|  |  |
| --- | --- |
|  | **Qatar University**  **College of Engineering**  **Department of Computer Science and Engineering** |

**CMPE263 Computer Architecture and Organization I**

**Course Project Report**

**Spring 2024**

project title

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

Submitted By:

1. Sultan Al-Maadid 202203236
2. Nasser Al-Ansari 202203172
3. Nasser Al-Marri 202008899
4. Abdelrahman Abdelwahed 202207121

Instructor Name: Eng. Amelle Bedair ([amlbdr@qu.edu.qa](mailto:amlbdr@qu.edu.qa))

Table of Contents

[General 2](#_Toc1156418368)

[Part I: Introduction 2](#_Toc594904951)

[Part II: CPU Architecture and Organization 3](#_Toc1485828166)

[Part III: Instruction Set 4](#_Toc1540795777)

[Part IV: Instruction Format 4](#_Toc1973497055)

[Part V: Assembly Programs 6](#_Toc1645664572)

[Part VI: Conclusion 9](#_Toc1606561508)

# General

* Do NOT use red text.
* Do NOT highlight.
* Title font size = 14
* Contents font size = 12
* Images must be clear.
* Each image must have a caption.
* Do NOT stretch images
* For images, do NOT change the “Wrap Text” format. Keep it “In Line with Text”.
* For tables, do NOT change the “Text Wrapping” format. Keep it “None.”
* All sections must be included
* Submit WORD document & the Logisim file

# Part I: Introduction

The Design revolves around a custom 7-bit CPU, with the primary constraint being the 7-bit processing power of the CPU. The architecture is designed to handle a fixed instruction width of 16-bits and supports 8 distinct instructions. The system is 4 general purpose registers with a width of 7-bits (7-bit CPU) and a special register Rx of 8-bits (8th bit is parity bit). It also supports an 8-bit addressable memory of 16-bit data width.

# Part II: CPU Architecture and Organization

The 7-bit architecture of the CPU starts with the register bank which contains 4 7-bit registers R1-R4, and an 8-bit Rx. The CPU also supports a 16-bit instruction width, designed to be fetched from ROM. The ALU supports 8 different operations (Transfer data, transfer immediate, add, sub, multiply, divide, odd parity, even parity) and the result will be back to one of the 7-bit registers R1-R4 the selector of the decoder will be the same as the Rd multiplexer, so it will be like two reference instruction (R1+R2 -->R1).

A computer screen shot of a computer

Description automatically generated

CPU

# Part III: Instruction Set

The instruction set of the CPU supports a range of operations including data transfer, arithmetic calculations, logic operations, shift instructions.

* **Data Transfer**: mvr (move register), mov imm (move immediate).
* **Arithmetic Operations**: add (addition), sub (subtraction).
* **Shift Operations**: logical shift left) and logical shift right.
* **Parity Operations**: Podd (Parity odd), Peven (Parity even).

# Part IV: Instruction Format

The format of the instruction goes as follows:

A diagram of a code

Description automatically generated

Figure 2: Instruction Format

From the sheets above, the first seven bits of an instruction represent the (Data In) which will be used in the instruction (Move Immediate). Also, the next 4 bits is the selectors for our CPU. Moreover, the next three bits is the opcode and define the functionality of the instruction (add, move, shift…) and the one of the bit is a selector for Podd and Peven. After that there is one bit remain 0 to make the start and the end of the program not interrupting and the most significant bit is the enable.

Op code:  
 **000:** MOV

**001:** MOV Imm

**010:** ADD

**011:** SUB

**100:** LSL

**101:** LRL

**110:** Podd

**111:** Peven

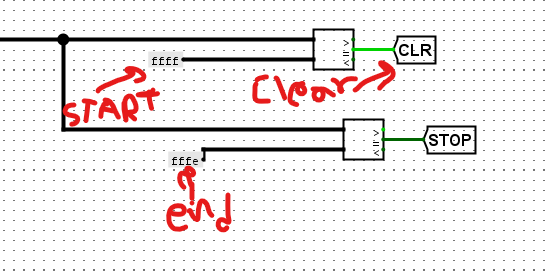
Start and stop:

The instruction **ffff** is the start of the program and it will rest all the registers back to zero. The instruction **fffe** stops the program and disable everything and light the led. Also, there is the Run bottom which will load the address of the program and start it.

A computer screen shot of a computer

Description automatically generated

Control Unit

A diagram of a circuit board

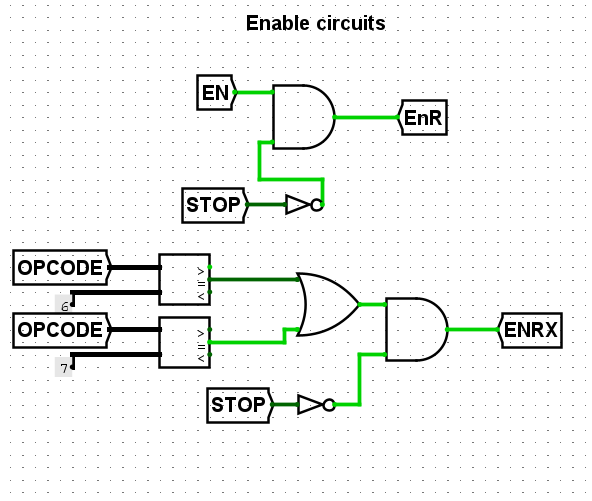
Description automatically generated

Enable:

we make some circuits to make sure Registers are disable in the end of the program. We also need to make sure that Rx is holding the right data, so we made a circuit that enables Rx when the opcode is 7 or 6 because the inputs (7 and 6) in the multiplexer that connected with the result and we make these two circuits to stop the program when it detect fffe (the end of our program).

A diagram of a circuit

Description automatically generatedA diagram of a circuit

Description automatically generated

Enable

# Part V: Assembly Programs

Using the format explained above, we will write a simple assembly code to test:

|  |  |  |
| --- | --- | --- |
| Program starting address | Operation | Code Description |
| 0x00 | Add two numbers | R1 = 53  R2 = 37  R1 = R1 + R2 |
| 0x12 | Subtract two numbers | R1 = 68  R2 = 35  R1 = R1 – R2 |
| 0x06 | Multiply two numbers using Shift Left operation | R1 = 7  R1 = 10 × R3 |
| 0x1b | Divide using Shift Right operation | R1 = 120  R1 = R1/ 4 |
| 0x24 | Insert the odd Parity | R1 = 120  Rx = odd(R1) |
| 0x30 | Insert an even parity | R1 = 121  Rx = even(R3) |

CODE 1(Program starting address 0x00):

ffff

8835

8a25

9080

fffe

explaining:

ffff: Start and clear all registers.

8835: MOV R1, #53

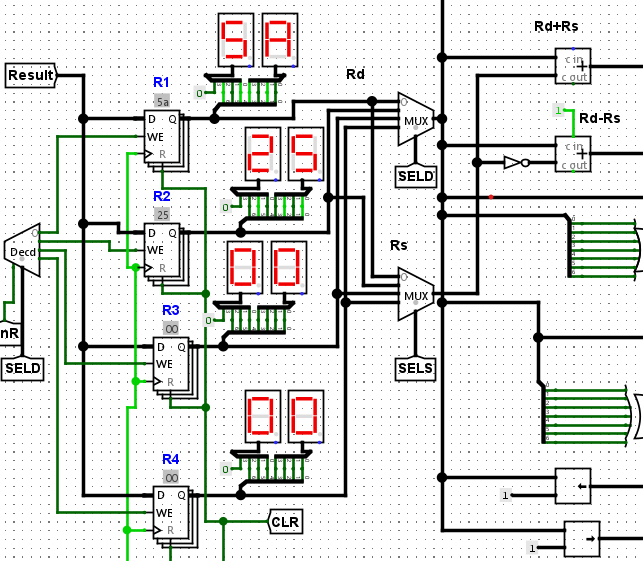
8a25: MOV R2, # 37

9080: ADD R1,R2

fffe: end and stop the program

Result:

The result will be 90 (5A in Hex) and it will be saved in R1.



contents after code 1

CODE 2:

ffff

8844

8a23

9880

fffe

explaining:

ffff: Start and clear all registers.

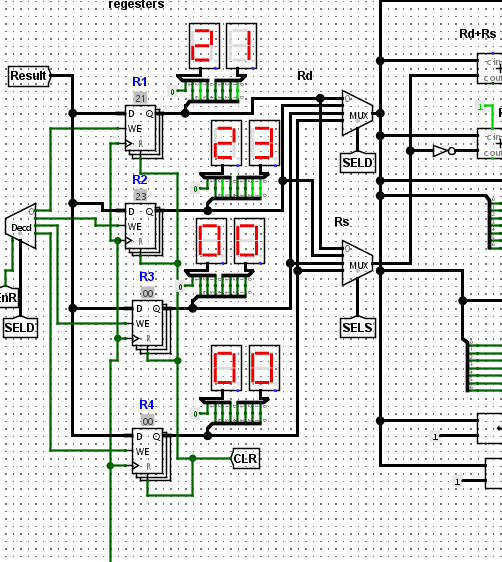
8844 : MOV R1, #68

8a23: MOV R2, # 35

9880: SUB R1,R2

fffe: end and stop the program

Results:



Code 2 results

CODE 3:

ffff

8807

8a07

a000

a000

a000

9080

9080

fffe

explaining:

ffff: Start and clear all registers.

8807: MOV R1, #7

8a07: MOV R2, # 7

a000: LSL , #1

9080: ADD R1,R2

fffe: end and stop the program

Results:

A diagram of a machine

Description automatically generated

Code 3 result

CODE 4:

ffff

8878

a800

a800

fffe

explaining:

ffff: Start and clear all registers.

8878: MOV R1, #120

a800: LSR, # 1

fffe: end and stop the program

Results:

A diagram of a circuit

Description automatically generated

CODE 5:

ffff

8878

b000

fffe

explaining:

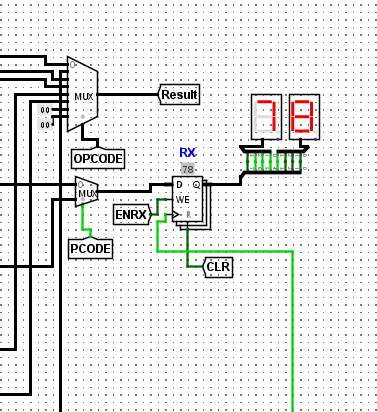
ffff: Start and clear all registers.

8878: MOV R1, #120

b000: add the odd parity bit and save it in Rx register

fffe: end and stop the program

Results:



CODE 5:

ffff

8879

b800

fffe

explaining:

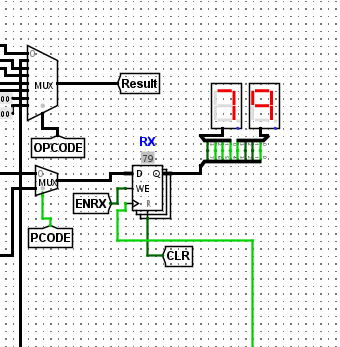
ffff: Start and clear all registers.

8879: MOV R1, #121

b800: add the even parity bit and save it in Rx register

fffe: end and stop the program

Results:



# Part VI: Conclusion

We designed and tested a basic CPU and it work like two reference instructions with a simple instruction set and limited registers, including a special parity register. There were some challenges included Rx register and how to make it work properly and some difficulties at designing the control unit.