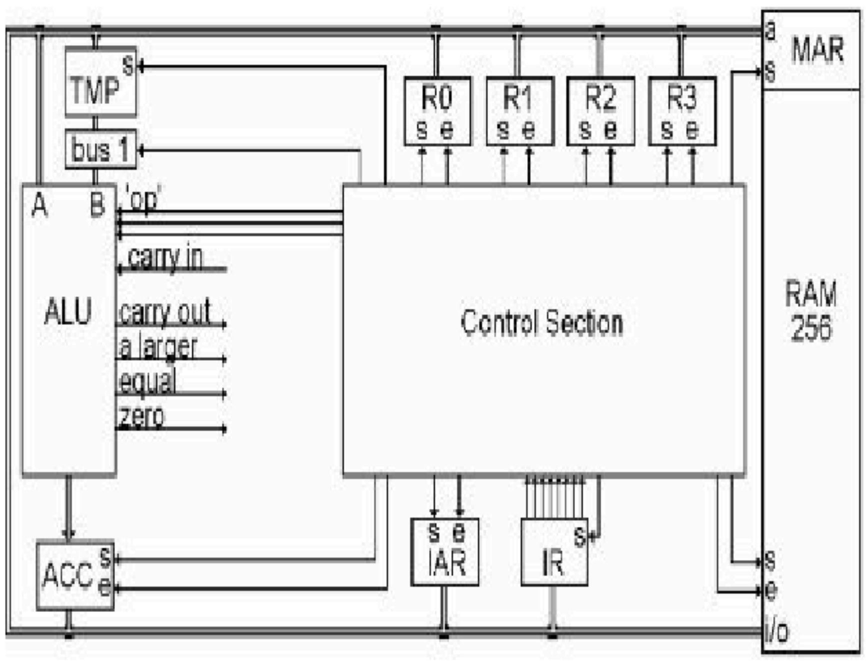
Building a Computer from Scratch in Logisim 🖥️🔧

Building a computer from scratch in Logisim is like setting up a magical kitchen where you can create various dishes (perform tasks) by following recipes (instructions). We've already established the fridge—known as RAM (Random Access Memory)—to store these recipes and ingredients. Now, it's time to construct the control unit, the kitchen's brain, which will read the recipes, make decisions, and ensure everything happens in the correct order. 🧠🍽️

Let's examine the overall structure using the diagram of our computer to understand how the RAM and control unit collaborate! 📊🤝



What's in the Diagram? 📐

The diagram illustrates our computer (CPU) with all its components connected:

- **RAM (256)**: The fridge on the right, capable of holding 256 locations for recipes and ingredients (each location stores an 8-bit number, like 00001111). 📦🔢
- **Control Section**: The central box is the control unit we're constructing. It acts as the head chef managing the kitchen. 👨‍🍳
- **Registers (R0, R1, R2, R3, TMP, ACC, MAR, IAR, IR)**: These small boxes surrounding the control unit function as storage jars in the kitchen, holding ingredients (numbers) we work with: 🏺🔢

    - **R0–R3**: General-purpose jars for numbers.
    - **TMP**: A temporary jar for holding numbers during processing.
    - **ACC**: The accumulator, where the cooking result is kept (like a plate for the finished dish). 🍽️
    - **MAR**: The memory address register, directing which fridge spot to access. 🗺️
    - **IAR**: The instruction address register, tracking the next recipe to read. 📚
    - **IR**: The instruction register, holding the current recipe being executed. 📝
- **ALU**: The oven on the left, performing arithmetic operations (like adding or comparing numbers) with two ingredients (A and B) and a method (op, such as ADD or NOT). It includes flags (carry in, carry out, larger, equal, zero) to indicate outcomes like "Did the sum exceed 255?" or "Are these numbers equal?" 🔢➕➖
- **Bus (Thick Lines)**: The conveyor belt transporting numbers (8-bit data) between the fridge, jars, and oven. 📦➡️🏺
- **Bus 1**: A special jar persistently holding the number 1, used for incrementing (like adding 1 to IAR for the next recipe). 🔢➕

What Did We Build So Far? The RAM (Fridge) 🧊

Our RAM resembles a large fridge with 256 shelves, each holding an 8-bit number (a byte, like 00001111). We can:

- **Store a number in the fridge**: Inform the fridge which shelf to use (via MAR) and place a number there (similar to storing an ingredient on shelf 10). 🗂️
- **Load a number from the fridge**: Direct the fridge to a specific shelf (using MAR), retrieving the number there (like picking an ingredient from shelf 10). 🥄

The RAM connects to the conveyor belt (bus), facilitating the movement of numbers to and from the fridge using the control unit. 🔄

What Are We Building Now? The Control Unit (Head Chef) 👨‍🍳

The control unit serves as the head chef ensuring the kitchen runs smoothly. It:

1. Retrieves a recipe from the fridge (RAM) and places it in the recipe jar (IR). 📜
2. Interprets the recipe to determine the necessary actions (like "Add two numbers" or "Retrieve an ingredient from the fridge"). 🤔
3. Selects the appropriate ingredients (from jars R0–R3) for use. 🏺
4. Performs the task:
    - For math operations (like ADD), it employs the oven (ALU). ➕➖
    - For retrieving or storing (Load or Store), it utilizes the fridge (RAM). 🧊
    - For acquiring a special ingredient (Data), it fetches it from the fridge and advances to the next recipe.
5. Resets the kitchen timer (stepper) to commence the next recipe. ⏲️

The control unit uses a timer (stepper) to sequence these steps (S1 to S12), akin to a clock ticking through tasks: "Step 1: Retrieve the recipe, Step 2: Interpret it, Step 3: Execute the task, etc." ⏰

How Do RAM and the Control Unit Work Together? 🤝

- **RAM Holds the Recipes and Ingredients**:
  - We insert a list of recipes (a program) in the fridge (RAM), like “Add the numbers in jars R0 and R1, then store the result on shelf 10.” 🍱🥗
  - We also store ingredients (numbers) in the fridge, such as “The number 5 is on shelf 20.” 🔢
- **The Control Unit Uses the RAM to Follow the Recipes**:
  - The control unit inspects the IAR address (like checking which recipe to read next, e.g., recipe #3 on shelf 3). 📚
  - It instructs the fridge (via MAR) to fetch that recipe and places it in IR. 📝
  - It interprets the recipe (IR) to decide the course of action:
    - If the recipe states “Load from shelf 20,” it directs the fridge to retrieve the number (5) from shelf 20 and deposits it in a jar (like R2). 🥄
    - If the recipe states “Store to shelf 10,” it transfers a number from a jar (like R0) and stores it on shelf 10 in the fridge. 📦
- **The Conveyor Belt (Bus) Connects Them**:
  - The bus serves as the conveyor belt, moving numbers between the fridge (RAM), jars (registers), and oven (ALU). The control unit ensures the correct numbers move at the appropriate time. ➡️

Why Is This Cool? 😎

- The RAM (fridge) operates as the storage room where all our recipes and ingredients reside. 🧊🔢
- The control unit (head chef) is the intelligent component that reads the recipes, determines actions, and ensures the kitchen (CPU) executes the program flawlessly. 👨‍🍳👩‍🍳
- Together, they convert a list of recipes (a program) into actions—like number addition, result storage, or special value retrieval—bringing our computer to life! 🚀

What’s Next? 🛠️

Having grasped the overall picture, we’ll delve into the specifics of the control unit. We’ll dissect it into 7 smaller stations (circuits) collaborating to enable the head chef (control unit) to perform its duties. Each station has a unique task, such as fetching recipes, interpreting them, or cooking in the oven. Let’s explore how they all interconnect in the next section! 🔄➡️

Understanding the Control Unit of a CPU

The control unit acts as the "brain" of the CPU, orchestrating the fetch and execution of instructions stored in RAM. It uses a stepper—a 12-step state machine (S1–S12)—to sequence operations, ensuring each instruction is processed systematically. The control unit comprises 7 circuits, each handling a specific task. This document explains how these circuits connect and interact to form a fully functional control unit, enabling the CPU to execute programs.

Components of the Control Unit

1. **Registers**:
   - R0–R3 (general-purpose)
   - TMP (temporary)
   - ACC (accumulator)
   - MAR (memory address)
   - IAR (instruction address)
   - IR (instruction)
2. **Memory**:
   - RAM (256 bytes, 8-bit)
3. **ALU**:
   - 8-bit arithmetic/logic unit with 3-bit operation input
4. **Bus**:
   - 8-bit shared data path
5. **Stepper**:
   - 12 steps (S1–S12), outputting one-hot signals
6. **Clock Signals**:
   - *clk e* (enable)
   - *clk s* (set) for timing

The 7 Circuits and Their Connections

Fetch Circuit (Steps 1–3) 🚀

- **Role**: Fetches the next instruction from RAM into the IR and increments the IAR.
- **Connections**:
  - Inputs: IAR (current instruction address), RAM (stores instructions), Bus 1 (constant 1 for IAR + 1)
  - Outputs: IR (holds the fetched instruction), IAR (updated address)
  - Interacts with: Stepper (S1–S3 signals), ALU (for IAR + 1), bus (data path)
- **How It Fits**: This circuit starts the fetch-execute cycle by loading the instruction into the IR, which the Instruction Decode Circuit then reads. It updates the IAR, preparing for the next fetch.

Instruction Decode Circuit 🔍

- **Role**: Decodes the instruction in the IR to determine the operation (ALU or non-ALU).
- **Connections**:
  - Inputs: IR (bits 0–3)
  - Outputs: Decoder outputs (0 = Load, 1 = Store, 2 = Data), IR bit 0 (ALU indicator)
  - Interacts with: ALU Execute Circuit (passes IR bit 0 and bits 1–3 for ALU ops), Load/Store and Data Execute Circuits (passes decoder outputs)

- **How It Fits**: Acts as the decision point, routing the instruction to the appropriate execution circuit (ALU, Load/Store, or Data) based on IR bits.

Register Selection Circuit 🔧

- **Role**: Selects registers (R0–R3) for Reg A and Reg B using IR bits 4–7.
- **Connections**:
    - Inputs: IR (bits 4–5 for Reg A, bits 6–7 for Reg B)
    - Outputs: Reg A and Reg B selection signals (R0–R3 Enable/Set)
    - Interacts with: ALU Execute, Load/Store, and Data Execute Circuits (provides register selection signals)
- **How It Fits**: Prepares the registers needed for execution, ensuring the correct data is accessed by downstream circuits.

ALU Execute Circuit (Steps 4–6) ➕

- **Role**: Executes ALU instructions (e.g., ADD, CMP) when IR bit 0 = 1.
- **Connections**:
    - Inputs: IR bit 0, bits 1–3 (ALU op), Reg A and Reg B (via Register Selection), Stepper (S4–S6)
    - Outputs: ACC (result), Reg B (updated value)
    - Interacts with: Instruction Decode (receives IR bit 0 and bits 1–3), Register Selection (accesses Reg A and Reg B), ALU (performs operation), bus (data path)
- **How It Fits**: Handles arithmetic/logic operations, using the ALU to process data from registers and storing the result, completing the execution phase for ALU instructions.

Load/Store Execute Circuit (Steps 4–5) 📥📤

- **Role**: Executes Load (RAM to register) and Store (register to RAM) instructions.
- **Connections**:
    - Inputs: Decoder outputs 0 (Load) and 1 (Store), Reg A and Reg B (via Register Selection), Stepper (S4–S5)
    - Outputs: RAM (for Store), Reg B (for Load)
    - Interacts with: Instruction Decode (receives decoder outputs), Register Selection (accesses Reg A and Reg B), RAM (reads/writes data), bus (data path)
- **How It Fits**: Manages memory operations, interfacing with RAM to load or store data, completing the execution phase for Load/Store instructions.

Data Execute Circuit (Steps 4–6) 📊

- **Role**: Executes the Data instruction, loading a value from the next RAM address into a register and skipping the data byte.
- **Connections**:
    - Inputs: Decoder output 2 (Data), IAR (current address), Reg B (via Register Selection), Stepper (S4–S6)
    - Outputs: Reg B (loaded value), IAR (updated address)
    - Interacts with: Instruction Decode (receives decoder output 2), Register Selection (accesses Reg B), Fetch Circuit (uses IAR, similar fetch-like steps), RAM (reads data), bus (data path)

- **How It Fits**: Handles special data loading, fetching a value and updating IAR, completing the execution phase for Data instructions.

Reset Circuit (Step 7) 🔄

- **Role**: Resets the stepper to S1 after execution.
- **Connections**:
  - Inputs: Stepper (S7)
  - Outputs: Stepper Reset signal
  - Interacts with: Stepper (resets to S1), Fetch Circuit (prepares for the next fetch)
- **How It Fits**: Closes the fetch-execute cycle by resetting the stepper, ensuring the CPU is ready to fetch the next instruction.

How the Circuits Form the Control Unit

Flow of Operation 🌐

1. **Fetch Circuit (S1–S3)**: Starts the cycle by loading an instruction from RAM into IR and incrementing IAR.
2. **Instruction Decode Circuit**: Reads IR, deciding whether it's an ALU instruction (bit 0 = 1) or a non-ALU instruction (bit 0 = 0, decoded into Load, Store, or Data).
3. **Register Selection Circuit**: Prepares the registers (Reg A and Reg B) needed for the instruction, based on IR bits 4–7.
4. **Execution**:
   - ALU Execute Circuit (S4–S6): Handles ALU instructions, performing operations like ADD and storing results.
   - Load/Store Execute Circuit (S4–S5): Handles memory operations, reading from or writing to RAM.
   - Data Execute Circuit (S4–S6): Handles data loading, fetching a value and skipping to the next instruction.
5. **Reset Circuit (S7)**: Resets the stepper to S1, restarting the cycle for the next instruction.

Shared Resources and Control Signals

- **Shared Resources**:
  - The bus is the shared data path, used by all circuits to move data between registers, RAM, and ALU.
  - The stepper provides timing signals (S1–S12), coordinating the actions of all circuits.
  - Registers (e.g., R0–R3, IR, IAR) and RAM are accessed by multiple circuits, with the Register Selection Circuit ensuring the correct registers are used.
- **Control Signals**:
  - Each circuit generates Enable and Set signals (e.g., Reg A Enable, RAM Set) using AND gates with stepper signals (S1–S7) and clk e/clk s.
  - OR gates combine these signals when a component (e.g., R0 Enable) is used by multiple circuits (e.g., Fetch, ALU Execute, Load/Store).

Conclusion

Together, these 7 circuits form a cohesive control unit that fetches instructions, decodes them, selects the necessary registers, executes the appropriate operation (ALU, Load/Store, or Data), and resets for the next cycle. The stepper ensures proper sequencing, while the bus and shared components enable data flow between circuits. This design allows the CPU to execute a program

stored in RAM, step by step, turning binary instructions into meaningful actions.