|  |  |
| --- | --- |
|  | **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. Fahrel Hidayat - 202206836
2. Syed Muhammad Hamza Iqbal - 202207864
3. Khalid Haji - 202010980
4. Mohammed Alam - 202211429

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

Table of Contents

Part I: [Introduction 3](#_Toc131949761)

[Part II: CPU Architecture and Organization 4](#_Part_II:_CPU)

[Part III: Instruction Set 5](#_Part_III:_Instruction)

[Part IV: Instruction Format ………………………………………………….…………………………………………………………….. 6](#_Part_IV:_Instruction)

[Part V: Assembly Programs 8](#_Part_V:_Assembly)

[Part VI: Conclusion 16](#_Part_VI:_Conclusion)

# Part I: Introduction

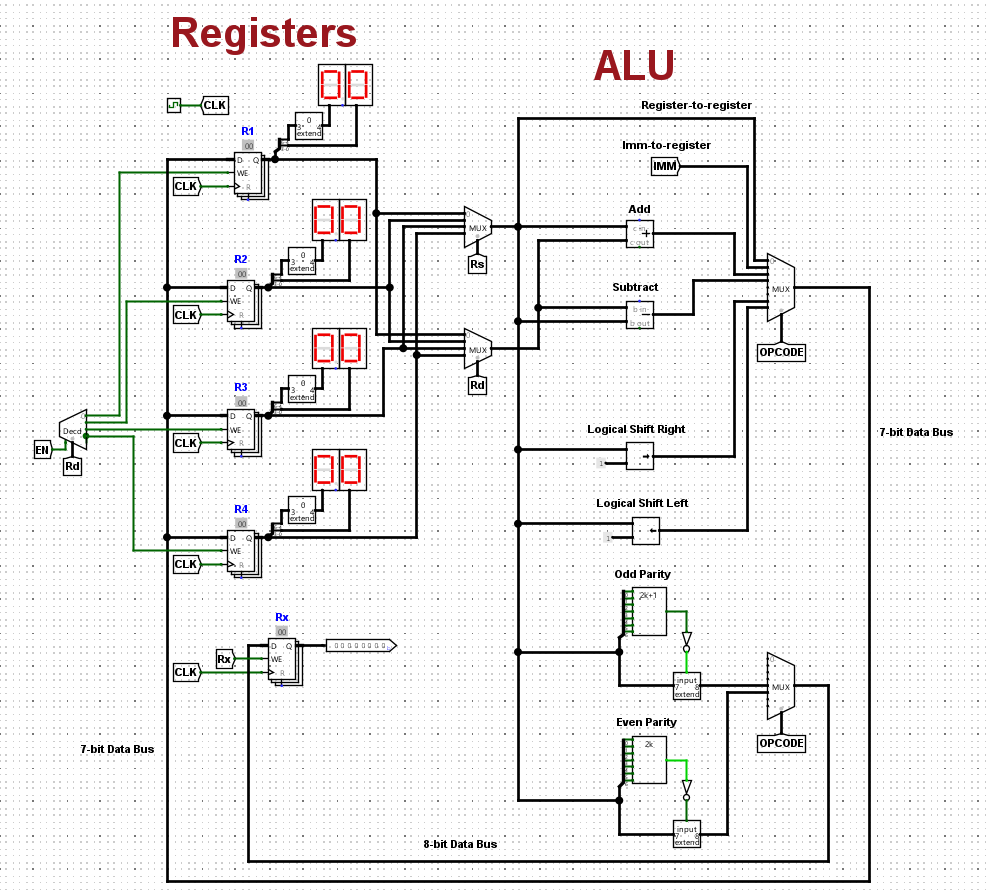
This project design is based on a very simple CPU in Logisim that can carry out multiple operations in the ALU which include arithmetic, transfer, shift, and parity operations. It has several 7-bit general purpose registers along with a 8-bit special purpose register used in operations related to parity.

The control unit consists of a program counter (PC) and a micro-ROM to execute complete programs without failure.

Multiplexers and decoders are implemented in the design to control data flow and enable registers respectively. A number of different wiring, input and output components are used throughout the design including splitters, button, LED, hex displays and bit-extenders.

Of the constraints adhered to in the design is the capability to shift either right or left only one bit at a time. Furthermore, the instruction length is fixed to 16-bits while the address consists of 8-bits which results in the size of the memory to be 28 x 16.

# Part II: CPU Architecture and Organization



**Figure 2.a: Registers and ALU**

Figure 2.a outlines the general-purpose registers and the special purpose register along with the ALU. Starting from the left, we have a 2 to 4 decoder which is responsible for the write enables of the general-purpose registers R1 to R4. These registers are of 7-bits and have two hex displays connected to display their content. We have two multiplexers with selects Rs and Rd which will direct data into the ALU to perform any operation.

The ALU is capable of carrying out 8 operations, these include register to register and immediate to register transfer, addition, subtraction, logical shift left and right, as well as even and odd parity error detection. The opcode to the multiplexer controls which result is allowed to enter onto the output 7-bit data bus which is connected back to the registers. The special purpose register Rx stores the result of any parity operation and is an 8-bit register.

A diagram of a control unit

Description automatically generated

**Figure 2.b: Control Unit**

The control unit is shown in figure 2.b. Its main components consist of a run button, a LED to indicate termination, a program counter, and a micro-ROM. The steps of executing any program begin at the starting address pin, where the user enters the starting address of the program from the memory map table. The next step is to hold on the run button for the duration of a clock pulse. The run button is designed using a J-K flip flop.

Once the run button is pressed, the starting address is loaded into the program counter, and it starts incrementing until the end instruction of 0x0000 is detected on the micro-ROM. The controlling of the execution of the program is implemented using a XOR gate which stops the program counter by disabling the enable. The current address enters the micro-ROM to execute the instructions stored in the specific address which are then using a splitter, distributed to the different fields in our instruction format.

# Part III: Instruction Set

The Instruction Set table for all 8 operations in the ALU:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OPERATION | | SYNTAX | RTL | OPCODE (3-Bits) |
| Transfer | Register-to-Register | MVR Rd, Rs | *Rd 🡨 Rs* | 000 |
| Imm-to-Register | MVI Rd, Imm | *Rd 🡨 Imm* | 001 |
| Arithmetic  (2-reference) | Add | ADD Rd, Rs | *Rd 🡨 Rd + Rs* | 010 |
| Subtract | SUB Rd, Rs | *Rd 🡨 Rd - Rs* | 011 |
| Parity | Podd | Podd Rs, 0/1 | *Rx 🡨 Rs[****P****XXXXXXX]* | 100 |
| Peven | Peven Rs, 0/1 | *Rx 🡨 Rs[****P****XXXXXXX]* | 101 |
| Shift | Logical Shift Right | LSR Rd, Rs | *Rd 🡨 LSR (Rs,1)* | 110 |
| Logical Shift Left | LSL Rd, Rs | *Rd 🡨 LSL (Rs, 1)* | 111 |

# Part IV: Instruction Format

16-bits fixed Instruction format showing the distribution of bits of the Assembly code.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| General Description of the Instruction Format. | | | | | |
| Opcode | EN | Rd | Rs | Rx | Immediate |
| 3-Bit | 1-Bit | 2-Bit | 2-Bit | 1-Bit | 7-Bit |
| Selects the Operation that is going to be executed | Controls the enabling of data storing to the General-purpose-registers | Destination; General-purpose-register | Source; General-purpose-register | Controls the enabling of data storing to Register Rx | Source; Immediate Value |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction Format of: MVI R1, 8 | | | | Transfers Decimal Value 8 to Register R1. | | |
| Opcode | EN | Rd | Rs | | Rx | Immediate |
| 3-Bit | 1-Bit | 2-Bit | 2-Bit | | 1-Bit | 7-Bit |
| 001 | 1 | 00 | xx | | 0 | 0001000 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction Format of: MVR R2, R1 | | | | Transfers Value of Register R1 to R2. | | |
| Opcode | EN | Rd | Rs | | Rx | Immediate |
| 3-Bit | 1-Bit | 2-Bit | 2-Bit | | 1-Bit | 7-Bit |
| 000 | 1 | 01 | 00 | | 0 | xxxxxxx |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction Format of: ADD R3, R2 | | | | Add Values of R3 and R2, Save the Result in R3. | | |
| Opcode | EN | Rd | Rs | | Rx | Immediate |
| 3-Bit | 1-Bit | 2-Bit | 2-Bit | | 1-Bit | 7-Bit |
| 010 | 1 | 10 | 01 | | 0 | xxxxxxx |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction Format of: SUB R4, R3 | | | | Subtract Value in R4 by Value in R3, Save the Result in R4. | | |
| Opcode | EN | Rd | Rs | | Rx | Immediate |
| 3-Bit | 1-Bit | 2-Bit | 2-Bit | | 1-Bit | 7-Bit |
| 011 | 1 | 11 | 10 | | 0 | xxxxxxx |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction Format of: Podd R1, 0/1 | | | | If numbers of 1s are odd in the Value of R1 set Parity Bit to 0, otherwise Set Parity Bit to 1. Save result in Special Purpose Register Rx. | | |
| Opcode | EN | Rd | Rs | | Rx | Immediate |
| 3-Bit | 1-Bit | 2-Bit | 2-Bit | | 1-Bit | 7-Bit |
| 100 | 0 | xx | 00 | | 1 | xxxxxxx |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction Format of: Peven R2, 0/1 | | | | If numbers of 1s are even in the Value of R2 set Parity Bit to 0, otherwise Set Parity Bit to 1. Save result in Special Purpose Register Rx. | | |
| Opcode | EN | Rd | Rs | | Rx | Immediate |
| 3-Bit | 1-Bit | 2-Bit | 2-Bit | | 1-Bit | 7-Bit |
| 101 | 0 | xx | 01 | | 1 | xxxxxxx |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction Format of: LSR R3, R2 | | | | Logical Shift Right 1-Bit for the Value of R2, Save the result in R3. | | |
| Opcode | EN | Rd | Rs | | Rx | Immediate |
| 3-Bit | 1-Bit | 2-Bit | 2-Bit | | 1-Bit | 7-Bit |
| 110 | 1 | 10 | 01 | | 0 | xxxxxxx |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction Format of: LSL R4, R3 | | | | Logical Shift Left 1-Bit for the Value of R3, Save the result in R4. | | |
| Opcode | EN | Rd | Rs | | Rx | Immediate |
| 3-Bit | 1-Bit | 2-Bit | 2-Bit | | 1-Bit | 7-Bit |
| 111 | 1 | 11 | 10 | | 0 | xxxxxxx |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| End Of Program Code/Operation | | | | If Instruction format is all 0’s, End the Program. | | |
| Opcode | EN | Rd | Rs | | Rx | Immediate |
| 3-Bit | 1-Bit | 2-Bit | 2-Bit | | 1-Bit | 7-Bit |
| 000 | 0 | 00 | 00 | | 0 | 0000000 |

# Part V: Assembly Programs

A Memory Map Table presenting the starting address of the 7 programs below with a brief description of their operation and general step-by-step instruction of the program (Code Description).

**Memory Map Table**

|  |  |  |
| --- | --- | --- |
| Program starting address | Operation | Code Description |
| 0x00 | Add two numbers (8+8) and saves result in R1. | R1 = 8  R2 = 8  R1 = R1 + R2 |
| 0x10 | Subtract two numbers  (27-14) and saves result in R1. | R3 = 27  R4 = 14  R3 = R3 - R4  R1 🡨 R3 |
| 0x20 | Performing an Odd Parity on 4. | R1 = 4  Rx = *odd*(R1) |
| 0x30 | Dividing using Shift Right  (8/4) and saves result in R4. | R1 = 8  R4 = R1/4 |
| 0x40 | Performing an Even Parity on 7. | R1 = 7  Rx = *even*(R1) |
| 0x50 | Multiply two numbers (4 × 4) using Shift Left operation and saves result in R3. | R1 = 4  R3 = R1 × 4 |
| 0x60 | Averages the numbers in R1 to R4 and saves result in R2 | R1 = 9  R2 = 9  R3 = 15  R4 = 15  R2 = (R1+R2+R3+R4)/4 |

Below are the detailed step-by-step instructions of the following 7 programs mentioned in the Memory Map Table showing how the operation is executed using assembly code with its corresponding Micro-Program (Machine Code). Also displaying the output of the program showing the content stored in the registers along with the LED lighting up as the program execution ends.

**SAMPLE PROGRAMS**

|  |  |  |
| --- | --- | --- |
| SAMPLE 1: R1 🡨 8 + 8 | | |
| Code Description | Assembly Code | Micro-Program (Machine Code) |
| Load the Immediate Value 8 into register R1 | MVI R1, 8 | 0011 0000 0000 1000 |
| Load the value in R1 to R2 | MVR R2, R1 | 0001 0100 0000 0000 |
| Add the value in R1 and R2 then store in R1 | ADD R1, R2 | 0101 0001 0000 0000 |
| Program execution stops | END | 0000 0000 0000 0000 |
| **SAMPLE OUTPUT:**    **Figure 5.a:** Sample output of the program and result is stored in R1.    **Figure 5.b:** LED lighten up indicating termination of the program. | | |

|  |  |  |
| --- | --- | --- |
| SAMPLE 2: R1 🡨 27 - 14 | | |
| Code Description | Assembly Code | Micro-Program (Machine Code) |
| Load the Immediate Value 27 into register R3 | MVI R3, 27 | 0011 1000 0001 1011 |
| Load the Immediate Value 14 into register R4 | MVI R4, 14 | 0011 1100 0000 1110 |
| Subtract the value in R3 by R4 then store in R3 | SUB R3, R4 | 0111 1011 0000 0000 |
| Load the value in R3 to R1 | MVR R1, R3 | 0001 0010 0000 0000 |
| Program execution stops | END | 0000 0000 0000 0000 |
| **SAMPLE OUTPUT:**    **Figure 5.c:** Sample output of the program and result is stored in R1.    **Figure 5.d:** LED lighten up indicating termination of the program. | | |

|  |  |  |
| --- | --- | --- |
| SAMPLE 3: Rx 🡨 *odd*(R1) | | |
| Code Description | Assembly Code | Micro-Program (Machine Code) |
| Load the Immediate Value 4 into register R1 | MVI R1, 4 | 0011 0000 0000 0100 |
| Inserts an Odd Parity on R1 and store in Rx | Podd R1 | 1000 0000 1000 0000 |
| Program execution stops | END | 0000 0000 0000 0000 |
| **SAMPLE OUTPUT:**    **Figure 5.e:** Immediate Value Stored in R1    **Figure 5.f:** Output of the Program. Result is stored in Rx    **Figure 5.g:** LED lighten up indicating termination of the program. | | |

|  |  |  |
| --- | --- | --- |
| SAMPLE 4: R4 🡨 8 / 4 | | |
| Code Description | Assembly Code | Micro-Program (Machine Code) |
| Load the Immediate Value 8 into register R1 | MVI R1, 8 | 0011 0000 0000 1000 |
| Logical Shift Right 1-bit the Value in R1 and store in R2 | LSR R2, R1 | 1101 0100 0000 0000 |
| Logical Shift Right 1-bit the Value in R2 and store in R4 | LSR R4, R2 | 1101 1101 0000 0000 |
| Program execution stops | END | 0000 0000 0000 0000 |
| **SAMPLE OUTPUT:**    **Figure 5.h:** Sample output of the program and result is stored in R4.    **Figure 5.i:** LED lighten up indicating termination of the program. | | |

|  |  |  |
| --- | --- | --- |
| SAMPLE 5: Rx 🡨 *even*(R1) | | |
| Code Description | Assembly Code | Micro-Program (Machine Code) |
| Load the Immediate Value 7 into register R1 | MVI R1, 7 | 0011 0000 0000 0111 |
| Inserts an Even Parity on R1 and store in Rx | Peven R1 | 1010 0000 1000 0000 |
| Program execution stops | END | 0000 0000 0000 0000 |
| **SAMPLE OUTPUT:**    **Figure 5.j:** Immediate Value Stored in R1    **Figure 5.k:** Output of the Program. Result is stored in Rx    **Figure 5.l:** LED lighten up indicating termination of the program. | | |

|  |  |  |
| --- | --- | --- |
| SAMPLE 6: R3 🡨 4 x 4 | | |
| Code Description | Assembly Code | Micro-Program (Machine Code) |
| Load the Immediate Value 4 into register R1 | MVI R1, 4 | 0011 0000 0000 0100 |
| Logical Shift Left 1-bit the Value in R1 and store in R2 | LSL R2, R1 | 1111 0100 0000 0000 |
| Logical Shift Right 1-bit the Value in R2 and store in R3 | LSR R3, R2 | 1101 1001 0000 0000 |
| Program execution stops | END | 0000 0000 0000 0000 |
| **SAMPLE OUTPUT:**    **Figure 5.m:** Sample output of the program and result is stored in R3.    **Figure 5.n:** LED lighten up indicating termination of the program. | | |

|  |  |  |
| --- | --- | --- |
| SAMPLE 7: R2 🡨 (9+9+15+15)/4 | | |
| Code Description | Assembly Code | Micro-Program (Machine Code) |
| Load the Immediate Value 9 into register R1 | MVI R1, 9 | 0011 0000 0000 1001 |
| Load the value in R1 to R2 | MVR R2, R1 | 0001 0100 0000 0000 |
| Add the value in R1 and R2 then store in R1 | ADD R1, R2 | 0101 0001 0000 0000 |
| Load the Immediate Value 15 into register R3 | MVI R3, 15 | 0011 1000 0000 1111 |
| Load the value in R3 to R4 | MVR R4, R3 | 0001 1110 0000 0000 |
| Add the value in R3 and R4 then store in R3 | ADD R3, R4 | 0101 1011 0000 0000 |
| Add the value in R1 and R3 then store in R1 | ADD R1, R3 | 0101 0010 0000 0000 |
| Logical Shift Right 1-bit the Value in R1 and store in R4 | LSR R4, R1 | 1101 1100 0000 0000 |
| Logical Shift Right 1-bit the Value in R4 and store in R2 | LSR R2, R4 | 1101 0111 0000 0000 |
| Program execution stops | END | 0000 0000 0000 0000 |
| **SAMPLE OUTPUT:**    **Figure 5.o:** Sample output of the program and result is stored in R2.    **Figure 5.p:** LED lighten up indicating termination of the program. | | |

# Part VI: Conclusion

In conclusion, this is a design for a simple 7-bit CPU using Logisim implementing the core sections of a CPU which include the ALU, ROM and registers. It is capable to carry out transfer, arithmetic, shift, and parity operations.

A deep understanding of designing using Logisim and its tools was gained. In addition, this design introduced us to the world of CPU designing.

No major difficulties were faced during the design and implementation of the CPU, although issues related to the Control Unit and the parity operations can be considered as obstacles that took some time and thinking. These include the implementation of the RUN button and termination of the program counter after complete execution of the program. Also, figuring out how the built-in parity components in Logisim functions by performing several tests with input, output pins, and splitters as well as incorporating additional components in order to successfully construct the desired parity operator.