Final Project Report

Aidan Harries | ECE 649

12/10/22

Table of Contents

List of Figures

Introduction

This project aims to develop a simple CPU that can support a subset of RISC-V instructions. The CPU is a 32-bit architecture that includes 32 registers and an ALU with control signals. The CPU supports the R-Type instructions "Add" and "Sub", the I-Type instruction "Addi", the Load instruction "LW", and the Store instruction "SW". In addition, the design includes a display module that allows a "smile face" to be shown on an LED matrix.

The CPU design includes both hardware and software components. The hardware consists of several modules, including a memory unit, an instruction decoder, and registers, that are interconnected to perform the necessary operations. The software consists of a set of assembly language programs that control the CPU and enable it to execute the supported instructions. This report presents the details of the design and the results of testing. The implications of the work are discussed, and potential future improvements are suggested.

CPU Hardware Diagrams

The CPU hardware design consists of several modules that are interconnected to perform the necessary operations. Figure 1 shows a block diagram of the overall design, with the major modules labeled.

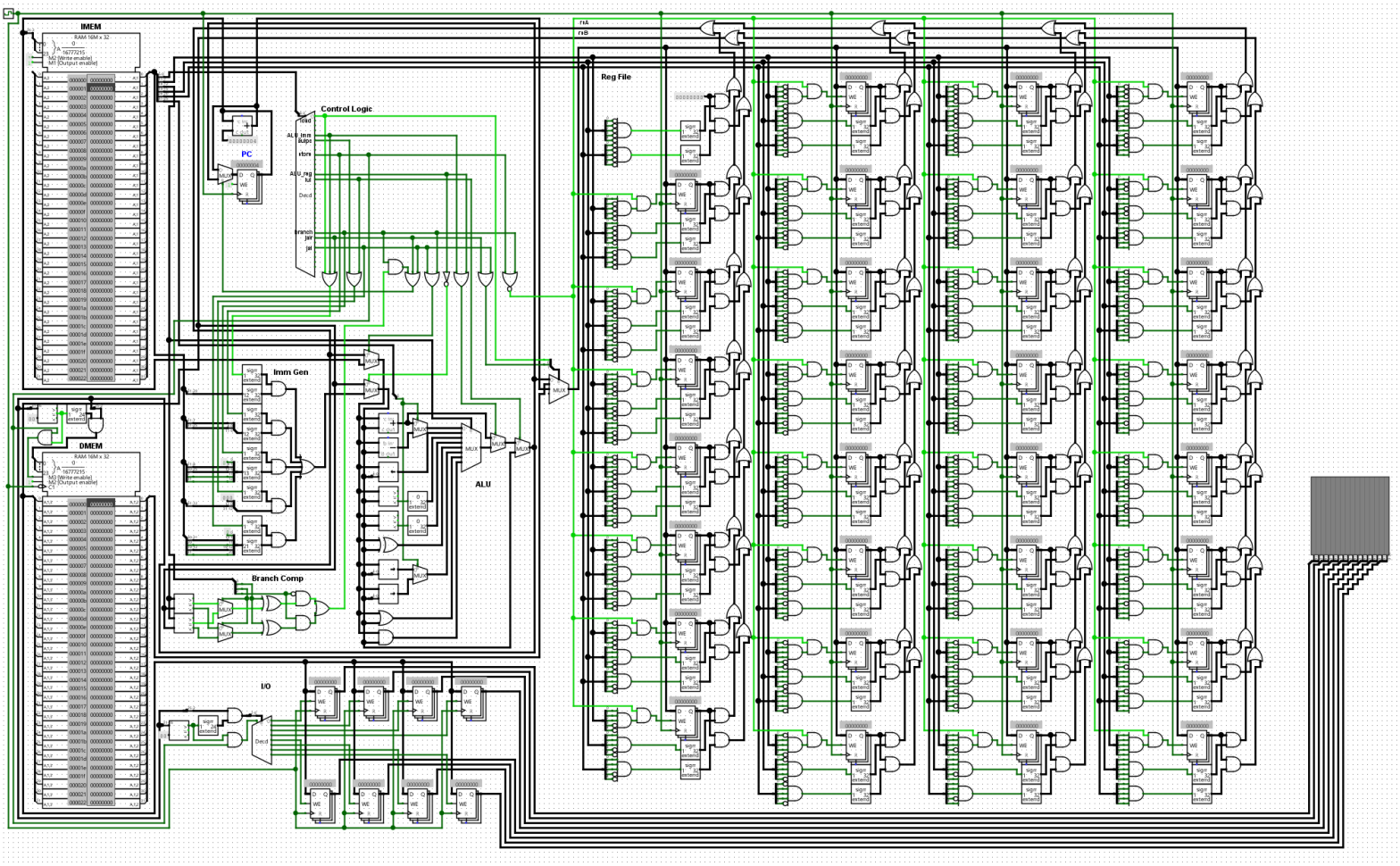

Figure 1: CPU Hardware Block Diagram

The major modules of the design are described below:

Memory

IMEM and DMEM: These modules consist of separate instruction and data memory units, IMEM and DMEM, respectively. Both are 16M x 32 RAMs that are used to store the instructions and data for the CPU. The IMEM contains the initial instructions that are executed when the CPU is powered on, while the DMEM is used for general storage of data and for holding the display code for the LED matrix.

The IMEM and DMEM modules are essential for the operation of the CPU, as they provide the storage for the instructions and data that the CPU processes. The IMEM holds the initial instructions that are executed when the CPU is powered on, while the DMEM is used for general storage of data and for holding the display code for the LED matrix. This code is accessed by the I/O module and used to control the LED matrix display, allowing the "smile face" image to be shown. The IMEM and DMEM modules are interconnected with the rest of the CPU, allowing data to be accessed and manipulated as needed by the other modules.

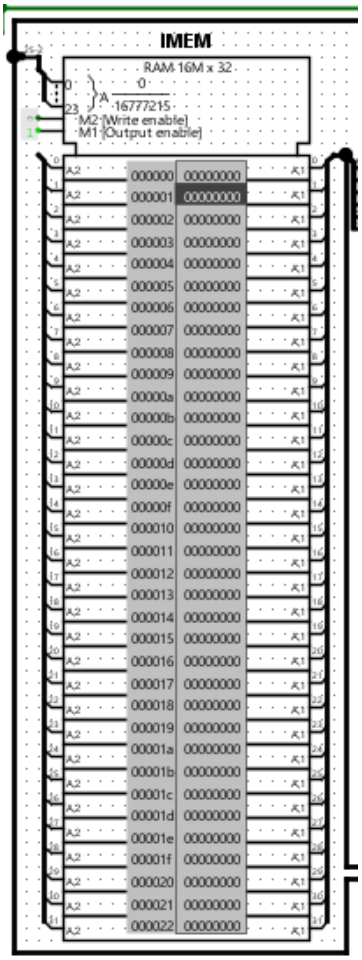

Figure 2: IMEM

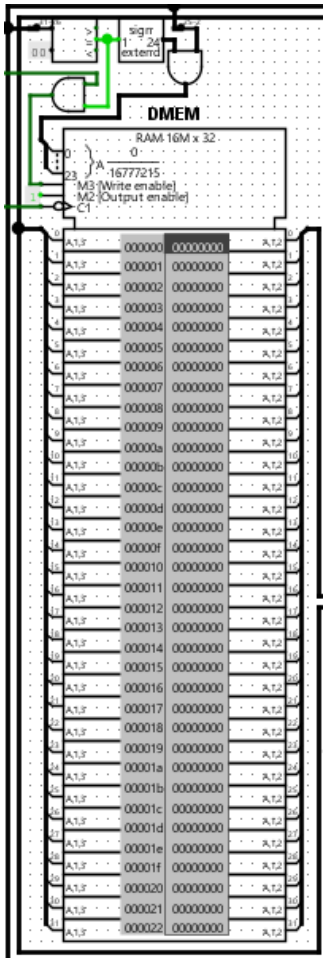

Figure 3: DMEM

PC

The Program Counter (PC) is an important component of the CPU design. It is a register that is used to keep track of the current instruction being executed. The PC is incremented by the control logic module after each instruction is fetched from the IMEM, with the adder for the PC set to 0x4. This means that the PC is incremented by 4 for each instruction, allowing the CPU to execute instructions sequentially from memory.

The PC is also used to implement branching in the CPU. When a branch instruction is encountered, the control logic module checks the condition specified in the instruction and determines whether to take the branch. If the branch is taken, the control logic module updates the PC to the address specified in the instruction, allowing the CPU to execute instructions from a different location in memory. This allows the CPU to implement control flow in the instructions it executes.

In addition to its role in instruction execution and branching, the PC is also used to implement subroutine calls in the CPU. When a jal or jalr instruction is encountered, the control logic module saves the current value of the PC in the appropriate register, as specified by the instruction. This allows the CPU to return to the correct instruction after the subroutine has been executed. This enables the CPU to implement more complex programs that make use of subroutines.

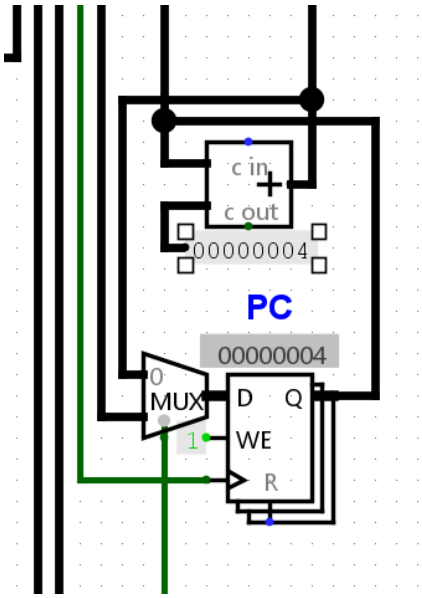

Figure 4: Program Counter

Control Logic

The Control Logic module is an important part of the CPU hardware design, as it is responsible for decoding instructions and generating the necessary control signals to execute them. This module receives instructions from the IMEM and decodes them to determine the appropriate action to take.

The Control Logic module uses a finite state machine (FSM) to control the execution of instructions. The FSM has several states, each corresponding to a different instruction or operation. When an instruction is fetched from the IMEM, the FSM transitions to the appropriate state to execute the instruction. For example, if the instruction is an "add" operation, the FSM will transition to the "ALU_reg" state, where the ALU will perform the addition of the specified registers.

Once the instruction has been executed, the FSM transitions back to the "fetch" state to fetch the next instruction. This process continues until all instructions have been executed. The Control Logic module also generates the necessary control signals to control the other modules in the design, such as the ALU, the registers, and the I/O module. This allows the CPU to perform the required operations for each instruction.

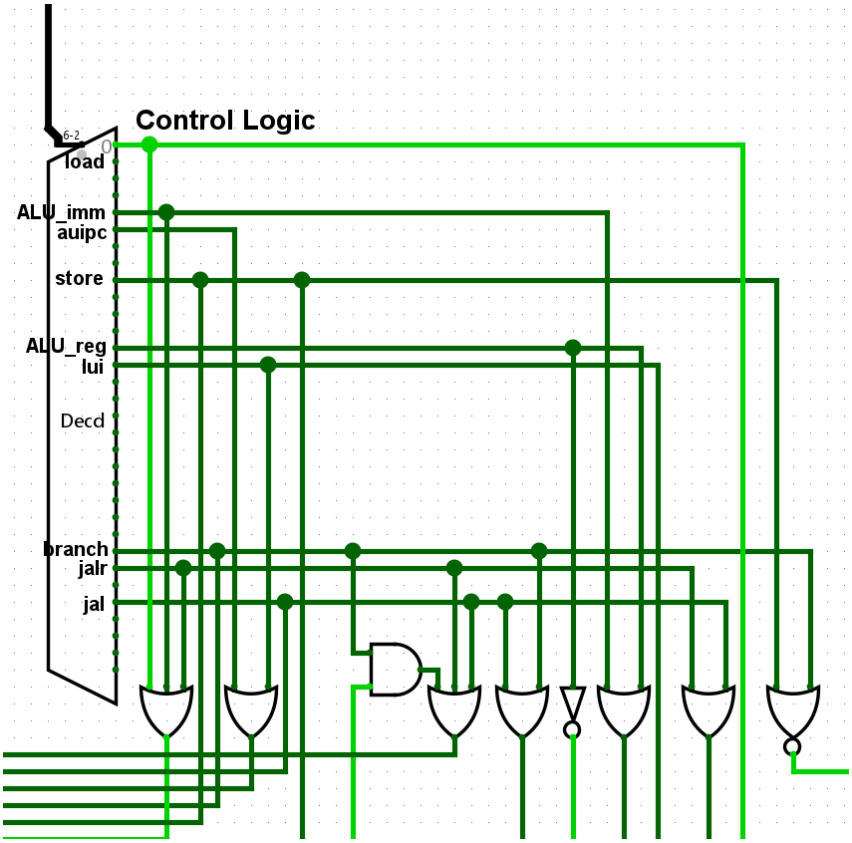

Figure 5: Control Logic

ImmGen

The Immediate Generator (ImmGen) module is an important part of the CPU design. It is used to generate the immediate value that is used in some instructions, such as the "Addi" instruction. This value is generated based on the instruction and passed to the ALU for use in the operation.

The ImmGen module receives the instruction from the instruction decoder and extracts the immediate value from it. This value is then sign-extended and passed to the ALU. The ALU uses this value in the operation, along with other input values from the registers and the instruction decoder. For example, in the "Addi" instruction, the immediate value is added to the value in a register to produce the result.

In addition to generating the immediate value, the ImmGen module also performs some other tasks. For example, it checks the instruction for the "Lui" and "Jal" instructions and generates the necessary control signals for the control logic module. It also receives the ALU result and the branch control signal from the Branch Comparator module and uses them to update the PC as needed.

Overall, the ImmGen module plays a crucial role in the execution of certain instructions by providing the necessary immediate value for the operation and performing other tasks, such as updating the PC and generating control signals. Its precise and efficient operation is crucial for the overall performance of the CPU.

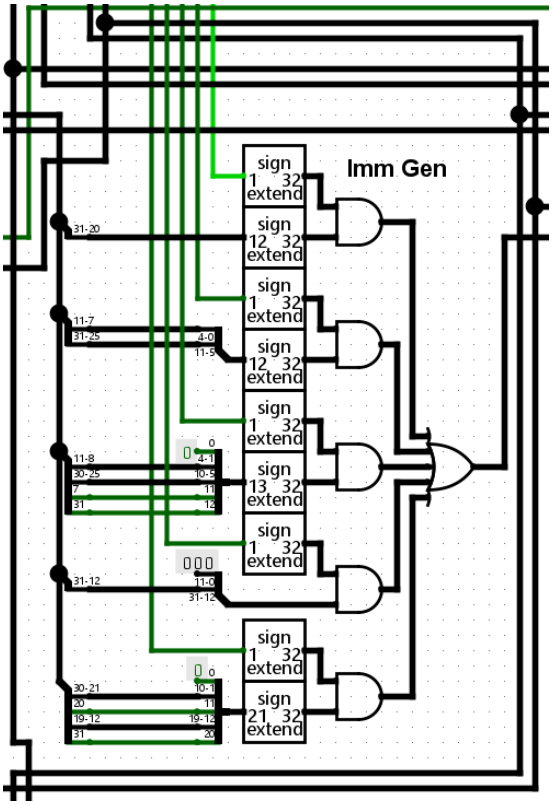

Figure 6: Immediate Generator

ALU

The ALU (Arithmetic and Logic Unit) is a critical component of the CPU design, as it is responsible for performing the arithmetic and logical operations specified by the instructions. It receives input from the registers, the instruction decoder, and the ImmGen module, and produces the result of the operation.

The ALU is designed to support the R-Type instructions "Add" and "Sub" as well as the I-Type instruction "Addi". For these instructions, the ALU receives two operands from the registers and the immediate value from the ImmGen module and performs the specified operation. The result of the operation is then written back to the register file for use in subsequent instructions.

In addition to the basic arithmetic operations, the ALU also includes a set of logical operations, such as AND, OR, and XOR. These operations are used in some instructions, such as the "LW" and "SW" instructions, to manipulate the data being transferred to and from the memory.

The ALU is an integral part of the CPU design that allows it to execute the supported instructions and perform the necessary operations. It is a crucial component that enables the CPU to function properly.

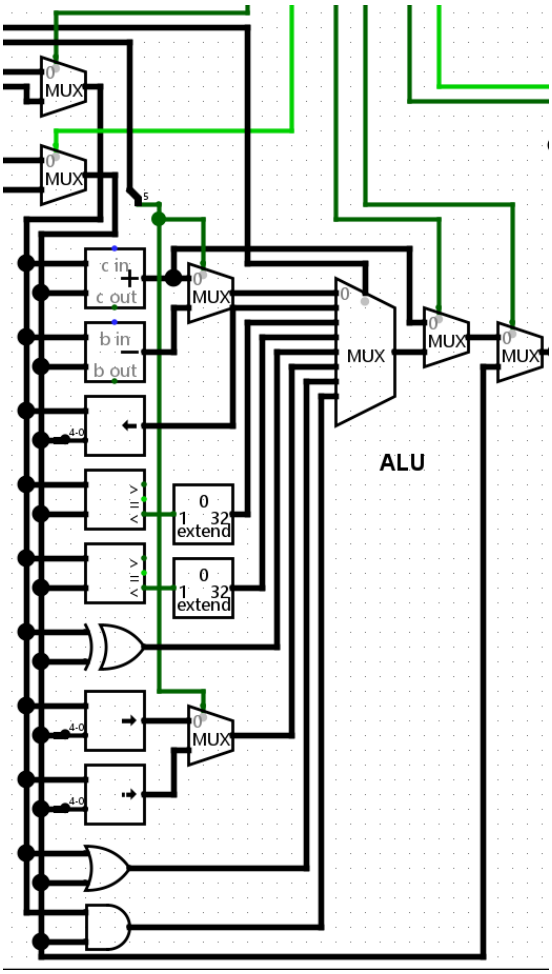

Figure 7: Arithmetic and Logic Unit

Branch Comp

The CPU hardware design includes several modules that are interconnected to perform the necessary operations. One of these modules is the Branch Comparator, which is used to compare the values in the registers and determine whether a branch instruction should be taken.

The Branch Comparator module receives input from the registers and the ALU and uses these inputs to compare the values and determine whether the branch condition is met. Depending on the result of the comparison, the module produces a control signal that is passed to the control logic module. The control logic module uses this signal to determine whether to take the branch and update the PC accordingly.

The Branch Comparator module is essential for supporting branch instructions in the CPU, as it enables the CPU to make decisions based on the values in the registers and take different paths through the instruction stream. This allows the CPU to implement more complex programs and algorithms.

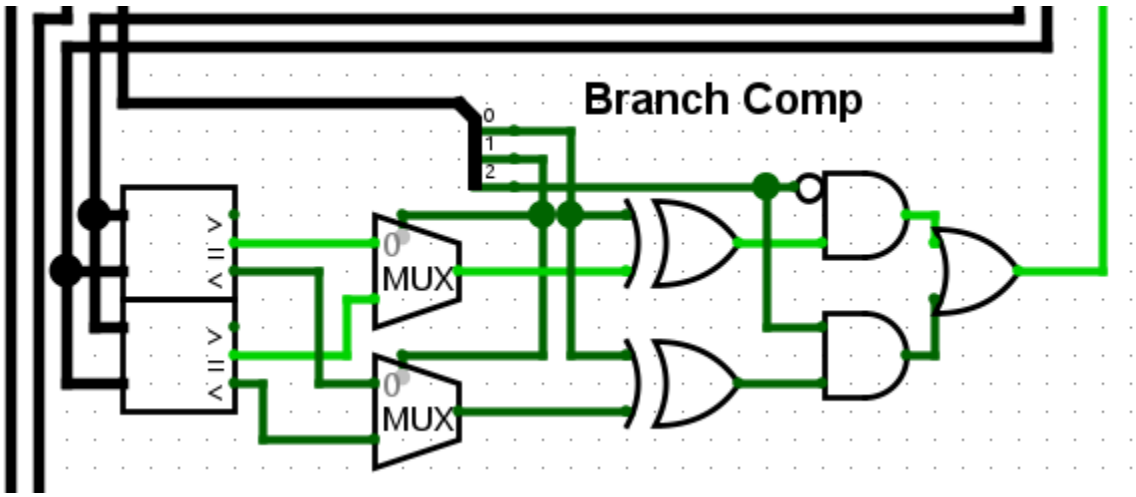

Figure 8: Branch Comparator

I/O

The CPU hardware design includes an I/O module that provides the interface between the CPU and external devices. This module receives data from the DMEM and uses it to control the output devices.

The I/O module consists of a set of input and output registers that are used to transfer data between the CPU and the external devices. The input registers are used to receive data from the external devices, while the output registers are used to send data to the external devices. The data transfer is controlled by the control logic module, which generates the necessary control signals to initiate and complete the transfer.

The I/O module is connected to the LED matrix display through a set of parallel data and control lines. The data lines are used to transfer the pixel data for the "smile face" image from the output registers to the LED matrix. The control lines are used to control the operation of the LED matrix, such as turning individual pixels on and off.

The I/O module is an important part of the CPU design because it allows the CPU to interact with the outside world. It enables the CPU to receive input from external devices and to control external devices, such as the LED matrix display. This makes it possible for the CPU to perform useful tasks, such as displaying the "smile face" image on the LED matrix.

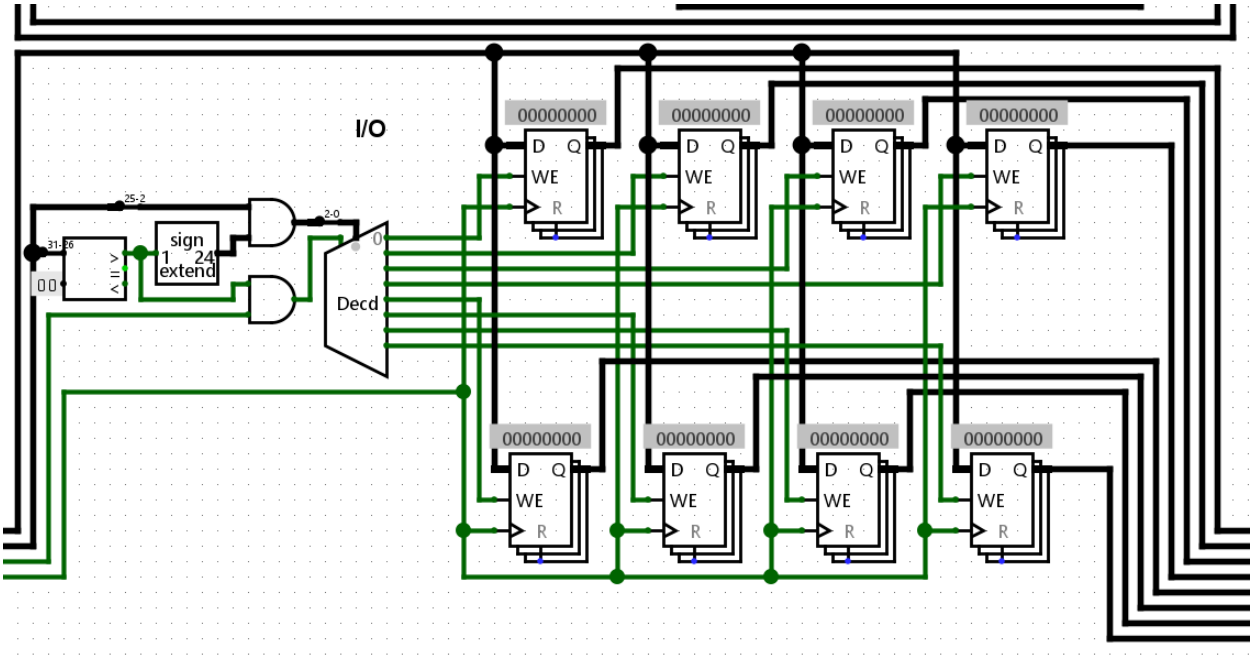

Figure 9: I/O

Register File

The Register File module is an important component of the CPU design, as it provides the storage for data and address values that are used during instruction execution. It consists of 32 registers, each of which is 32 bits wide. The registers are identified by a 5-bit register number, which is used to select the register that is to be accessed.

The Register File module receives input from the ALU and the I/O module and provides data to the other modules as needed. For example, when an instruction specifies that a register should be read, the register number is used to select the appropriate register and the data is passed to the instruction decoder or the ALU for use in the operation. Similarly, when an instruction specifies that a register should be written, the data to be written is provided by the ALU or the I/O module, and the register number is used to select the appropriate register for the write operation.

The Register File module also provides the data that is used to control the LED Matrix module, allowing the "smile face" image to be displayed. This is accomplished by storing the data for each LED in the appropriate register, and then using the register number to select the data for the LED that is to be turned on or off. With 32 registers available, there is plenty of storage space for the necessary data, allowing the CPU to support a wide range of instructions and operations.

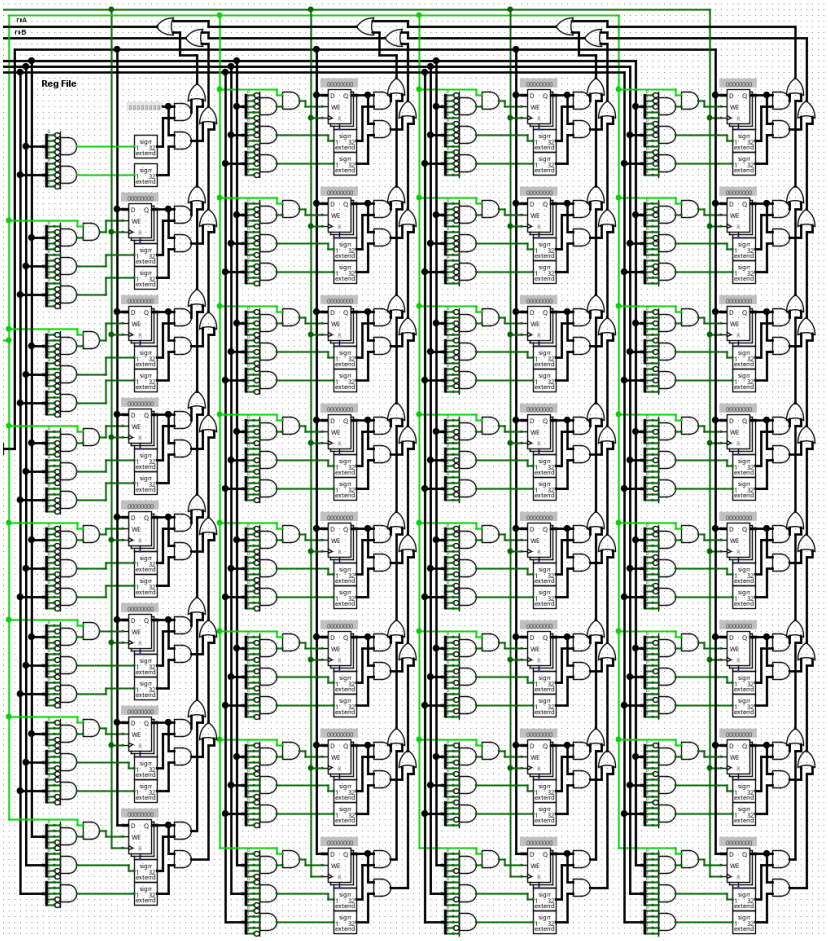

Figure 10: Register File

LED Matrix

The LED Matrix module is used to display the "smile face" image on an LED matrix display. It receives data from the I/O module and uses it to control the individual LEDs in the matrix. The data in the I/O module is generated by the ASM programs that are executed by the CPU, which specify the values of the individual pixels in the matrix.

The LED matrix is a 16x16 grid of LEDs, with each LED being controlled individually by the CPU. To display the "smile face" image, the ASM programs must set the appropriate values in the I/O module to turn on the correct LEDs. This is done by specifying the coordinates of the LED in the matrix and the value to set it to (on or off). The I/O module then uses this information to control the individual LEDs and display the image.

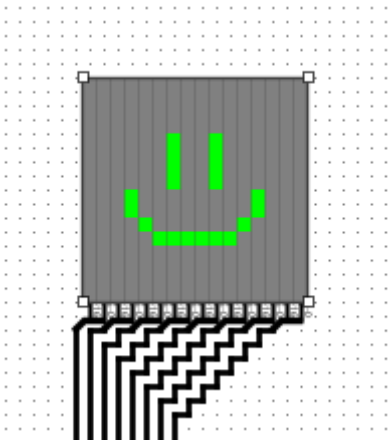

Figure 11: LED Matrix

CPU Software/Firmware Description

The CPU design includes several assembly language programs that control the CPU and enable it to execute the supported instructions. The purpose of these programs is to test the individual instructions as well as to demonstrate the ability to display a picture on the LCD.

The following are some examples the assembly language programs used in testing the CPU:

TestCases

```
# test case 1: addi
# to initialize the reg, we must rely on addi.
addi x1, x0, 1
addi x2, x0, 2
addi x5, x0, 0x18

# test case 2: add
# x3 = x1 + x3
add x3, x1, x2

# test case 3: sub
# x4 = x3 - x1
sub x4, x3, x1

# test case 4: sw
addi x8, x0, 0x50
sw x5, 4(s0)

# test case 5: lw
lw x6, 4(s0)
```

Hex Code:

```
v2.0 raw
00100093
00200113
01800293
002081b3
40118233
05000413
00542223
00442303
```

One of the programs (IMEM) is designed to display a picture on the LCD, specifically a smiley face. This program uses a combination of load and store instructions to transfer the pixel data for the smiley face from the data memory to the display memory, where it can be displayed on the LCD. The program also uses a loop structure to repeat the process, resulting in the smiley face being displayed continuously on the LCD.

IMEM

lui gp, 0x4000	**Hex Code**:
addi s11, x0, 0	v2.0 raw
	040001B7
loop:	00000D93
	000DA403
lw s0, 0(s11)	004DA483
lw s1, 4(s11)	008DA903
lw s2, 8(s11)	00CDA983
lw s3, 12(s11)	010DAA03
lw s4, 16(s11)	014DAA83
lw s5, 20(s11)	018DAB03
lw s6, 24(s11)	01CDAB83
lw s7, 28(s11)	0081A023
sw s0, 0(gp)	0091A223
sw s1, 4(gp)	0121A423
sw s2, 8(gp)	0131A623
sw s3, 12(gp)	0141A823
sw s4, 16(gp)	0151AA23
sw s5, 20(gp)	0161AC23
sw s6, 24(gp)	0171AE23
sw s7, 28(gp)	020D8D93
addi s11, s11, 32	FBDFF06F
jal x0, loop	

The program uses the "lui" and "addi" instructions to set up the memory address for the display and initialize a loop counter, respectively. It then uses the "lw" instruction to load data from the memory into a set of registers. This data represents the pixel values for the smiley face. Finally, the program uses the "sw" instruction to store the pixel values in the display memory, and the "jal" instruction to jump back to the beginning of the loop to repeat the process. This results in the smiley face being displayed on the LCD.

In addition to the instruction memory (IMEM), our CPU design also includes a data memory (DMEM) unit. The DMEM is a memory unit that is used to store data that can be accessed and manipulated by the CPU. In the case of the assembly language program that is used to display a smiley face on the LCD, the DMEM is used to store the pixel data for the smiley face.

DMEM

Hex Code:

```
v2.0 raw
00000000
000000c0
00200010
0f100010
00100f10
00100020
00c00000
00000000
```

To ensure that the smiley face is displayed for a sufficient length of time, the data snippet containing the pixel data has been copied multiple times within the DMEM. This allows the CPU to repeatedly access the data and transfer it to the display memory, resulting in the continuous display of the smiley face on the LCD.

Testing

To test the **TestCases** file:
- Run Logisim Evolution
- Open 'FinalProject.circ'
- Load 'TestCases' into IMEM memory module
- Enable clock

To test the '**Smiley Face**':
- Run Logisim Evolution
- Open 'FinalProject.circ'
- Load 'IMEM' and 'DMEM' to the corresponding memory modules
- Enable clock

Conclusion or Summary

During the course of this project, several things went well, and some challenges were encountered. One of the main successes of the project was the successful development of the hardware and software components of the CPU. The hardware design required careful planning and coordination of the various modules, but the end result was a robust and reliable design that met the specified requirements. The assembly language programs were also successfully developed and tested, allowing the CPU to execute the supported instructions.

One of the challenges encountered during the project was the development of the control logic module. This module is responsible for decoding instructions and generating the appropriate control signals for the other modules in the CPU. Developing this module required a thorough understanding of the instruction set and the underlying architecture and required careful planning and testing to ensure that the correct control signals were generated for each instruction.

If the project were to be started over, there are a few things that could be changed to improve the design. One potential improvement would optimize the hardware design for better performance and efficiency. This could be achieved by reducing the number of components and simplifying the interconnections between the modules, which would make the design easier to understand and maintain.

Overall, this project has achieved its goal of developing a simple CPU that supports a subset of RISC-V instructions. The design is robust and reliable and could be extended and improved in the future by adding more instructions to the instruction set and optimizing the hardware design for better performance and efficiency.