**ISTANBUL TECHNICAL UNIVERSITY**

**COMPUTER ENGINEERING DEPARTMENT**

# BLG 222E
# COMPUTER ORGANIZATION
# PROJECT 1 REPORT

**CRN** : 22599

**LECTURER** : Mustafa Ersel KAMAŞAK, PHD

## GROUP MEMBERS:

150220078 : Alperen AKBAŞ

150230046 : Ömer Faruk EKMEKÇİ (Group Representative)

**SPRING 2025**

# Contents

# 1 INTRODUCTION

In this project, we, Ömer Faruk Ekmekçi and Alperen Akbaş, implemented a basic computer using Verilog hardware description language. Inside this implementation, we can make arithmetic and logical operations. These operations include increment and decrement, adding and substracting numbers, negating, logical, arithmetic and circular shifting and OR, AND, XOR and NAND operations.

## 1.1 Task Distribution

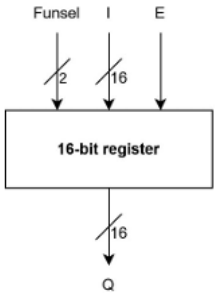
Part 1, Part 2 and a portion of Part 3 were done by Ömer Faruk Ekmekçi and Part 3 and Part 4 were completed by Alperen Akbaş. Troubleshooting throughout the project was done in collaboration. Ömer Faruk EKMEKÇİ is chosen to be the group representative.

# 2 Project Parts

## 2.1 Part 1

### 2.1.1 Part 1a

For this part, we designed a 16-bit register with 4 functionalities which will be controlled by a FunSel signal and an Enable input. Phi symbol means "Don't care". Functionality and illustration of the circuit can be seen in the figure below. We will reuse this register in later parts.

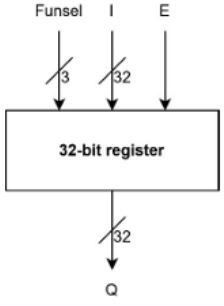



| E | FunSel | Q⁺ |
|---|---|---|
| 0 | ϕ | Q (Retain value) |
| 1 | 00 | Q-1 (Decrement) |
| 1 | 01 | Q+1 (Increment) |
| 1 | 10 | I (Load) |
| 1 | 11 | 0 (Clear) |

Figure 1: Illustration and functionality of the 16-bit register

### 2.1.2 Part 1b

For this part, we designed a 32-bit register with 8 functionalities which will be controlled by a FunSel signal and an Enable input. Phi symbol means "Don't care". Functionality and illustration of the circuit can be seen in the figure below. We will reuse this register in later parts as well.



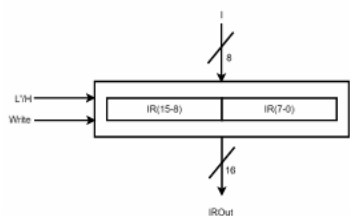

| E | FunSel | Q⁺ |
|---|---|---|
| 0 | ϕ | Q (Retain value) |
| 1 | 000 | Q-1 (Decrement) |
| 1 | 001 | Q+1 (Increment) |
| 1 | 010 | I (Load) |
| 1 | 011 | 0 (Clear) |
| 1 | 100 | Q (31-8) ← Clear, Q (7-0) ← I (7-0) |
| 1 | 101 | Q (31-16) ← Clear, Q (15-0) ← I (15-0) |
| 1 | 110 | Q (31-8) ← Q (23-0) (8-bit Left Shift)<br>Q (7-0) ← I (7-0) |
| 1 | 111 | Q (31-16) ← Sign Extend (I (15)) (16-bit sign extend)<br>Q (15-0) ← I (15-0) |

Figure 2: Illustration and functionality of the 32-bit register

## 2.2 Part 2

### 2.2.1 Part 2a

We designed a 16-bit Instruction Register for this part. Our IR can store 16 bits data but takes 8 bit inputs. Functionalities will be controlled by Write and L'H inputs. Phi symbol means "Don't care". L'H signal decides whether the lower or upper half will be loaded from the bus. Functionality and illustration of the circuit can be seen in the figure below.

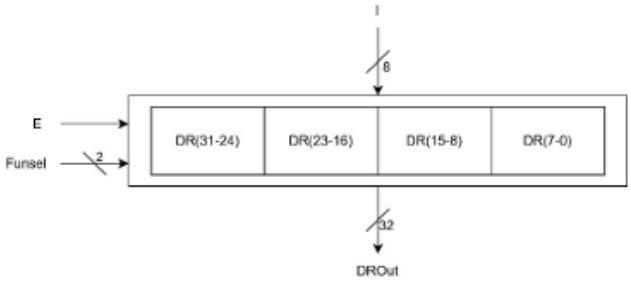



| L'H | Write | IR+ |
|---|---|---|
| ϕ | 0 | IR (Retain value) |
| 0 | 1 | IR (7-0) ← I (Load LSB) |
| 1 | 1 | IR (15-8) ← I (Load MSB) |

Figure 3: Illustration and functionality of the Instruction Register

### 2.2.2 Part 2b

For this part of the experiment, we designed a 16-bit Data Register. This register stores 32 bits data but takes 8 bit inputs, similar to Instruction Register. Functionalities will be controlled by FunSel inputs. Phi symbol means "Don't care". Functionality and illustration of the circuit can be seen in the figure below.




| E | FunSel | DR+ |
|---|---|---|
| 0 | ϕ | DR (Retain value) |
| 1 | 00 | DR (31-8) ← Sign Extend (I (7)) (8-bit sign extend)<br>DR (7-0) ← I (7-0) |
| 1 | 01 | DR (31-8) ← Clear,<br>DR (7-0) ← I (7-0) |
| 1 | 10 | DR (31-8) ← DR (23-0) (8-bit Left Shift)<br>DR (7-0) ← I (7-0) |
| 1 | 11 | DR (23-0) ← DR (31-8) (8-bit Right Shift)<br>DR (31-24) ← I (7-0) |

Figure 4: Illustration and functionality of the Data Register

### 2.2.3 Part 2c

This section was the hardest part of Part 2 of the experiment and took a long time to understand and implement. Our task was designing and implementing a Register File consisting of 8 registers in total, 4 of them being 32 bit general purpose registers and 4 of them being scratch registers. Illustration of the Register File can be seen in the figure below.

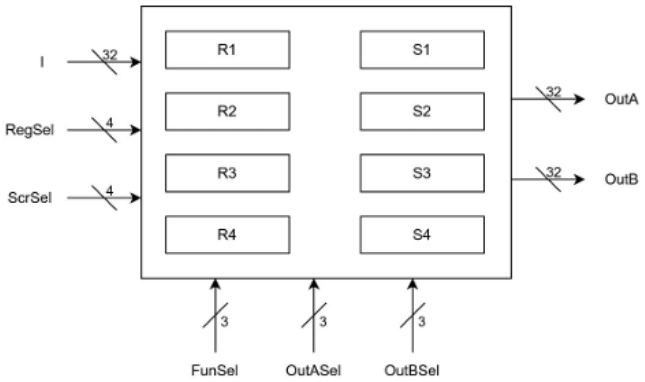



Figure 5: Illustration of the Register File

OutASel and OutBSel are the selector inputs of our Register File. These inputs will decide which of the included 8 registers output the their data to OutA and OutB. These selections can be seen in the figure below.

Also, which of the 8 functions the Register File is capable of will be selected by FunSel inputs before these functions are applied to the data in the R1, R2, R3, R4, S1, S2, S3 and S4 registers. This selection is also given in the figure below.

| OutASel | OutA |
|---|---|
| 000 | R1 |
| 001 | R2 |
| 010 | R3 |
| 011 | R4 |
| 100 | S1 |
| 101 | S2 |
| 110 | S3 |
| 111 | S4 |

| OutBSel | OutB |
|---|---|
| 000 | R1 |
| 001 | R2 |
| 010 | R3 |
| 011 | R4 |
| 100 | S1 |
| 101 | S2 |
| 110 | S3 |
| 111 | S4 |

| FunSel | $R_x^+$ (Next State) |
|---|---|
| 000 | $R_x$-1 (Decrement) |
| 001 | $R_x$ +1 (Increment) |
| 010 | I (Load) |
| 011 | 0 (Clear) |
| 100 | $R_x$ (31-8) ← Clear, $R_x$ (7-0) ← I (7-0) |
| 101 | $R_x$ (31-16) ← Clear, $R_x$ (15-0) ← I (15-0) |
| 110 | $R_x$ (31-8) ← $R_x$ (23-0) (8-bit Left Shift)<br>$R_x$ (7-0) ← I (7-0) |
| 111 | $R_x$ (31-16) ← Sign Extend (I (15)) (16-bit sign extend)<br>$R_x$ (15-0) ← I (15-0) |

Figure 6: Functionality of the Register File

ReqSel control inputs select which of the 8 included registers will be active or passive. Active registers will have their data be the inputs to the functions selected by FunSel inputs. Outputs of these functions will be outputted through the OutA.

| RegSel | Enable General Purpose Registers | RegSel | Enable General Purpose Registers |
|---|---|---|---|
| 0000 | NO general purpose register is enabled. (All registers retain their values.) | 1000 | Only R1 is enabled. (Function selected by FunSel will be applied to R1.) |
| 0001 | Only R4 is enabled. (Function selected by FunSel will be applied to R4.) | 1001 | R1 and R4 are enabled. (Function selected by FunSel will be applied to R1 and R4.) |
| 0010 | Only R3 is enabled. (Function selected by FunSel will be applied to R3.) | 1010 | R1 and R3 are enabled. (Function selected by FunSel will be applied to R1 and R3.) |
| 0011 | R3 and R4 are enabled. (Function selected by FunSel will be applied to R3 and R4.) | 1011 | R1, R3, and R4 are enabled. (Function selected by FunSel will be applied to R1, R3, and R4.) |
| 0100 | Only R2 is enabled. (Function selected by FunSel will be applied to R2.) | 1100 | R1 and R2 are enabled. (Function selected by FunSel will be applied to R1 and R2.) |
| 0101 | R2 and R4 are enabled. (Function selected by FunSel will be applied to R2 and R4.) | 1101 | R1, R2, and R4 are enabled. (Function selected by FunSel will be applied to R1, R2, and R4.) |
| 0110 | R2 and R3 are enabled. (Function selected by FunSel will be applied to R2 and R3.) | 1110 | R1, R2 and R3 are enabled. (Function selected by FunSel will be applied to R1, R2, and R3.) |
| 0111 | R2, R3, and R4 are enabled. (Function selected by FunSel will be applied to R2, R3, and R4.) | 1111 | All general purpose registers are enabled. (Function selected by FunSel will be applied to R1, R2, R3 and R4.) |

Figure 7: R Register Selection with ReqSel inputs

Similar to the previous part, ScrSel control inputs select which of the 8 included registers will be active or passive. Active registers will have their data be the inputs to the functions selected by FunSel inputs. Outputs of these functions will be outputted through the OutB.

| ScrSel | Enable General Purpose Registers | ScrSel | Enable General Purpose Registers |
|---|---|---|---|
| 0000 | NO general purpose register is enabled. (All registers retain their values.) | 1000 | Only S1 is enabled. (Function selected by FunSel will be applied to S1.) |
| 0001 | Only S4 is enabled. (Function selected by FunSel will be applied to S4.) | 1001 | S1 and S4 are enabled. (Function selected by FunSel will be applied to S1 and S4.) |
| 0010 | Only S3 is enabled. (Function selected by FunSel will be applied to S3.) | 1010 | S1 and S3 are enabled. (Function selected by FunSel will be applied to S1 and S3.) |
| 0011 | S3 and S4 are enabled. (Function selected by FunSel will be applied to S3 and S4.) | 1011 | S1, S3, and S4 are enabled. (Function selected by FunSel will be applied to S1, S3, and S4.) |
| 0100 | Only S2 is enabled. (Function selected by FunSel will be applied to S2.) | 1100 | S1 and S2 are enabled. (Function selected by FunSel will be applied to S1 and S2.) |
| 0101 | S2 and S4 are enabled. (Function selected by FunSel will be applied to S2 and S4.) | 1101 | S1, S2, and S4 are enabled. (Function selected by FunSel will be applied to S1, S2, and S4.) |
| 0110 | S2 and S3 are enabled. (Function selected by FunSel will be applied to S2 and S3.) | 1110 | S1, S2, and S3 are enabled. (Function selected by FunSel will be applied to S1, S2, and S3.) |
| 0111 | S2, S3, and S4 are enabled. (Function selected by FunSel will be applied to S2, S3, and S4.) | 1111 | All general purpose registers are enabled. (Function selected by FunSel will be applied to S1, S2, S3, and S4.) |

Figure 8: S Register Selection with ScrSel inputs

### 2.2.4 Part 2d

For this part, we designed and implemented an Address Register File. This register consists of three 16-bit address registers, a Stack Pointer, a Program Counter and an Adress Register.

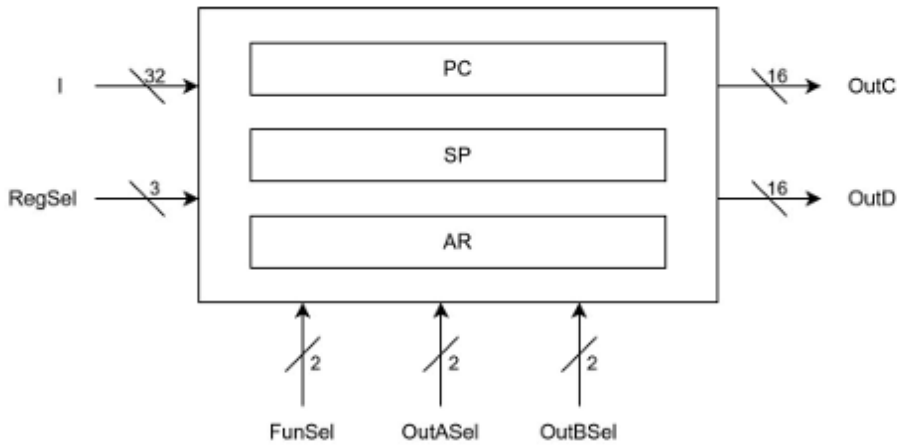



Figure 9: Illustration of the Address Register File

The inputs from OutCSel and OutDSel are used to select which register's data will be modified using the implemented functions and the result will be fed through OutC and OutD. How this operation will be conducted is shown in the figure below.

How the functions will be selected with the FunSel bits and how to determine which registers will be selected using ReqSel inputs are also given in the figure below.

| OutCSel | OutC |
|---|---|
| 00 | PC |
| 01 | SP |
| 10 | AR |
| 11 | AR |

| OutDSel | OutD |
|---|---|
| 00 | PC |
| 01 | SP |
| 10 | AR |
| 11 | AR |

| FunSel | $R_x^+$ (Next State) |
|---|---|
| 00 | $R_x$-1 (Decrement) |
| 01 | $R_x$ +1 (Increment) |
| 10 | I (Load) |
| 11 | 0 (Clear) |

| ScrSel | Enable General Purpose Registers | ScrSel | Enable General Purpose Registers |
|---|---|---|---|
| 000 | NO registers is enabled. (All registers retain their values.) | 100 | Only PC is enabled. (Function selected by FunSel will be applied to PC.) |
| 001 | Only AR is enabled. (Function selected by FunSel will be applied to AR.) | 101 | PC and AR are enabled. (Function selected by FunSel will be applied to PC.) |
| 010 | Only SP is enabled. (Function selected by FunSel will be applied to SP.) | 110 | PC and SP are enabled. (Function selected by FunSel will be applied to PC and SP.) |
| 011 | SP and AR are enabled. (Function selected by FunSel will be applied to SP and AR.) | 111 | All registers are enabled. (Function selected by FunSel will be applied to PC, SP, and AR.) |

Figure 10: Functionality of the Address Register File

## 2.3 Part 3

For this part, we were tasked with designing and implementing an Arithmetic Logic Unit with two 32-bit inputs and a 32-bit output along with multiple sets of selector bits and flags.

These flags include Zero (Z), Carry (C), Negative (N), and Overflow (O). The ALU operates based on the FunSel input and uses 2's complement meethod in the arithmetic operations. Flag updates occur only when WF (Write Flag) is set.

Z is set if ALUOut is zero.

C is set if a carry is generated during operations.

N is set if the result is negative.

O is set if overflow occurs.

Flags are stored in a register in this order: "ZCNO"

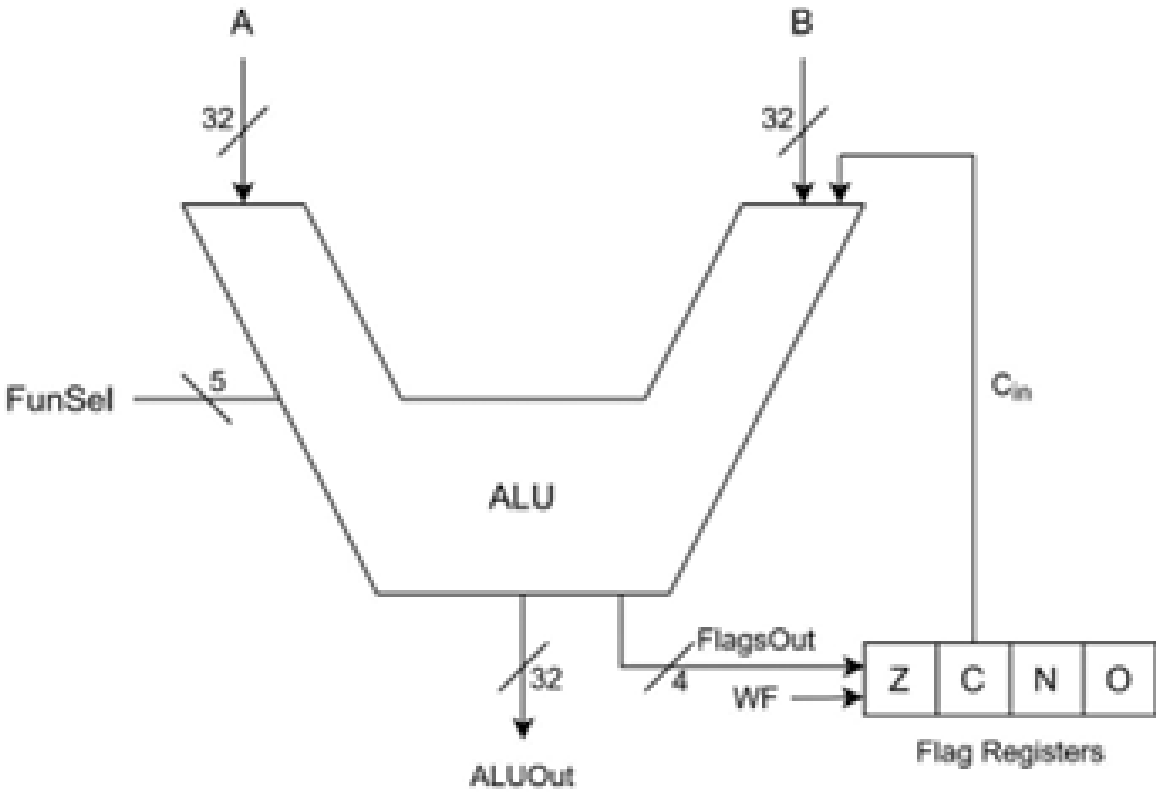



Figure 11: Illustration of the Arithmetic Logic Unit

The tables below depict which functions will be used depending on the input from FunSel bits. Also the information on which flags will be disabled for what operations can be found in the figure.

| FunSel | ALUOut | Z | C | N | O |
|---|---|---|---|---|---|
| 00000 | A (16-bit) | + | - | + | - |
| 00001 | B (16-bit) | + | - | + | - |
| 00010 | NOT A (16-bit) | + | - | + | - |
| 00011 | NOT B (16-bit) | + | - | + | - |
| 00100 | A + B (16-bit) | + | + | + | + |
| 00101 | A + B + Carry (16-bit) | + | + | + | + |
| 00110 | A – B (16-bit) | + | + | + | + |
| 00111 | A AND B (16-bit) | + | - | + | - |
| 01000 | A OR B (16-bit) | + | - | + | - |
| 01001 | A XOR B (16-bit) | + | - | + | - |
| 01010 | A NAND B (16-bit) | + | - | + | - |
| 01011 | LSL A (16-bit) | + | + | + | - |
| 01100 | LSR A (16-bit) | + | + | + | - |
| 01101 | ASR A (16-bit) | + | - | - | - |
| 01110 | CSL A (16-bit) | + | + | + | - |
| 01111 | CSR A (16-bit) | + | + | + | - |

| FunSel | ALUOut | Z | C | N | O |
|---|---|---|---|---|---|
| 10000 | A (32-bit) | + | - | + | - |
| 10001 | B (32-bit) | + | - | + | - |
| 10010 | NOT A (32-bit) | + | - | + | - |
| 10011 | NOT B (32-bit) | + | - | + | - |
| 10100 | A + B (32-bit) | + | + | + | + |
| 10101 | A + B + Carry (32-bit) | + | + | + | + |
| 10110 | A – B (32-bit) | + | + | + | + |
| 10111 | A AND B (32-bit) | + | - | + | - |
| 11000 | A OR B (32-bit) | + | - | + | - |
| 11001 | A XOR B (32-bit) | + | - | + | - |
| 11010 | A NAND B (32-bit) | + | - | + | - |
| 11011 | LSL A (32-bit) | + | + | + | - |
| 11100 | LSR A (32-bit) | + | + | + | - |
| 11101 | ASR A (32-bit) | + | - | - | - |
| 11110 | CSL A (32-bit) | + | + | + | - |
| 11111 | CSR A (32-bit) | + | + | + | - |

Figure 12: Characteristic table of the Arithmetic Logic Unit

## 2.4 Part 4

In this part of the experiment, our task was implementing the circuit shown in the figure, using all the registers and other components we designed so far. This is a very complicated system and it uses a single clock throughout the whole structure, therefore it was a challenge implementing it. Despite not designing any new components, it was difficult to organize everything into a working basic computer like we were tasked to do.

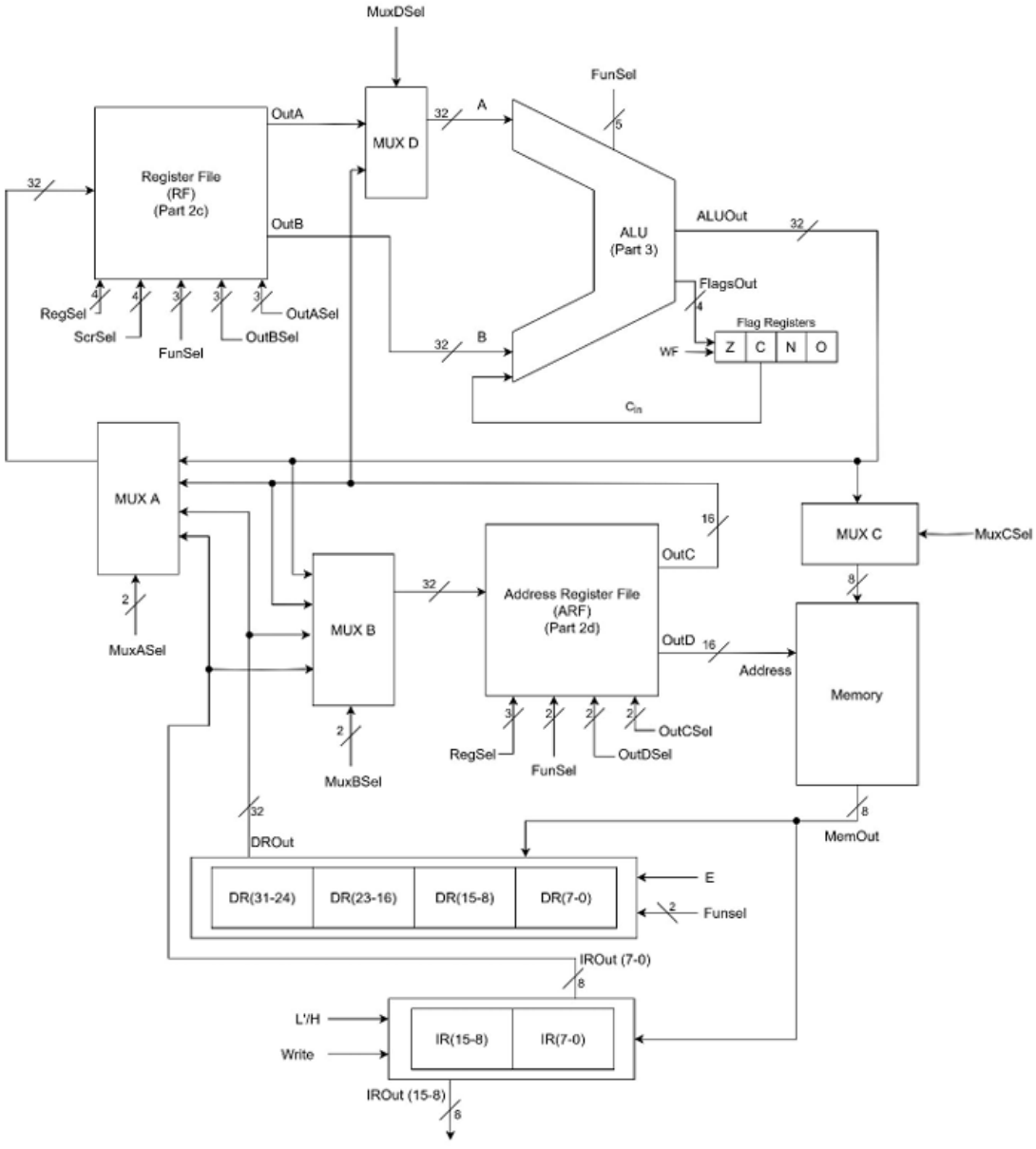



Figure 13: A Basic Computer with a single clock, consisting of all the components we designed so far

# 3 RESULTS

In this section, the schematics will be presented, starting with Part 2.

## 3.1 Part 2

## 3.2 Part 2a

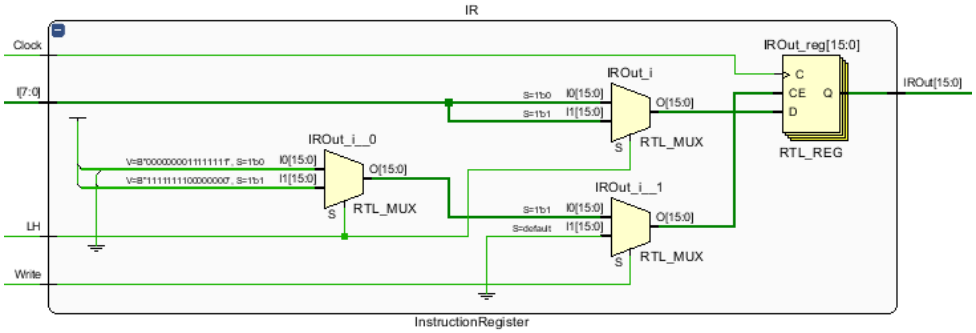



Figure 14: Schematic for IR

## 3.3 Part 2b

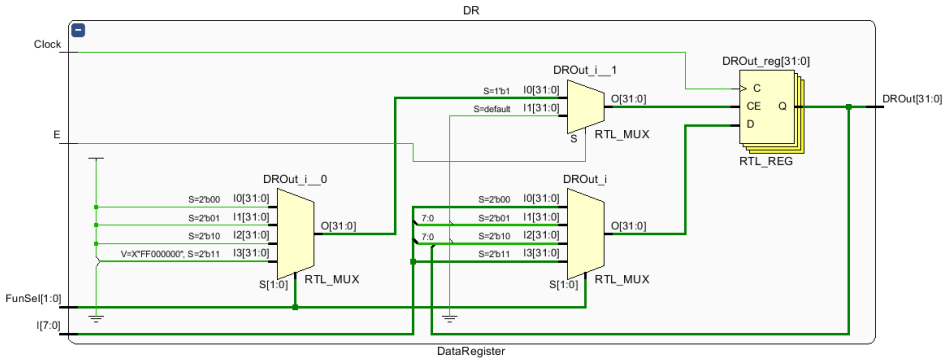



Figure 15: Schematic for DR

## 3.4 Part 2c

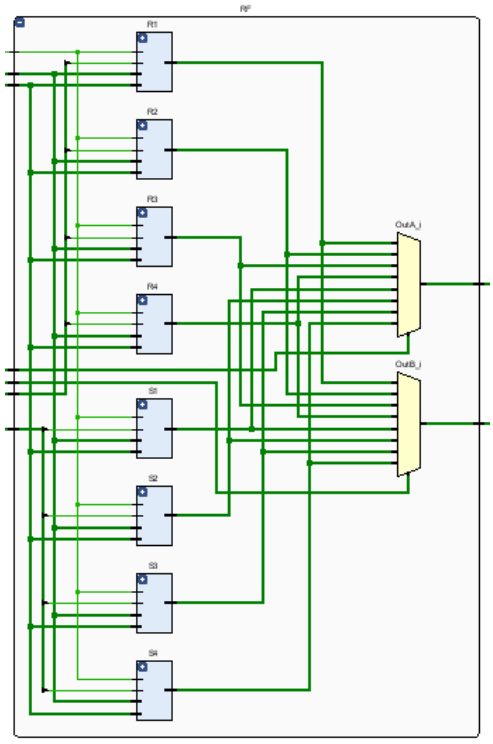



Figure 16: Schematic for RF

## 3.5 Part 2d

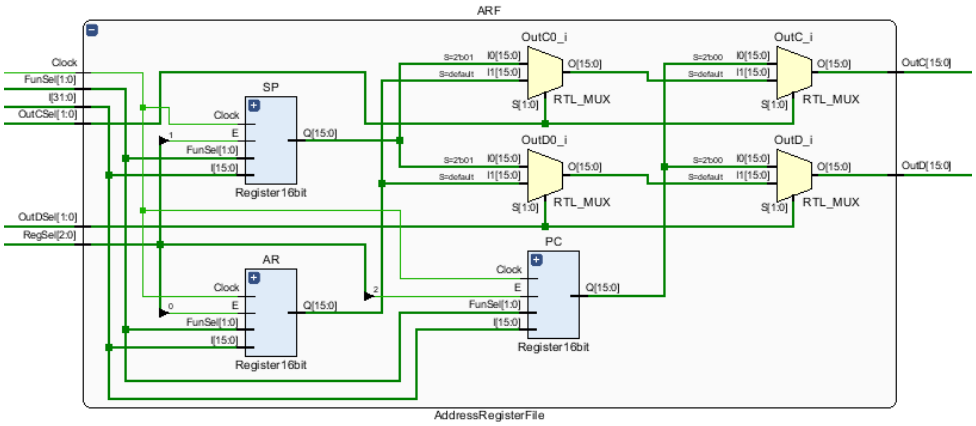



Figure 17: Schematic for ARF

## 3.6 Part 3

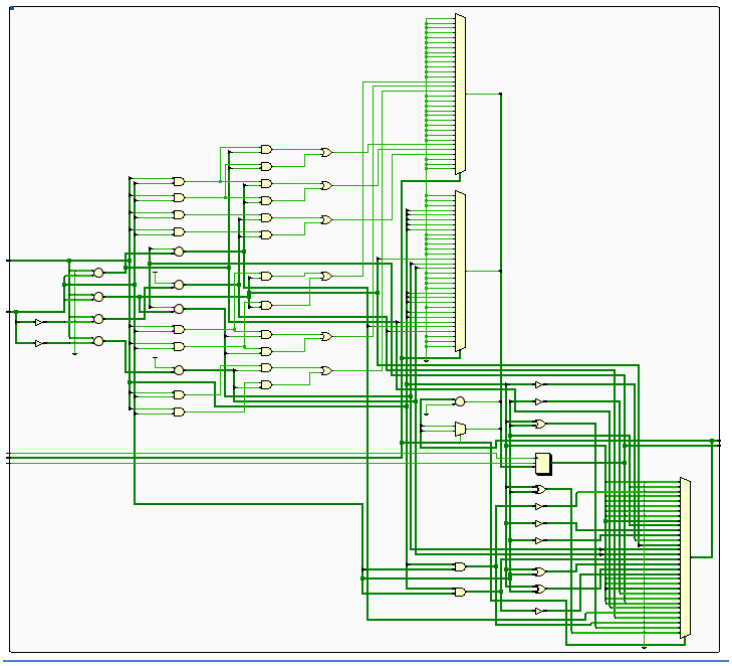


Figure 18: Schematic for ALU

## 3.7 Part 4

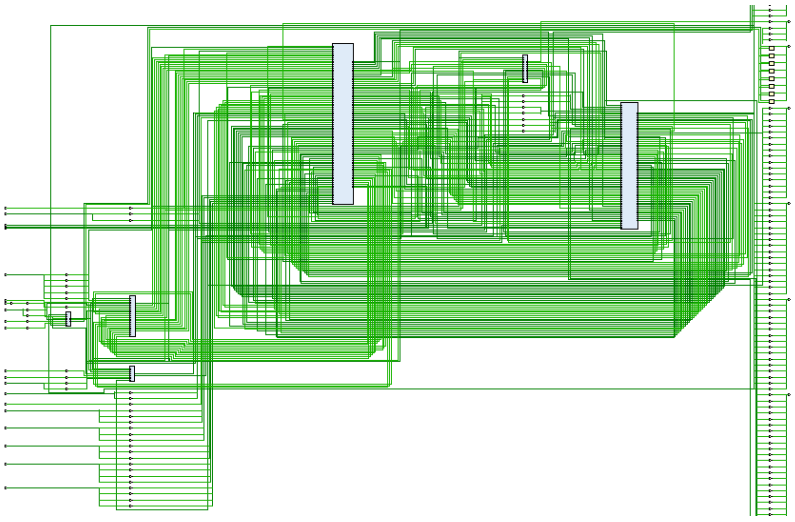


Figure 19: Schematic for the entire circuit

# 4 DISCUSSION

While doing this project, we gained a better understanding of how low-level hardware components work in collaboration to form the core system of a functioning computer. Starting from basic register designs and adders, we got to designing a complete ALU. This project helped us strengthen our theoretical knowledge with real-world Verilog implementations.

One of the hardest challenges we faced was ensuring clock signals worked correctly throughout the system and all the modules, especially in the register file and ALU, where many, many control lines and operations had to be selected and used in the functions. Troubleshooting these sections required a lot of care and fixing the errors one by one.

In addition to that, the design and implementation of the Address Register File and the required logic for the flags in the ALU emphasized the importance of doing the planning beforehand, and being very careful while doing so. Small moments of lach of caution led to wrong outputs multiple times.

Working together while troubleshooting improved our efficiency significantly. We covered each other's mistakes and worked very quickly. By dividing the workload and responsibilities and frequently checking on each other, we managed to fing and fix issues quickly.

Overall, this project provided us with invaluable experience in hardware design and using Verilog. It improved our appreciation of how processors run instructions at a very low level, and contributed to our skills in modular design, debugging, and digital circuits integration.

# 5 CONCLUSION

In this project, we didn't face many issues during the first and second parts. Everything went well until we started implementing the ALU. That's where we started to face problems. Small syntax errors that was caused by a moment's blunder took a lot of time to fix. One of the biggest difficulties we faced was implementing the flag checks. At first, we were moving slowly because we were confused about the homework instructions. After spending some time figuring it out, we finally understood what to do and managed to implement the necessary changes.

We learned a lot about Verilog during this project. It was our first time using a hardware description language, it is even lower level than C, which we've been taking courses for for two semesters. It was intriguing to see how things work at such a low level. We also had the chance to work with components like adders, registers, multiplexers, and more. While we learned about these parts in our digital circuits course, this was the first time we actually designed them from scratch.

Overall, the project helped us build the bridges between theory and real world use cases. It gave us a better understanding of how computers really work under the hood, and now we feel we are more knowledgable about digital design. Even though there were points where coding was frustrating, it feels good to see our designs come together and function properly in the end.

# REFERENCES

[1] BLG222E Computer Organization Project 1 Homework Definition