

**32-bit Single Cycle MIPS Processor**

Achraf Mohamed Abobakr B.N.:12

Zyad Billal Gamal B.N.:25

Mohad Mohamed Hosni B.N.:50

Youssef Rashad Farouk B.N.:58

Youssef Mohamed Saeed B.N.:60



supervised by:

DR: Lamiaa Elrefaae

ENG: Shaimaa Yosry

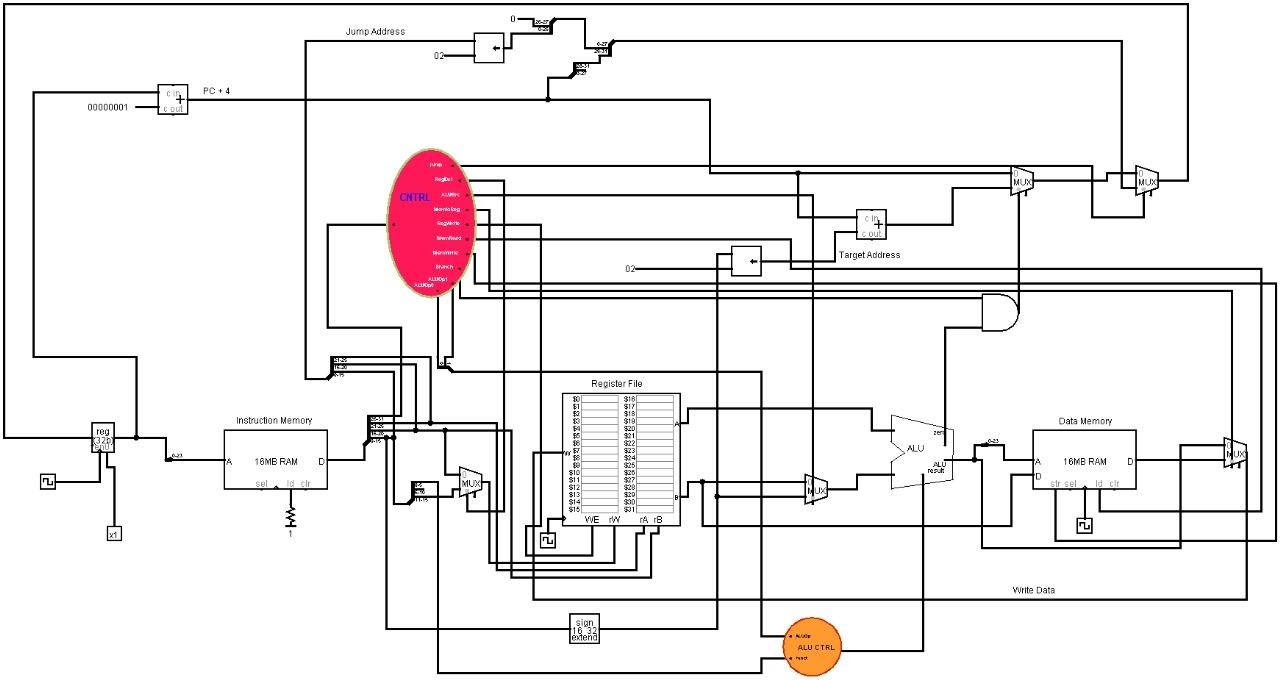
32-bit Single Cycle MIPS Processor

**Introduction:**

This project showcases a single-cycle MIPS processor implementation using Logisim, focused on supporting core R-type and J-type instructions. The processor executes each instruction in a single clock cycle, combining the instruction fetch, decode, execute, and write-back stages into one step.

Key components of the processor include the program counter (PC), instruction memory, control unit, register file, ALU (Arithmetic Logic Unit), and necessary multiplexers and adders for control and data path routing. The processor supports basic R-type operations (such as add, sub, and, or) beq sw lw from the I-format and the jump (j) instruction from the J-type format.

The design simplifies the processor architecture making it a focused and efficient model for understanding core datapath operations in a MIPS architecture. This implementation serves as a solid foundation for extending to multi-cycle or pipelined designs in future work.



* Design and Implementation

# Instruction Execution Stages in a Single-Cycle MIPS Processor:

This single-cycle MIPS processor executes each instruction in one clock cycle by passing it through five key stages. Below is a description of each stage:

## Instruction Fetch (IF)

In this stage, the processor retrieves the next instruction to execute. The Program Counter (PC) provides the address to the Instruction Memory, which outputs the 32-bit instruction. After fetching, the PC is incremented by 4 to point to the next instruction in sequence.

## Instruction Decode / Register Read (ID)

The fetched instruction is decoded to determine the type of operation and which registers are involved. The Register File is accessed to read the values of two source registers (rs and rt) as specified in the instruction. Simultaneously, the Control Unit interprets the opcode and generates the appropriate control signals for the following stages.

## Execution (EX)

During execution, the Arithmetic Logic Unit (ALU) performs the required operation on the register operands. For R-type instructions, the specific function (such as add, sub, and, or, slt) is selected based on the function code and ALU Control signals. The ALU produces a result that will either be stored or written back.

## Memory Access (MEM)

In this stage, memory instructions would access the Data Memory. The address computed by the ALU is used to read or write data. However, in this specific implementation—which supports only R-type and J-type instructions—this stage is generally inactive or bypassed.

## Write Back (WB)

The final stage involves writing the result of the operation back into the Register File. For R- type instructions, the ALU result is written to the destination register (rd). This occurs only if the RegWrite control signal is active, ensuring correct data flow.

## Datapath Overview:

The datapath of the single-cycle MIPS processor consists of interconnected components that collectively carry out instruction execution. Major elements include:

***Program Counter (PC):*** Holds the address of the current instruction.

***Instruction Memory (IM):*** Outputs the instruction located at the address provided by the PC.

***Register File:*** Reads source operands and writes back results.

***Control Unit:*** Generates control signals based on the instruction opcode. ***ALU Control:*** Determines the specific operation the ALU should perform. ***ALU:*** Executes arithmetic and logic operations.

***Data Memory (optional):*** Used for load/store instructions (not active in this implementation).

***Multiplexers and Adders:*** Direct and combine data and control paths as needed.

Each stage passes its results to the next through this datapath, allowing the processor to complete one instruction per clock cycle efficiently.

# Components:

The main components of the CPU include:

**Program Counter (PC) Register** – Holds the address of the current instruction and is updated each cycle based on the instruction type (sequential, jump, or branch).

**Instruction Memory** – Stores and provides the instruction at the address specified by the PC.

Register File – Contains 32 general-purpose registers. It supports two simultaneous reads and one write per cycle.

**ALU (Arithmetic Logic Unit)** – Executes arithmetic and logical operations based on the instruction type.

**ALU Control** – Determines the specific operation the ALU should perform, based on the instruction's function field and ALUOp control signals

**Data Memory** – Used for load (lw) and store (sw) instructions to read from or write to memory.

**Main Control Unit** – Decodes the instruction opcode and generates control signals for the rest of the datapath.

**Multiplexers (MUXes)** – Used throughout the design to select between data sources (e.g., ALU vs. memory result, PC+4 vs. branch/jump address), replacing the need for a separate "PC control" module.

# control:

There are three main control components in the CPU: the Main Control Unit, the ALU Control Unit, and the PC update logic implemented through multiplexers.

The Main Control Unit interprets the opcode of the current instruction and generates the necessary control signals to guide the datapath elements. These signals determine actions such as register writing, memory access, ALU operations, and branching. Key outputs include RegWrite, MemRead, MemWrite, MemtoReg, ALUSrc, ALUOp, Jump, and Branch.

The ALU Control Unit takes in the ALUOp signal from the Main Control Unit and the function field (funct) from R-type instructions. Based on these, it selects the appropriate ALU operation (e.g., add, sub, and, slt). For I-type instructions, it uses the opcode directly to configure the ALU operation. The ALU Control also determines the inputs to the ALU: operand A comes from a register, and operand B comes either from a register or an immediate value, depending on the ALUSrc control signal.

Instead of a separate PC Control unit, the next instruction address is determined using a combination of multiplexers and control signals (Jump, Branch, and the ALU's Zero signal). These elements decide whether the PC should increment normally (PC + 4), jump to a target address, or branch to a new location in the case of a beq instruction. The selected address is then written back into the PC register for the next cycle.

This control scheme allows the processor to manage instruction flow effectively while simplifying the overall design.

# Implementation Details:

Our CPU was implemented using a modular single-cycle architecture, with each major component connected to form a complete datapath capable of executing a subset of MIPS instructions. Key implementation points include:

The PC register stores the current instruction address and is updated each clock cycle. It receives input from a multiplexer that selects between PC + 4, a branch target (for beq), or a jump address, based on control signals such as Jump and Branch, and the ALU's Zero signal.

The Instruction Memory takes the PC value and outputs the corresponding instruction, which is then decoded by the Main Control Unit. This unit sets signals such as RegDst, MemRead, MemWrite, MemtoReg, ALUSrc, ALUOp, and branching/jump control lines.

The Register File provides two source operands to the ALU (or stores a result depending on the instruction), supporting both R-type and I-type formats.

The ALU Control Unit interprets the ALUOp signals from the main control and, for R-type instructions, uses the funct field to determine the specific ALU operation. For I-type instructions, it maps the opcode directly to an ALU function (e.g., addi uses addition).

The ALU performs computations like addition, subtraction, and logical operations. Its output is written either to the Register File or Data Memory, depending on the instruction type.

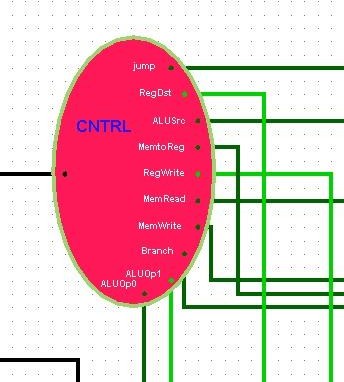
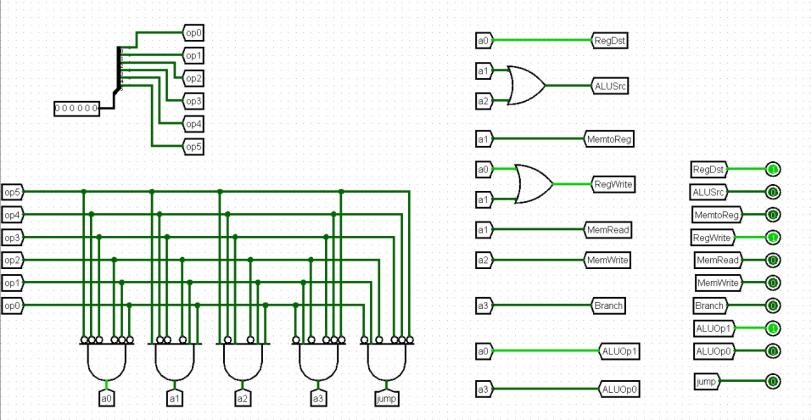
Data Memory is accessed only during lw (load word) and sw (store word) instructions. It receives control from MemRead and MemWrite signals.

We did not implement the instructions bne, xor, or or in our design. Therefore, the control logic and ALU control do not support them.

The overall datapath was implemented using basic logic components, multiplexers, memory modules, and control units. All control signals are generated directly from the instruction fields and control units, without using a separate "PC Control" module.

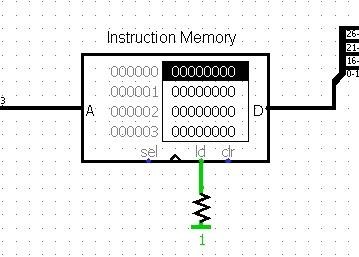
1. **Component Circuits and Overall Datapath:**

# Main Control:

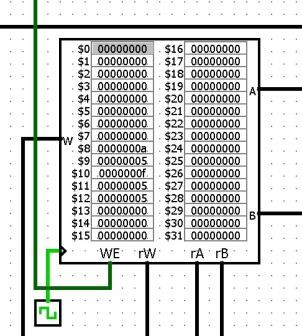
The Main Control Unit decodes the opcode field of the instruction and generates control signals for the rest of the processor. These signals determine the behavior of the datapath elements (e.g., RegDst, ALUOp, MemWrite, RegWrite) based on instruction type (R-type or J-type in your case).

# Instruction Memory:

****

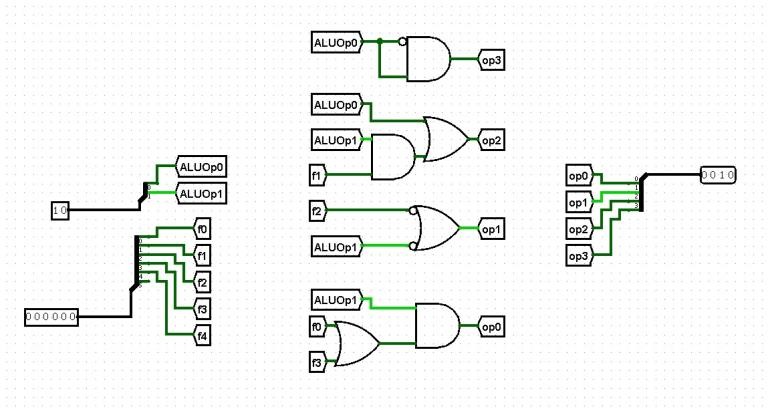
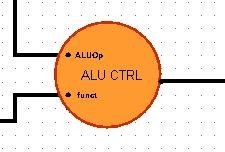
The Instruction Memory stores the program's instructions. Given a Program Counter (PC) address, it outputs the 32-bit instruction located at that address. It is read-only and does not change during execution.

# Register File:



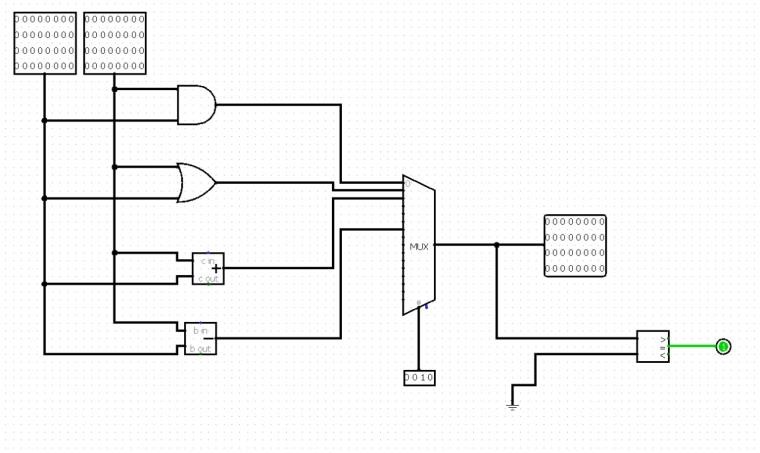
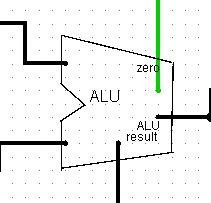
The Register File contains 32 general-purpose registers. It reads two source registers (rs, rt) and writes to one destination register (rd) if the RegWrite signal is enabled. It’s accessed in every instruction for reading and writing operands and results.

# ALU Control:

****

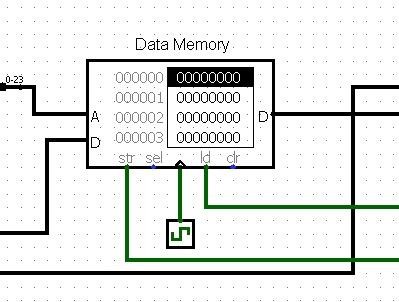
The ALU Control Unit receives ALUOp signals from the main control and the function field (for R-type instructions) from the instruction. It outputs a specific control code that tells the ALU what operation to perform (e.g., add, subtract, and, or, slt).

# ALU:

****

The ALU performs the required arithmetic or logic operation on two 32-bit inputs. It produces a 32-bit result and a Zero flag (used for branches, though not needed in your R- and J-type implementation). Operations depend on the ALU Control signal.

# Data Memory

****

The Data Memory stores and retrieves data when executing memory instructions like lw or sw. In your case (without I-type support), it may be unused. If included, it supports reading or writing data based on the MemRead and MemWrite control signals.

# Write Back

The Write Back stage is the final step in the instruction execution cycle. In this stage, the result of the computation performed by the ALU (for R-type instructions) is written back into the register file.

Conclusion:

In this project, we successfully designed and implemented a single-cycle MIPS processor using Logisim. The processor supports R-type, I-type, and J-type instructions, and follows a structured five-stage execution process: Instruction Fetch, Instruction Decode, Execution, Memory Access, and Write Back. Each instruction passes through these stages within a single clock cycle, ensuring straightforward and consistent operation.

Our processor handles a subset of MIPS instructions, including arithmetic operations, memory access (load and store), and conditional branching (such as beq), demonstrating a functional implementation of all three instruction formats. The datapath was carefully constructed to accommodate various instruction types by using control signals, multiplexers, and functional units like the ALU and register file.

This implementation highlights key computer architecture concepts such as control logic, datapath design, and instruction execution flow. It provides a solid foundation for further improvements, such as adding pipeline stages, hazard handling, or support for more complex instructions in future work.