Khulna University of Engineering & Technology

Department of Computer Science and Engineering

Course No: CSE 2114

Course Name: Computer Architecture and Organization

Designing a 25-bit CPU

In Logisim

Submitted by

Utsa Roy

Roll: 2207027

Group: A1

Submission Date:

ABSTRACT

This project documents the design, simulation, and implementation of a complete digital mini computer system, using the Logisim tool. The primary objective is to demonstrate a practical understanding of fundamental computer architecture principles by constructing a functional 25-bit computer from the ground up. The architecture features a custom Instruction Set Architecture (ISA), a hardwired control unit, an Arithmetic Logic Unit (ALU) capable of arithmetic and logical operations, and a register set including an accumulator, program counter, and memory registers, all interconnected via a common bus system.

OBJECTIVES

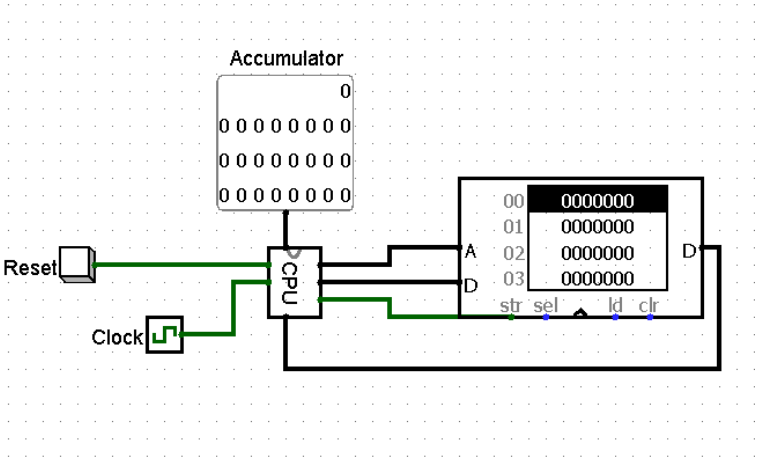
- ✓ To develop a thorough understanding of essential CPU components, the memory interface, and the overall operational flow of a processor.
- ✓ To explore the concept of instruction sets and analyze their impact on CPU architecture and performance.
- ✓ To efficiently encode instructions within a constrained bit-length using various encoding schemes and methodologies.
- ✓ To enhance problem-solving skills by debugging and optimizing simple CPU simulations and more complex digital systems.

INTRODUCTION

A **Central Processing Unit (CPU)** is the core of any computing system, responsible for executing instructions and performing arithmetic and logical operations. In this project, we designed and implemented a **25-bit CPU in Logisim**, consisting of multiple essential components:

1. **Arithmetic Logic Unit (ALU)** – Performs arithmetic and logical operations on 25-bit operands.
2. **Registers** – Temporary storage locations for instruction processing.
3. **Memory** – Stores instructions and data for execution.
4. **Control Unit** – Decodes and executes instructions by generating control signals.

Each of these components plays a vital role in processing data and managing instruction execution efficiently.



The **CPU design** includes:

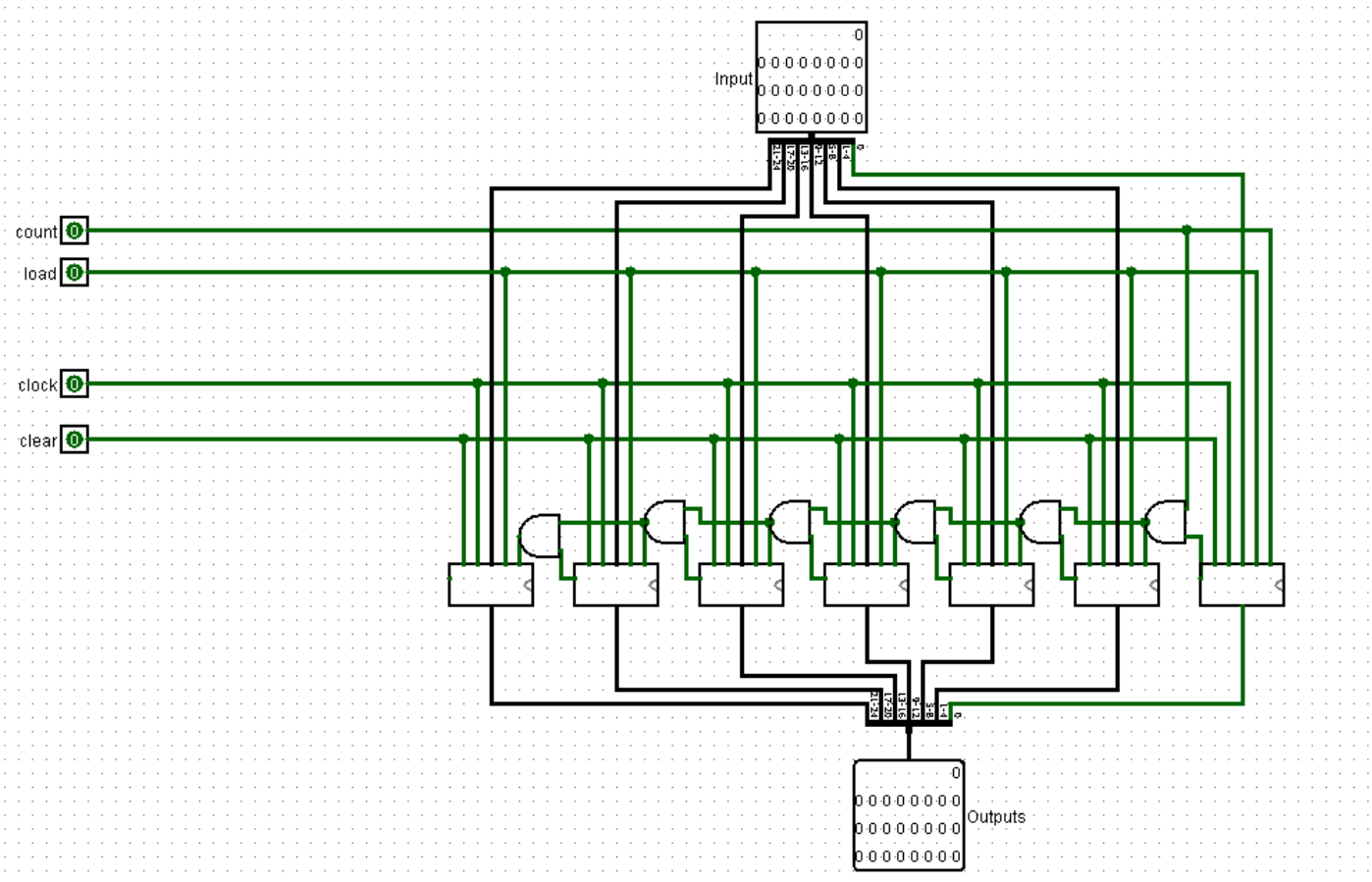
- A **25-bit data bus and address bus** for handling memory and computation.
- **Registers** such as o **MBR (Memory Buffer Register)** o **MAR (Memory Address Register)**
  - **PC (Program Counter)**
  - **IR (Instruction Register)**
  - **Load Register with Counter**
  - **PLR (Parallel Load Register)**
- An **ALU** capable of performing fundamental operations.
- A **control unit** that processes and executes instructions using predefined control signals.

REGISTERS IN THE CPU

Registers are essential components of a CPU that temporarily store data and control information during execution. These registers facilitate the movement of data between memory and the processing units, making them crucial for fast and efficient computation.

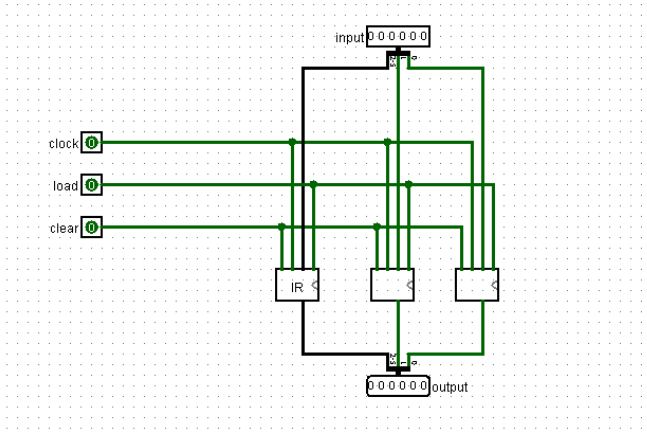
25 Bit Memory Buffer Register (MBR):

The **MBR** temporarily holds data fetched from memory before it is processed by the ALU or stored back into memory. It serves as an intermediary between memory and the CPU, allowing smooth data transfer.



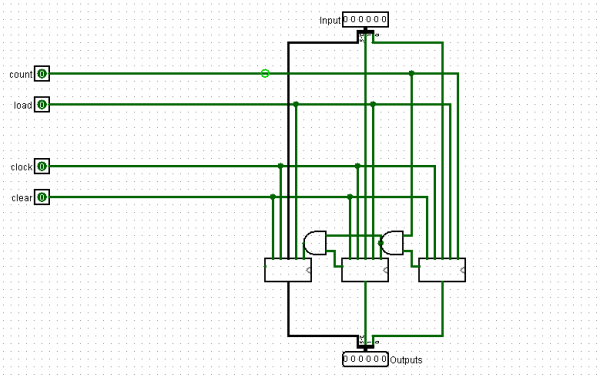
6 Bit Memory Address Register (MAR):

The **MAR** stores the address of the memory location the CPU wants to access, either for reading data into the CPU or writing results back to memory. It is responsible for sending memory addresses during instruction execution, ensuring that the right data is fetched or stored.



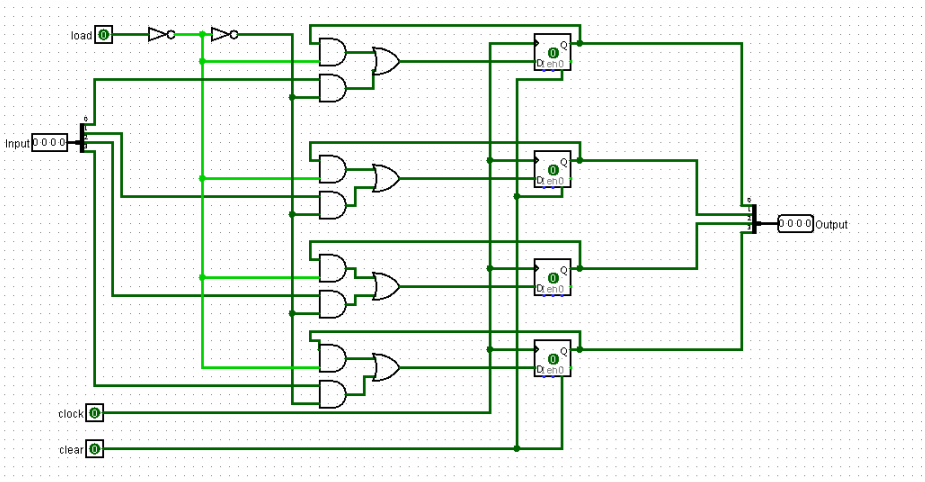
Program Counter (PC):

The **PC** stores the address of the next instruction to be executed. It automatically increments after each instruction, ensuring the program progresses through its instructions. However, the PC can be altered during jump or branch operations to modify the program's flow.

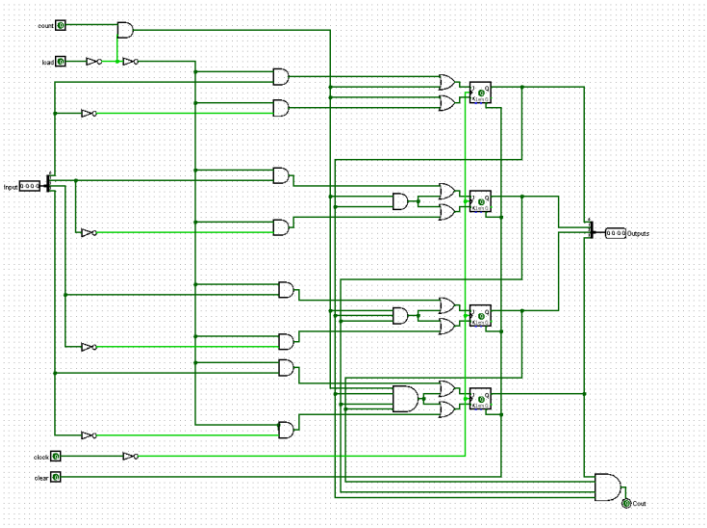


4 Bit Instruction Register (IR):

The **IR** holds the current instruction being executed. After fetching an instruction from memory, the control unit decodes it to determine what operation needs to be performed, such as a data transfer, arithmetic operation, or jump.

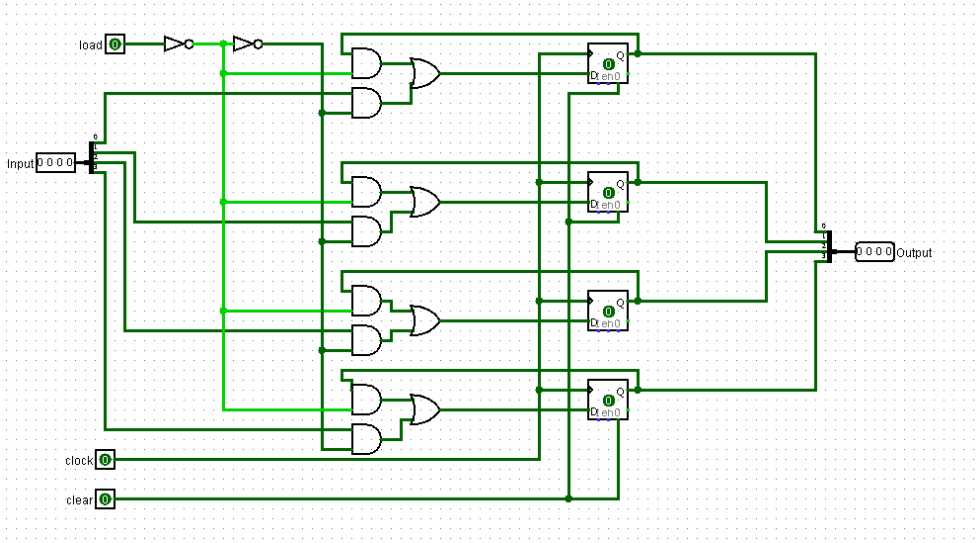


Parallel Load Register with Counter:



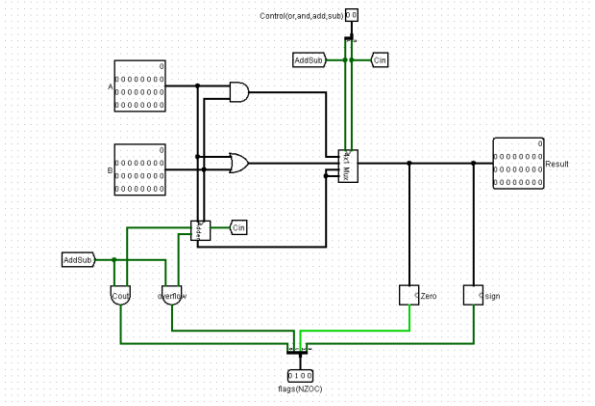
Parallel Load Register (PLR):

The **PLR** stores the required privilege level for a given process. It is used to ensure that a process does not exceed its assigned privilege level, enforcing access control policies and securing sensitive operations.



Arithmetic Logic Unit (ALU)

The **ALU** is responsible for performing fundamental mathematical and logical operations. In this **25-bit CPU**, the ALU supports a variety of operations that form the core of most programs.



INSTRUCTION SET

The instruction set consists of 25-bit.

24	23 20	19 6	5 0
0	Opcode	Not used	Address
0	XXXX	XXX…XX	XXXXXX

The bits that are given 24th and 5-19th are the unused bits. These are kept unused to read/write the instructions only by seeing the hex values.

Opcode is of 4 bits, so there are $2^4 = 16$ instructions.

Opcode	Hex Value	Mnemonic	Definition
0000	0	AND	ACC <- ACC & M
0001	1	ADD	ACC <- ACC + M
0002	2	STO	M <- ACC
0003	3	BUN	PC <- M
0004	4	BSB	M <- PC, PC <- M+1
0005	5	LOAD	ACC <- M
0006	6	ISZ	M <- M+1 and PC++ if M=0
0007	7	JZ	JUMP if ACC = 0
0008	8	PUSH	ACC <- Stack
0009	9	POP	Stack -> ACC
000a	A	HALT	Stop Execution
000b	B	NEG	ACC <- (-ACC)
000c	C	MUL	ACC <- M x ACC
000d	D	reserved	N/A
000e	E	reserved	N/A
000f	F	JN	Jump if ACC is Neg

CONTROL UNIT

The **Control Unit (CU)** is responsible for decoding instructions and generating signals to direct data movement between different components. It follows a **finite state machine (FSM) approach**, ensuring that each instruction is executed in multiple steps.

CONTROL UNIT AND INSTRUCTION EXECUTION PROCESS

The execution of an instruction follows a structured process, ensuring smooth operation.

Instruction Fetch:

The **Program Counter (PC)** holds the memory address of the next instruction. This address is sent to the **Memory Address Register (MAR)**, and the instruction is fetched into the **Instruction Register (IR)**.

Instruction Decode:

The **Control Unit (CU)** decodes the instruction by identifying the **opcode, source registers, and destination registers**. Based on this decoding, the CU determines which operation needs to be performed.

Instruction Execution:

If the instruction involves arithmetic or logical operations, the **ALU** performs the required computation. For memory-related instructions, the **LOAD/STORE** mechanism handles data transfer between registers and memory. If the instruction is **ISZ**, the selected register is incremented, and if the result is zero, the next instruction is skipped.

Write Back and PC Update:

After execution, the result is written back to the appropriate register or memory location. The **Program Counter (PC)** is updated to point to the next instruction, ensuring continuous Execution.

Sample Code to Run on RAM

The following raw machine code represents a simple program stored in **RAM**, which executes basic operations:

v2.0 raw
0600004 0100005 0200008 0700000 0000009 0000003

Explanation of Instructions (LOAD, ADD, STORE)

- **0600004** → **LOAD** the value from memory address 000004 into the **Accumulator.**
- **0100005** → **ADD** the value from memory address 000005 to the **Accumulator**.
- **0200008** → **STORE** the accumulated result into memory address 000008.
- **0700000** → **JZ** (Jump to address 000000 if the last operation resulted in zero).
- **0000009** → Represents data stored in memory.
- **0000003** → Represents additional memory data.

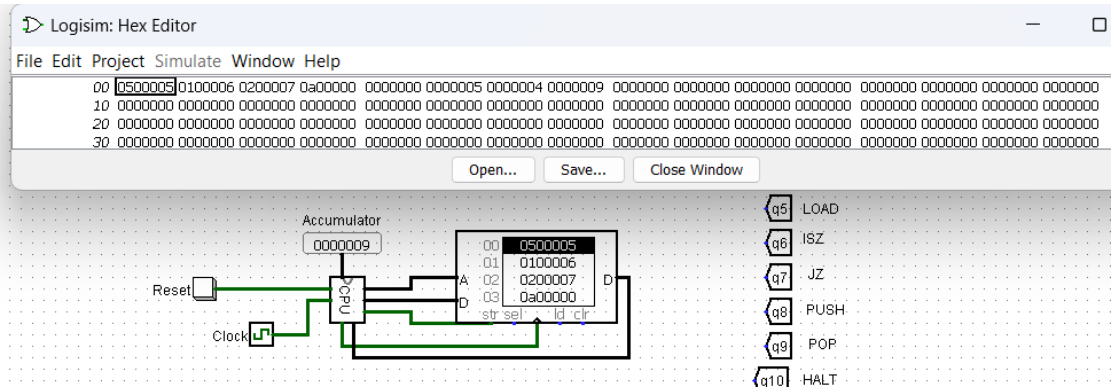
This sequence **loads data, performs an addition, stores the result, and halts execution**.

v2.0 raw
0500005 0100006 0200007 0a00000 0000000 0000005 0000004

Explanation of Instructions (ISZ)

- **0500004** → **LOAD** the value from memory address 000004 into the **Accumulator**.
- **0100005** → **ADD** the value from memory address 000005 to the **Accumulator**.
- **0200008** → **STORE** the accumulated result into memory address 000007.
- **0a00000** → **Halt** the execution.
- **0000000** → No operation.
- **0000005** → Represents data stored in memory.
- **0000004** → Represents additional memory data.

This sequence **loads data, performs an addition, stores the result, and halts execution**.



DISCUSSION

Through this project, we explored the **fundamental principles of computer architecture** by designing a functional **25-bit CPU**. The step-by-step implementation in **Logisim** demonstrated how different CPU components work together to execute instructions.

One of the **key challenges** was ensuring proper control signal synchronization between the **ALU, registers, and memory**. Debugging instruction execution required **tracing data flow through the registers and verifying control unit outputs**.

The **AND operation** was successfully implemented, demonstrating how logical operations can be executed efficiently within an ALU. Similarly, the **LOAD and STORE** instructions confirmed the proper working of memory transfers.

This project provided deeper insights into **instruction cycle execution, CPU design, and digital logic operations**, reinforcing theoretical concepts with practical application.

CONCLUSION

The implementation of a **25-bit computer** in Logisim provided valuable hands-on experience in **processor architecture and digital system design**. This project enhanced understanding of:

- **Instruction execution cycles** and how a CPU processes data.
- **Register and memory interactions** in a computing system.
- **ALU functionality**, including arithmetic and logical operations.
- **Control unit logic**, responsible for orchestrating CPU processes.

Overall, the project successfully demonstrated a working model of a **25-bit CPU**, highlighting the importance of structured data flow and control logic in computer architecture.

REFERENCES:

- **Logisim Documentation**
- **Class Lecture and Slide**