Josh Brown, Abby Prince
3/22/2024
CS-274

UrCPU Written Report

Introduction:

In this section, we provide an overview of the problem statement and outline our strategy for designing the UrCPU. We discuss the importance of memory reliability and the core functionalities required of the microprocessor. The UrCPU project sought to address the need for a reliable microprocessor capable of executing programs while ensuring memory integrity. Our strategy involved designing a von Neumann style microprocessor with a focus on memory reliability through the implementation of parity-check error correction code. The core functionalities of the UrCPU include storing short programs in memory, executing programs by fetching and transforming data, and correcting memory errors.

Our first goal was to work through the majority of the ALU submodules by compiling all our individual work into one. Once we had a good amount of material here, we were able to move onto the construction of the control unit and the memory bank. With only two people in our group, we experienced an overload of work that we had to do all at once, so although communication was easy, fulfilling every goal was a challenge.

Hardware design decisions:

During the design phase, we made several crucial hardware design decisions to ensure the functionality and efficiency of the UrCPU. We discuss the overall design of the UrCPU, including the configuration of registers, the structure of the ALU, the implementation of the memory bank with parity based ECC, and the design of the control unit. The UrCPU follows a von Neumann architecture, where instructions and data share the same memory. This architecture simplifies the design and facilitates program execution by allowing instructions to be treated as data. Additionally, the use of single memory system reduces complexity and resource requirements.

The UrCPU features a set of general-purpose registers and program segment/pointer registers. These registers play a crucial role in storing data, memory addresses, and program pointers. By providing separate program segment registers for instruction, static data, and dynamically allocatable memory, the UrCPU effectively organizes memory resources, enabling efficient program execution.

The ALU is responsible for executing numerical operations and logical computations. To support a wide range of operations, the ALU is designed to operate in either full-word mode or half-word mode, depending on the requirements of the instruction. The ALU's capability to handle arithmetic, logic, bit shift, and comparison operations ensures the versatility of the UrCPU in executing diverse programs. The ISA allows the ALU to work in both full-word and half-word modes. In both modes, the ALU’s functionality encompasses a wide spectrum of operations, including arithmetic computations like addition and subtraction as well as logical operations such as AND, OR, XOR, and NOT. By tailoring the algorithm to different modes of the ALU, the program is ensured optimal performance and efficient operations within the constraints of the selected word size.

The control unit of the UrCPU plays a pivotal role in coordinating the execution of instructions and managing various hardware components. It can be implemented either as a statically programmed finite automaton or a microprogrammed control unit. The control unit ensures that instructions are executed in the correct sequence and that relevant flags are set at each stage of instruction execution.

The ISA of the UrCPU is designed to accommodate a wide range of instructions, enabling efficient program execution. Each instruction is encoded as a 20-bit word, with specific OP codes representing different operations. The ISA supports essential operations such as program flow control, logic operations, bit shifts, arithmetic operations, and comparisons, providing the necessary functionalities for executing programs effectively.

These hardware design decisions collectively contribute to the functionality, efficiency, and reliability of the UrCPU, enabling it to execute programs reliably while ensuring memory integrity. By carefully considering the design of each hardware component, the UrCPU achieves its objectives of providing a versatile and robust microprocessor for various computing tasks.

Instruction set architecture design:

The ISA of the UrCPU was designed to facilitate program execution and optimized resource utilization. We encoded OP codes corresponding to the machine language of the UrCPU, with each instruction stored as a 20-bit word. This section provides examples of machine language programs and discusses the allocation of bits per OP code. The encoding scheme ensures compact representation of instructions, with each instruction occupying a 20-bit word. This allocation of bits per OP code allows for a balance between the number of available instructions and the precision required for encoding various operations.

In designing the ISA, careful consideration was given to incorporating a range of operations to support the computational needs of a wide array of programs. Key operations include arithmetic computations (addition, subtraction, etc.) and logical operations (AND, OR, XOR, etc.). Additionally, the ISA includes instructions for memory access, control flow manipulation, and bit-level operations like shifting and rotating.

In the example program (file name: exprogram.v), a simple state machine is implemented to mimic the behavior of the UrCPU executing the respective programs. The programs load operands from memory, perform operations, and store the results back into memory. The modules feature a clock input (‘clk’), a reset input (‘rst’), and a memory array to simulate the memory of the UrCPU.

Implementation:

During the implementation phase of the UrCPU design, the translation of design decisions into functional Verilog code enabled us to realize the architecture’s core functionalities. One notable advantage of our design approach was the modular organization, which facilated the development of individual components such as the ALU, memory bank, and control unit. This modularity not only enhanced code readability and maintainability but also allowed for easier debugging and testing of each module in isolation.

Despite the advantages offered by our design decisions, we encountered several challenges during implementation that required careful consideration and resolution. One significant challenge involved ensuring the correct functioning of the ALU, especially in handling complex arithmetic and logical operations efficiently. We addressed this challenge through testing and validation of the ALU’s operation under various input conditions to identify any potential errors or inconsistencies.

To address the challenges encountered during implementation and further enhace the design, several potential solutions could be explored. For instance, leveraging simulation tools and hardware emulation platforms can facilitate comprehensive testing and validation of the UrCPU’s functionality, enabling early detection of issues.

Discussion and conclusions:

The UrCPU project represents a significant achievement in microprocessor design and implementation. We have demonstrated the feasibility of designing a reliable microprocessor capable of executing programs while ensuring memory integrity. We have demonstrated the capability to design and implement a microprocessor capable of executing programs while ensuring memory integrity. The modularity and versatility in the UrCPU's architecture lay a solid foundation for future advancements and innovations through this project.

Moving forward, there are numerous opportunities to extend and enhance the functionality of the UrCPU architecture. One potential advancement is the exploration of additional features and instructions to augment its capabilities. Optimizing the performance of the UrCPU through techniques such as pipeline optimization and cache management strategies could further enhance its efficiency and competitiveness in modern computing environments. Exploring opportunities for energy efficiency and power optimization could contribute to the sustainability of the UrCPU architecture.

In terms of scalability, UrCPU's modular design and flexible architecture could enable seamless integration of new components and extensions, allowing for iterative refinement over time. The open ended nature of the UrCPU's design encourages more work to be done when necessary.