Lab 8

Madiba Hudson-Quansah

Question 1

Component-Level Understanding
1. How does the register file maintain data integrity during simultaneous read and write operations?
2. Explain the purpose of the ALU control signals and how they determine operation type.
3. What are the key differences between synchronous and asynchronous design in digital systems?

Solution:

1. It prevent simultaneous writes to the same register by using a write enable signal. The register file uses a clock signal to synchronize read and write operations, ensuring that data is not corrupted during simultaneous access.

2. The ALU determines which ALU operations are performed based on the control signals. These signals are generated by the control unit and specify the operation to be performed, such as addition, subtraction, or logical operations.

3. Synchronous designs i.e. single clocked systems use a global clock signal to synchronize all operations, while asynchronous designs do not rely on a clock signal and instead use handshaking signals to control data flow. Synchronous designs are generally easier to design and debug, while asynchronous designs can be more efficient in terms of power and speed.

Question 2

Architectural Design
1. How does the control unit manage the state transitions during instruction execution?
2. Discuss the trade-offs between complexity and performance in CPU design.
3. Does the current control unit handle ORI (OR-Immediate)? Why or why not?

Solution:

1. The control unit manages state transitions by using a finite state machine (FSM) that changes states based on the current instruction and control signals. It generates the necessary control signals to direct the data flow and operations of the ALU, register file, and other components.

2. Increased performance in CPU design from the single cycle variant comes with increased complexity in the control unit and data path. More complex designs can lead to longer design times, more difficult debugging, and increased power consumption. However, they can also provide better performance and efficiency.

3. It does not handle ORI because it has no when clause to any other ALU operation than ADDI.

Question 3

Testing and Verification
1. How do testbenches help validate the functionality of individual components and the entire system?
2. What challenges might you encounter when testing more complex instruction sets?

Solution:

1. Testbenches provide a controlled environment to simulate the behavior of individual components and the entire system. They allow for the generation of various input scenarios and the observation of output responses, helping to identify functional errors and verify that the design meets specifications.

2. Generating the test cases that properly cover all possible scenarios was challenging

Question 4

Technical Deep Dive
1. Analyze the zero flag implementation in the ALU. How does it support branch instructions?
2. Explain the sign-extension mechanism for immediate values in the datapath.
3. Describe how the ALU control unit decodes different instruction types.

Solution:

1. The zero flag is set when the ALU output is zero, indicating that the result of an operation was zero. This flag is used in branch instructions to determine whether to take a branch based on the result of a previous operation.

2. The sign-extension mechanism takes a smaller immediate value and extends its sign bit to match the size of the target register. This ensures that negative values are correctly represented in the larger register.

3. The ALU control unit decodes different instruction types by interpreting the opcode and function code fields in the instruction. It generates specific control signals based on these fields to direct the ALU to perform the appropriate operation.

Question 5

Performance and Optimization
1. What performance limitations exist in this CPU design?
2. How could you modify the architecture to improve instruction throughput?
3. Discuss potential bottlenecks in the current design.

Solution:

1. The performance limitations include the single-cycle execution of instructions, which can lead to longer cycle times and reduced throughput, the lack of overlapping instruction execution also does not allow for instruction level parallelism

2. To improve instruction throughput, a pipelined design coutd be implemented, allowing multiple instructions to be in different stages of execution simultaneously.

3. Potential bottlenecks include the ALU, which may become a limiting factor if multiple instructions require ALU operations simultaneously, and the register file, which may not be able to handle multiple read/write requests efficiently.

Question 6

Practical Implications
1. How do concepts like pipelining, cache, and branch prediction enhance CPU performance?
2. What real-world applications might benefit from understanding such low-level digital design?

Solution:

1. Pipelining allows for greater instruction level parallelism increasing the throughput of the CPU, cache memory reduces the time it takes to access frequently used data, and branch prediction helps to minimize the performance impact of branch instructions by predicting the outcome of branches and preloading instructions.