Course Project

Announcement Date	:	10 – March –2024
Submission Deadline	:	25 – April – 2024
Team size	:	Max of four members per team

Project Title

Design of 7-bit CPU using Logisim by integrating ALU, Registers and ROM.

Introduction

The goal of this project is to design a programmable 7-bit CPU with four 7-bit general purpose registers; R1, R2, R3, and R4 and one special-purpose register; Rx.

Design and simulate in Logisim using the build-in components; e.g. registers, adders, subtractors, shifters, decoders, ROM, etc.

Objectives

Design a 7-bit programmable CPU architecture with

- An ALU to execute the required instructions (refer to Table 2)
- A data path with 7-bit general-purpose registers; R1 to R4
- One Special-purpose register which contains the 7-bits data and the Parity bit as set by the code
- A special-purpose register; Rx (more details are given later)
- A micro-programmed Control Unit with a $2^8 \times 16$ bits addressable ROM
- The Program Counter stops once all the instructions of a program are executed.
- Add a RUN button to start program execution.
- Add an END LED to indicate program termination.
- Connect a display unit to each register to show its contents

Methodology

1. Design a CPU architecture using *Logisim* simulator.
2. Design the instruction set including the opcode and operands for each instruction
3. Implement the CPU using 7-bit general-purpose registers, ALU and CU.
4. Connect all registers with ALU for arithmetic and logical operations.
5. Use the Program Counter to generate the addresses of ROM.

Expected Outcome

This project is expected to produce a programmable 7-bit CPU with 8 instructions, 16-bit fixed-length instructions, and 7-bit registers numbered R1 through R4. In addition, an 8-bit addressable memory with a 16-bit width will be developed as part of the project.

To test the operation of the digital system, you should;

1. Write a sample program to add two numbers
2. Write a sample program to subtract two numbers
3. Write a sample program to multiply two numbers using Shift Left Operation
4. Write a sample program to divide using Shift Right Operation
5. Write a sample program to insert the Even Parity / Odd Parity to the 7-bits data where the MSB will be Parity bit. Store the data with parity bit in an 8-bit register; Rx.

Table 1: - Design Requirements

Specifications	Description
Number of instructions	8
Width of the instruction	16-bits fixed-length
CPU registers	7-bits general-purpose registers; R1 to R4, each of 7-bits 8-bits Rx (7 data + 1 Parity)
Memory Size	2^8×16 bits, i.e. 8-bits address × 16-bits width
Data Type	7-bit unsigned integer

Table 2: - Basic Instruction Set Examples

Operation	Operation	Syntax	In RTL
Transfer	Register-to-register	MVR Rd, Rs	Rd ← Rs
	Imm-to-register	MVI Rd, Imm	Rd ← Imm
Arithmetic (2-ference)	Add	ADD Rd, Rs	Rd ← Rd + Rs
	Subtract	SUB Rd, Rs	Rd ← Rd – Rs
Parity	Podd Peven	Podd Rs, 0/1 Peven Rs, 0/1	*Rx* ← Rs[PXXXXXXX] *Rx* ← Rs[PXXXXXXX]
Shift	Logical Shift Right	LSR Rd, Rs	Rd ← LSR (Rs, 1)
	Logical Shift Left	LSL Rd, Rs	Rd ← LSL (Rs, 1)

Notes: -

- Imm ≡ Immediate data, e.g. MOV R2, 5 ; R2 ← 5
- P is Parity bit
- The hardware can only execute a single bit LSR / LSL operation

Design Operation

1. Based on the given instructions, write the assembly code.
2. Based on your instruction format, convert the assembly code to micro-code
3. Upload the micro-code into the memory (ROM)
4. Hit on the RUN button. The CPU should execute the code and saves the result as instructed by the code
5. When program execution is complete, the results will be available in register(s).

Sample of Assembly Code

With the given instructions, we can write the assembly code to multiply the contents of R1 by 5 and save the result in R3 without changing the value of R1.

```
MVI   R1, 12      ; R1 ← 12
MVR   R2, R1      ; R2 ← 12
LSL   R2, 2       ; R2 ← 4×12
ADD   R2, R1      ; R2 ← 5×12
```

Program Execution

To run any program; the user will

- Load the μPC with the program starting address. The ROM memory table that you will provide should indicate the starting address of all programs saved in the μROM
- Hit the RUN button
- The μPC starts counting up and the μROM will generate the control signals
- The μPC stops (find any way for making this happen) when it detects the end of the program (the clock keeps ticking) and the DONE LED turns ON.

Conclusion

This course project will provide a hands-on experience in designing a 7-bit CPU architecture and execute various instructions on ALU. This project will enhance students' understanding of computer architecture and hardware design, preparing them for future careers in computer engineering and related fields.

The Logisim Design Guidelines

- Keep the design neat. Reduce long wiring and replace them with **Tunnels**
- If using Tunnels, use descriptive names.
- Do not disperse the components. Arrange similar items close to each other. For example, align the registers on one side, the ALU on the other side.
- Label each component; e.g. register (R1, R2, …), ALU, micro-ROM, etc.
- Use one and only one clock generator.
- Add the RUN button, and the DONE LED.
- To the **Data Input** of the μPC (Micro Program Counter), connect an input pin. This will be used to load the counter with the program starting address.
- On page-3, a list of programs is required. All should be saved in the ROM. It would nice if you add one or two extra examples doing a more complicated arithmetic operations; calculating the average of four numbers.
- Use a certain code to indicate the end of a program, e.g. all 0s or all 1s or any special code of your choice.

Testing the System

To run any program; the user will

- Load the μPC with the program starting address. The ROM memory table that you will provide should indicate the starting address of all programs saved in the μROM
- Hit the RUN button
- The μPC starts counting up and the μROM will generate the control signals
- The μPC stops (find any way for making this happen) when it detects the end of the program (the clock keeps ticking) and the DONE LED turns ON.