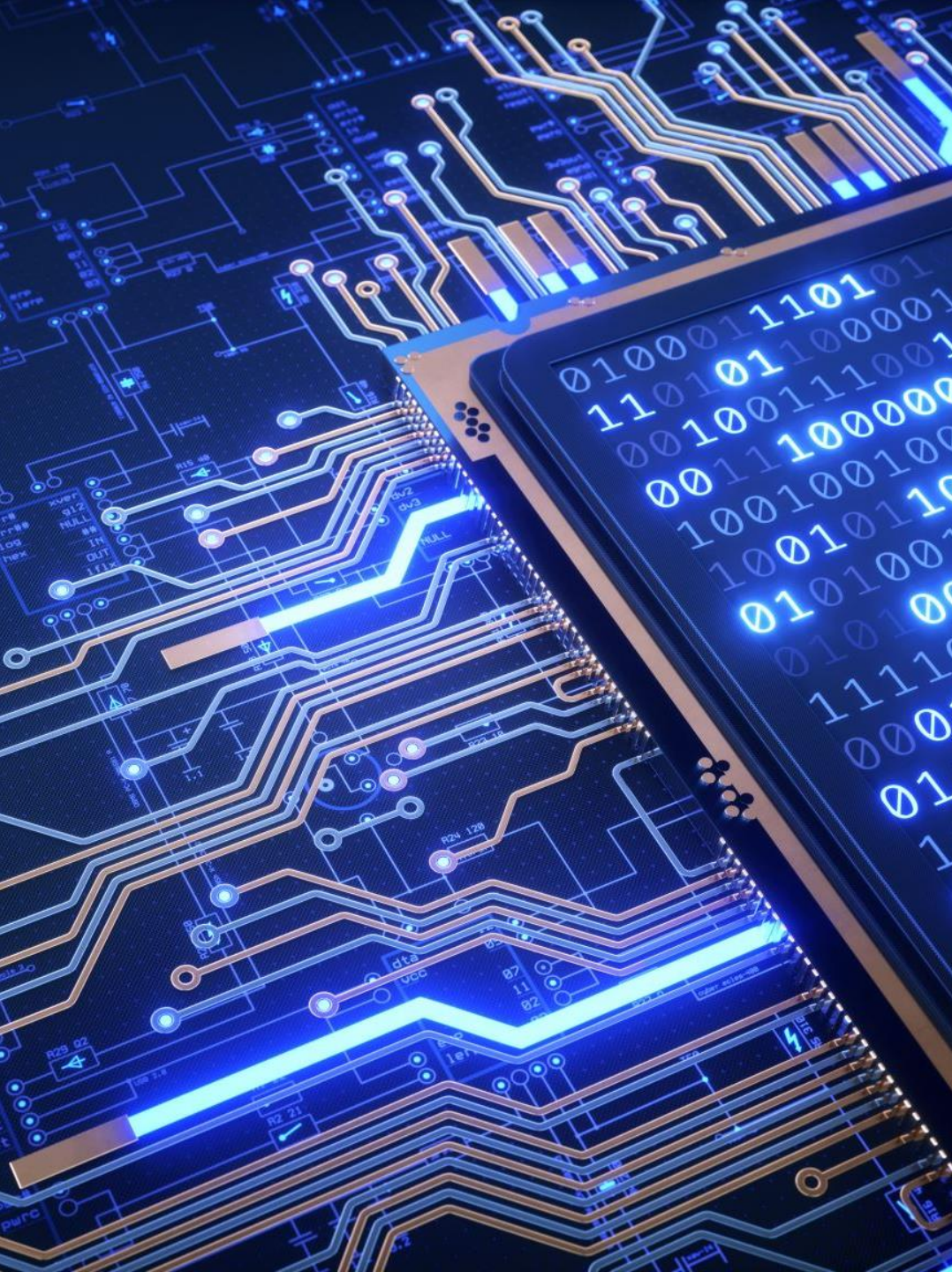


ILA Status: Idle

Name	Value
alu_input_1[31:0]	3157579e
alu_input_2[31:0]	0ea0ee0d
wd_mux_output[31:0]	0ea0ee0d
alu_result[31:0]	0ea0ee0d
instruction_out[31:0]	003103b3
pc_out[4:0]	02
read_data_reg_file_2_out[31:0]	0ea0ee0d
write_addr_data_mem_out[4:0]	0d



Updated at: 2023-Jul-19 15:42:37



Conclusion

- A four stage non-pipelined RISC-V processor was designed and implemented on FPGA.
- A total of 24 instructions based on the instruction set RV32I has been implemented.
- The instructions contains Arithmetic, Immediate, Load, Store and branch operations
- Based on the simulation and hardware test results, the RISC processor is functional.

Further improvements

- Converting the processor from a Non-pipelined to a Pipelined form to improve performance.
- Adding new instructions to improve its usability.

