32-Bit Custom CPU

This project showcases a custom-designed 8-bit CPU built in Logisim Evolution, utilizing a 29-bit data bus and a 32-bit instruction format. It supports a streamlined yet versatile instruction set and includes essential computing features such as ALU operations, memory access, and control flow management.

🔧 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

• Contains 8 general-purpose registers: R0 to R7

• Each register can store and manipulate 29-bit data

ALU Capabilities

Supports the following arithmetic and logic instructions:

• ADD: Addition

• SUB: Subtraction

• AND: Bitwise AND

• OR: Bitwise OR

• INC: Increment

• DEC: Decrement

• CMP: Compare two registers

Instruction Execution Format

Instruction	Description
MVI Rn, val	Move immediate value to register Rn
ADD Rn, Rm	Add contents of Rm to Rn
STR Rn, [addr]	Store contents of Rn to memory address
LOD Rn, [addr]	Load value from memory address into Rn
JMP addr	Unconditional jump to address

JZ addr	Jump if result is zero
JN addr	Jump if result is negative

Memory Interface

- Memory operations support a 17-bit address field
- Enables read/write access to data memory

Control Logic

- A complete 32-bit instruction decoder
- Integrated Program Counter (PC) to manage instruction sequencing

Multiplier (Multiplier_C.circ)

Function

- Performs 8-bit × 8-bit multiplication
- Supports integer arithmetic operations

Inputs

- Two 8-bit operands from general-purpose registers

Output

- A 16-bit product stored in a dedicated register or split across two general-purpose registers

Integration

- Invoked through a dedicated opcode (e.g., MUL Rn, Rm)
- Optionally accessible via a special I/O port for expanded ALU functionality

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**:
The CPU retrieves the instruction from memory using the address in the Program Counter (PC).
2. **Decode**:
The instruction is parsed to determine the operation (opcode) and the operands (registers or immediate values).
3. **Execute**:
The specified operation is performed by the ALU, memory unit, or control unit.
4. **Write-back**:
The result of the operation is written back to a register or memory location as required.
5. **Advance PC**:
The Program Counter is incremented or updated to point to the next instruction.

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

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

Group No. 15

1. Nayem Ahmed
   Reg. No: 2020331048
   Session: 2020–21
2. Sanjoy Das
   Reg. No: 2020331023
   Session: 2020–21
3. Mustakim Billah
   Reg. No: 2020331097
   Session: 2020–21
4. Tajul Islam Tarek
   Reg. No: 2020331067
   Session: 2020–21
5. Mossa Manisa
   Reg. No: 2020331058
   Session: 2020–21