32-Bit Custom CPU

This project implements a custom 8-bit CPU with a *29-bit data bus* and a *32-bit instruction format*, built using Logisim Evolution. It features a compact yet expressive instruction set and integrates core computing capabilities such as ALU operations, memory access, and control flow.

Project Overview

- *Instruction Width:* 32 bits
- *Data Bus Width:* 29 bits
- *Register Width:* 8 bits
- *Instruction Set:* 16 operations (0x00 to 0x0F)
- *Development Tool:* Logisim Evolution
- *Files Included:*
  - CPU_C.circ: Main CPU design
  - Multiplier_C.circ: Dedicated 8-bit multiplier component

System Components

CPU (CPU_C.circ)

- *Registers:* At least 8 general-purpose registers (e.g., R0–R7)
- *ALU Capabilities:*
  - ADD, SUB, AND, OR, INC, DEC, CMP
- *Instruction Execution:*
  - MVI Rn, val: Load immediate
  - ADD Rn, Rm: Add registers
  - STR Rn, [addr]: Store register to memory
  - LOD Rn, [addr]: Load from memory to register
  - JMP addr: Unconditional jump
  - JZ/JN addr: Conditional jump (if zero / if negative)
- *Memory Interface:* Read and write via 17-bit memory address field
- *Control Logic:* Full 32-bit instruction decoder and program counter

Multiplier (Multiplier_C.circ)

- *Function:* Performs 8-bit × 8-bit multiplication
- *Inputs:* Two 8-bit registers
- *Output:* 16-bit product
- *Integration:* Accessed via a dedicated opcode or I/O port for expanded ALU operations

Instruction Encoding

Each 32-bit instruction is structured as:

09100005 → MVI R1, 5 ; R1 = 5
0B100020 → STR R1, [0x20] ; Mem[0x20] = R1
0E000009 → JMP 0x09 ; Jump to address 0x09
00120000 → ADD R1, R2 ; R1 = R1 + R2

Execution Cycle

1. *Fetch*: Retrieve instruction from memory.
2. *Decode*: Parse opcode and operands.
3. *Execute*: Perform ALU, memory, or control operation.
4. *Write-back*: Store result in register/memory.
5. *Advance PC*: Update program counter.

Educational Objectives

- Learn core computer architecture principles
- Understand CPU datapaths and control signals
- Practice with digital logic and circuit design
- Explore low-level assembly-style programming

Future Improvements

- Add multi-cycle instruction support (e.g., MUL)
- Implement stack and subroutine calls
- Introduce interrupts and I/O handling
- Develop an assembler for converting mnemonics to binary

File Structure

8-bit-cpu-project/
├── CPU_C.circ # Main CPU design in Logisim Evolution
├── Multiplier_C.circ # Multiplier circuit for arithmetic extension
├── README.md # Project documentation

Sample Instruction Set

Opcode	Mnemonic	Description
00	ADD	Add registers
01	SUB	Subtract registers
02	AND	Bitwise AND
03	OR	Bitwise OR
04	INC	Increment register
05	DEC	Decrement register
06	CMP	Compare
07	NOP	No operation
08	MOV	Register to register
09	MVI	Move immediate
0A	LOD	Load from memory
0B	STR	Store to memory
0C	JZ	Jump if zero
0D	JN	Jump if negative
0E	JMP	Unconditional jump
0F	HLT	Halt execution

Contact

For questions or suggestions, please open an issue or submit a pull request to the following GitHub Repository:

GitHub Repository Link

Meet Our Team

Group No. 04

Rayhanul Amin Tanvir
rayhanulamint2@gmail.com
Reg. No: 2020331002
Session: 2020-21

Md Rezaul Karim Sumon
rezaul0818@gmail.com
Reg. No: 2020331029
Session: 2020-21

Mohammad Ridowan Sikder
ridowan.cse@gmail.com
Reg. No: 2020331043
Session: 2020-21

Khalid Bin Selim
khalid44@student.sust.edu
Reg. No: 2020331044
Session: 2020-21

Gazi Mujtaba Rafid
rafidjess13@gmail.com
Reg. No: 2020331051
Session: 2020-21
