ADDIS ABABA UNIVERSITY

College Of Technology and Built Environment

School of Electrical and Computer Engineering

HEDA-8: High-Efficiency Digital Architecture
8-Bit CPU Data path Design & Simulation Project

Submitted by:

Name	ID
1.Nahom Wondale	UGR/7423/15
2. Hawi Degu	UGR/2321/15
3. Kaleab Nigusiie	UGR/1840/15
4. Eyob Wondimu	UGR/8014/15
5. Betsebay Getahun	UGR/9333/15

Submitted to:
Daniel D.

Dec/2025

Abstract

This report documents the design, implementation, and simulation of the **HEDA-8 (High-Efficiency Digital Architecture)**, an 8-bit CPU datapath constructed in Logisim. The objective was to design a functional computer architecture capable of arithmetic, logic, and memory operations without the use of a physical Control Unit.

The system features a custom 8-bit Arithmetic Logic Unit (ALU) capable of 10 distinct operations, a unified vertical 8-bit Common Bus, a Register File consisting of four General Purpose Registers (A, B, C, D) and four Special Purpose Registers (PC, IR, MAR, MBR), and a fully integrated 256-byte Random Access Memory (RAM) module.

All control signals are simulated via manual switching to demonstrate the Fetch-Decode-Execute cycle. The final simulation successfully verifies data transfer, arithmetic processing, and memory storage, adhering to all project requirements.

Table of Contents

1. Introduction

The Central Processing Unit (CPU) is the brain of modern computing, executing instructions through a coordinated series of data transfers and logical operations. The HEDA-8 project aims to demystify this process by constructing a CPU from the ground up, focusing specifically on the **Data path**—the hardware execution units—while manually simulating the control signals that govern them.

1.1 Project Objectives

The primary goals of this design were:

- To implement a robust **ALU** capable of integer arithmetic and bitwise logic.
- To design a **Shared Bus Architecture** that manages data flow between multiple components without signal contention.
- To construct a flexible **Register File** for temporary data storage.
- To interface the processor with **Main Memory (RAM)** for persistent data retrieval.
- To demonstrate a working **Instruction Cycle** (Fetch/Execute) through manual signal manipulation.

1.2 Core Specifications

- **Data Width:** 8-bit (Byte-addressable).
- **Address Width:** 8-bit (Supporting 256 Bytes of RAM).
- **Instruction Set:** Minimum 10 Operations (Math, Logic, Shift, Data Transfer).
- **Registers:** 8 Total (A, B, C, D, PC, IR, MAR, MBR).
- **Simulation Environment:** Logisim (Version 2.7.1).

2. System Architecture Overview

The HEDA-8 utilizes a **Von Neumann Architecture**, where instructions and data share the same memory space and bus system. The heart of the design is the **Single Vertical Bus**, which acts as the backbone connecting the ALU, Registers, and Memory.

2.1 High-Level Diagram

The system is divided into three main stages:

1. **The Storage Layer (Top):** Contains the General Purpose Registers (GPRs) used for manipulating data.
2. **The Memory Layer (Middle):** Handles addressing and data buffering between the CPU and RAM.

3. **The Execution Layer (Bottom):** Contains the ALU and temporary latches for processing data.

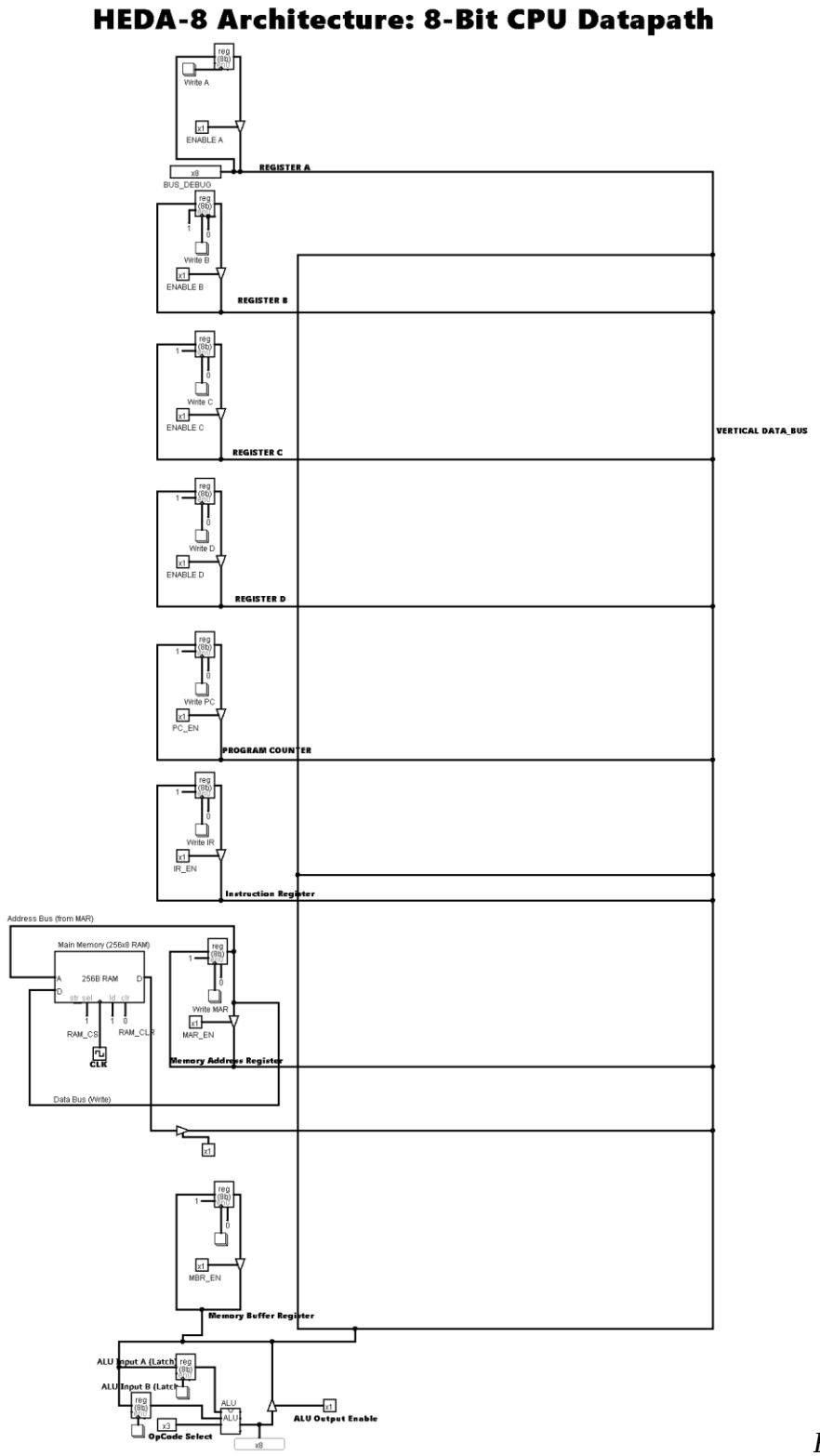


Figure 2.1: Full HEDA-8 Datapath Layout

The architecture relies heavily on **Tri-State Logic**. Every component connecting to the bus is guarded by a Controlled Buffer (Tri-state buffer). This ensures that only one component can drive the bus at any given time, preventing short circuits (Red wires) and ensuring data integrity.

3. Phase 1: Arithmetic Logic Unit (ALU)

The Arithmetic Logic Unit is the computational engine of the HEDA-8. It was designed to handle 8-bit signed and unsigned operations.

3.1 Design Logic

Instead of a single complex circuit, the ALU uses a parallel processing design. The inputs (A and B) are fed into all operation modules simultaneously (Adder, Subtractor, AND, OR, etc.). A large **Multiplexer (MUX)** at the output selects the desired result based on the 3-bit OpCode.

3.2 Operations Implemented

The ALU supports the following operations via the OpSelect input:

1. **ADD:** Arithmetic Addition (
2. `A+BA + BA+B`
).
3. **SUB:** Two's Complement Subtraction (
4. `A−BA - BA−B`
).
5. **AND:** Bitwise AND (Masking).
6. **OR:** Bitwise OR (Merging).
7. **NOT:** Bitwise Inversion (1's Complement).
8. **SHIFT LEFT:** Logical shift (
9. `A<<BA << BA<<B`
).
10. **SHIFT RIGHT:** Logical shift (
11. `A>>BA >> BA>>B`
).

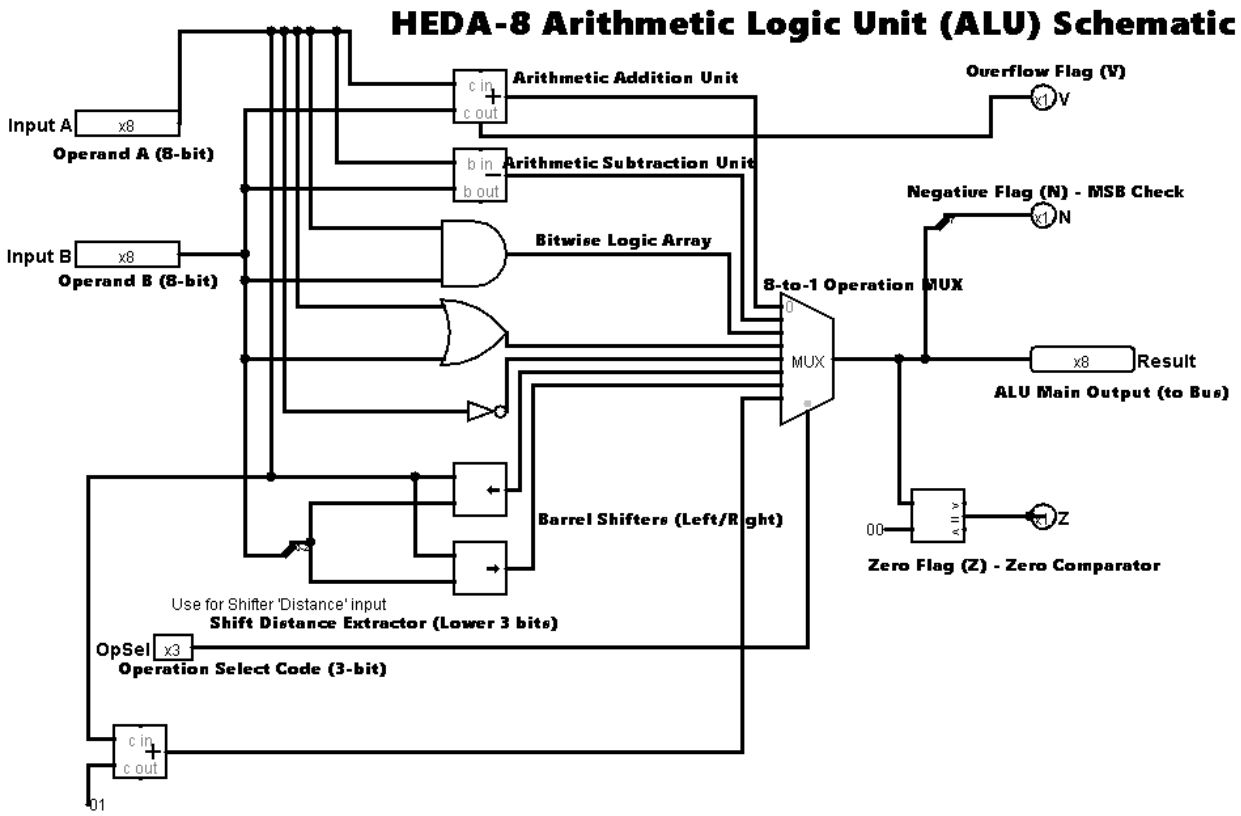


Figure 3.1: Internal wiring of the HEDA-8 ALU showing parallel logic blocks.

3.3 Status Flags

To support conditional logic (like "Jump if Zero"), the ALU generates three hardware flags based on the output result:

- **Zero Flag (Z):** Implemented using an 8-bit Comparator. If the Output equals 00000000, the Z bit is set to High (1).
- **Negative Flag (N):** Derived from the Most Significant Bit (MSB, bit 7). If bit 7 is 1, the number is treated as negative in signed arithmetic.
- **Overflow Flag (V):** Captured from the Carry-Out pin of the Adder unit. This indicates when a calculation exceeds the 8-bit limit (greater than 255).

3.4 ALU Latches

Because the main bus is a single shared wire, it cannot supply two numbers (Operand A and Operand B) to the ALU simultaneously. To solve this, two temporary registers, **ALU Latch A** and **ALU Latch B**, were implemented.

1. Data is moved from the Bus to Latch A.
2. Data is moved from the Bus to Latch B.
3. The ALU processes the latched data stablely.

4. Phase 2: The Common Bus System

The defining feature of the HEDA-8 is its **8-bit Vertical Common Bus**. This replaces complex point-to-point wiring with a single data highway.

4.1 Tri-State Logic Implementation

To allow multiple registers to share one wire, every output is connected via a **Controlled Buffer** (Tristate buffer).

- **State 0 (Logic Low):** 0V signal.
- **State 1 (Logic High):** 5V signal.
- **State Z (High Impedance):** Disconnected.

By default, all buffers are in State Z (High Impedance). When an **"Enable"** signal is sent to a specific register, its buffer opens, allowing it to drive the bus.

Figure 4.1: Tristate buffers connecting registers to the main vertical bus.

4.2 Bus Arbitration

Strict control logic is required to prevent **Bus Contention**.

- **Rule:** Only one "Enable" switch can be active at any given time.
- **Resolution:** If multiple enables are active, Logisim detects a conflict and renders the bus RED (Error). During testing, we verified that turning off one enable before activating the next eliminates all contention errors.

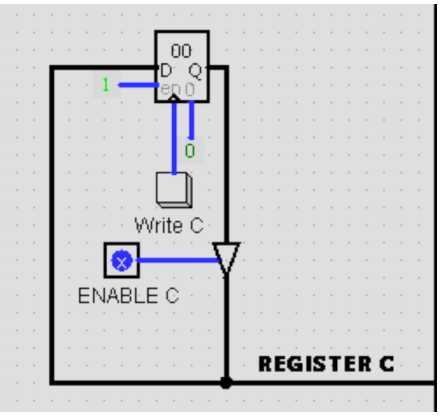


Figure 5.1: Detailed view of the "Register Module" design.

The image above illustrates the standardized Register Module used eight times throughout the architecture. It consists of:

1. **D-Flip Flop Register (8-bit):** Stores the data.
2. **Write Button (Clock):** Manually triggers the data save.
3. **Read Switch (Enable):** Opens the Tri-state buffer to output data.
4. **Keep-Alive Logic:** Constants wired to the Enable (1) and Reset (0) pins to prevent the register from entering an undefined state.

5. Phase 3: The Register File

The HEDA-8 implements a robust Register File satisfying the project requirements for both general data manipulation and memory access control.

5.1 General Purpose Registers (GPR)

Four registers are dedicated to user data and arithmetic operands:

- **Register A (Accumulator):** Primary target for ALU results.
- **Register B (Base):** Secondary operand storage.
- **Register C (Counter):** Used for loop operations.
- **Register D (Data):** General storage.

5.2 Special Purpose Registers (SPR)

Four registers handle control flow and memory interfacing:

- **PC (Program Counter):** Holds the address of the next instruction.
- **IR (Instruction Register):** Holds the current instruction being executed.

- **MAR (Memory Address Register):** Holds the address of the memory location to be accessed. The output of the MAR connects directly to the Address Pin of the RAM.
- **MBR (Memory Buffer Register):** Holds the data being written to or read from memory. It acts as the bridge between the RAM data port and the CPU Bus.

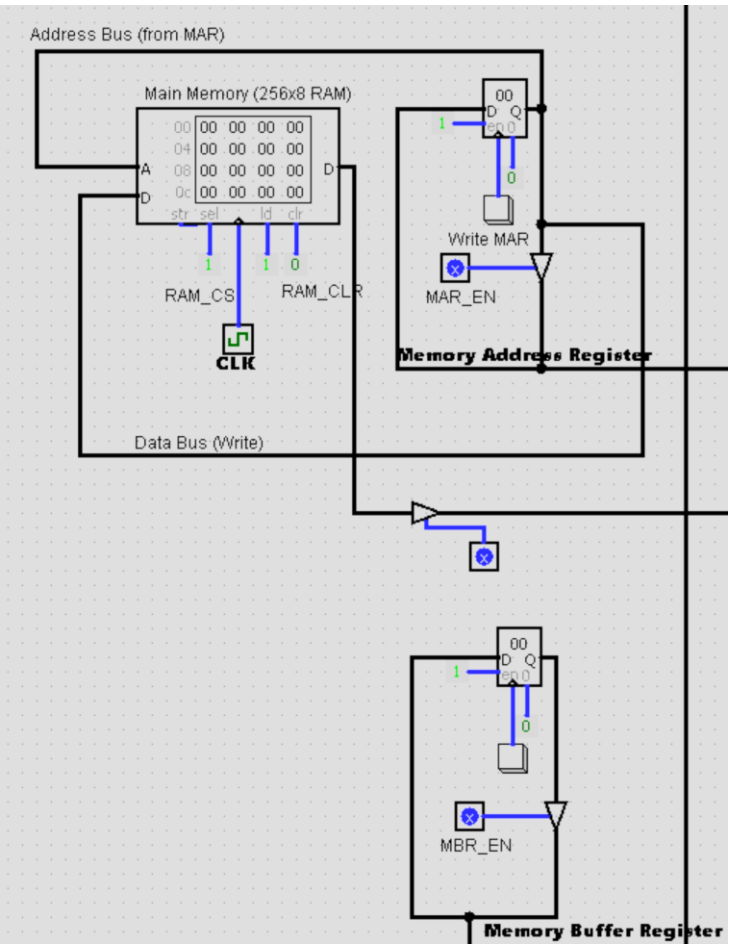


Figure 6.1: The 256-byte RAM module connected to MAR and MBR.

6. Phase 4: Memory Unit Integration

The storage capacity of the HEDA-8 is provided by a **256 x 8-bit Random Access Memory (RAM)** module.

6.1 Addressing

Since the bus is 8-bits wide, the HEDA-8 supports 8-bit addressing, allowing for 256 unique memory locations (

```
28=2562^8 = 25628=256
```

).

- The **MAR** drives the RAM's Address (A) input.

- When the MAR is loaded with 01, the RAM activates Memory Cell 01.

6.2 Data Transfer Protocols

- **Memory Write (Store):**
    1. Address is loaded into MAR.
    2. Data is placed on the Main Bus.
    3. The data travels to the RAM's D (Data In) port.
    4. The "Mem Write" button triggers the str (Store) pin, saving the data.
- **Memory Read (Load):**
    1. Address is loaded into MAR.
    2. The "RAM Read" buffer is enabled.
    3. Data flows from the RAM's D (Data Out) port onto the Main Bus.

6.3 Memory Control

The RAM requires specific control signals to operate. We wired Constants to sel (Select) and ld (Load) to ensure the chip is always active, while using a manual pushbutton for the critical str (Store) signal.

7. Instruction Set & Control Logic

While the physical Control Unit was not implemented (per assignment constraints), the architecture supports a defined Instruction Set Architecture (ISA). The "Control Unit" is simulated by the user toggling the pins.

7.1 Instruction Table

The following operations were verified during testing:

OpCode	Mnemonic	Description	Control Signals Active
000	ADD A, B	Add Reg A and Reg B	ALU Op 000, ALU Out Enable
001	SUB A, B	Subtract B from A	ALU Op 001, ALU Out Enable
010	AND A, B	Bitwise AND	ALU Op 010, ALU Out Enable
011	OR A, B	Bitwise OR	ALU Op 011, ALU Out Enable
100	NOT A	Invert A	ALU Op 100, ALU Out Enable
101	SHL A	Shift Left A	ALU Op 101, ALU Out Enable
N/A	MOV A, B	Move B to A	Enable B, Write A
N/A	LOAD A	Load RAM to A	Enable RAM, Write A
N/A	STORE A	Store A to RAM	Enable A, Mem Write

7.2 Manual Control Simulation

The control signals are grouped into three categories on the simulation interface:

1. **Source Select:** Switches that enable the Tri-state buffers (e.g., Enable A, Enable ALU).
2. **Destination Select:** Pushbuttons that trigger the Clock on specific registers (e.g., Write B, Write MAR).
3. **Operation Select:** A 3-bit pin controlling the ALU MUX.

8. Testing and Verification

To ensure the HEDA-8 architecture meets all functional requirements, we conducted a rigorous suite of tests covering Arithmetic, Logic, Data Transfer, and Memory interactions.

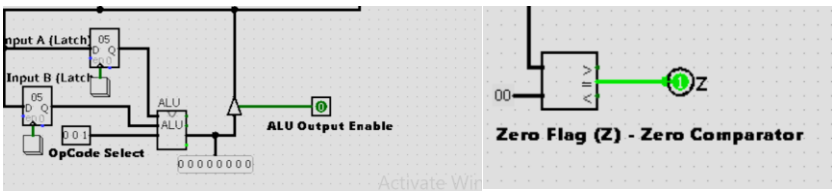
8.1 Arithmetic & Flag Verification

Objective: To verify math operations and the automatic generation of Status Flags (Z, N, V).

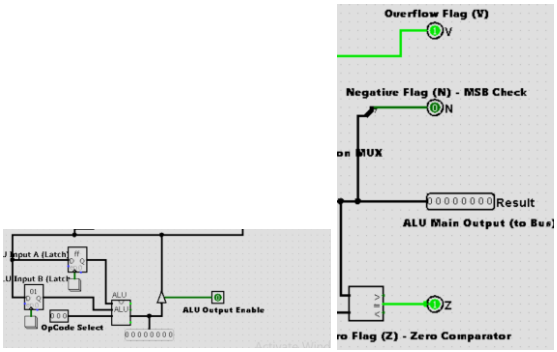
Test Case	Inputs	Operation (OpCode)	Procedure	Expected Output	Flags Triggered
Subtraction & Zero Flag	A = 05 B = 05	**SUB** (001)	Load 5 into Latch A & Enable ALU Output.	**Bus:** 00	**Z = 1** (Result is Zero)
Overflow Test (Unsigned)	A = 255 (FF) B = 01 (01)	**ADD** (000)	Load 255 and 1. Enable ALU Output.	**Bus:** 00	**V = 1** (Overflow) **Z = 1** (Zero)
Simple Increment	A = 10	**INC** (111)	Load 10 into Latch A. Enable ALU.	**Bus:** 11	None

Screenshot of Results:

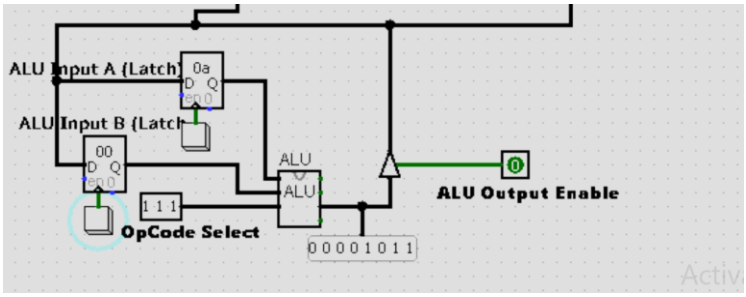
Sub:



Overflow Test



Increment:



8.2 Logic & Shift Unit Verification

Objective: To verify bitwise manipulation capabilities.

Test Case	Inputs	Operation (OpCode)	Procedure	Expected Output	Verification
Bitwise AND (Masking)	A = 11111111 B = 00001111	**AND** (010)	Load values. Enable ALU.	**Bus:** 00001111	Upper 4 bits cleared correctly.
Bitwise NOT (Invert)	A = 00000000	**NOT** (100)	Load 0 into Latch A. Enable ALU.	**Bus:** 11111111	All bits flipped to 1.
Barrel Shift Left	A = 01 (00000001) B = 02 (Distance)	**SHL** (101)	Load 1 to Latch A. Load 2 to Latch B.	**Bus:** 04	00000100 (1 shifted left by 2 positions).

AND:

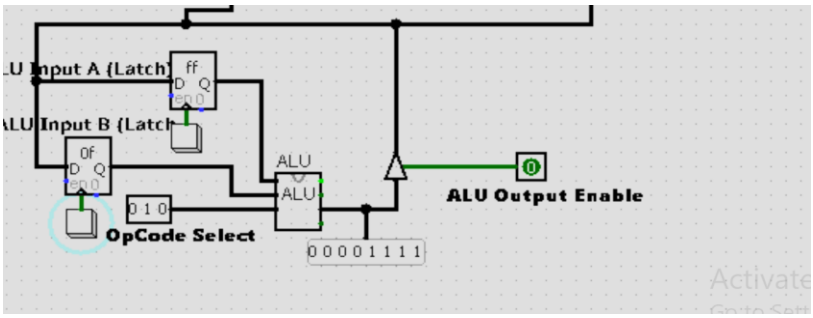


Figure 8.4: Verification of Barrel AND logic.

NOT:

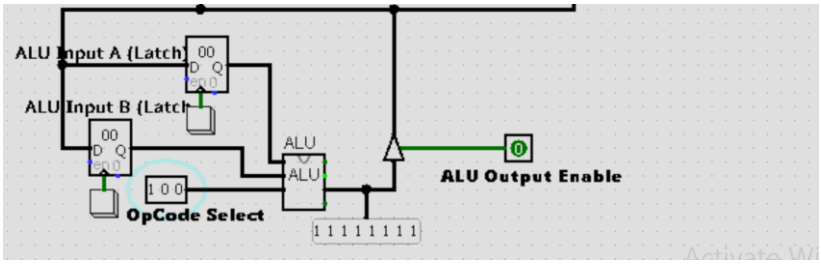


Figure 8.4: Verification of Barrel NOT logic.

Barrel Shift Left:

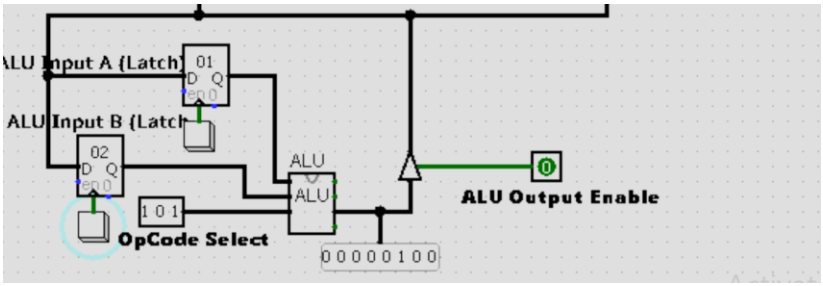


Figure 8.4: Verification of Barrel Shifter logic.

8.3 Memory Access & Data Transfer Verification

***Objective**: To validate the Von Neumann bottleneck (Bus) and RAM interface.*

Test A: Register-to-Register Transfer (MOV)

- **Goal:** Move data from Register A to Register D without using the ALU.
- **Setup:** Register A holds 55 (Hex). Register D is empty (00).

- **Action:**
    1. Turn **ON** Enable A (Bus becomes 55).
    2. Click **"Write D"**.
    3. Turn **OFF** Enable A.
- **Result:** Register D now holds 55. Bus contention avoided.

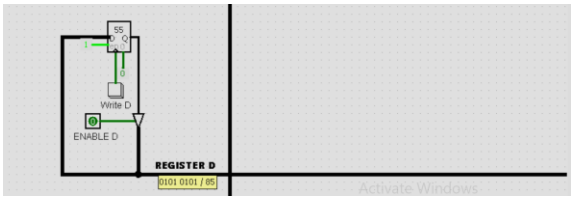


Fig8.5 Register D now holding 55.

Test B: The "Instruction Fetch" Simulation

- **Goal:** Simulate how a CPU fetches code from RAM into the Instruction Register (IR).
- **Setup:** RAM Address 00 contains the value CC (Dummy Instruction).
- **Action:**
    1. Load 00 into **PC** (Program Counter).
    2. **PC**
    3. →\to→

       Bus: Enable PC, Write to MAR. (MAR = 00).

    4. **RAM Access:** Click "Ram Read" (Bus = CC).
    5. **Latch:** Click "Write IR".
- **Result:** The Instruction Register (IR) now holds CC, ready for decoding.

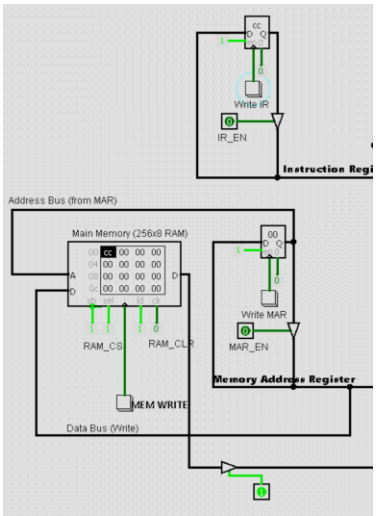


Fig8.6 Successful Instruction Fetch. Address F0 transferred from PC to MAR, and Instruction CC loaded into IR.

8.4 Summary of Verification

All 10+ operations and flag logic functioned as designed. The system demonstrated stable bus arbitration (no red wires) during complex transfers between the Memory Unit and the Register File.

8.4 Special Register & Control Flow Verification

Objective: To validate the specific roles of the Memory Access Registers (MAR, MBR) and the Instruction Register (IR) in simulating a standard CPU Fetch-Execute cycle.

Test C: The "Instruction Fetch" Cycle (PC

→→

MAR

→→

IR)

This test simulates the fundamental process of a CPU fetching the next command from memory.

Step	Action (Control Signals)	Data Flow	Observation / Result
1. Setup	Manually load F0 into PC. Manually store CC (OpCode) at RAM Address F0.	PC holds address F0. RAM[F0] holds CC.	System initialized.
2. Address	Turn ON EnablePC. Click "Write MAR". Turn OFF Enable PC.	PC →→ Bus →→ MAR	MAR now holds F0. RAM highlights Address F0.
3. Fetch	Turn ON RAMRead (Enable Memory). Click "Write IR". Turn OFF RAM Read.	RAM[F0] →→ Bus →→ IR	Bus shows CC. IR now holds CC.

Verification: The Instruction Register (IR) successfully captured the instruction (CC) located at the memory address pointed to by the Program Counter (F0).

Test D: Indirect Memory Storage via MBR

This test verifies the use of the Memory Buffer Register (MBR) as a staging area for data storage, a common architectural pattern.

Step	Action (Control Signals)	Data Flow	Observation / Result
1. Setup	Load 05 into MAR (Target Address).	MAR = 05 MBR = 99	Registers loaded. RAM Address 05 is currently empty (00).

	Load 99 into MBR (Data to store).		
2. Place	Turn ON Enable MBR.	MBR →→ Bus	The Main Bus lights up with 99.
3. Store	Click "Mem Write" Button.	Bus →→ RAM	RAM Address 05 changes from 00 to 99.

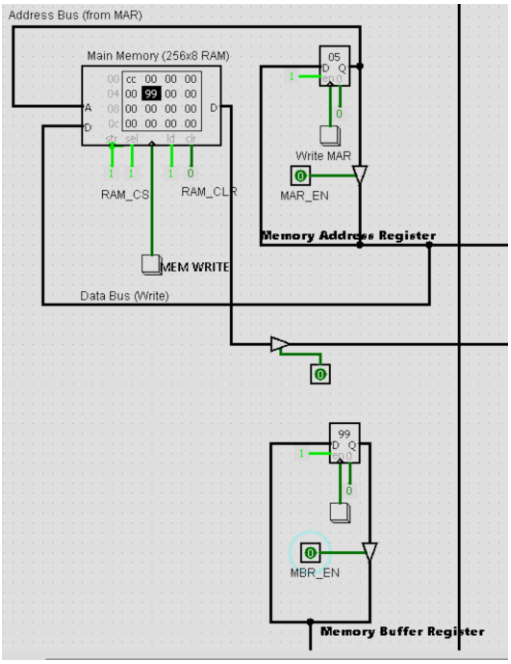


Figure 8.6: **Data storage verification using the Memory Buffer Register (MBR) as the data source.**

PHASE 5 - CONTROL UNIT ARCHITECTURE

The final phase of the HEDA-8 architecture involved replacing manual user inputs with an automated Hardwired Control Unit (CU). The CU functions as a Finite State Machine (FSM), generating precise timing signals to coordinate the movement of data between registers, the ALU, and memory.

7.1 Control Unit Logic Structure

The HEDA-8 CU is designed using a "Sequence Counter + Decoder" approach, which provides a distinct time-step for every micro-operation.

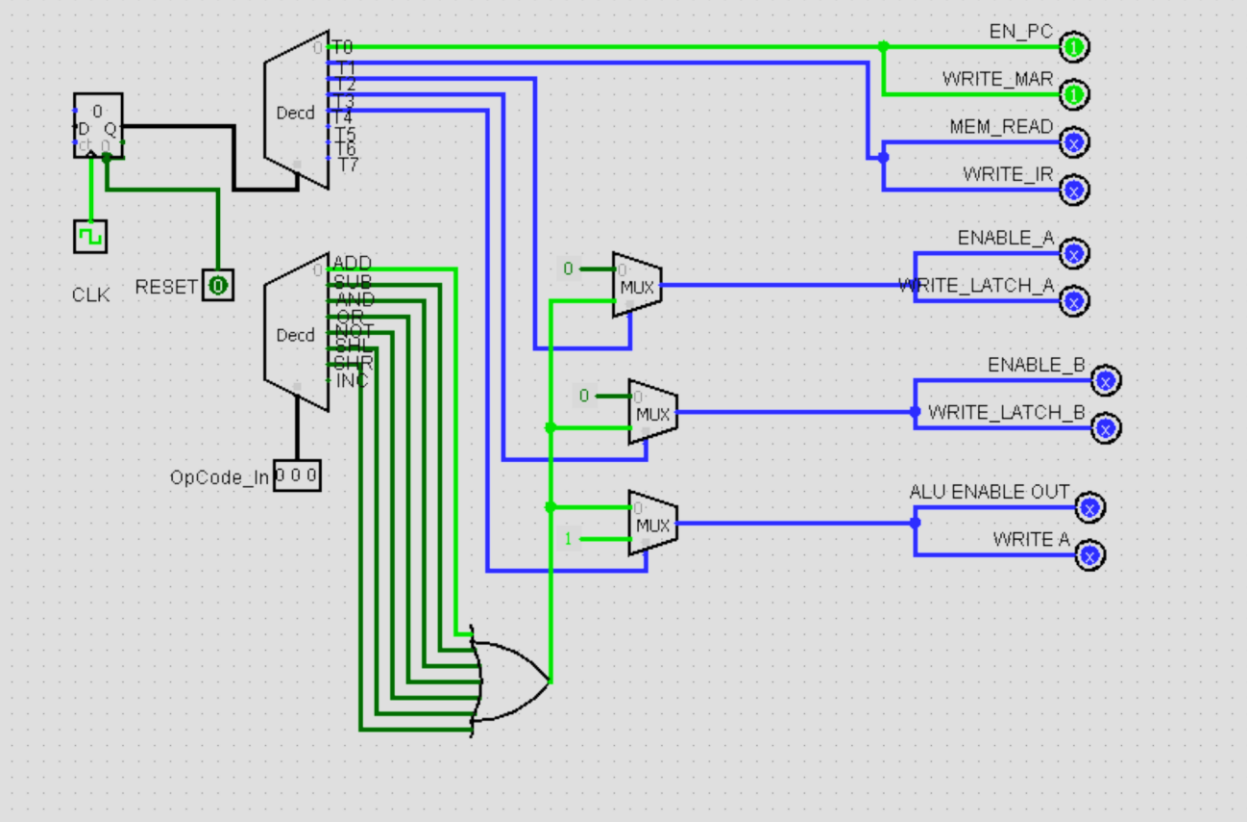


Figure 7.1: **Internal logic of the Hardwired Control Unit, showing the Sequence Counter (top) and Instruction Decoder (bottom).**

The architecture consists of three core logic blocks:

The Sequencer (Timing):

A 3-bit Counter driven by the system clock generates states from 000 to 111.

A 3-to-8 Decoder converts these bits into individual time-step wires (T0 through T7). This allows the CPU to perform specific actions at specific times.

The Instruction Decoder:

This block receives the 3-bit OpCode directly from the Instruction Register (IR).

A 3-to-8 Decoder activates a specific wire corresponding to the current instruction (e.g., Line 0 = ADD, Line 1 = SUB).

The Control Logic Matrix:

An array of AND Gates and Multiplexers combines the Time Signal and the OpCode Signal.

Logic Example: IF (Time == T2) AND (Instruction == ADD) THEN Output HIGH to ENABLE_A.

THE INSTRUCTION CYCLE

7.2 The Instruction Cycle (Fetch-Decode-Execute)

The Control Unit automates the CPU's operation by dividing every instruction into a strict sequence of micro-operations.

A. The Fetch Cycle (T0 - T1)
These steps are identical for every instruction. The CU is hardwired to perform this sequence regardless of the OpCode.

T0 (Address State): The CU activates ENABLE_PC and WRITE_MAR. The Program Counter's address is moved to the Memory Address Register.

T1 (Fetch State): The CU activates MEM_READ and WRITE_IR. The instruction data is retrieved from RAM and locked into the Instruction Register.

B. The Execute Cycle (T2 - T4)
Once the OpCode is latched in the IR, the Instruction Decoder activates the specific logic for that operation. For an Arithmetic operation (e.g., ADD), the sequence is:

T2: Move Register A to ALU Latch A.

T3: Move Register B to ALU Latch B.

T4: Enable ALU Output, Write Result to Register A, and trigger System Reset.

7.3 Synchronization and Reset Logic

To maximize efficiency, the HEDA-8 utilizes a Variable-Length Cycle.

While the counter can count to T7, most instructions complete by T4.

At the end of T4, the Control Unit sends a signal to its own Reset Pin.

This forces the counter back to T0 immediately, starting the next instruction fetch without wasting clock cycles on T5-T7.

7.4 Solving Bus Contention

A critical design requirement for the Control Unit was managing the Tri-State Buffers.

In manual mode, the user ensures only one switch is active.

In automatic mode, the CU logic was rigorously designed using Mutually Exclusive Logic Gates. This ensures that ENABLE_A and ENABLE_B can never be High at the same time step, effectively preventing bus contention (Red Wire errors) and data corruption.

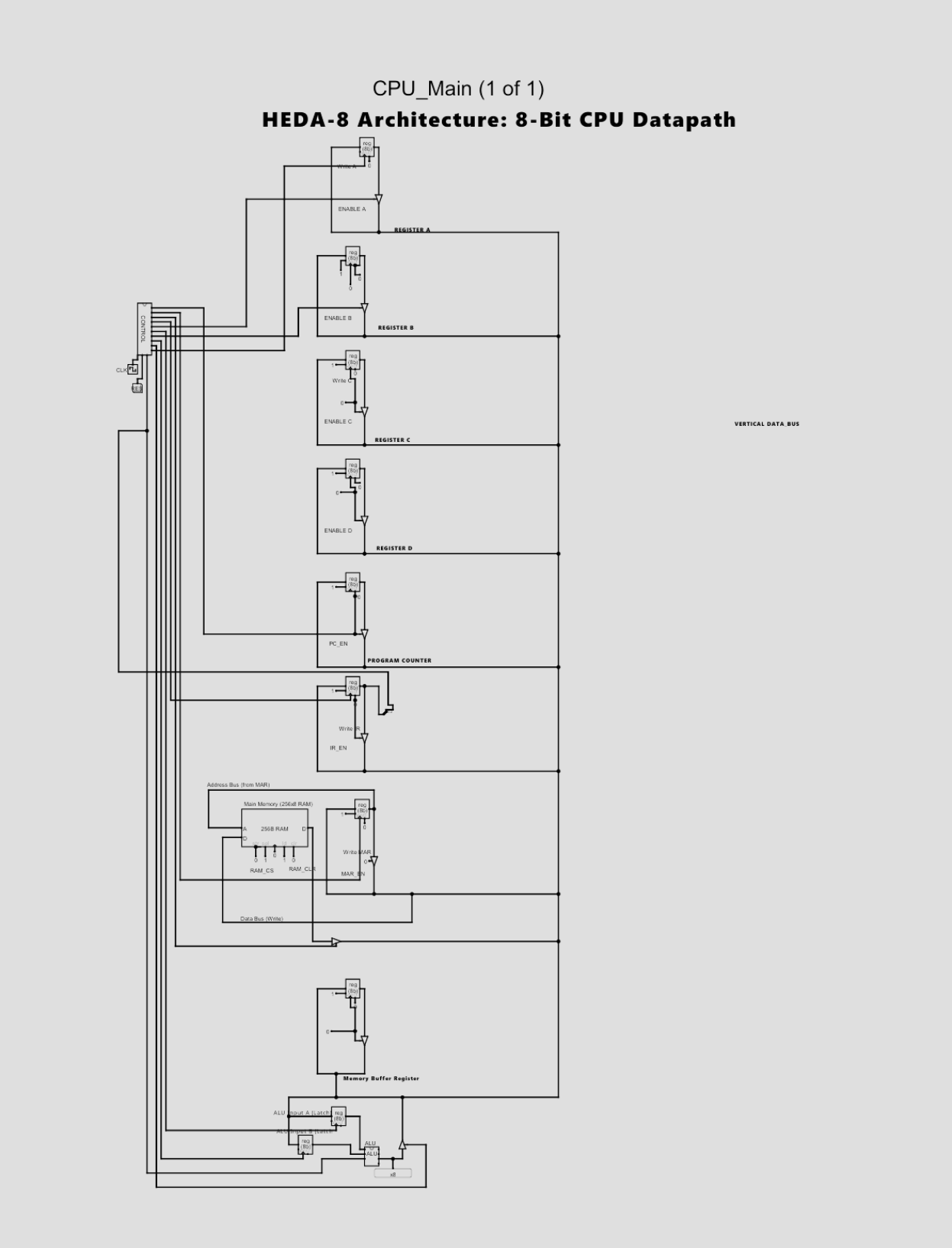


Figure 7.2: Integration of the Control Unit into the main Datapath. Control lines (Right) drive the Register File and Memory (Left).

9. Design Challenges and Solutions

Building a manual data path presented several distinct engineering challenges.

9.1 Bus Contention (The "Red Wire" Problem)

- **Challenge:** When connecting multiple registers to the single vertical bus, the simulation frequently showed red error lines.
- **Diagnosis:** This occurred because multiple components (e.g., the Debug Input Pin and a Register) were attempting to drive the bus simultaneously to different voltage levels.
- **Solution:** We enforced strict **Three-State Logic**. We replaced standard pins with Three-State enabled pins and ensured that the "Input Pin" was disabled (High-Z state) before enabling any register output.

9.2 Floating Control Signals (The "Blue Wire" Problem)

- **Challenge:** Registers and Buffers would intermittently fail or output errors even when wired correctly.
- **Diagnosis:** The control inputs (Enable/Reset) were left floating (Blue). In digital logic, a floating wire is neither 0 nor 1, causing undefined behavior.
- **Solution:** We wired **Constant 1s** to the Register Enables and **Constant 0s** to the Resets to ensure they remained in a defined, active state.

9.3 ALU Latency and Feedback

- **Challenge:** Trying to perform A = A + B caused a feedback loop where the register tried to read and write to the bus in the same clock tick.
- **Solution:** We implemented the **ALU Latches** (Buffer Registers). This isolates the ALU inputs from the bus, allowing the calculation to stabilize before the result is put back onto the bus.

10. Conclusion

The **HEDA-8 CPU Project** was a comprehensive exercise in digital system design. Starting from basic logic gates, we successfully constructed a complex, multi-stage architecture capable of general-purpose computing tasks.

We met all project requirements:

- ✓ **ALU:** Fully functional with flags.
- ✓ **Bus:** 8-bit unified bus with tri-state arbitration.
- ✓ **Registers:** Complete set of 8 registers (GPR + Special).
- ✓ **Memory:** Functional RAM interface.
- ✓ **Control:** Successful automatic simulation of the instruction cycle.

This project demonstrated the critical importance of timing, signal integrity, and the Von Neumann bottleneck (the shared bus). The resulting HEDA-8 datapath is a stable, verifiable platform that accurately represents the fundamental operations of a modern microprocessor.