ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION PROJECT 1 REPORT

CRN	:	21336
LECTURER	:	Mustafa Kamaşak

GROUP MEMBERS:

150200097	:	Mustafa Can Çalışkan
150200016	:	Yusuf Şahin

SPRING 2024

Contents

1 INTRODUCTION **1**

2 MATERIALS AND METHODS **1**

2.1 16-bit Register (Part 1) 1

2.2 Instruction Register (Part 2a) 2

2.3 Register File (Part 2b) 2

2.4 Address Register File (Part 2c) 3

2.5 Arithmetic Logic Unit (ALU) (Part 3) 4

2.6 Arithmetic Logic Unit System (Part 4) 5

3 RESULTS [15 points] **6**

3.1 16-bit Register (Part 1) 6

3.2 Instruction Register (Part 2a) 6

3.3 Register File (Part 2b) 7

3.4 Address Register File (Part 2c) 7

3.5 Arithmetic Logic Unit (ALU) (Part 3) 7

3.6 Arithmetic Logic Unit System (Part 4) 8

4 DISCUSSION **9**

5 CONCLUSION **9**

REFERENCES **10**

1 INTRODUCTION

In this project, we have implemented an Arithmetic Logic Unit System gradually. We have used register file, instruction register, address register file, Arithmetic Logic Unit (ALU), and ALU System. All of these components and their functionalities have been designed by us.

2 MATERIALS AND METHODS

We have arranged the task division so that Can Çalışkan will implement 16-bit register, register file, and 16-bit IR register while Yusuf Şahin will handle address register file, Arithmetic Logic Unit (ALU), and Arithmetic Logic Unit System. The report was entirely prepared through a collaborative effort.

2.1 16-bit Register (Part 1)

In this section, a 16-bit register was designed. The register includes a 16-bit input I, a 3-bit FunSel, a 1-bit Enable, and a 16-bit output Q. The register performs operations based on the operations provided in FunSel within an "always" block corresponding to each case of FunSel. To ensure compatibility with other parts of the system, the register executes its operations during the positive (rising) edge of the clock intervals.

The schematic of the designed register can be seen in Figure 1.

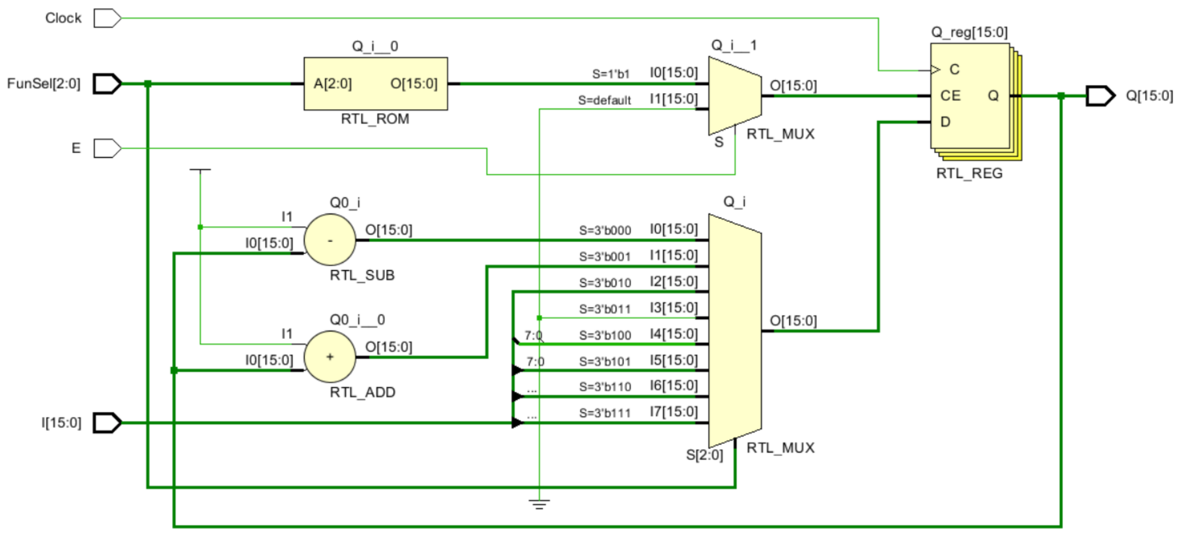


Figure 1: Register Schematic

2.2 Instruction Register (Part 2a)

In this section, we designed the instruction register (IR). The module takes 8-bit I, 1-bit L/H, and 1-bit Write as inputs, and includes 16-bit IROut as output. The register performs operations based on the operations provided in FunSel within an "always" block corresponding to each case of W and L/H. Since the instruction register itself is a register and needs to synchronize with other parts of the system, the "always" block operates on positive (rising) clock edges.

The schematic of the designed instruction register can be seen in Figure 2.

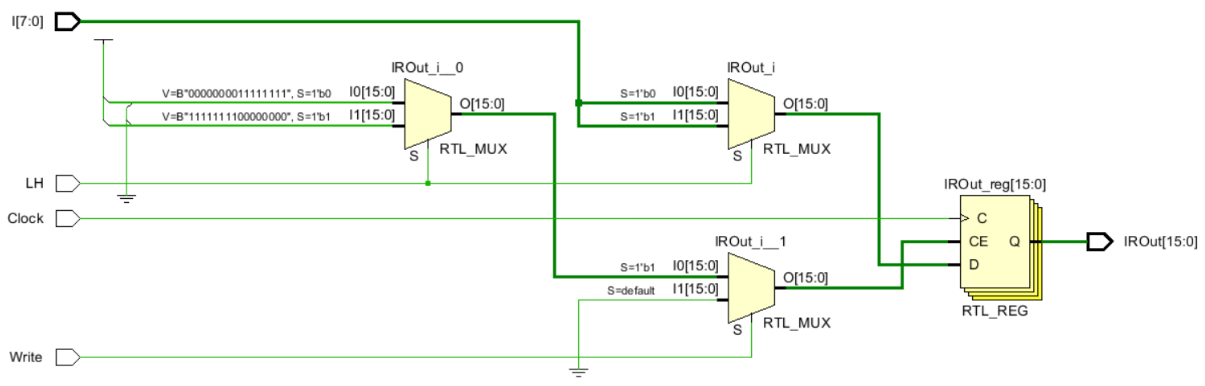


Figure 2: Instruction Register (IR) Schematic

2.3 Register File (Part 2b)

In this section, we designed the register file. The module takes 16-bit I as input and includes 16-bit OutA and 16-bit OutB as outputs. Additionally, the module features 4-bit inputs for RegSel and ScrSel to select the active registers, a 3-bit FunSel to determine the operations to be performed on the active registers, and finally, 3-bit inputs for OutASel and OutBSel to specify which registers will be output as OutA and OutB from the module. We designed the module by importing the Register module we previously wrote to incorporate the registers it possesses (R1-R4, S1-S4).

The schematic of the designed register file can be seen in Figure 3.

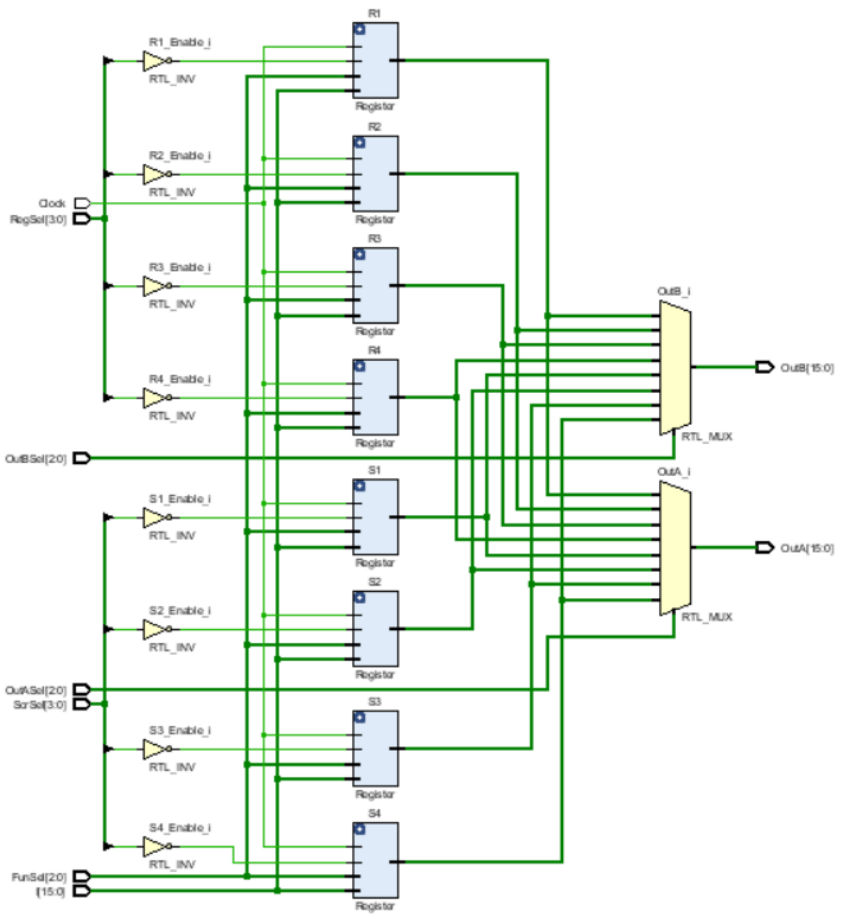


Figure 3: Register File Schematic

2.4 Address Register File (Part 2c)

In this section, we designed the address register file (ARF). The module includes a 16-bit input and 16-bit OutC and OutD each. Additionally, similar to the register file module, it features 3-bit RegSel to determine the active registers and 2-bit OutCSel and OutDSel to specify which registers will be output through OutC and OutD from the module. We designed the module by importing the Register module we previously wrote to incorporate the registers it possesses (AR, SP, PC).

The schematic of the designed address register file can be seen in Figure 4.

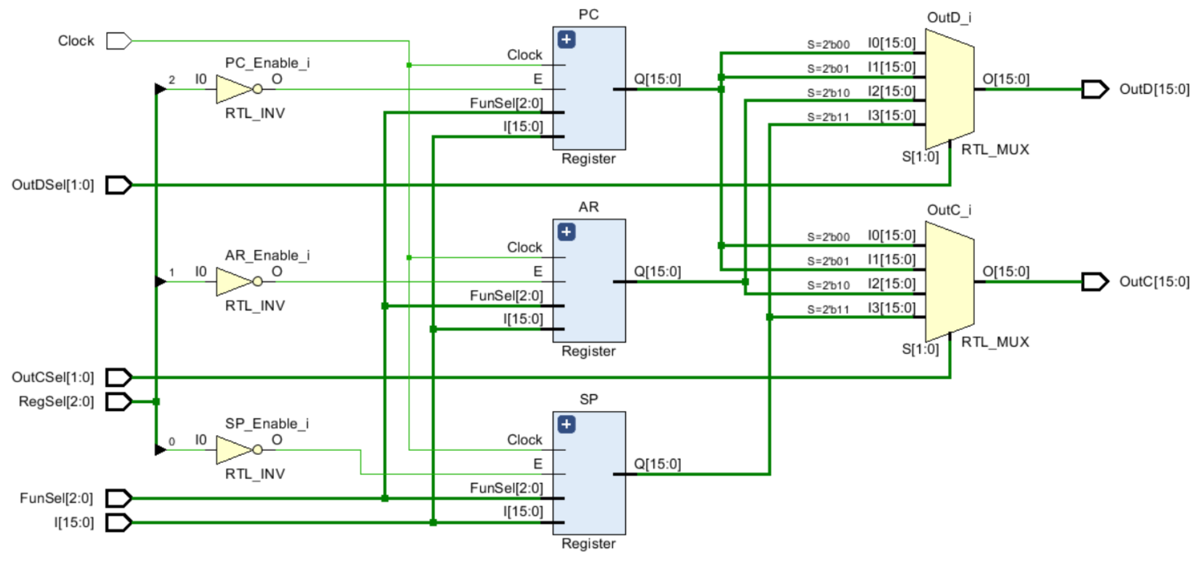


Figure 4: Address Register File Schematic

2.5 Arithmetic Logic Unit (ALU) (Part 3)

In this section, we have implemented the design of the Arithmetic Logic Unit (ALU). The ALU takes 16-bit inputs A and B, and produces a 16-bit output ALUOut. To determine which operations will be performed with A and B, it includes a 5-bit FunSel. Additionally, the module contains a Flags Register which is modified based on the result of these operations on ALUOut. If WF is provided as input to the module and is set to 1, placement into the Flags Register can be performed. For the Flags Register to operate compatibly with other components in the system, it performs operations on a positive (rising) edge.

The schematic of the designed arithmetic logic unit can be seen in Figure 5.

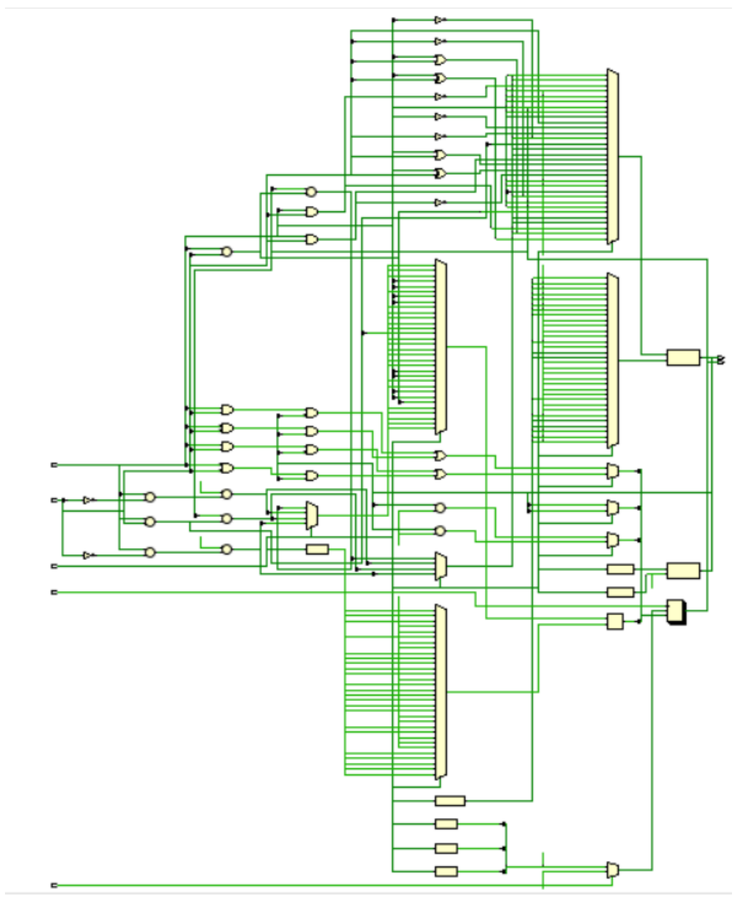


Figure 5: Arithmetic Logic Unit Schematic

2.6 Arithmetic Logic Unit System (Part 4)

As a final section, previously designed modules were combined by incorporating the Memory module. The connections between them were established using A, B, and C multiplexers designed using "always" blocks.

The schematic of the designed arithmetic logic unit system can be seen in Figure 6.

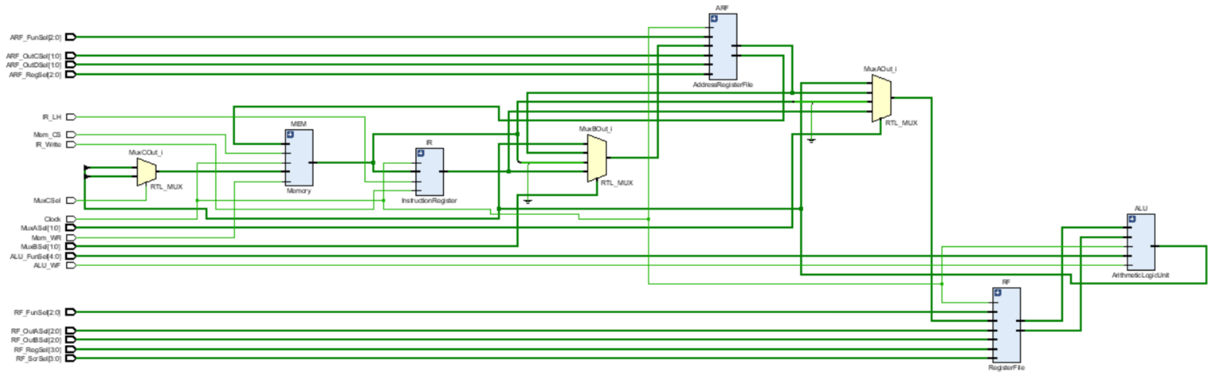


Figure 6: Arithmetic Logic Unit System Schematic

3 RESULTS [15 points]

We utilized not only the pre-existing simulation files provided beforehand but also our own simulation files during the stage of simulating the project and obtaining results.

3.1 16-bit Register (Part 1)

The results of the test prepared for the register can be seen in Figure 7.

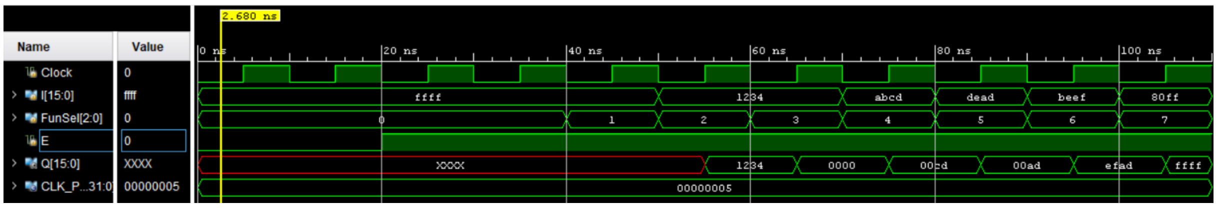


Figure 7: Register Test

As evident from the test, initially when Enable is 1 and FunSel is 0x2 (Load), Q remains in an undefined state until the Load operation is executed. Subsequently, it successfully executes the operations of Clear, Left Half Clear-Right Half Load, Right Half Load, Left Half Load, and Left Half Sign Extended-Right Half Load in sequence. The clock period is set to 5ns.

3.2 Instruction Register (Part 2a)

The results of the test prepared for the instruction register can be seen in Figure 8.

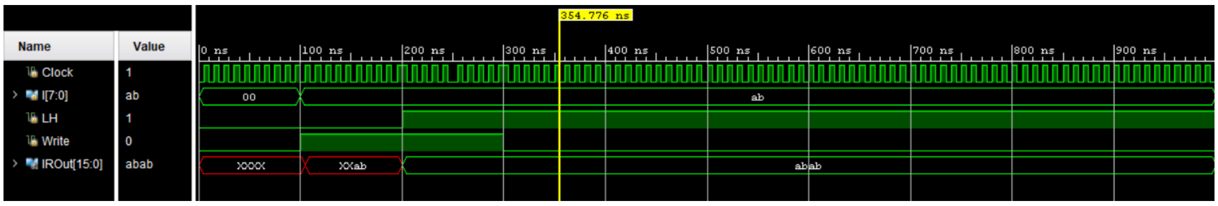
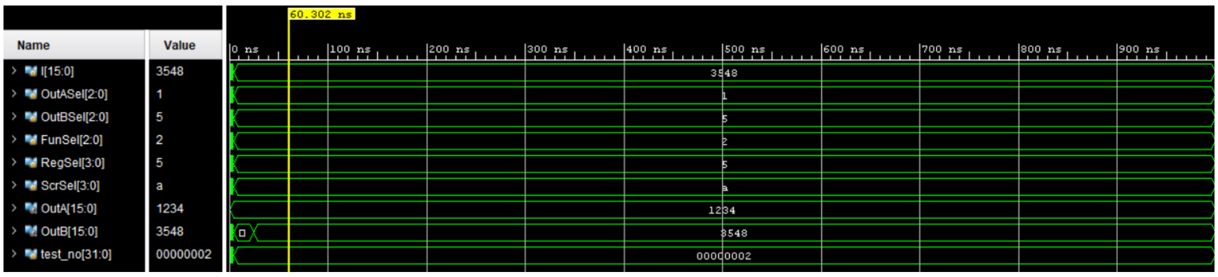


Figure 8: Instruction Register (IR) Test

As evident from the test, initially, the value of Q remains undefined until Write becomes 1. Once Write becomes 1, when L/H is 0, it loads the right half of Q, and when L/H is 1, it loads the left half of Q.

3.3 Register File (Part 2b)

The results of the test prepared for the register file can be seen in Figure 9.



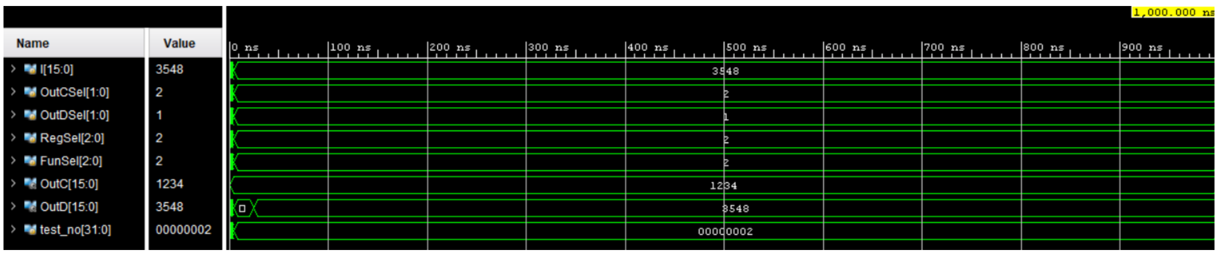
Name	Value
I[15:0]	3548
OutASel[2:0]	1
OutBSel[2:0]	5
FunSel[2:0]	2
RegSel[3:0]	5
ScrSel[3:0]	a
OutA[15:0]	1234
OutB[15:0]	3548
test_no[31:0]	00000002

Figure 9: Register File Test

The expected results were obtained in tests conducted based on control inputs.

3.4 Address Register File (Part 2c)

The results of the test prepared for the address register file can be seen in Figure 10.



Name	Value
I[15:0]	3548
OutCSel[1:0]	2
OutDSel[1:0]	1
RegSel[2:0]	2
FunSel[2:0]	2
OutC[15:0]	1234
OutD[15:0]	3548
test_no[31:0]	00000002

Figure 10: Address Register File Test

Initially, the 16-bit input I is set to the hexadecimal value 3548, which seems to be a constant for the duration of the simulation. The simulation starts with OutCSel and OutDSel—the control signals responsible for selecting the output registers—set to values 2 and 1, respectively. These selectors likely correspond to specific registers from which the output values OutC and OutD are read. OutC begins with a value of 1234, then subsequently changes to 3548, matching the input I. This suggests an operation has occurred, possibly a load operation that has written the input I into the register corresponding to OutC.

3.5 Arithmetic Logic Unit (ALU) (Part 3)

The results of the test prepared for the arithmetic logic unit can be seen in Figure 11.

The ALU begins its operation with input values of 0x7777 for A and 0x8889 for B. It is configured to perform the function designated by FunSel value 15, and the WF is high, enabling the ALU to update its status flags. As the operation completes, the Z flag is

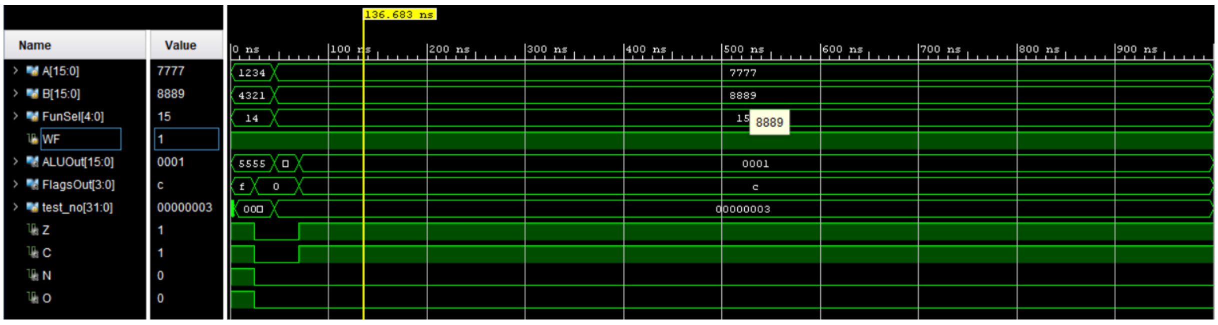


Figure 11: Arithmetic Logic Unit Test

set, indicating a zero result from the operation, and the C flag is set, indicating a carry out. No negative or overflow conditions are flagged. The ALUOut briefly shows a result of 0001 before becoming undefined, which could be due to the ALU transitioning to the next operation or due to the timing of signal changes within the simulation environment. The test number increments, suggesting that the testbench has moved on to the next predefined test case, changing FunSel to 14 to initiate a different operation by the ALU.

3.6 Arithmetic Logic Unit System (Part 4)

The results of the test prepared for the arithmetic logic unit system can be seen in Figure 12.

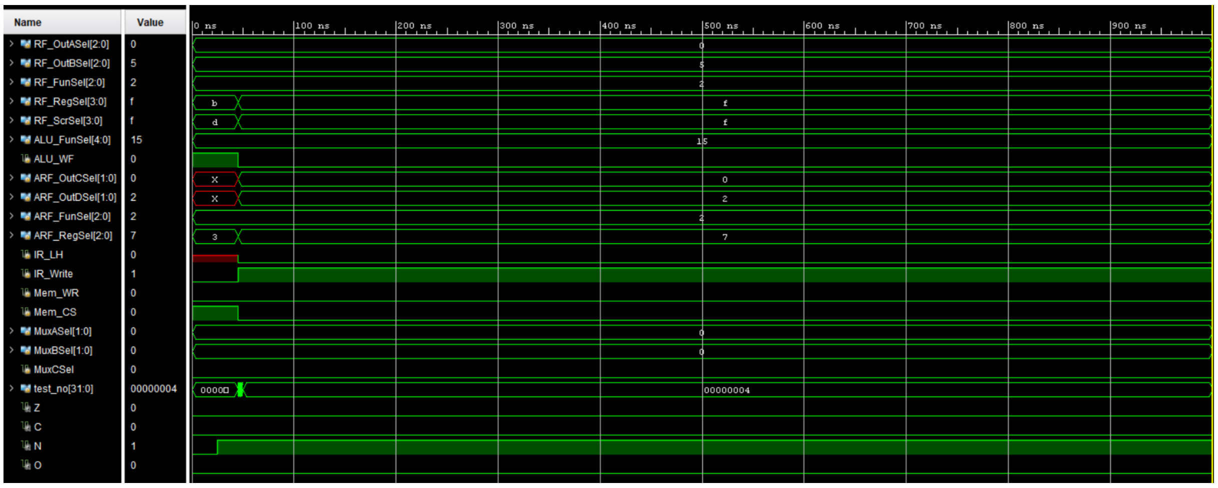


Figure 12: Arithmetic Logic Unit System Test

The expected results were obtained in tests conducted based on control inputs.

4 DISCUSSION

Throughout the project, we ensured compatibility with system components and rigorously tested each module. Challenges included synchronizing signals across components, addressed through careful timing considerations and the use of “always” blocks synchronized with positive clock edges. The project provided valuable hands-on experience with digital design principles and tools such as Xilinx Vivado and Verilog, laying a solid foundation for future exploration in computer architecture and hardware design.

5 CONCLUSION

This project is vital for understanding the basics of a arithmetic logic unit system. With this project, we have extended our knowledge and insight on how 0 and 1s can build a computer with its components. Also, this project has given information to us for the upcoming courses such as Microprocessor Systems and Computer Architecture.

The difficulties we have seen during this project were mostly about being sure on clock signals. Using the same single clock is a non-negligible requirement, therefore we had to be careful of it. However, we have solved it using the always “posedge” blocks properly. We have also checked the outputs at each step before getting forward on the next step. This was crucial because we have implemented an interconnected, multi-component system and every component had to work without any errors. Therefore, we had a dedicated and special interest on unit tests.

REFERENCES