2023 Annual Conference & Exposition
Baltimore Convention Center, MD | June 25 - 28, 2023
The Harbor of Engineering Education for 130 Years
ASEE
Paper ID #36738

Capstone Project: CPU Design with Multiplexer

Prof. Yumin Zhang, Southeast Missouri State University

Yumin Zhang is a professor in the Department of Engineering and Technology, Southeast Missouri State University. His research interests include semiconductor devices, electronic circuits, neural networks, and engineering education.

Capstone Project: CPU Design with Multiplexer

Anthony F. Di Mauro, Michael C. Hawkins, Bradley K. Lindsey, Yumin Zhang

Department of Engineering and Technology
Southeast Missouri State University
Cape Girardeau, MO 63701

Abstract

For a capstone design project, we designed and implemented an 8-bit CPU on an Altera DE2 FPGA board. The CPU uses 4-bit opcodes and can execute 16 instructions, including basic arithmetic and logic operations. We simplified the control unit by implementing it with a multiplexer. We fully tested the CPU by inputting instructions and data through the DE2 board switches, and displaying the results on the seven-segment displays and LEDs. This project provides a valuable opportunity for students majoring in electrical engineering and computer science to gain insight into the relationship between machine codes and their corresponding operations.

Introduction

The CPU is the core component responsible for information processing, making it a crucial topic for students majoring in electrical engineering and computer science to comprehend. Unfortunately, the structure of a CPU is often highly complex, making it difficult for those outside of computer engineering to grasp its intricacies. Although some efforts have been made to design simplified CPUs [1, 2], they can still be quite challenging for many students to fully understand.

A CPU consists of three fundamental components: the arithmetic logic unit (ALU), control unit (CU), and registers. Of these components, the CU is typically the most complex to design. Traditionally, control units are implemented using finite state machines, while pipelined structures are used in advanced architectures [3]. However, given the limited time available for a capstone design project, these approaches can prove to be overly challenging. Instead, we opted for a simpler approach: instruction sets are decoded using multiplexers. This method provides a clear interface between machine codes and their corresponding operations, making it easily understandable even for students with limited digital electronics backgrounds.

Instruction Set and Display

Designing a CPU typically begins with defining the instruction set, which then informs the overall hardware architecture. For our capstone design project, we limited the opcodes to 4 bits, which allowed us to design 16 different operations as listed in Table 1. Once the instruction set has been finalized, the components within the ALU can be designed to match. While the ALU is considered the heart of a processor, its design is relatively straightforward and the necessary logic circuits can be found in many reference materials [4, 5]. Similarly, register design is also quite simple, as various types of registers are well-covered in many digital logic circuit textbooks [6, 7].

Table 1. Opcodes in instruction set.

Operation	Opcode			
	Bit 3	Bit 2	Bit 1	Bit 0
Addition	0	0	0	0
Subtraction	0	0	0	1
Multiplication	0	0	1	0
Integer Division	0	0	1	1
Shift Left	0	1	0	0
Shift Right	0	1	0	1
Rotate Left	0	1	1	0
Rotate Right	0	1	1	1
Logic AND	1	0	0	0
Logic OR	1	0	0	1
Logic XOR	1	0	1	0
Logic NOR	1	0	1	1
Logic NAND	1	1	0	0
Logic XNOR	1	1	0	1
Comparison of A > B	1	1	1	0
Comparison of A = B	1	1	1	1

These instructions can be grouped into two primary categories: arithmetic operations and logic operations. We have demonstrated the functionality of these operations in a video featuring the DE2 board [8], and additional details regarding the I/O interface are shown in Fig.1. The yellow labels indicate the switches used for inputs, the displays of the two operands and the output values.

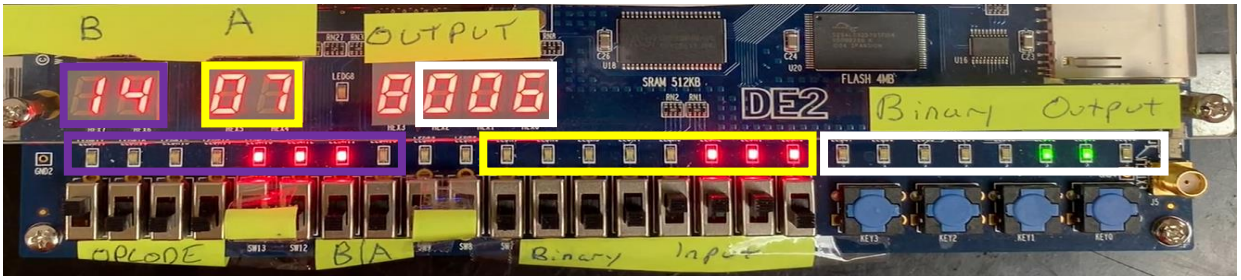


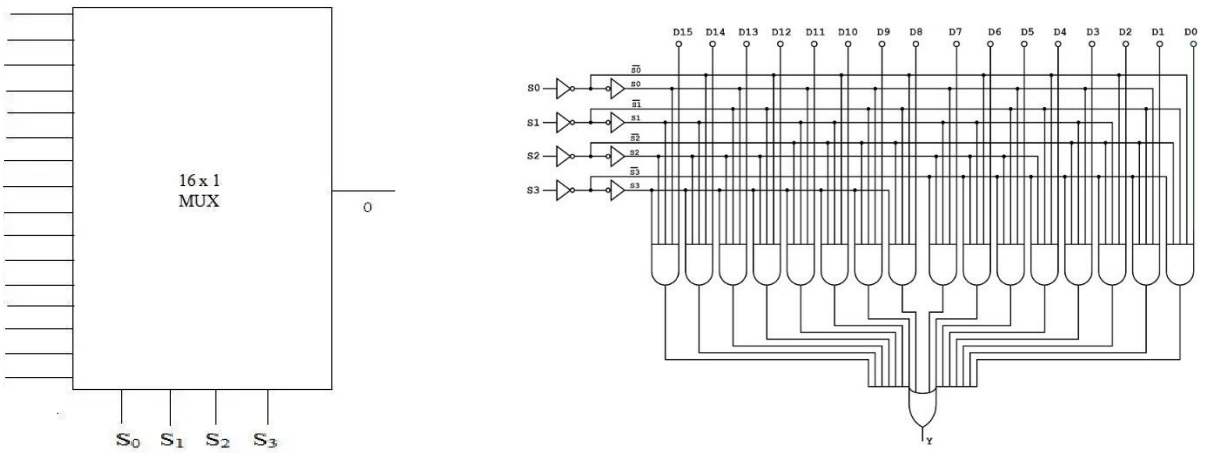
Fig. 1. Input and output in DE2 FPGA board.

As shown in Fig. 1, the switches 0-7 on the right are used to input operands **A** or **B**, and their selection is controlled by switches 10-11. These two binary numbers are stored in two registers and displayed in the LEDs above the switches. In addition, the decoded decimal numbers of **A** and **B** are displayed in the 7-segment displays. Specifically, the binary number of **A** is displayed in LEDs 0-7, while its decimal value is displayed in the 7-segment displays 4-5, highlighted with yellow frames. Similarly, the binary number of **B** is displayed in LEDs 10-17, and its decimal value is displayed in 7-segment displays 6-7 on the left, highlighted with purple frames. The Opcode is input from switches 13-17 on the left side. The binary output value is displayed in the group of eight LEDs on the right, while its decimal value is displayed in the three 7-segment displays 0-2 on the right, highlighted with white frames.

Conceptual Design

In terms of CPU design, each operation within the instruction set can be viewed as a separate module. The adder is the most frequently utilized component within an ALU, as it is involved in numerous operations. Subtraction can be implemented as addition when the subtrahend is represented as a negative number in 2's complement format. Additionally, integer multiplication can be achieved through a combination of left shifting and addition/subtraction. To illustrate this point, the multiplier can be expressed as the sum of several terms, such as: $5 = 2^2 + 1$ or $11 = 2^3 + 2^2 - 1$. Multiplying by 2^n can be accomplished by simply shifting the bits to the left, and then the result can be obtained via the addition/subtraction operation.

By implementing each instruction as a separate module, operations can be performed in parallel. This means that all 16 results can be obtained once the input data has been processed. To select the appropriate output, the control unit simply needs to utilize a 16-1 multiplexer with the 4-bit opcodes serving as the selection bits, which is shown in Fig. 2.



(a) (b)

Fig. 2. Control unit implemented with a multiplexer: (a) block diagram and (b) logic circuit.

In Fig. 2, we can see the circuit diagram for the multiplexer mentioned earlier. The 16 input bits represent the results from the 16 modules in the ALU, while the opcodes serve as the selection

bits (S_3, S_2, S_1, S_0) at the bottom of Fig. 2(a). The logic circuit in Fig. 2(b) shows that selection is accomplished through the use of AND/OR gates, with two equations from Boolean algebra utilized: $X \cdot 0 = 0$ and $X + 0 = X$. Each AND gate in the circuit has five input bits, one from a module in the ALU and four from the selection bits.

For example, the four control bits of the rightmost AND gate are ($\bar{S}_3, \bar{S}_2, \bar{S}_1, \bar{S}_0$). If any of the selection bits (S_3, S_2, S_1, S_0) is '1', the output of the AND gate will be '0', effectively blocking input data. Therefore, the condition for the data D0 to become the output of the multiplexer is that the four selection bits are (0 0 0 0), which corresponds to the opcode for addition found in Table 1. As such, D0 should be the result of the 8-bit adder.

In the same way, we can see that the second AND gate from the right, with D1 as the data input, has the control bits of ($\bar{S}_3, \bar{S}_2, \bar{S}_1, S_0$). Therefore, D1 can become the output of the multiplexer provided the opcode is (0 0 0 1), which corresponds to the operation of subtraction. Similarly, each of the 16 opcodes in the instruction set corresponds to a specific AND gate in the circuit, and only one of them becomes active for each opcode in the instruction set.

VHDL Design

In this project, the Arithmetic Logic Unit (ALU) and Control Unit (CU) have been combined and implemented as a single component named "alu_code" at the top level, shown as a block in the middle of the second column in Fig. 3.

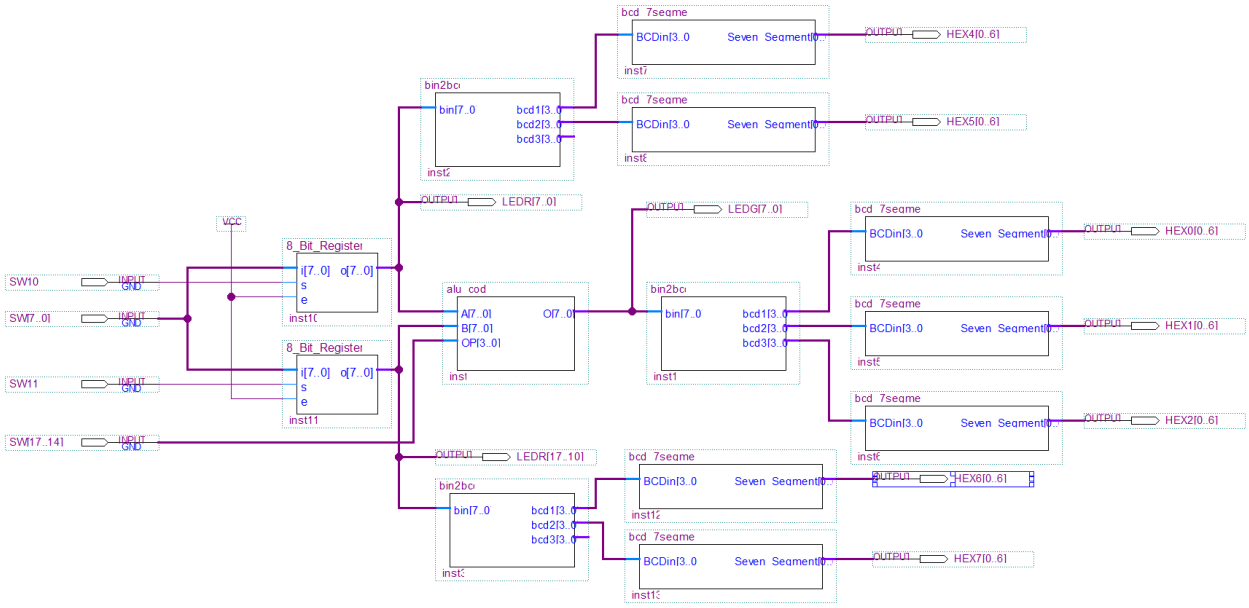


Fig. 3. Top Level CPU Circuit

To the left of the ALU-CU block, there are three input buses, which are connected to two registers, A and B, as well as the switches for input machine code. The output bus of the ALU-CU block is connected to a decoder that converts an 8-bit binary number into a decimal number.

Since an 8-bit binary number can represent a positive integer between 0 and 255, the decoder on the right side of the ALU-CU block is connected to three 7-segment display modules. For instance, if the result from the ALU-CU block is 123, the three 7-segment displays will respectively show (1-2-3).

To display the input numbers, the outputs from registers A and B are connected to two decoders located above and below the ALU-CU block in Fig. 3. Since the DE2 board has only eight 7-segment displays, with three reserved for displaying the CPU results, each register (A or B) has only two 7-segment displays available for output. Therefore, the third output pin of these two decoders is not used. Although the decoders can only display up to 99, the input number range is still between 0-255.

The ALU-CU component is designed by using VHDL, and the entity is shown in Fig. 4. In this code, **A** and **B** are the two 8-bit inputs from the two input registers, **OP** represents the 4-bit opcode, and **O** is the 8-bit output. In the architecture, the function of the ALU-CU is implemented with a case-statement structure, where each of the 16 opcodes is specified. This can be considered as the translation of the instruction set. Using the ieee.numeric_std package, all arithmetic operations can be easily handled.

```
entity alu_code is

generic (
      constant N: natural := 1  -- number of shited or rotated bits
    );

port (  -- the alu connections to external circuitry:
   A  : in STD_LOGIC_VECTOR(7 downto 0);   -- operand A
   B  : in STD_LOGIC_VECTOR(7 downto 0);   -- operand B
   OP : in STD_LOGIC_VECTOR(3 downto 0); -- opcode
   O  : out STD_LOGIC_VECTOR(7 downto 0));  -- operation result
end alu_code;
```

Fig. 4. Interface of ALU-CU module

Discussion

Although computers and microcontrollers are ubiquitous in modern society, many people, including programmers and engineers, do not fully understand how they work. Computer architecture is a complex subject, and some attempts to simplify it using a simplified version of a CPU [9] are still demanding for most people. This capstone design project offers a straightforward approach that can be easily understood by anyone with a basic background in digital logic circuits. As a result, it can be integrated into courses on microcontroller and computer organization, making it an accessible and valuable teaching tool.

Due to the limited time available for this capstone project during the pandemic, several features of the CPU were not included. For instance, there are various flags that are required for conditional jumps, such as zero, carry, negative, and overflow. Furthermore, our CPU is designed to work only in the "discrete mode," where the opcodes and data are manually inputted.

However, a more functional version of the CPU can be created by incorporating an on-board clock and memory, which is the topic for a future capstone project.

Conclusion

As a capstone project, we successfully designed and implemented an 8-bit CPU using the DE2 FPGA board. The CPU has 16 operations in its instruction set, each defined by a 4-bit opcode. With the use of VHDL code, we were able to realize the arithmetic functions of the ALU-CU at a higher level, bypassing the need for circuit-level design. We tested all the operations in the instruction set and verified the results through the use of the 7-segment displays and LEDs on the DE2 board.

References

[1] Ronald Hayne, "An instructional processor design using VHDL and an FPGA", *Proceedings of 118th ASEE Annual Conference*, Vancouver, BC, Canada, June 26 - 29, 2011.

[2] Karim Salman, Michael Anderton, "5-step design methodology for a general purpose CPU using standard CPLDs/FPGAs", *Proceedings of 112th ASEE Annual Conference*, Portland, Oregon, June 12 - 15, 2005.

[3] David Patterson, John Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 6th ed. Amsterdam, Netherland: Morgan Kaufmann, 2020. ISBN: 978-0128201091.

[4] M. Morris Mano, Charles Kime, Tom Martin, *Logic & Computer Design Fundamentals*, 5th ed. Boston, MA: Pearson, 2015. ISBN: 978-0133760637.

[5] David Harris, Sarah Harris, *Digital Design and Computer Architecture*, 2nd ed. Amsterdam, Netherland: Morgan Kaufmann, 2012. ISBN: 978-0123944245.

[6] Ronald Tocci, Neal Widmer, Greg Moss, *Digital Systems: Principles and Applications*, 12th ed. Boston, MA: Pearson, 2016. ISBN: 978-0134220130.

[7] Brock J. LaMeres, *Introduction to Logic Circuits & Logic Design with VHDL*, 2nd ed. Cham, Switzerland: Springer, 2019. ISBN: 978-3030124885.

[8] Demonstration video: https://youtu.be/MuV13OMyMwg

[9] J. Clark Scott, *But How Do It Know? The Basic Principles of Computers for Everyone*. John C Scott, 2009. ISBN: 978-0615303765.