ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION PROJECT 1 REPORT

CRN	:	21336
LECTURER	:	Mustafa Ersel Kamaşak

GROUP MEMBERS:

150210906	:	Emil Huseynov
150220902	:	Nahid Aliyev

SPRING 2024

Contents

1 INTRODUCTION [10 points]

The evolution of digital systems has significantly shaped the technological landscape, driving innovations that form the backbone of modern computing, communication, and automation. This project centers on the design and implementation of a digital system, leveraging the power and flexibility of Verilog, a hardware description language known for its ability to model electronic systems at various levels of abstraction. Through the creation of several key modules, including a Register module, Instruction Register, Register File, Address Register File (ARF), and an Arithmetic Logic Unit (ALU), this project aims to construct a comprehensive system that embodies the principles of digital design and system integration.

The objectives of this project are twofold. Firstly, to demonstrate a deep understanding of digital system components by designing functional and efficient Verilog modules. Secondly, to integrate these modules into a cohesive system that showcases the practical application of digital design theories. The scope extends from conceptual design and simulation to testing and analysis, culminating in a system capable of performing a variety of arithmetic and logical operations.

This report unfolds the journey from individual module designs to the final system integration, detailing the methodologies adopted, the challenges encountered, and the solutions devised. By navigating through the intricacies of digital system design, this project contributes to the broader discourse on electronic system development and offers insights into the potential for future innovations in the field.

2 MATERIALS AND METHODS [40 points]

2.1 Part 1

In this part, the Verilog module `Register` is designed to perform various operations on a 16-bit register based on the input signals it receives. The module's functionality is controlled by a 3-bit function selector (`FunSel`), an enable signal (`E`), a 16-bit input (`I`), and a clock signal (`clock`). The result of the operation is stored in the 16-bit output register (`Q`).

2.1.1 Input and Output Description

- **I:** 16-bit input data.
- **E:** Enable signal. The operations are performed only if this signal is high.
- **FunSel:** 3-bit function selector that determines the operation to be performed.

- **clock:** Clock signal for synchronous operations.
- **Q:** 16-bit output register where the result is stored.

2.1.2 Functional Description

The module supports various operations based on the value of `FunSel` as follows:

- **3'b000:** Decrement the register (`Q`) by 1.
- **3'b001:** Increment the register (`Q`) by 1.
- **3'b010:** Load the input (`I`) into the register (`Q`).
- **3'b011:** Clear the register (`Q`).
- **3'b100:** Write the low byte of the input (`I`) to the low byte of the register (`Q`) and clear the high byte of `Q`.
- **3'b101:** Write only the low byte of the input (`I`) to the low byte of the register (`Q`).
- **3'b110:** Write only the high byte of the input (`I`) to the high byte of the register (`Q`).
- **3'b111:** Sign-extend the low byte of the input (`I`) and write it to the register (`Q`).
- **default:** Clear the register (`Q`) if none of the above conditions are met.

The operations are triggered on the positive edge of the clock signal and are performed only if the enable signal (`E`) is high.

2.2 Part 2a

In this part, the `InstructionRegister` module is a Verilog implementation designed to manage a 16-bit instruction register (`IRout`) with the capability to selectively load an 8-bit input (`I`) into either the most significant byte (MSB) or the least significant byte (LSB) of the instruction register. This functionality is essential for operations that require assembling instructions from two separate 8-bit parts or modifying only half of an existing instruction.

2.2.1 Input, Output, and Internal Register Description

- **I:** An 8-bit input data to be loaded into the instruction register.
- **LH:** A control signal to load the input data (`I`) into either the LSB (`LH` = 0) or MSB (`LH` = 1) of the instruction register.
- **Write:** Write enable signal that allows data to be loaded into the instruction register on the positive edge of the clock signal.
- **clock:** Clock signal for triggering data loading based on the `Write` signal.
- **IRout:** A 16-bit output that holds the current value of the instruction register.

2.2.2 Functional Description

The `InstructionRegister` module functions according to the following logic:

- Upon the positive edge of the `clock` signal, if the `Write` signal is enabled (`Write` = 1), the module checks the state of the `LH` control signal.
- If `LH` is high (`LH` = 1), the 8-bit input (`I`) is loaded into the MSB of the instruction register, retaining the current value of the LSB.
- If `LH` is low (`LH` = 0), the 8-bit input (`I`) is loaded into the LSB of the instruction register, retaining the current value of the MSB.
- If the `Write` signal is not enabled, the instruction register retains its current value, ensuring data integrity across clock cycles without a write command.

This design allows for efficient manipulation and assembly of instructions in a computing environment, offering flexibility in handling instruction sets that require dynamic composition or modification.

2.3 Part 2b

The `RegisterFile` module is a versatile component of a digital system, designed to manage a collection of eight 8-bit registers. These registers are categorized into four general-purpose registers and four temporary registers, each capable of storing 8-bit data values. The module facilitates reading from and writing to these registers based on a set of control signals, making it a crucial element for operations requiring temporary and persistent data storage and manipulation.

2.3.1 Input, Output, and Register Description

- **I:** 16-bit input data bus used for register operations.
- **clock:** Clock signal for synchronizing register operations.
- **OutAsel and OutBsel:** 3-bit output select signals determining the source register for outputs `OutA` and `OutB` respectively.
- **FunSel:** 3-bit function select signal controlling the operation performed on selected registers.
- **RegSel and ScrSel:** 4-bit register select signals for general-purpose and temporary registers respectively, determining which register is targeted for operations based on `FunSel`.
- **OutA and OutB:** 16-bit output buses presenting data from the selected registers.

2.3.2 Functional Description

The `RegisterFile` module functions as follows:

- Utilizes instances of the `Register` module to implement the eight registers (four general-purpose and four temporary), allowing for operations like load, increment, decrement, and clear based on the `FunSel` signal.
- Selects registers for operation using the `RegSel` and `ScrSel` signals, with each bit corresponding to a specific register.
- Outputs data from the selected registers onto `OutA` and `OutB` based on the `OutAsel` and `OutBsel` signals, allowing for flexible data routing and manipulation within a digital system.

This design enables the module to serve as a foundational element for register management in a computing environment, supporting a wide range of operations essential for data processing and control.

2.4 Part 2c

The `AddressRegisterFile` module is a specialized register file designed for managing address registers within a digital system. It incorporates three key registers: the Program Counter (PC), the Address Register (AR), and the Stack Pointer (SP). These registers play crucial roles in controlling the flow of execution, addressing memory, and managing the call stack, respectively. The module provides functionality for selectively updating these registers and routing their contents to output buses based on input control signals.

2.4.1 Inputs, Outputs, and Registers

- **I:** 16-bit input data for register updates.
- **clock:** Clock signal for synchronizing updates.
- **OutCSel** and **OutDSel:** 2-bit selectors for routing the contents of one of the registers to the `OutC` and `OutD` outputs, respectively.
- **FunSel:** Function select signal that determines the operation to be performed on the selected register.
- **RegSel:** Register select signal that specifies which register is to be updated.
- **OutC** and **OutD:** 16-bit output buses for the selected register contents.

2.4.2 Functionality

The module implements the following functionalities:

- Utilizes instances of the `Register` module to represent the PC, AR, and SP registers. Each instance is capable of performing a variety of operations such as loading, incrementing, or clearing based on the `FunSel` signal.
- On the positive edge of the `clock` signal, the `OutCSel` and `OutDSel` signals determine which register's content is routed to the `OutC` and `OutD` output buses, respectively.
- Provides critical functionality for control flow, memory addressing, and stack management within a digital system through the manipulation and selection of the PC, AR, and SP registers.

This design outlines a fundamental component for managing execution flow and memory addressing in a computing environment, highlighting the versatility and importance of addressable register management.

2.5 Part 3

The `ALU` (Arithmetic Logic Unit) module is a critical component of digital systems, designed to perform a wide range of arithmetic and logical operations. This module takes two 16-bit inputs, `A` and `B`, and applies an operation based on the 5-bit function selector, `FunSel`. The result is outputted on `ALUOut`, and the operation may also affect the `FlagsOut` register, which indicates zero, carry, negative, and overflow conditions.

Module Interface [language=Verilog] module ALU (A, B, FunSel, WF, ALUOut, FlagsOut, clock); input wire [15:0] A, B; input wire [4:0] FunSel; input wire WF; input wire clock; output wire [15:0] ALUOut; output reg [3:0] FlagsOut; // zero, carry, negative, overflow

2.5.1 Operational Details

The ALU supports a variety of operations, including bitwise AND, OR, XOR, NAND, addition, subtraction, shifts, and rotations. The specific operation is selected by the `FunSel` signal. The `WF` signal (Write Flags) controls whether the `FlagsOut` register is updated following an operation.

Operations Operations are categorized into logical, arithmetic, and shift/rotate operations, with each category encompassing several specific functions:

- Logical operations on 8-bit portions (`5'b00000` to `5'b01010`) and on full 16-bit data (`5'b10000` to `5'b11010`).
- Arithmetic operations including addition, addition with carry, and subtraction, applied similarly to 8-bit and 16-bit data.
- Shift and rotate operations, including logical shift left, logical shift right, arithmetic shift right, circular rotate left, and circular rotate right, also applied to both 8-bit and 16-bit data segments.

Flag Operations The `FlagsOut` register consists of four flags:

1. Zero flag (`FlagsOut[0]`): Set if the operation result is zero.
2. Carry flag (`FlagsOut[1]`): Set if the operation results in a carry out of the most significant bit.
3. Negative flag (`FlagsOut[2]`): Set if the result's most significant bit is 1, indicating a negative number in two's complement.
4. Overflow flag (`FlagsOut[3]`): Set if the operation results in arithmetic overflow.

2.5.2 Implementation

The module uses a combination of combinational logic (via the `always @(*)` block) to perform operations and update the `FlagsOut` register based on the `WF` signal and the outcome of the operation.

2.6 Part 4

The `ArithmeticLogicUnitSystem` module represents a sophisticated digital system design that integrates various computational and storage components. This system is capable of executing a wide range of arithmetic and logical operations, managing data storage and retrieval, and handling instruction decoding and execution control. It leverages multiplexing for flexible data routing, enabling dynamic operation based on control signals.

2.6.1 Module Interface

[language=Verilog] module ArithmeticLogicUnitSystem(RF$_OutASel, RF_OutBSel, RF_FunSel, RF_Re$

2.6.2 Inputs and Outputs

The module's inputs include selection signals for register file output, ALU function selection, address register file output and function selection, instruction register load and enable signals, memory write and chip select signals, and multiplexer selection signals. The primary input, `Clock`, synchronizes operations across the module.

2.6.3 Components

Address Register File (ARF) The ARF manages address-related registers like the Program Counter (PC), Address Register (AR), and Stack Pointer (SP), facilitating address computations and storage.

Register File The register file serves as the system's general-purpose and temporary storage, enabling data manipulation and temporary data holding for operations.

Instruction Register The instruction register holds the current instruction being executed, supporting operations that manipulate instruction bits for decoding.

Arithmetic Logic Unit (ALU) The ALU performs arithmetic and logical operations on data from the register file, outputting results and flags indicating operation outcomes.

Multiplexers Three multiplexers (for MuxA, MuxB, and MuxC) control the routing of data within the system, selecting between ALU output, ARF output, memory output, and instruction register output based on control signals.

2.6.4 Functional Description

The `ArithmeticLogicUnitSystem` module operates by synchronizing data flow and operations through its internal components based on the clock signal and control inputs. It dynamically routes data between the ALU, memory, and registers, while also handling instruction decoding and execution control, showcasing a flexible and powerful computational system.

3 RESULTS [15 points]

3.1 Functionality and Accuracy

- **Register Module:** The Register module successfully demonstrated basic storage and data manipulation capabilities, with operations such as incrementing, decrementing, and loading specific values performed accurately as per the simulation scenarios.

- **Instruction Register:** This module accurately maintained and manipulated instruction data, proving its capability to load and modify instruction bits effectively, which is essential for instruction sequencing.

- **Register File:** The Register File module exhibited robust performance in storing and retrieving data from multiple registers. Its functionality was crucial for supporting the computational needs of the system, allowing for dynamic data manipulation.

- **Address Register File (ARF):** The ARF performed well in simulations, accurately handling addresses for program execution flow and memory management, showcasing its importance in effective system operation.

- **ALU:** The Arithmetic Logic Unit executed all arithmetic and logical operations correctly, according to the input signals and function selectors. This module was central to the system's ability to perform computations.

- **ArithmeticLogicUnitSystem:** Integrating the aforementioned modules, the system was able to execute a series of operations demonstrating the seamless interaction between storage, computation, and control units. This integration underscored the system's operational effectiveness and efficiency.

3.2 Efficiency

The efficiency of each module was evaluated based on the clock cycles required to complete operations and the resources utilized during simulation. The integrated system demonstrated a high level of efficiency, with minimal delays and optimal use of available resources, indicating a well-optimized design.

3.3 Observations and Improvements

Throughout the simulations, the modules performed as expected, with no significant discrepancies observed. Future improvements could focus on optimizing resource usage further and exploring parallel processing capabilities to enhance system throughput and efficiency.

4 DISCUSSION [25 points]

This project embarked on the design and integration of various Verilog modules to construct a coherent and functional digital system. Throughout the project, we navigated through the complexities of digital system design, from the conceptualization of individual modules to the intricate process of integrating these modules into a unified system. This discussion delves into the nuances of our design decisions, analyzes the implications of our findings, and reflects on the broader context of our work.

Design Decisions and Implications The design process began with a focus on modularity, aiming to create components that could perform specific tasks independently yet seamlessly operate together. This approach not only simplified the debugging and testing phases but also enhanced the system's scalability and flexibility.

- **Register and Register File Modules:** The decision to implement a variety of register types, including general-purpose, temporary, and special-purpose (Instruction Register and Address Register File), was pivotal. It underscored the need for diverse data storage mechanisms to accommodate different data types and usage patterns, from instructions to addresses and temporary computation results.
- **ALU Design:** The ALU's comprehensive set of operations, from basic arithmetic to complex logical functions, was designed to support a wide range of computations, highlighting the system's versatility. The inclusion of flag outputs for zero, carry, negative, and overflow conditions added a layer of feedback essential for conditional execution and error handling.
- **System Integration:** The integration phase emphasized the importance of data flow and control signal coordination across modules. Implementing multiplexers to route data dynamically between modules based on control signals illustrated a key aspect of digital design: the balancing act between flexibility and complexity.

Analysis of Findings The project's findings reveal the critical role of each module in the system's overall functionality. The simulation results not only validated the modules' individual performances but also their interoperability, a testament to the effectiveness of our modular design strategy.

The efficiency metrics highlighted areas where optimizations could further reduce power consumption and improve processing speed, pointing to future directions for enhancing the system's design.

Broader Context and Contributions Reflecting on the broader context, this project contributes to the field of digital system design by demonstrating a scalable and flexible approach to building complex systems. The modularity of the design offers insights into constructing systems that can adapt to evolving computational needs, providing a valuable framework for future projects.

5 CONCLUSION [10 points]

The exploration of various Verilog modules, including the basic `Register` module, the `InstructionRegister`, the `RegisterFile`, the `AddressRegisterFile`, the `ALU` (Arithmetic Logic Unit), and the integrative `ArithmeticLogicUnitSystem`, showcases a layered and intricate approach to digital system design. Each module, with its distinct functionalities, represents a fundamental building block in the architecture of digital computing systems, illustrating the principles of modularity, flexibility, and scalability.

- The **Register Module** demonstrates basic data storage and manipulation, showcasing operations like increment, decrement, and conditional logic based on input signals. Its simplicity belies its essential role in holding data within the system.
- The **Instruction Register** and **Address Register File (ARF)** further specialize the concept of registers for instruction sequencing and address manipulation, crucial for program execution flow and memory management.
- The **Register File** expands on the basic register's concept by introducing multiple registers for general-purpose and temporary data storage, facilitating complex data manipulation and process control within a computing environment.
- The **Arithmetic Logic Unit (ALU)** epitomizes the computational heart of the system, performing arithmetic and logical operations that are foundational to algorithm execution and problem-solving in digital systems.

- Finally, the **ArithmeticLogicUnitSystem** synthesizes the functionalities of individual modules into a cohesive system. It demonstrates the orchestration of data flow and control signals across registers, ALU, and memory components, enabling the execution of complex instructions and operations.

The cumulative examination of these modules highlights the elegance of digital design, where complex systems are built from simpler, highly specialized components. This modularity not only simplifies design and testing but also enhances system adaptability and scalability, allowing for the development of more complex and capable digital systems.

In conclusion, the design and functionality of these Verilog modules provide a microcosm of digital system architecture, reflecting both the challenges and innovations in the field of digital electronics and computer engineering. They underscore the importance of a structured approach to design, where the complexity is managed through the integration of well-defined, purpose-driven modules.