ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 1 REPORT

CRN : 21336

LECTURER : Mustafa Ersel Kamaşak

GROUP MEMBERS:

150210061 : METİN ERTEKİN KÜÇÜK

150200059 : MEHMET ALİ BALIKÇI

SPRING 2024

Contents

1 TASK DISTRIBUTION

Metin Ertekin Küçük: Report Preparing, Part 1, 2, 3, 4
Mehmet Ali Balıkçı: Part 1, 2, 3, 4

2 INTRODUCTION

In this project, we created a simple computer that can read data from memory, perform various operations on that data (such as addition, subtraction, shift, etc.), and then store that data in memory. A register file to choose the appropriate register, an address register file to implement the program counter, and an ALU to perform those operations are all going to be designed as part of the implementation. To create our computer, we implemented each component one at a time.

3 MATERIALS AND METHODS

In this project, we implemented a system consisting of various modules in Verilog HDL. Each module was designed to perform specific functionalities as described in the project requirements. The project involved the following parts:

- 1. Designing a 16-bit register with 8 functionalities controlled by 3-bit control signals (FunSel) and an enable input (E).
- 2. Designing a register file structure containing multiple registers, including a 16-bit IR register and a system with general-purpose and scratch registers.
- 3. Designing an address register file system with three 16-bit address registers: program counter (PC), address register (AR), and stack pointer (SP).
- 4. Implementing an Arithmetic Logic Unit (ALU) capable of performing various arithmetic and logic operations on two 16-bit inputs.
- 5. Integrating all systems together and making well working computer system.

Each part of the project involved designing individual modules, simulating them, and ensuring their correct functionality. The simulation was carried out for each combination of inputs to verify the correctness of the design.

3.1 List of Control Inputs and Corresponding Functions

- Part 1: 16-bit Register

  - FunSel: 3-bit control signal selecting one of the 8 functionalities.

  - E: Enable input controlling whether the register operates based on the selected functionality.

- Part 2a: 16-bit IR Register

  - L'H: Control signal determining whether to load the lower or higher half of the 16-bit data.

  - Write: Signal to write data into the IR register.

- Part 2b: System with General Purpose and Scratch Registers

  - OutASel, OutBSel: Control inputs to select the output registers.

  - RegSel, ScrSel: 4-bit signals selecting the registers for applying the specified function.

  - FunSel: 3-bit control input determining the function to be applied to the selected registers.

- Part 2c: Address Register File (ARF) System

  - RegSel: Control input selecting the address registers (PC, AR, SP) to apply the specified function.

  - FunSel: 3-bit control input determining the function to be applied to the selected address registers.

  - OutCSel, OutDSel: Control inputs to select the output registers for the outputs OutC and OutD.

- Part 3: Arithmetic Logic Unit (ALU)

  - FunSel: 5-bit control input selecting one of the 32 ALU functions.

  - A, B: 16-bit inputs to the ALU.

  - WF: Write Flag signal controlling whether to update the zero, negative, carry, and overflow flags.

  - ALUOut: Output of the ALU containing the result of the selected operation.

  - Z, C, N, O: Output flags indicating zero, carry, negative, and overflow conditions, respectively.

3.2 Explanations for Each Constructed Part

- 1. 16-bit Register: Designed a register with various functionalities controlled by control signals. Implemented logic to perform operations such as increment, decrement, load, clear, and write.
- 2. Register File System: Created a structure containing general-purpose and scratch registers. Developed logic to select registers based on control inputs and perform specified operations on them.
- 3. Address Register File (ARF) System: Designed a system comprising address registers with functionality similar to the register file system but tailored for address registers such as program counter, address register, and stack pointer.
- 4. Arithmetic Logic Unit (ALU): Implemented an ALU capable of performing arithmetic and logic operations on two 16-bit inputs. Designed control logic to select operations and update flags based on the result.

Overall, each part of the project involved careful design and simulation to ensure proper functionality and adherence to the project requirements.

4 RESULTS

As a result, we observe some graphs, as a result of verilog simulation files:

- 16-bit register test result:

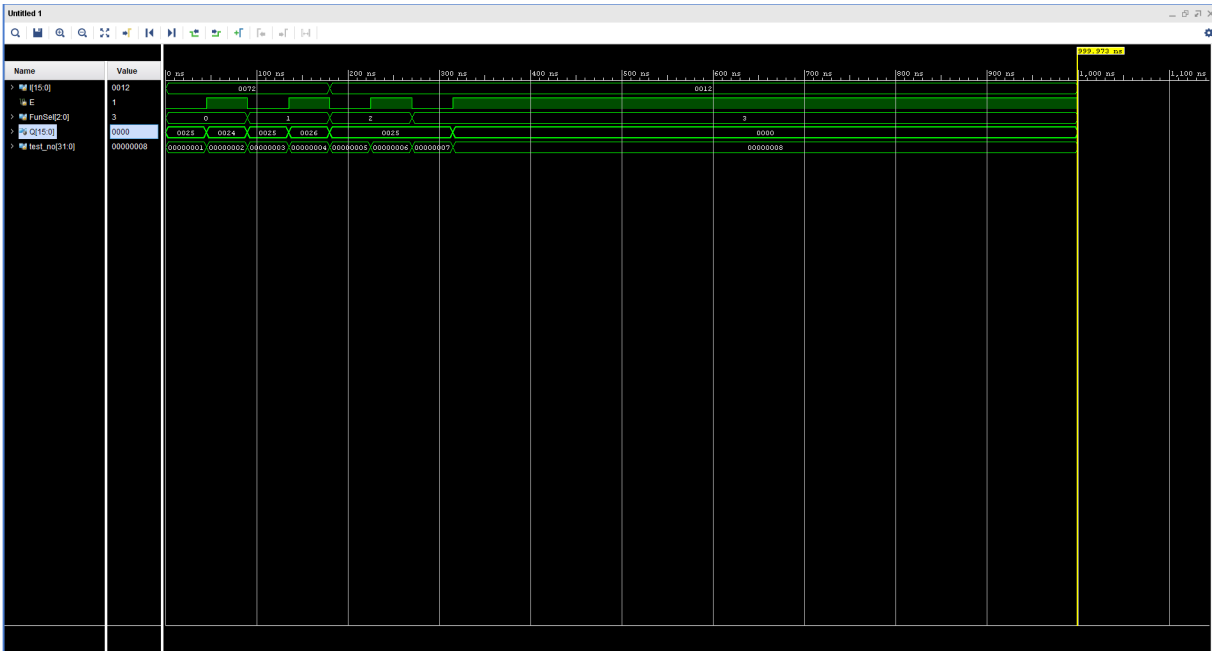

Figure 1: 16 – *bitregister*

- Register system test result:

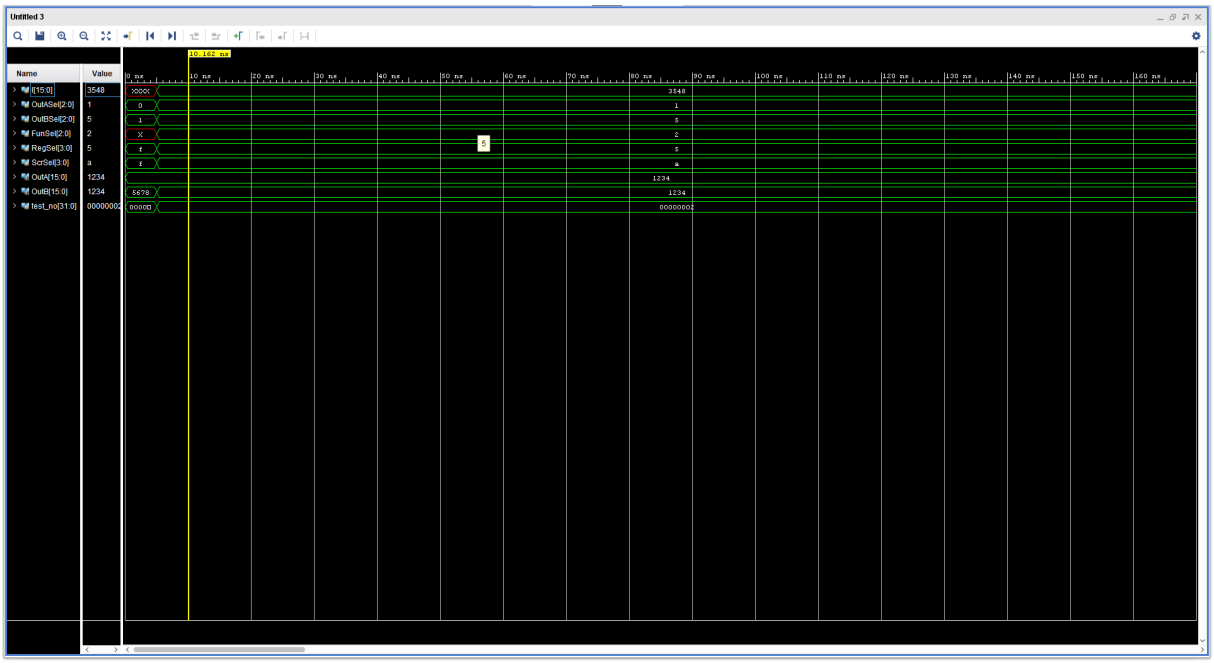

Figure 2: *registersystem*

- Instruction register test result:

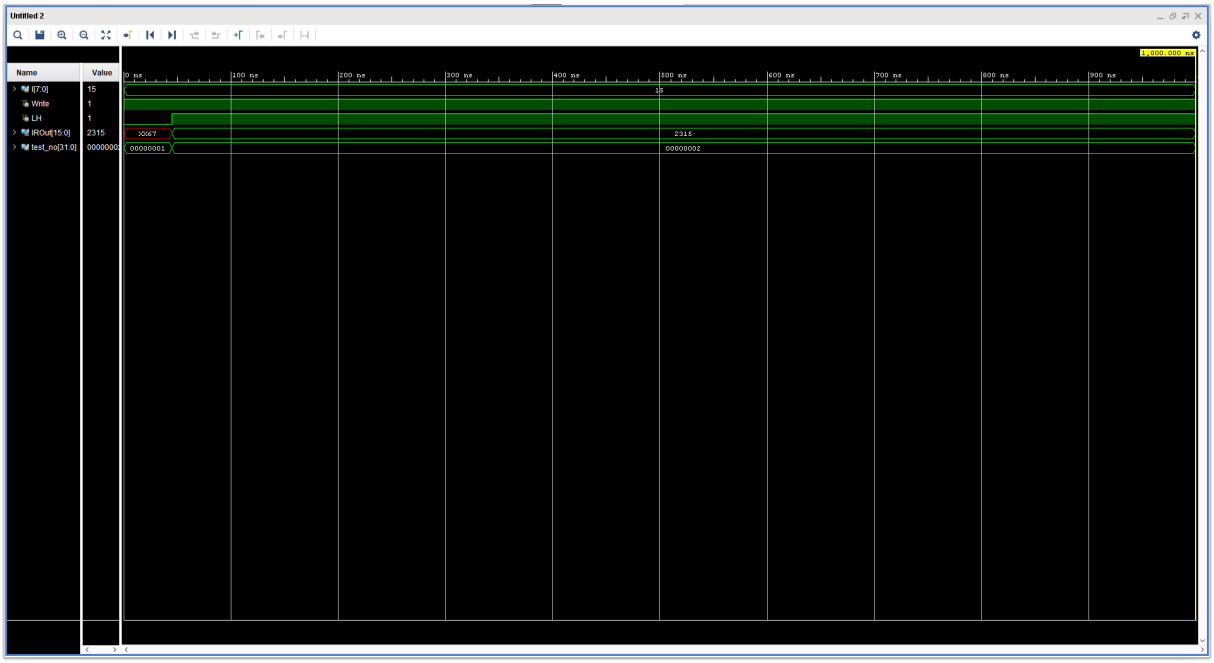

Figure 3: *IRregister*

- Address Register File test result:

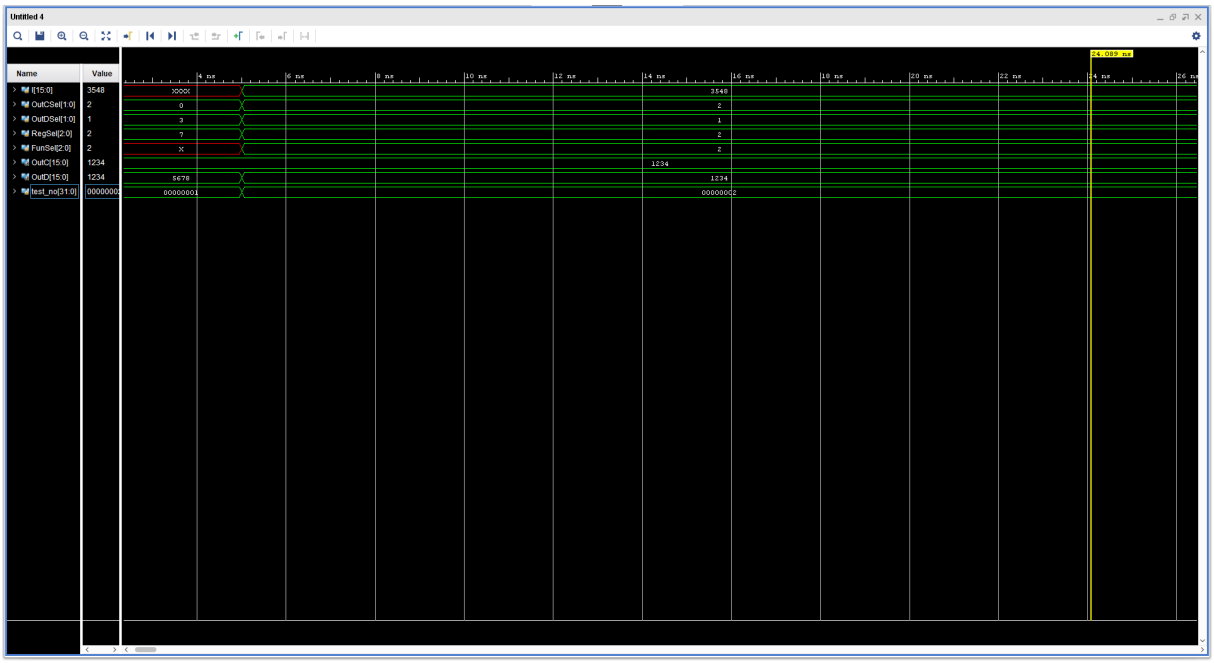

Figure 4: *adressregisterfile*

- Arithmetic Logic Unit test result:

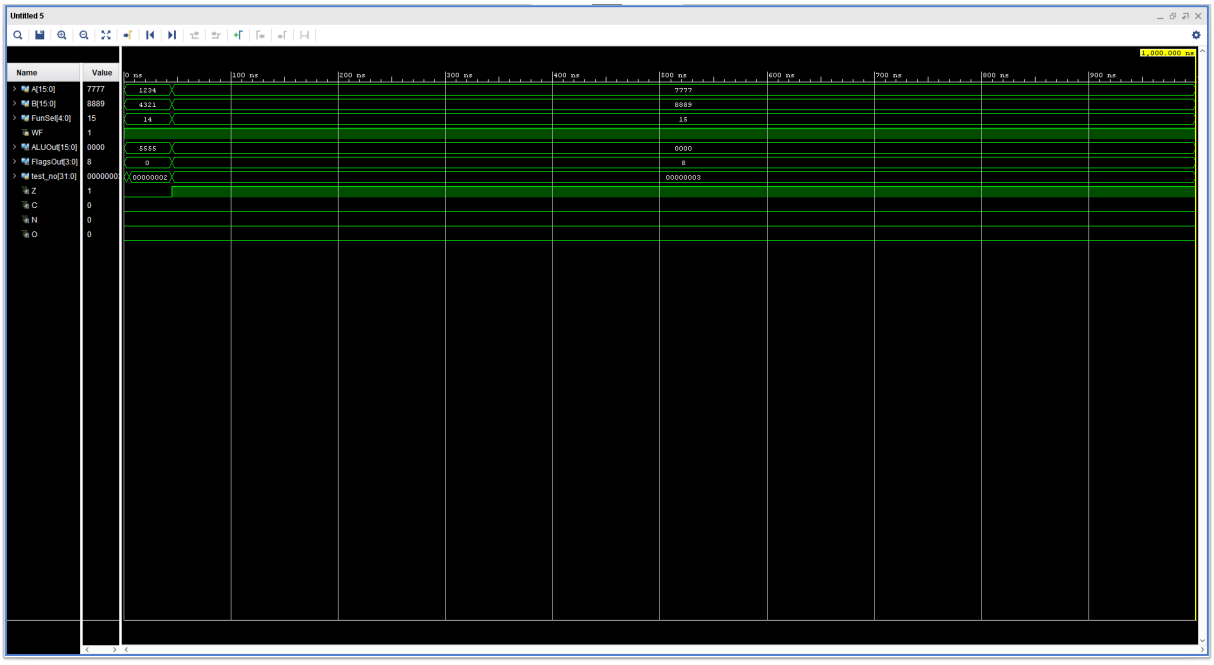

Figure 5: *ArithmeticLogicUnit*

- Arithmetic Logic Unit System test result:

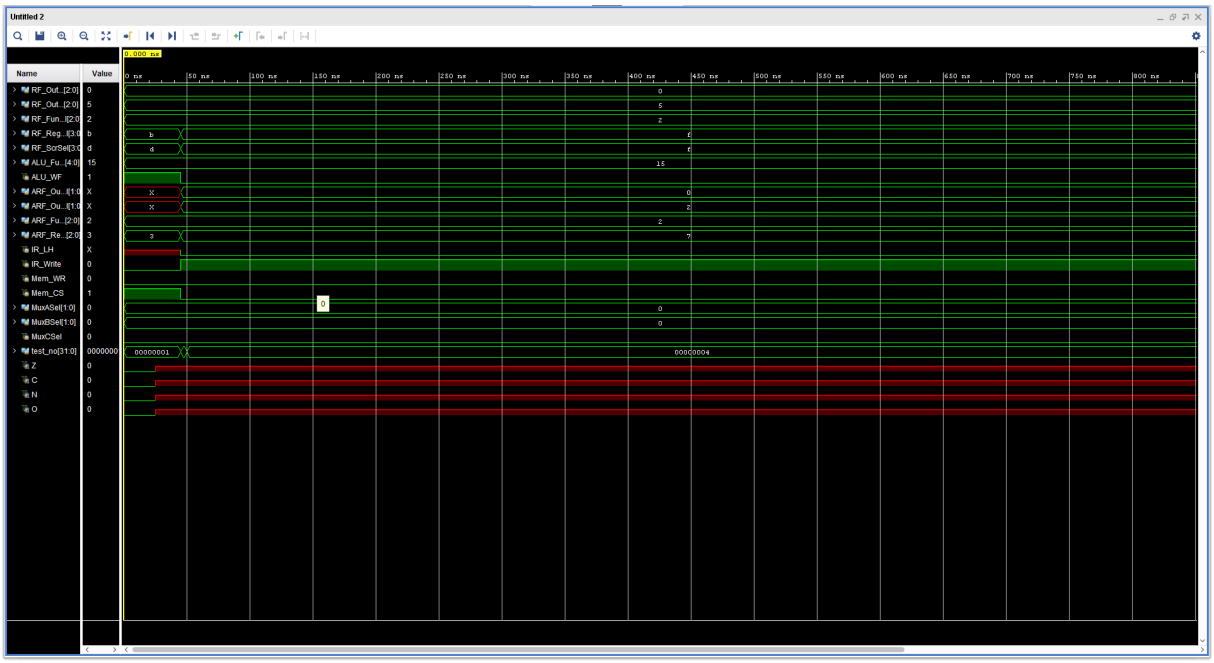

Figure 6: *ALUSystem*

5 DISCUSSION

During the project, we undertook the task of designing and implementing various modules to create a system capable of performing register operations, managing register files, executing arithmetic and logic operations, and handling address registers. Here's a detailed explanation, analysis, and interpretation:

- 16-bit Register Design:

  Explanation: We designed a 16-bit register capable of performing eight different functionalities controlled by 3-bit control signals and an enable input. These functionalities included retaining the current value, incrementing, decrementing, loading data, clearing, and various write operations.

  Analysis: By implementing this register, we provided a versatile component that can be reused in subsequent parts of the project. The flexibility of choosing different operations based on control signals allows for efficient utilization in various contexts.

  Interpretation: This module serves as a fundamental building block for the entire system, providing the capability to manipulate data and perform essential operations. Its versatility ensures adaptability to different requirements in subsequent parts of the project.

- Register File System Design:

Explanation: We created a register file structure containing both general-purpose and scratch registers. The system allowed for selecting specific registers based on control inputs and applying various operations to them.

Analysis: This design offers modularity and scalability by organizing registers into a structured system. By enabling selective operations on registers and supporting both general-purpose and scratch registers, it provides flexibility for diverse applications.

Interpretation: The register file system facilitates efficient data management and manipulation within the system. It enables the handling of multiple registers simultaneously and streamlines access to data for subsequent processing.

- Address Register File (ARF) System Design:

  Explanation: We extended the register file concept to address registers, including the program counter (PC), address register (AR), and stack pointer (SP). Similar to the register file system, it allows for selective operations on address registers based on control inputs.

  Analysis: By incorporating address registers into the system, we addressed the specific requirements of managing program flow and memory addresses. The modular design facilitates efficient handling of address-related operations and simplifies system control.

  Interpretation: The ARF system plays a critical role in program execution and memory management. It ensures proper handling of addresses, supports program flow control, and facilitates interaction with memory elements, contributing to the overall functionality and efficiency of the system.

- Arithmetic Logic Unit (ALU) Design:

  Explanation: We implemented an ALU capable of performing various arithmetic and logic operations on two 16-bit inputs. The ALU function was determined by a 5-bit control input, with additional flags indicating zero, carry, negative, and overflow conditions.

  Analysis: The ALU serves as the computational core of the system, performing essential arithmetic and logic operations required for data processing. Its modular design allows for efficient integration into the overall system architecture, supporting diverse computation needs.

  Interpretation: By incorporating the ALU, we provided the system with the capability to execute a wide range of arithmetic and logic operations. The flags generated by the ALU facilitate decision-making and error detection, enhancing the system's reliability and functionality.

6 CONCLUSION

In conclusion, we connect all the systems together and made well-balanced basic computer that can make basic arithmetic operations with some essential registers.

It was quite difficult to get the Vivado version to be compatible and to understand this IDE, and it took a lot of effort to overcome this. In addition, it took a lot of time to make sense of the complex instructions and we could not conclude all the system integration but in the end, we learnt how a simple computer works and how to simulate it in a simple way.

REFERENCES