ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 1 REPORT

CRN : 21334 - 21336

LECTURER : Gökhan İnce

Mustafa Kamaşak

GROUP MEMBERS:

150200060 : Ömer Yıldız

150190052 : Utku Kadir Somer

SPRING 2024

Contents

1 INTRODUCTION [10 points]

In this project, we have implemented (or simulated) a primitive computer using Verilog hardware description language. We are able to do logical and arithmetic operations within this system. Additionally, we have the option to perform these operations sequentially and store the outcomes in a memory or register. Hopefully, the system functions with the test inputs provided.

2 MATERIALS AND METHODS [40 points]

2.1 Task Distribution:

All parts of project has been implemented in collaboration and the debugging - reporting parts was also done as group.

3 PART 1: Designing a 16-bit register

For the first part we designed and implemented a 16-bit register that is utilized later on in the implementation of the Register File and the Address Register File (ARF). The generated 16 bit register has 8 functions when enabled; clear, load, decrement, increment, clear and write low, only write low, only write high, sign extention and write low. The elaborated design for our implementation can be seen below:

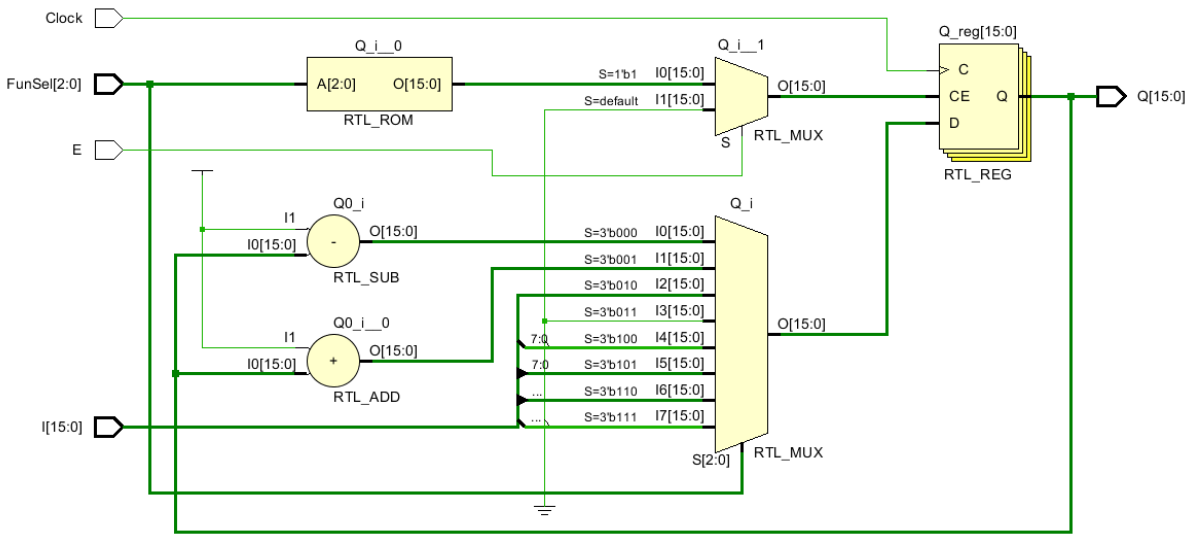

Figure 1: schematic for registers

4 PART 2: Designing a register file

4.1 Part 2-a: Designing a 16-bit IR register

In this part of the project, a 16-bit Instruction Register (IR) is designed and implemented, capable of storing 16-bit binary instructions by using only an 8-bit data input bus. A new feature in the design of the register is the use of a control signal, L'H, dictated to load the 8-bit data in either half or the higher half (bits 15-8) of the register. This allows for flexibility in instruction loading while maintaining a compact data path.

Operationally, the IR has a Write control signal that, when asserted, enables the writing of data into the register. The loading of either the higher or lower byte is contingent on the state of the L'H signal.

It is logic "0" for the L'H; then, its write operation will transfer data in the lower byte. When logic changes to "1," the same write operation is to be carried out at the higher byte of IR.

For the data integrity, the value in the IR will remain as it was in the case that the write signal is not active. Such a controlled two-phase approach to loading data, under the influence of the L'H signal, is optimized for our register design used by varied computational tasks that often require manipulation of either part of the instruction register. Its ease belies the efficacy of this approach, as it allows one 8-bit bus to serve both the data and instruction needs without requiring a wider bus, which would have wasted valuable resources and made the architecture more complex. Our design, therefore, realizes high functionality and efficiency within the constraints of an 8-bit data input bus.

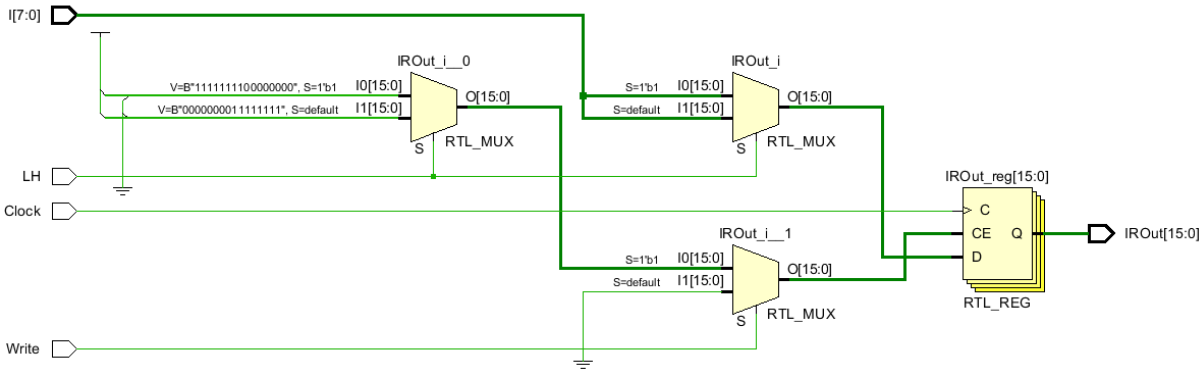

Figure 2: schematic for Instruction register (IR)

4.2 Part 2-b: Designing a register system

For Part 2b, we design a register file system that consists of eight 16-bit registers, of which four are general-purpose registers (R1 to R4) and the other four are scratch registers (S1 to S4). Our schematic shows the connections and controls that guide the operations of the register file.

It uses a set of control signals, composed of OutASel, OutBSel, RegSel, ScrSel, and FunSel, where every signal has an individual contribution in the choosing of registers and in function determination. "OutASel" and "OutBSel" are selector signals, the one that decides which register data is to be routed onto the respective output lines OutA and OutB. This allows the dynamically controlled data flow to be materialized within our system, ensuring that, whenever necessary, the required register data can be tapped into.

RegSel controls the scratch registers selected for operations to be executed and controls the general-purpose registers by it. Such selectors help enable selective, particular registers based on the input of binary into them, therefore allowing for manipulations targeting registers.

The 3-bit FunSel signal is used to specify the function to be applied to the selected registers. These operations are, but not limited to, increment, decrement, load, clear, for the given register. The more specific functions include writing to the low or high byte and sign extension based on the input data.

Our design thus ensures modularity for the possibility of scalability and probable reuse in future parts of the project. We have tested with a high level of rigor the input combination one by one to ensure that through simulation, there is integrity and functionality in the register file system. The project has been designed at a strong level, and it met the project requirements, creating solid grounds for the remaining parts of the project.

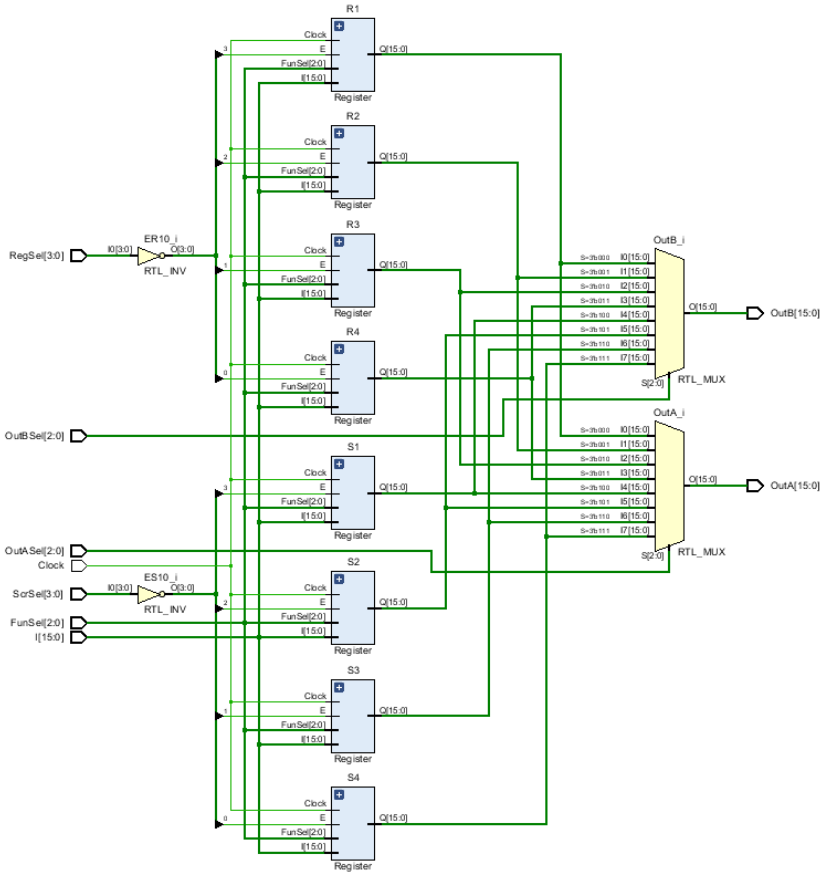

Figure 3: schematic for Register File

4.3 Part 2-c: Designing a address register file (ARF)

we have implemented part 2-c, which is the design of a system that created an Address Register File (ARF). The registers of this system are three: 16-bit registers of addresses - the Program Counter (PC), the Address Register (AR), and the Stack Pointer (SP). The mentioned registers are the basic necessary components for controlling flow operations within a computer.

The design to be implemented is expected to allow manipulation of these registers, with operation as part 2b while implementing the general-purpose and scratch registers, enabled through the use of FunSel signals and RegSel. The RegSel signal, when asserted, determines which of the address registers are the target for these operations, while the FunSel signal is a 3-bit input that designates the operation to be performed on the selected registers.

OutDSel and OutCSel form a selection line controlling outputs from the ARF system, enabling data from the selected registers for both to get routed to the OutD and OutC output lines, respectively, for them to be available in the succeeding stages of our computer organization.

The architecture is designed in such a manner that all the three register files should work consistently under one clock signal, which is very much required for the synchronization of the system. Some of the components of our ARF system were placed in a careful plan and rigorous test to confirm that it is behaving as expected and reflecting our ability to design complex digital systems.

This achievement sets the stage for the integration of the ARF with other constituents of our project: the ALU and Register File system, in order to work as a whole, a single-unit computer. This includes an engineering and functionality modular design reflecting deep knowledge and practical skills acquired from all parts of this project.

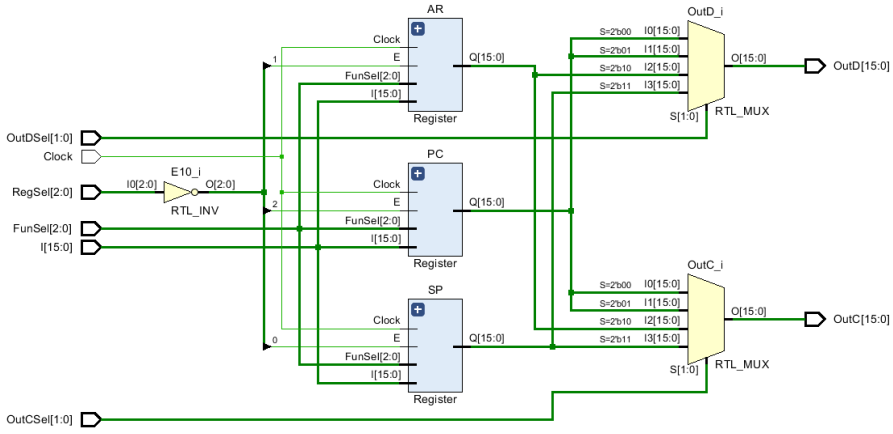

Figure 4: schematic for Address Register File

5 Part 3: ARITHMETIC LOGIC UNIT (ALU)

For this part we designed and implemented the ALU. This unit uses a pair of 16-bit inputs to carry out an array of calculations and logic , returning a 16-bit solution. Plus, it's raises flags through a 4-bit status alert—zero, negative, carry, and overflow—cluing us into the result's character.

The ALU follows the lead of the 4-bit FunSel signal, acting as a mission controller that selects the precise task at hand, from basic math like addition and subtraction to more intricate bitwise maneuvers and shifts.

And our ALU knows when to flag up. Guided by the Write Flag signal, it only raises the signal flags when there's something to report.. A zero flag for a null result, a carry flag when there's an excess, a negative flag for below-zero outcomes, and an overflow flag for when the numbers just can't fit in their digital box.

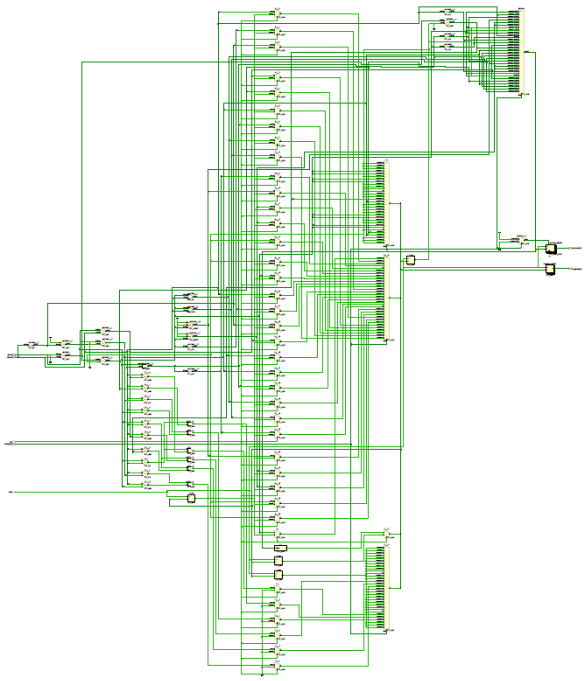

Figure 5: schematic for ALU

6 Part 4: ALU SYSTEM

In our part 4 of the project, we effectively implemented the final overall organization, putting the various components designed from the preceding parts to be operated effectively under one clock. This system makes use of the flexibility in our Arithmetic Logic Unit (ALU) and the organization of information storage linked to our Register File (RF), including the Address Register File (ARF), with its responsibilities to do with control flow operations, and the Instruction Register (IR) for management over programs instructions.

All this works in coordination with the help of multiple multiplexers (MUXes) that choose the paths of data on control signals. MuxASel and MuxBSel are signals that select inputs to the ALU from the immediate, ARF outputs, and memory and lower byte of IR. Similarly, MuxCSel selects a part of the ALU output to be guided towards the stages in the system. ALU input memory data is extended accordingly in a way to be operated correctly.

This clock, common to every one of these operations, is then used to sync the operations. In turn, this assures the concurring working of all the subsystems; therefore, preventing conflicts in data and, in turn, loss of integrity in the operations obtained across the system. The IR holds the current instruction, and its low byte can be directly used either by the ALU as an input or an address for memory, depending on the current

operation being executed.

Having done this part of the task correctly, our system now represents the functional, holistic entity capable of carrying out a vast set of computing operations from simple arithmetic to complex logical operations. This integration highlights the modular and functional design approach that was the trademark of our project, showing comprehensive knowledge and practical skills achieved throughout the flow of this project.

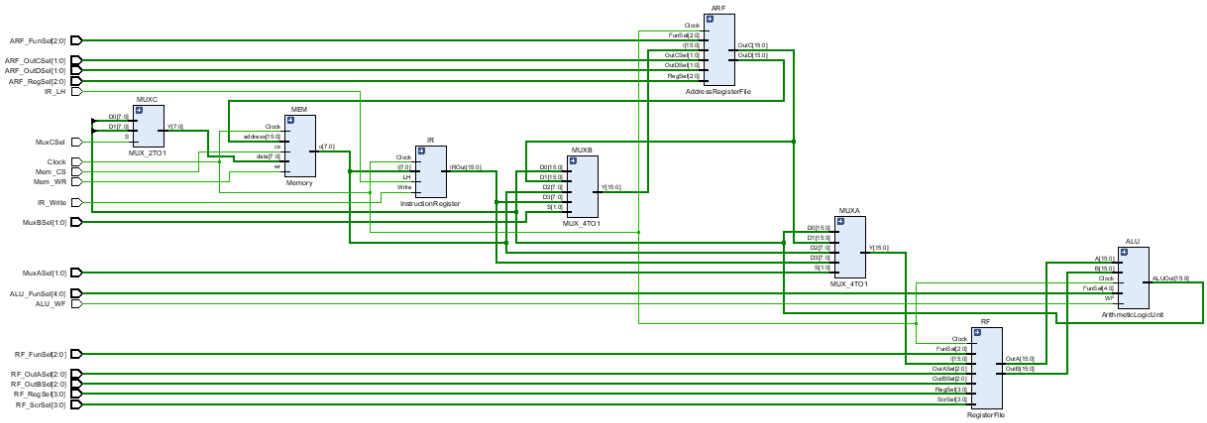

Figure 6: schematic for whole system

7 RESULTS [15 points]

```
Register Simulation Started
[PASS] Test No: 1, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 2, Component: Q, Actual Value: 0x0024, Expected Value: 0x0024
[PASS] Test No: 3, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 4, Component: Q, Actual Value: 0x0026, Expected Value: 0x0026
[PASS] Test No: 5, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 6, Component: Q, Actual Value: 0x0012, Expected Value: 0x0012
[PASS] Test No: 7, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 8, Component: Q, Actual Value: 0x0000, Expected Value: 0x0000
Register Simulation Finished
0 Test Failed
8 Test Passed
```

Figure 7: test result of registers

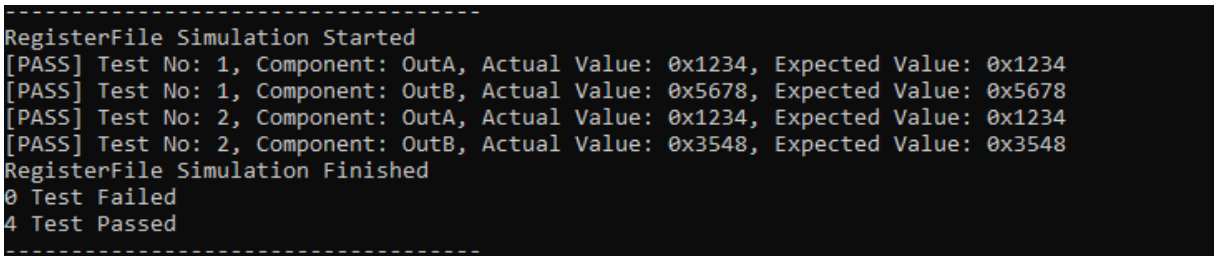

Figure 8: test result of register file

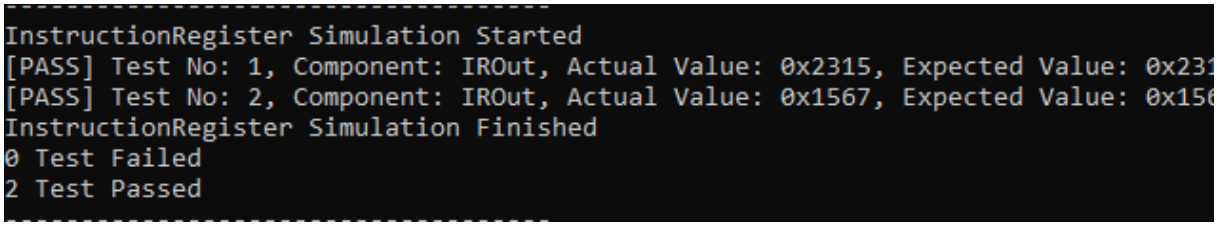

Figure 9: test result of IR

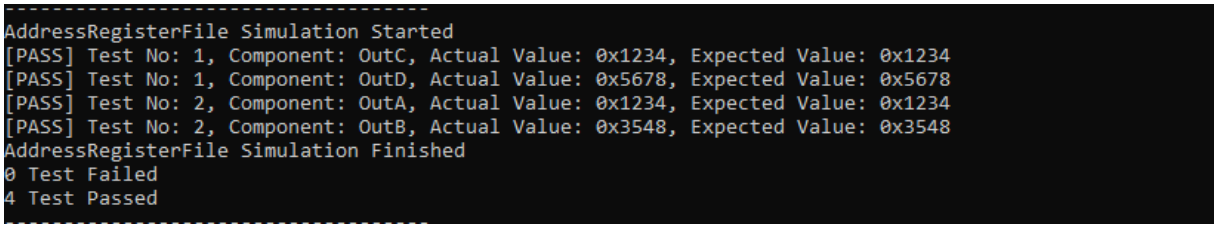

Figure 10: test result of address register file

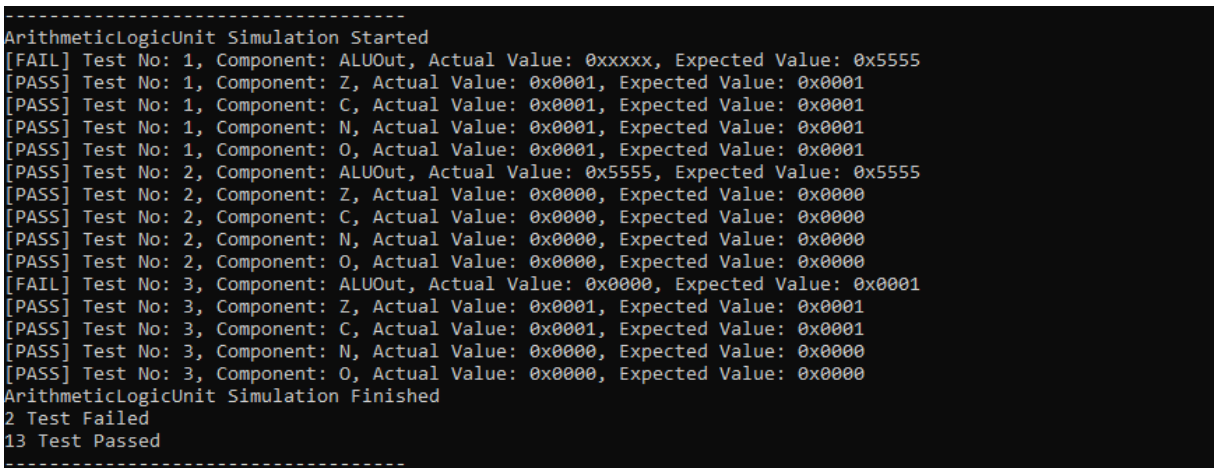

Figure 11: test result of ALU

```
----------------------------------
ArithmeticLogicUnitSystem Simulation Started
[PASS] Test No: 1, Component: OutA, Actual Value: 0x7777, Expected Value: 0x7777
[PASS] Test No: 1, Component: OutB, Actual Value: 0x8887, Expected Value: 0x8887
[FAIL] Test No: 1, Component: ALUOut, Actual Value: 0xxxxx, Expected Value: 0xfffe
[PASS] Test No: 1, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: N, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
[FAIL] Test No: 1, Component: MuxAOut, Actual Value: 0xxxxx, Expected Value: 0xfffe
[FAIL] Test No: 1, Component: MuxBOut, Actual Value: 0xxxxx, Expected Value: 0xfffe
[FAIL] Test No: 1, Component: MuxCOut, Actual Value: 0x00xx, Expected Value: 0x00fe
[PASS] Test No: 1, Component: R2, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: S3, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: OutA, Actual Value: 0x7777, Expected Value: 0x7777
[PASS] Test No: 2, Component: OutB, Actual Value: 0x8887, Expected Value: 0x8887
[PASS] Test No: 2, Component: ALUOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: N, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 2, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: MuxAOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: MuxBOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: MuxCOut, Actual Value: 0x00fe, Expected Value: 0x00fe
[FAIL] Test No: 2, Component: R2, Actual Value: 0xxxxx, Expected Value: 0xfffe
[FAIL] Test No: 2, Component: S3, Actual Value: 0xxxxx, Expected Value: 0xfffe
[FAIL] Test No: 2, Component: PC, Actual Value: 0xxxxx, Expected Value: 0xfffe
[PASS] Test No: 3, Component: OutC, Actual Value: 0x1254, Expected Value: 0x1254
[PASS] Test No: 3, Component: Address, Actual Value: 0x0023, Expected Value: 0x0023
[PASS] Test No: 3, Component: Memout, Actual Value: 0x0015, Expected Value: 0x0015
[PASS] Test No: 3, Component: IROut, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 4, Component: OutC, Actual Value: 0x1254, Expected Value: 0x1254
[PASS] Test No: 4, Component: Address, Actual Value: 0x0023, Expected Value: 0x0023
[PASS] Test No: 4, Component: Memout, Actual Value: 0x0015, Expected Value: 0x0015
[PASS] Test No: 4, Component: IROut, Actual Value: 0x0015, Expected Value: 0x0015
ArithmeticLogicUnitSystem Simulation Finished
7 Test Failed
26 Test Passed
----------------------------------
exit
INFO: [Common 17-206] Exiting xsim at Thu Apr  4 18:28:40 2024...

C:\Users\u_k_s\comporg\project_2\Submission>
```

Figure 12: test result of System

8 DISCUSSION [25 points]

In the completion of the project on computer organization, the objective was to design and implement registers and register files, segmented into four parts. Each part aimed at addressing a specific aspect of computer organization, with an emphasis on modular design for reusability and comprehensive simulation to ensure correctness. Below is a factual summary of the work conducted across the different segments.

- Part 1: 16-bit Register Design

  The project commenced with the design of a 16-bit register capable of executing eight distinct operations dictated by 3-bit control signals and an enable input. The design's versatility was demonstrated through its capacity to retain its value, increment, decrement, load specific data, clear, and perform various bitwise operations. This foundational component was critical, serving as a building block for subsequent

parts of the project.

- Part 2: Register File Design

  Subsequent to the register design, a register file was developed, consisting of a 16-bit IR register and a system comprising general-purpose and scratch registers. The IR register was designed to store 16-bit binary data, accommodating only 8-bit inputs, thereby introducing a mechanism to select between the lower and higher halves of the register for data storage. The complexity increased with the design of a system for selecting and operating on multiple registers, requiring precise control logic to ensure the correct execution of desired functionalities.

- Part 3: Arithmetic Logic Unit (ALU) Design

  The third segment focused on the design of an ALU with two 16-bit inputs and outputs, along with a 4-bit output for status flags (zero, negative, carry, and overflow). This unit was tasked with performing a broad spectrum of arithmetic and logic operations, employing 2's complement logic for arithmetic operations. The challenge was to accurately implement the operations and flag updates based on the outcomes of these operations.

- Part 4: System Integration

  The final part involved integrating the previously designed components into a cohesive system operating under a single clock. This step required meticulous attention to the interactions between components, ensuring coherent data flow and correct operation as per the control signals. The integration tested the design's modularity and the robustness of individual components within a unified system.

Throughout the project, the focus remained on achieving functional correctness through detailed simulation testing for each component and the integrated system. Challenges encountered during the design and implementation phases were addressed through iterative testing and refinement, ensuring compliance with the specified requirements.

This project has contributed to a deeper understanding of computer organization, emphasizing the importance of modular design, simulation testing, and the intricacies involved in system integration.

9 CONCLUSION [10 points]

Throughout the project we did not face any problems till the ALU part,except the sign extention part - since we didn't know that while we are extended sign, should we

always use 0 or should we also use 1's for the negative integers. Afterwards we learned how to do it using the updates that were posted on ninova. We also found it troublesome to deal with the faulty testbench to test our whole ALU system. As for what we learned, we gained knowledge in writing modules using behavioral verilog and understood how an ALU system behaves using 2's complement logic.

REFERENCES