Designing and building an 8-bit CPU

Ian Cho

TABLE OF CONTENTS

1. INTRODUCTION

This project documents the design and implementation of a discrete-logic 8-bit CPU designed first on the KiCAD software and prototyped on breadboards, with current efforts to create PCBs for it. The processor uses TTL components (74LS series) and a microcoded control unit stored in parallel EEPROMs. With an 8-bit data bus, adjustable clock, and a compact instruction set, the system exposes the classic fetch–decode–execute cycle with visual instrumentation for bug fixes: more than 80 LEDs trace control lines, bus traffic, and register contents in real time.

This project draws heavy inspiration from Ben Eater's 8-bit computer design, a well-known educational build that demonstrates the fundamentals of computer architecture using discrete logic. While his tutorials and schematics provide a clear foundation, this implementation adapts and extends the design to fit my own goals, with modifications in components, organization, presentation, and debugging approaches. The result remains faithful to the core teaching philosophy—making the invisible processes of computation visible through LEDs and modular wiring—while also reflecting the unique challenges of building and troubleshooting my own version.

2. GOAL

- Build an 8-bit computer on breadboards and a follow-on PCB.
- Implement a microcoded control unit that sequences each instruction into micro-operations.
- Provide manual and continuous clocking for step-through debugging and real-time execution.
- Surface internal state via LEDs and 7-segment displays for clarity.

3. RESOURCES

The most significant resource for this project is Ben Eater's 8-bit computer series (https://eater.net/8bit). His detailed schematics and tutorials were invaluable in understanding the architecture and sequence of operations. However, reproducing and adapting the design required extensive independent troubleshooting and debugging, especially now, as the build is being attempted to be migrated into a PCB prototype.

- **Lucid Chart** – block diagram representation
- **KiCad** – schematic design & PCB layout.
- **Arduino IDE** – used with a custom Arduino Nano EEPROM programmer.
- **Custom EEPROM Programmer** – Arduino Nano plus shift-register glue to write binary images into **28C16** EEPROMs.
- Standard lab tools: multimeter, oscilloscope, wire tools, assorted TTL ICs, LEDs, switches, and passive components.

4. SYSTEM CHARACTERISTICS

Architecture

- **Bus width:** 8-bit shared data bus
- **Clock speed:** Adjustable from sub-Hz (manual stepping) to ~100 kHz (continuous), limited by 74LS propagation delays
- **Supply:** 5 V DC regulated
- **Indicators:** >80 LEDs for control, bus, and register visualization

Instruction Set (representative)
LDA, STA, ADD, SUB, JMP, Jcc (conditional branch), OUT, HLT

Major Components

- **Clock Module:** 555 timer oscillator with potentiometer frequency control
- **Registers:** Two 74HCT173s (4-bit D register) and 74HCT245N (octal D-type) for A, B, Instruction, and Output registers
- **ALU:** 74LS283 adders + logic (supports ADD, SUB via two's-complement), basic bitwise ops (AND/OR/XOR)
- **RAM:** 74189 ×2 (16 bytes of RAM, e.g., 128 bytes)
- **Program Counter:** 74HCT161 (8-bit PC)

- **Control Logic:** 28C16 ×2 storing microcode; sequenced by step counter
- **Bus Transceivers:** 74LS245 ×6 to prevent contention and isolate modules

Component Table

Component	Part Number	Function	Notes
Clock Module	555 Timer + gates	System clock source	Adjustable via potentiometer
General Registers	74HCT173 x 2 + 74HCT245N	Temporary data storage	4-bit D-type with LEDs tapped
Output Register	74HCT273 + 28C16 + 74HCT139 + 74LS76 + 555 timer + 7-segment display	Latches value to displays	Drives LEDs & 7-segment displays
ALU	74LS283 + 74HCT245 + 74HCT173 + gates	Arithmetic & logic	ADD/SUB via XOR + carry-in; AND/OR/XOR
RAM	74189 ×2 + 74HCT157 x 3 + 74HCT245N + inverters + switches	Program/data storage	128 bytes total
Program Counter	74HCT161 + 74HCT245N	Instruction sequencing	8-bit counter
Control Logic	28C16 ×2 + 74HCT138 + 74HCT161D + LEDs + gates + inverters	Microcoded instruction control	Programmed with Arduino Nano
Bus Transceivers (included within the modules)	74LS245 ×6	Data-bus isolation	Avoids bus contention

5. Block Diagram

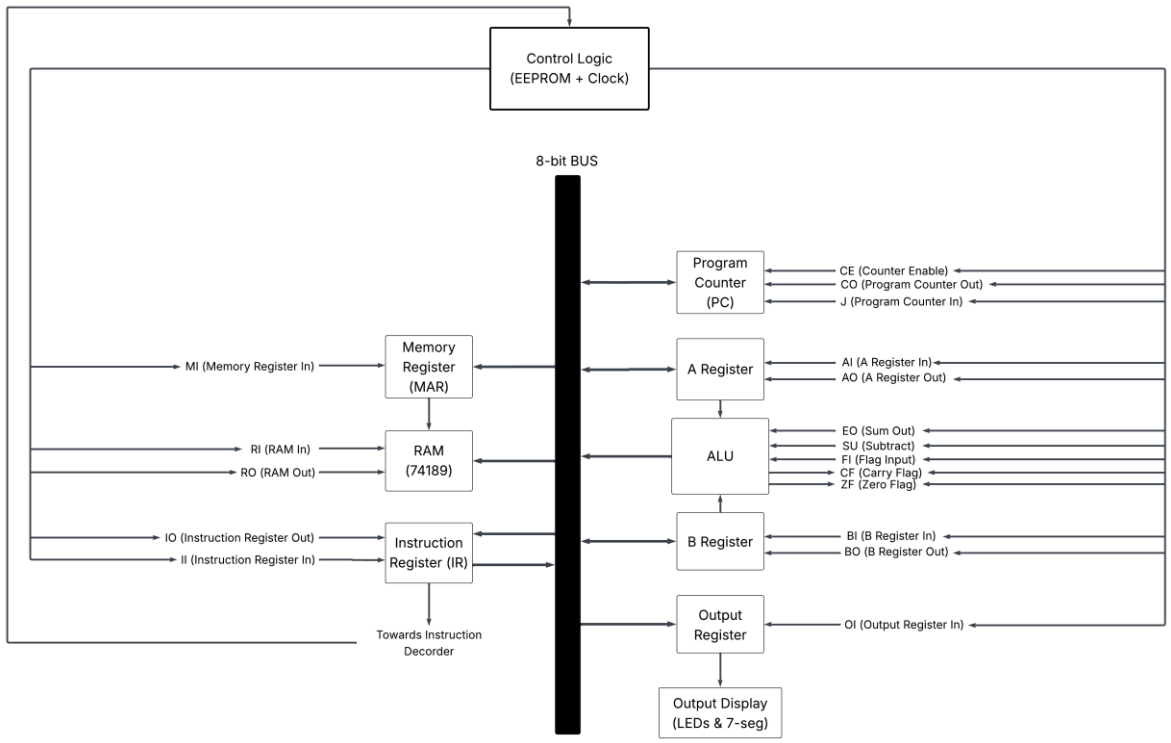

Figure 1: system-level block diagram of the 8-bit CPU architecture

Before examining the individual modules in detail, a system-level block diagram of the CPU is presented (Figure 1). This diagram captures the processor's overall architecture, showing the Program Counter, Memory, Registers, ALU, Output Register, and Control Logic, all connected through the central 8-bit bus.

The diagram highlights the shared bus as the backbone of communication, with each module interfacing to it under the direction of the microcoded control logic. Arrows indicate the direction of data flow and control signals such as load, enable, or output. The Control Logic, implemented with EEPROMs and a step counter, ensures that only one component drives the bus at any time, coordinating the fetch–decode–execute cycle across the system.

The block diagram provides a clear overview of module interactions before delving into individual schematics. It allows readers to see how the CPU fits together as a whole, helping them interpret the more detailed wiring diagrams that follow.

6. BUILD PROGRESS

6.1. Clock Module

6.1.1. Automatic Mode

A 555 timer is configured in astable mode to generate the system clock. A panel potentiometer varies the RC constant to sweep from <1 Hz to ~100 kHz (practical limit set by TTL timing).

6.1.2. Manual/Single-Step Mode

A debounced pushbutton injects single pulses for step-through debugging. Classic switch bounce is suppressed using a 555 in monostable configuration (or equivalent debounce network).

6.1.3. Clock Schematic

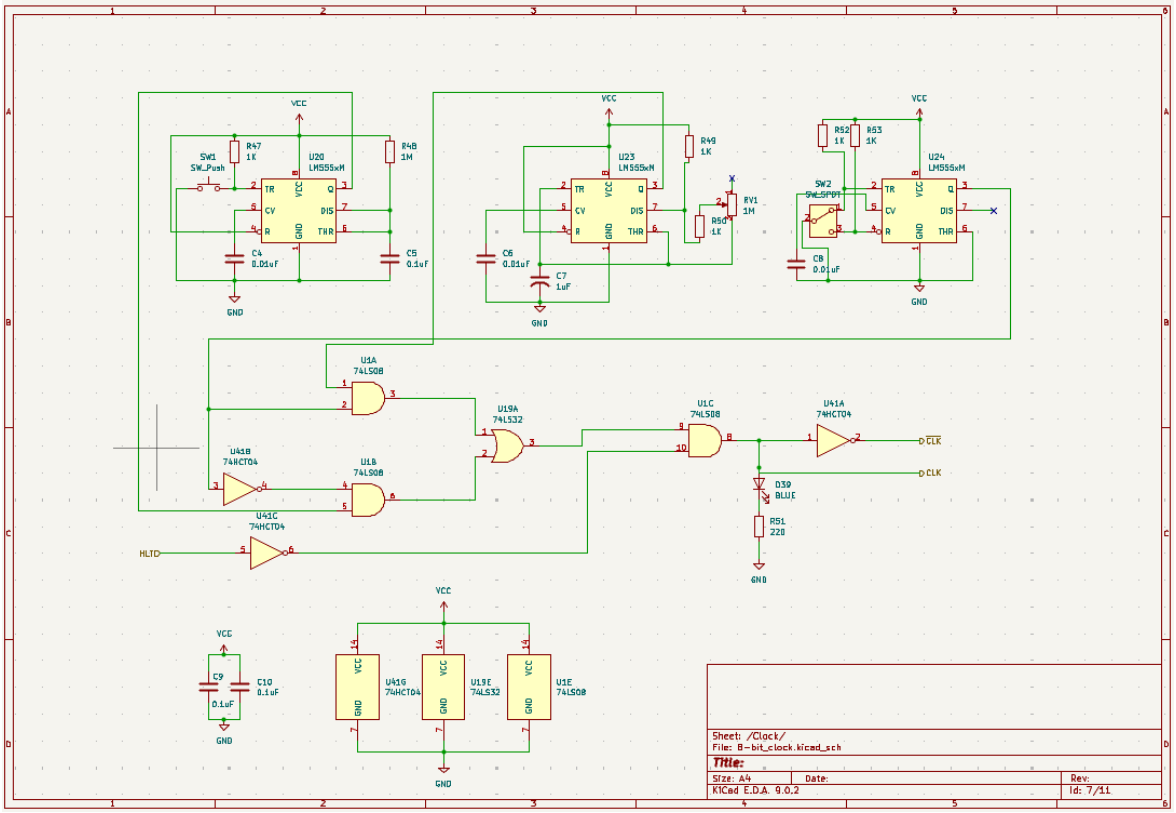

Figure 2: Clock schematic created on KiCAD

6.2. Register Modules

6.2.1. Bus Interfacing

A central feature of this CPU design is the use of a shared 8-bit tri-stated data bus. Every module that needs to exchange data, registers, memory, ALU, and output devices, connects to this bus. However, to avoid electrical interference, it is critical that only one module drives the bus at any given moment, while all others remain disconnected (high-impedance state).

To achieve this, the design relies on 74LS245 bus transceivers. Each transceiver acts as a gatekeeper, isolating the internal register outputs from the bus until an enable signal allows data to flow. This arrangement ensures reliable communication and prevents multiple outputs from shorting against each other.

For visibility and debugging, the system also includes LED taps attached to register outputs. These LEDs show the current latched contents in real time, allowing the user to monitor what values are stored without affecting bus operation. Since the LEDs draw current, they are either connected through resistors or rely on the inherent current-limiting of certain TTL devices to avoid loading the bus. This design decision balances transparency for the user with electrical safety for the system.

In summary, the bus-transceiver arrangement provides both electrical integrity and observability, ensuring modules can share a single bus without interference while still offering immediate feedback to the operator.

6.2.2. A/B/Instruction Registers

The CPU relies on several registers to hold operands and instructions during execution. Each is implemented with 74HCT173 4-bit D-type registers, which provide edge-triggered latching, enabling controlled data storage synchronized to the system clock.

- **A & B Registers**:
  Two cascaded 74HCT173 ICs form the A and B operand registers, each capable of holding an 8-bit value. These registers are typically loaded from the system bus when their respective load enable signals are active. Once latched, their outputs drive two paths:
    - LED indicators, which visually represent the binary value stored in each register.

    - 74LS245 transceivers, which allow the register contents to be selectively placed onto the system bus when required for further operations (such as feeding into the ALU).

  This dual visibility and bus-access design make the A/B registers highly flexible: they serve as both working storage for computations and as sources of data to be reused elsewhere in the CPU.

- **Instruction Register (IR)**:
  The Instruction Register is critical for sequencing program execution. It stores the current 8-bit instruction word fetched from memory. Its role is split into two distinct functions:
    - The upper 4 bits (opcode) are routed directly to the microcode control logic, where they determine which instruction is being executed.
    - The lower 4 bits (operand or immediate value) can either be interpreted internally (for branching or addressing) or placed back on the bus through a transceiver to be used as an address, data value, or control parameter.

This separation of opcode and operand in the IR (Instruction Register) reflects the CPU's fetch-decode-execute cycle: fetch the full instruction into the IR, decode the opcode to determine the operation, and then use the operand to guide execution. Together with the A and B registers, the IR ensures that both instructions and data can be cleanly stored, interpreted, and shared across the bus at the right time.

6.2.3. Register Schematics

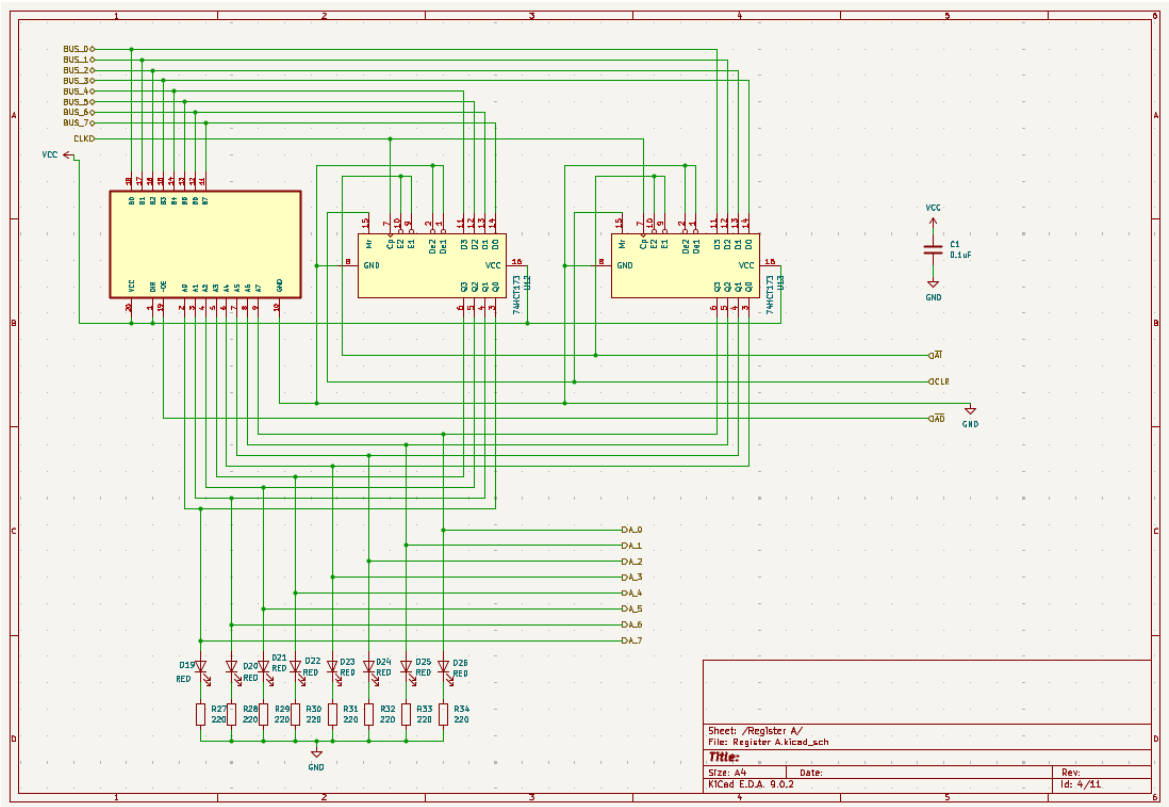

Figure 3: Schematic of Register A created on KiCAD

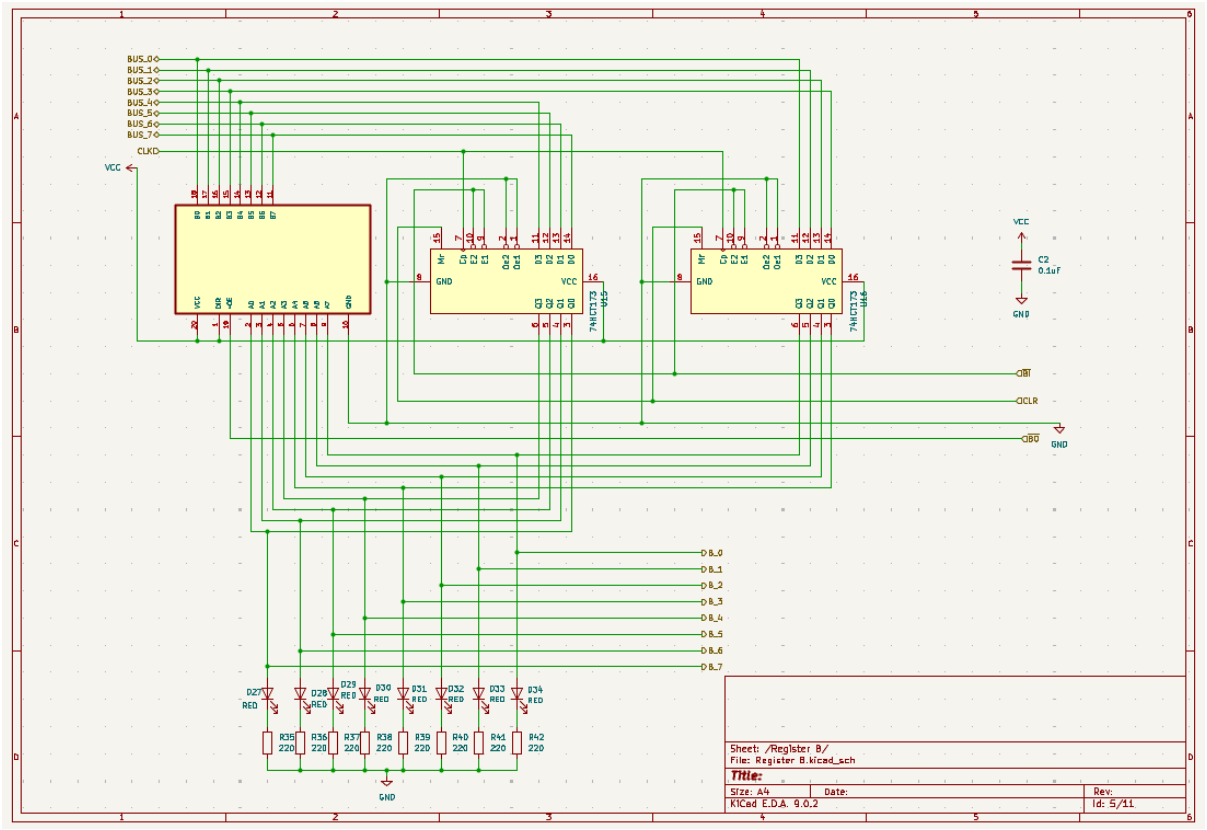

Figure 4: Schematic of Register B created on KiCAD

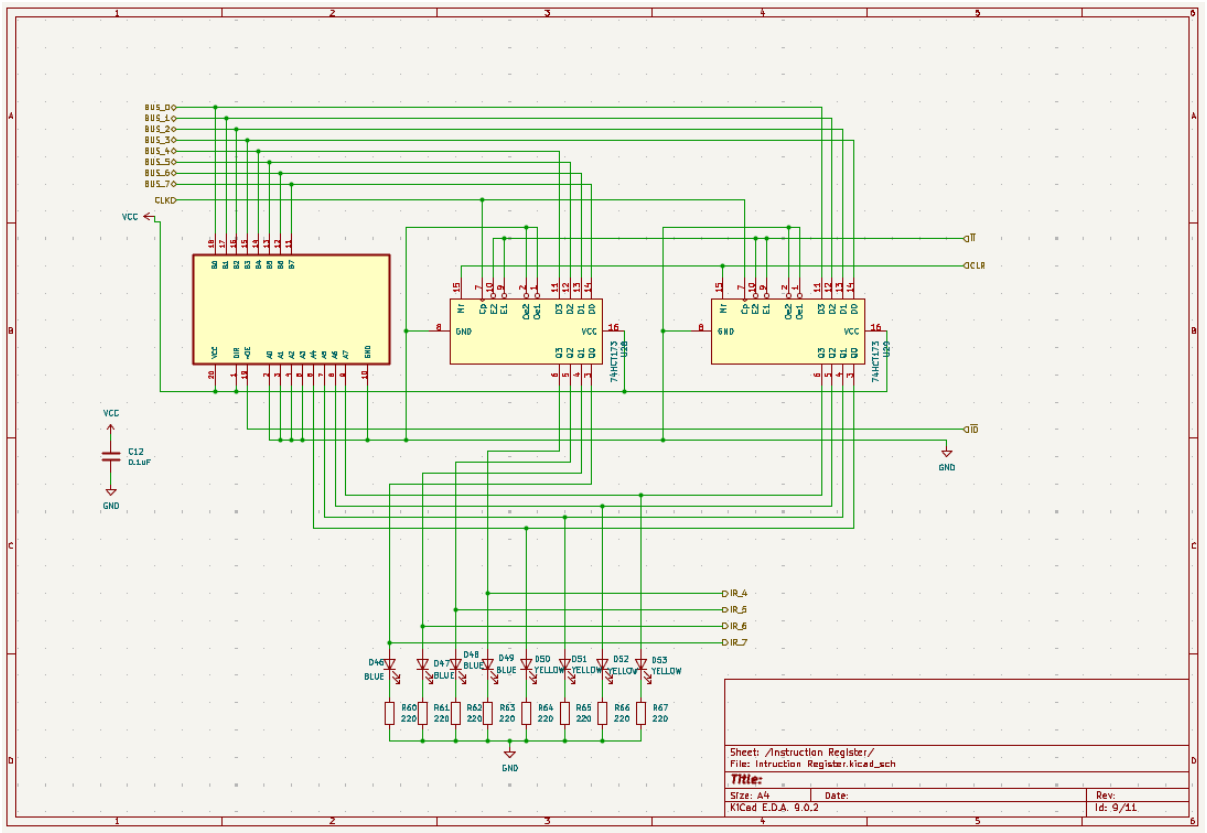

Figure 5: Schematic of the Instruction Register created on KiCAD

6.3. Arithmetic Logic Unit (ALU)

6.3.1. ALU Overview

The ALU is the computational core of the CPU, responsible for performing arithmetic and logical operations on 8-bit operands. It accepts two inputs (from the A and B registers), processes them through adders and logic circuitry, and places the result onto the shared system bus or into the output latch.

6.3.2. Core Components

Input Register (74HCT173)

- Temporary latch for operand values loaded from the system bus.
- Holds one of the ALU's two inputs (commonly the B operand).
- Controlled via load (LD) and clear (CLR) signals.

Operand Multiplexing & XOR Gates (74LS08 AND, 74LS32 OR, 74LS86 XOR)

- B operand bits are routed through XOR gates.

- When the SUB control line is low, B passes through unchanged.
- When SUB is high, B is inverted bit-wise, forming the two's complement required for subtraction.
- The same SUB line also drives the carry-in of the adder, completing the A – B operation.

Binary Adders (74LS283 × 2)

- Two 4-bit adders cascaded to form an 8-bit adder.
- Inputs: A operand (direct) and conditioned B operand (via XOR).
- Outputs: 8-bit sum result plus carry/overflow status.
- Supports addition, subtraction, and arithmetic with carry.

Logic Operations (74LS08, 74LS32, 74LS04)

- Additional AND, OR, and NOT operations implemented alongside the adder.
- Selected by control signals from the control logic.
- Allows the ALU to perform bitwise operations in addition to arithmetic.

Output Latch (74LS245 Bus Transceiver)

- Buffers ALU output onto the shared system bus only when enabled.
- Provides tri-state isolation to prevent bus contention.
- LED indicators connected in parallel to visualize ALU results in real time.

6.3.3. Supported Operations

- **Addition (A + B)** – Standard binary addition with optional carry.
- **Subtraction (A – B)** – Achieved by inverting B through XOR gates and setting carry-in high.
- **Bitwise AND / OR / NOT / XOR** – Logical operations implemented via dedicated gate circuitry.
- **Pass-Through** – ALU can directly forward operands under certain control conditions.

6.3.4. Indicators and Flags

- **Result LEDs**: 8 LEDs tied to the ALU output lines for real-time visibility of computation results.
- **Carry Flag (C)**: Taken from the MSB carry-out; in subtraction mode indicates borrow.

- **Zero Flag (Z)**: Asserted when all ALU outputs are zero, detected via NOR/AND gate logic.

6.3.5. ALU Schematic

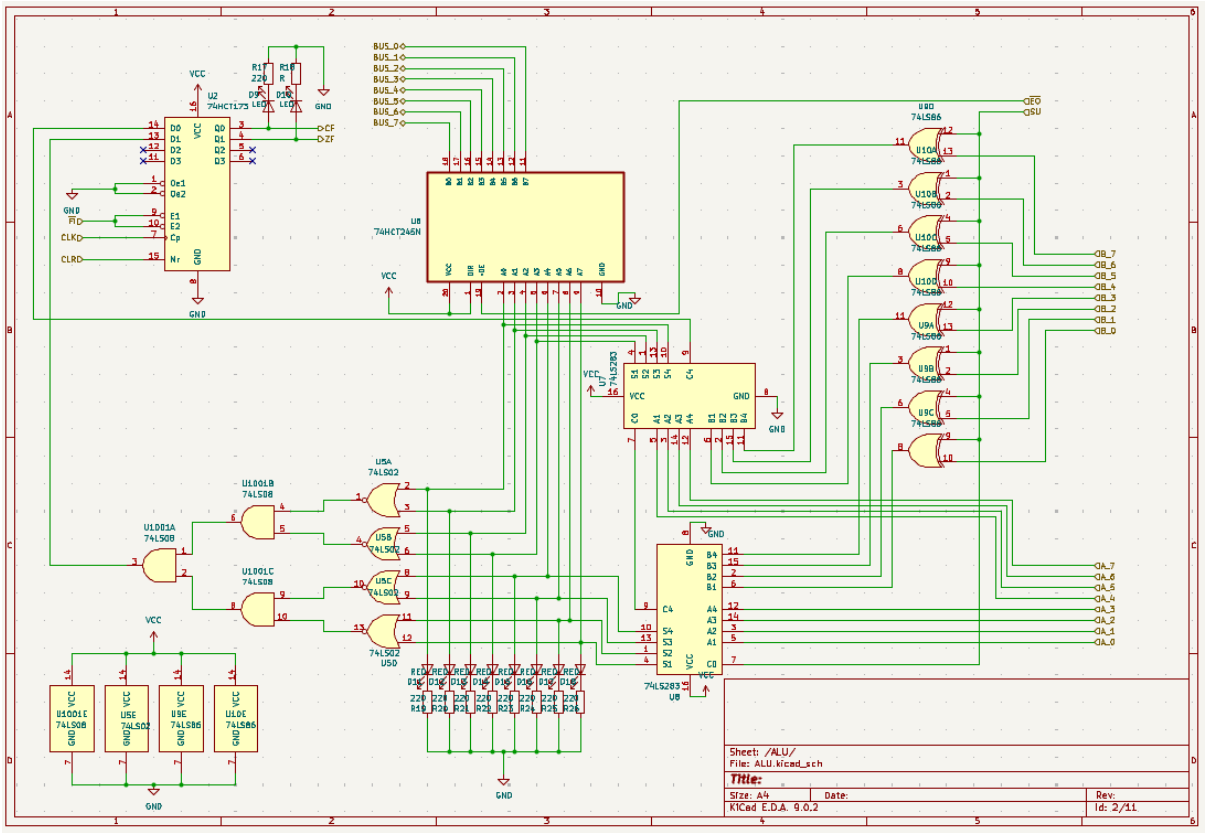

Figure 6: Schematic of the ALU created on KiCAD

6.4. Random Access Memory (RAM)

6.4.1. RAM Design Overview

Two 74189 static RAMs form 16×8 (replicated to 128 bytes total across the design), with data inversion corrected using 74LS04 inverters. The RAM can be written either from front-panel switches (Program Mode) or from the CPU during execution (Run Mode).

6.4.2. Build Process

- Tie CS low to enable devices continuously; WE is asserted (active-low) only on write events.
- Address lines are shared across both RAM ICs; each stores a nibble.
- Outputs pass through inverters, then a 74LS245 to the bus.

6.4.3. Program Mode & Run Mode

- **Program Mode:** DIP switches set address and data; a debounced write button asserts WE on a clock edge to store bytes sequentially.
- **Run Mode:** The CPU's PC and control unit own the bus; DIP inputs are ignored.

6.4.4. Adding Selection Logic

74LS157 multiplexers select between DIP-switch inputs and bus-sourced address/data, controlled by a front-panel Program/Run switch. External pull-ups prevent floating inputs.

6.4.5. RAM schematic

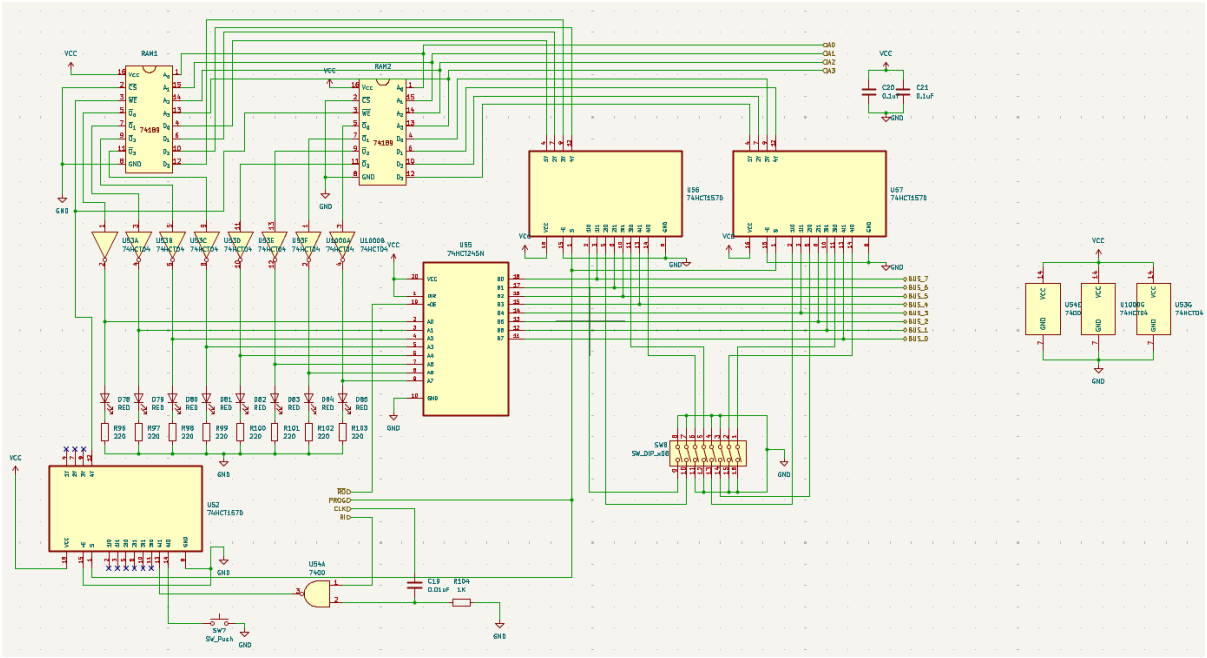

Figure 7: Schematic of the RAM module created on KiCAD

6.5. Program Counter (PC)

6.5.1. Program Counter Overview

Program counters keep track of which instruction is currently being executed and where the next one will be found. At the beginning of program execution, the counter starts at address 0. With each instruction cycle, it advances step by step, pointing to address 1, then 2, and so on. When a jump instruction occurs, the counter is forced to a new value, so execution continues from a specific memory location instead of sequential order.

In this design, the program counter is implemented using a 74HCT161 4-bit synchronous counter combined with a 74HCT245 bus transceiver. The counter handles the sequential incrementing of addresses, while the transceiver manages safe communication between the PC and the system bus. Much like other registers in the CPU, the PC is equipped with control inputs:

- A **Load** signal, which allows a new address (for jumps or branches) to be written directly into the counter.
- An **Enable** signal, which permits controlled incrementation rather than advancing on every clock pulse.
- A **Clear/Reset** line, which initializes the counter back to zero before execution begins.

For monitoring purposes, LEDs are connected to the output lines of the counter, providing a visual indication of the current address in binary form. This makes it possible to trace the program flow in real time as the counter advances or changes due to a jump.

5.5.2. Program Counter Schematic

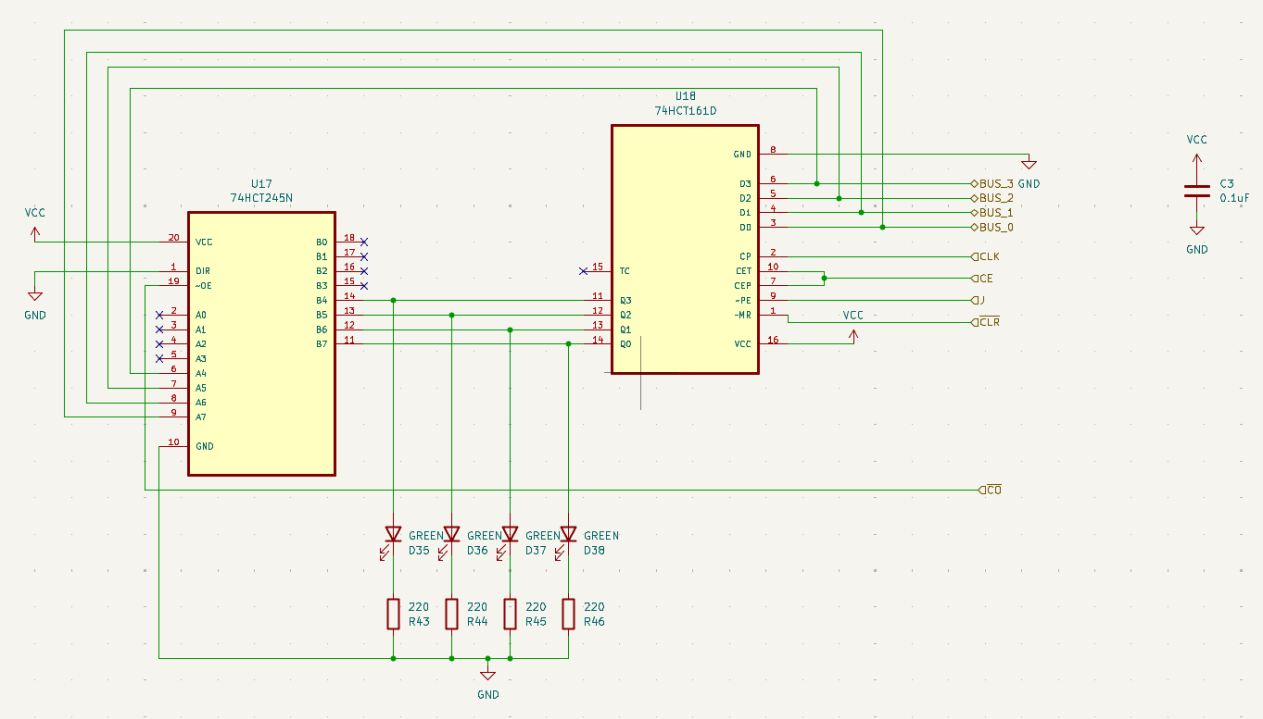

Figure 8: Schematic of the Program Counter module created on KiCAD

6.6. Output Register

6.6.1. Output Register Design Overview

The output register provides the interface between the CPU's internal 8-bit data bus and a human-readable display system. Its primary function is to latch the results of computations and present them on a set of 7-segment displays, enabling real-time observation of CPU operations.

6.6.2. Main Components

The Output Module converts binary results from the CPU into a human-readable decimal display. It consists of the following key components:

- **Latch/Output Register (74HC273)**
  An octal D-type flip-flop that captures 8 bits from the CPU bus on the rising clock edge when the load signal (ELD) is active. It provides a stable output to the display system, independent of bus activity.
- **EEPROM (28C16)**
  Functions as a programmable lookup table for binary-to-7-segment decoding. By mapping each 8-bit binary input to the proper segment drive pattern, the EEPROM replaces hardwired combinational logic and supports flexible signed/unsigned modes.
- **Multiplexing & Display Driver Logic**
  A 555 timer generates a high-frequency scan clock. Two 74LS76 JK flip-flops form a 2-bit counter, and a 74ACT139 decoder activates one digit at a time. This ensures only one 7-segment display is driven during each scan cycle while maintaining the appearance of continuous illumination through persistence of vision.
- **7-Segment Display Array (CA56-12CGWA)**
  A four-digit array used to show results. Three digits represent numerical output, with the fourth optionally reserved for a sign indicator.

Together, these components form the pathway from bus data capture to final display, translating raw binary into a readable output.

6.6.3. Operation Sequence

The sequence of operations in the Output Module proceeds as follows:

1. **Data Capture**
   The CPU places a result on the data bus. Control logic asserts the load signal (ELD), clocking the value into the 74HC273 output register.
2. **Binary-to-Display Decoding**
   The latched value is presented to the 28C16 EEPROM, which converts the binary number into corresponding 7-segment patterns. Mode inputs allow selection between signed and unsigned representations.
3. **Digit Multiplexing**
   The 555 timer advances the 2-bit counter, cycling through digits. The 74ACT139 decoder enables one display at a time, while the EEPROM updates its outputs in sync with the scan.
4. **Final Output**
   The four 7-segment digits update rapidly enough to appear continuously lit, showing the CPU's 8-bit result as a decimal number (with sign if enabled). LEDs alongside the output register provide additional binary visibility.

6.6.4. Design Benefits:

- **Efficient resource usage:** Instead of multiple decoders, one EEPROM provides flexible segment decoding for all digits.
- **Flexible formats:** Signed/unsigned display selectable via mode control.
- **Human-readable results:** Numeric display allows easy verification of arithmetic programs.
- **Isolation:** Output register prevents bus contention, as it only reads from the bus and does not write back.

6.6.5. Output Register Schematic:

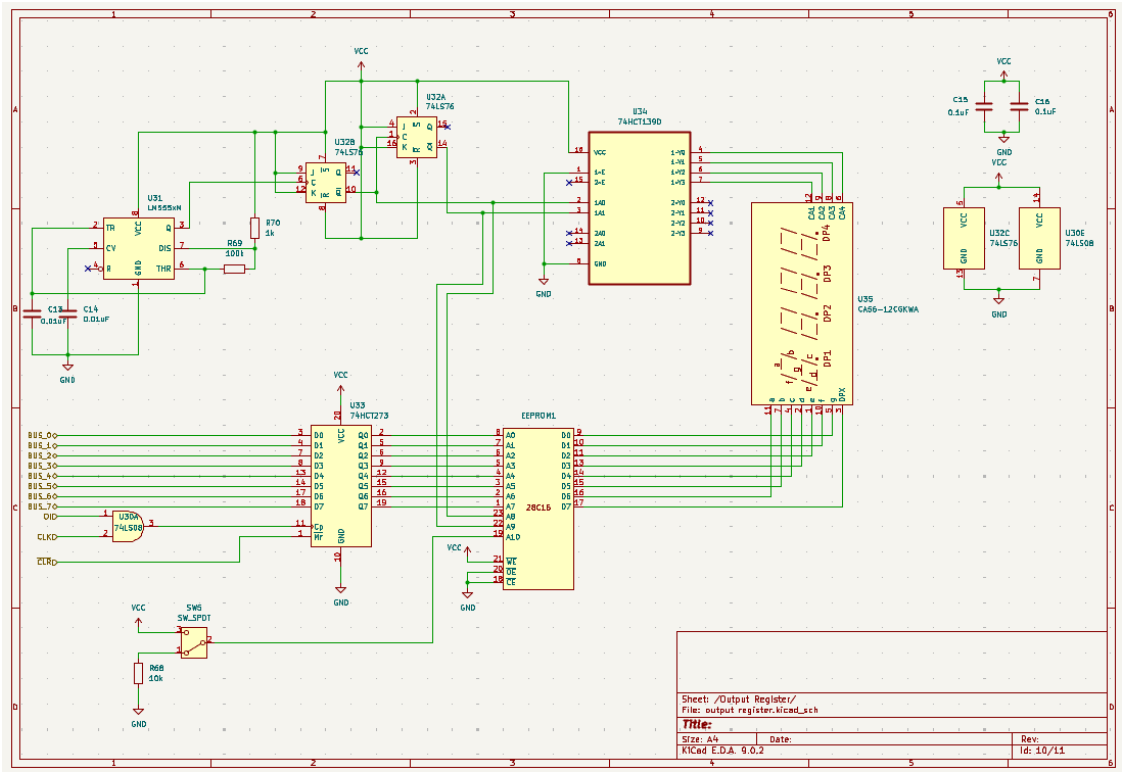

Figure 9: Schematic of the Output Register created on KiCAD

6.7. Control Logic

6.7.1. Control Logic Design Overview

The control logic is the heart of the CPU, orchestrating every step of instruction execution. It interprets opcodes, generates timing signals, and produces the micro-operations that direct data flow across the bus. This schematic implements a microcoded control unit using EEPROMs, counters, and decoding logic. The microcoded control unit converts each instruction into 5 micro-steps (expandable to 8) using EEPROM lookup.

6.7.2. Purpose and Function

- Provides fetch–decode–execute sequencing for all instructions.
- Translates instruction opcodes + step counter + flags into control line activations.
- Ensures only the correct modules (registers, ALU, RAM, PC) are enabled at the right moment.
- Allows for conditional execution through flag-dependent branching.

6.7.3. Core Components

1. **Instruction Register (IR)** – Built from a 74HC173 register.
    a. Latches the 8-bit instruction fetched from memory.
    b. Splits into opcode (upper nibble) and operand/address (lower nibble).
2. **Step Counter (74HC161 + Decoder)**
    a. Advances with each clock cycle, generating micro-steps for instruction sequencing.
    b. Typically cycles through 5 microinstructions per instruction.
    c. Decoder provides visible step states via LEDs.
3. **EEPROMs (28C16 × 2)**
    a. Store the microcode that maps (opcode + step + flags) → (control signals).
    b. One EEPROM outputs the first 8 control lines, the second drives the remaining 8.
    c. Easily reprogrammable to expand or refine the instruction set.
4. **Flag Register & Logic**
    a. Zero and Carry flags derived from ALU outputs.
    b. Stored in a latch (e.g., 74HC173).
    c. Fed into EEPROM address lines to enable conditional jumps (JC, JZ).
5. **Output Indicators (LEDs + Drivers)**
    a. Each control line is tapped to LEDs for visualization.
    b. Active-low lines normalized with inverters for consistency.
    c. Allows real-time monitoring of instruction sequencing.

6.7.4. Operation

1. **Fetch Phase**
    a. Step counter triggers control lines to load PC → MAR, then RAM → IR.
    b. Instruction opcode is presented to the EEPROM address inputs.
2. **Decode Phase**
    a. Opcode bits combine with step counter state and flags to select a row in microcode.
    b. EEPROM outputs the corresponding control signals.
3. **Execute Phase**
    a. Depending on the instruction, EEPROM asserts signals to enable ALU ops, register loads, memory writes, or program counter jumps.
    b. Step counter resets when instruction completes, looping back to the next fetch cycle.

6.7.4. Control Logic Schematic

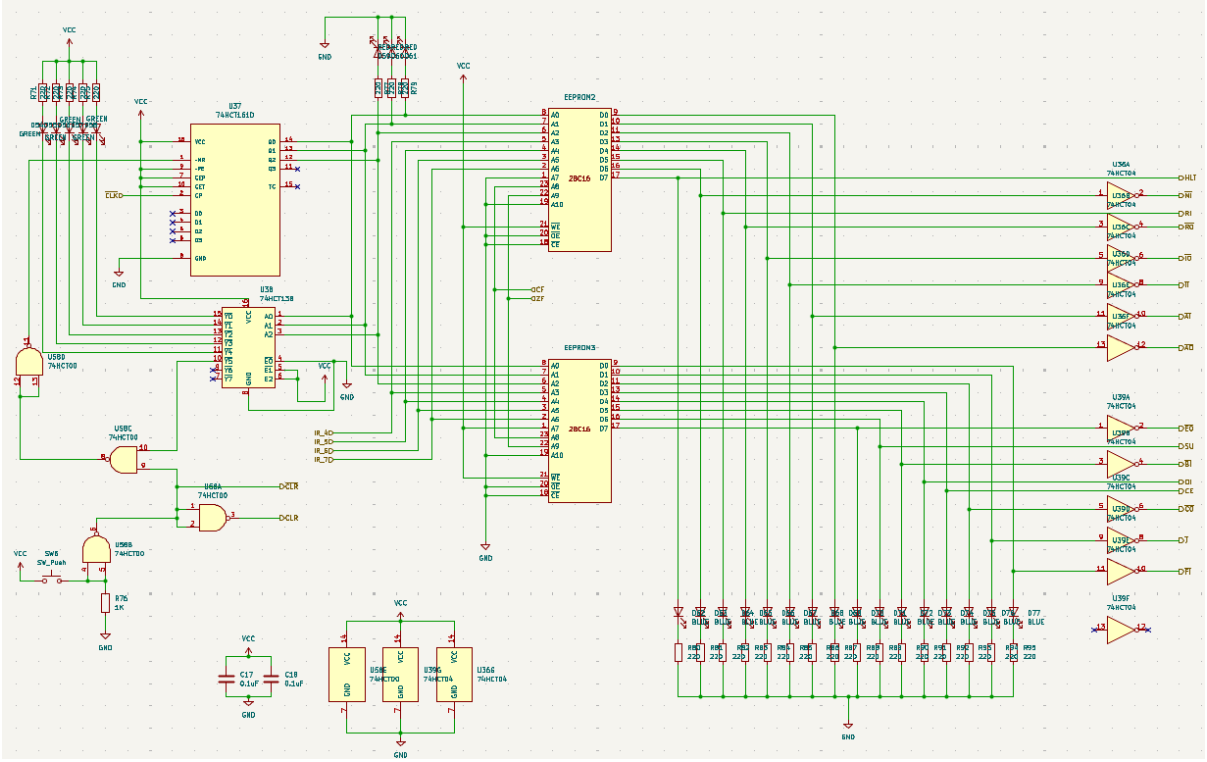

Figure 10: Schematic of the Control Logic module created on KiCAD

6.8. Complete 8-bit CPU

The complete schematic (shown in Figure 11) brings together all modules of the 8-bit CPU into a single diagram, showing how the system achieves the fetch–decode–execute cycle in hardware. Each block described previously—Clock, Registers, ALU, RAM, Program Counter, Output, and Control Logic—is wired to the shared 8-bit bus, with transceivers and control lines ensuring proper sequencing.

6.8.1. How the Circuit Works Together

1. **Clock and Control Logic**
    a. The clock provides the system's timing, advancing each micro-operation step by step.
    b. The control logic, implemented with EEPROMs and a step counter, interprets the current instruction (via the Instruction Register) and issues control signals to the bus and modules.
2. **Instruction Flow**
    a. The Program Counter outputs the address of the next instruction onto the bus.

   b. The Memory Register and RAM fetch the instruction, which is latched into the Instruction Register.
   c. The Instruction Register splits into opcode and operand: the opcode drives the control logic, while the operand can be placed on the bus as an address or data value.
3. **Data Flow**
   a. Operands are loaded into the A and B registers from the bus.
   b. The ALU performs arithmetic or logic on the values from A and B, and places the result back onto the bus when enabled.
   c. Results can be written to memory, moved into registers, or latched into the Output Register.
4. **Output**
   a. The Output Register captures bus values and drives LEDs and 7-segment displays, giving real-time visibility into program execution.

6.8.2. Important Notes

- The **shared bus** is the backbone: only one device may drive it at a time, controlled by enable lines.
- The **control unit** is microcoded: each instruction is broken down into micro-steps stored in EEPROM.
- **Flags (Zero, Carry)** from the ALU feed back into the control logic, enabling conditional instructions (e.g., JZ, JC).
- LEDs throughout the circuit provide debugging insight into register contents, bus traffic, and control signals.

Together, these components form a complete, working 8-bit CPU capable of executing programs in real hardware.

6.8.3. Complete Schematic of the 8-bit CPU

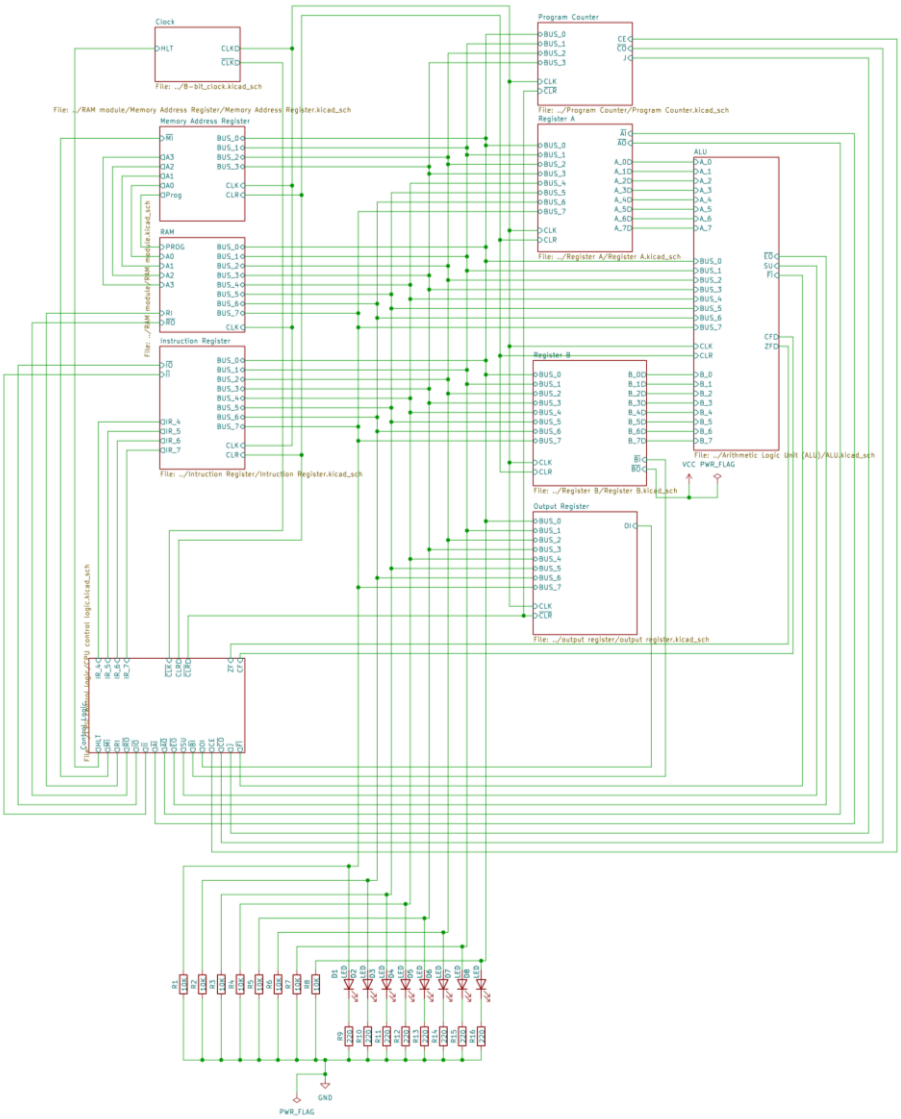

Figure 11: Schematic of all previous modules wired together created in KiCAD

6. OPERATING INSTRUCTIONS

1. Connect a regulated **5 V** supply to the PCB.
2. Select **Program Mode**; use DIP switches to set address and data; press **WRITE** to store bytes.
3. Switch to **Run Mode**.
4. Choose **Manual** (single-step) or **Continuous** clock.
5. Press **RESET** to clear PC and registers; start the clock.

6. Observe execution on LEDs and 7-segment displays.

7. SAMPLE PROGRAM & EXPECTED OUTPUT

Program

LDA 14
ADD 15
OUT
HLT

Behavior
The CPU fetches operands from addresses 14 and 15, sums them in the ALU, latches the result into the Output Register, and displays the value on LEDs and the 7-segment display, then halts.

8. LIMITATIONS

- **Memory Size:** 128 bytes total.
- **Speed:** Bounded by 74LS propagation and board wiring (~100 kHz practical).
- **Debugging Complexity:** Longer programs require careful tracing of micro-steps.
- **EEPROM Wear:** 28C16 devices have finite write endurance; re-microcoding should be done judiciously.

9. CURRENT BUILD (Notes & Improvements)

- Currently in the process of migrating from breadboard prototype to KiCad-routed PCB.
- Added extensive **decoupling** and **bypass capacitors** for supply stability.
- Normalized control signals and added **comprehensive LED instrumentation**.
- Built an **Arduino Nano–based EEPROM programmer** to streamline microcode updates.

10. CONCLUSION

This 8-bit CPU re-creates the classic von Neumann pipeline with discrete TTL and a microcoded control store, offering a transparent view of instruction execution. The adjustable clock, step counter, and rich LED/7-segment instrumentation make it a powerful teaching platform for computer architecture and digital design. Despite constraints in memory and speed, the system reliably executes a compact instruction set and can be extended with additional microcode, I/O devices, and addressing modes.