4-Bit Microprocessor/CPU

ECE 2700 Final Project Report

List of Authors (Matthew Irvine, Joshua Duncan, George Trupiano, David Navarre)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
E-Mails: MatthewIrvine@oakland.edu, JoshuaDuncan@oakland.edu , Gtrupiano@oakland.edu, dnavarre@oakland.edu

***Abstract*— The aim of this project was to create a simplified microprocessor that would provide a representation and enable understanding of the capabilities and functionality of the device. With basic study, even those who are uninitiated towards microprocessors can explore the structure and use of CPUs. This CPU was created by coding in the software Vivado, using VHSIC Hardware Description Language (VHDL).**

I. Introduction

This project uses an FPGA board to create a microprocessor. A microprocessor is an electronic component that can perform complex functions and arithmetic operations based on a specific set of commands. The user is able to input a 4-bit unsigned number, along with a 6-bit command–which functions as the instruction for the microprocessor through the board's switches. After a user inputs a number and command, that user will then press the BTNC button on the board in order to perform one of many possible functions based on the command input. These functions include loading the inputted number from the board, inserting a number into a register, performing a variety of arithmetic or boolean operations between two numbers, outputting a number, in addition to a variety of other operations. The output of the functions is displayed on the board's built-in LEDs. Various programmed components make up the microprocessor and are what allow it to perform the aforementioned functions. These components include an Algebraic Logic Unit (ALU), registers, a multiplexor, and a control unit.

The inspiration for this project was the Intel 4004 (shown in Fig. 1) which was made in 1971. This was the first ever consumer grade microprocessor and it completely revolutionized the modern computer, and electronics in general. Today, almost every modern electronic device is powered by microprocessors. They can be found in cars, planes, and even common home appliances [1].

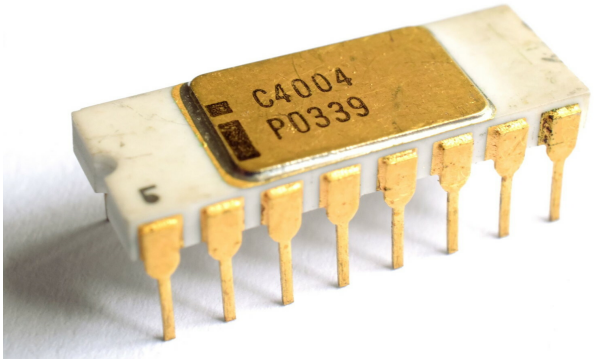

Figure 1: Intel 4004
This figure shows the Intel 4004 chip [1].

As can be seen in Fig. 2, the block diagram of the 4004 chip is relatively similar to that of the CPU created in this project and shares many of the same components. Notably present are registers, a data bus, multiplexers, an ALU, and a control unit. It is interesting that even after 50 years of development, the fundamentals of the microprocessor remain constant.

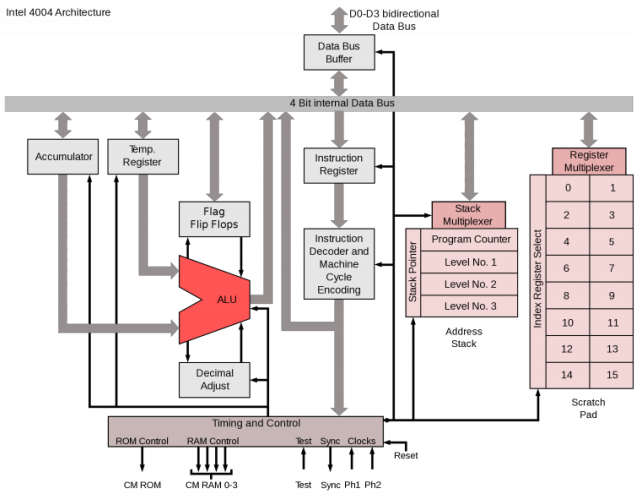

Figure 2: Intel 4004 Block Diagram
The figure shows the block diagram of the Intel 4004 [2].

As expected by the march of time, CPUs have advanced far beyond 4-bits since the 4004, but one can safely conclude

that modern electronics may not be in their present state without the 4004 pioneering the technology.

A microprocessor applies much of what is learned from the ECE 2700 course. All of the components used in the circuit were explored in depth during the class and this project applied those lesions yet forced the author's of this report to expand upon those teachings.

II. Methodology

For this project, an ALU, and a control circuit with specific instruction sets were used, along with multiple registers for the purpose of storing data. A multiplexor was also used to select specific registers, which allows that specific data to be recalled to perform the operation that is called on by the user. The control circuit further includes a finite state machine that allows a user to choose and execute the desired instruction.

The state machine also includes functionality to enable the register that houses the desired data, as well as the register to store the data after the execution of an algebraic or bitwise computation. The FSM is essential to this implementation; without it there would be no way to pass through specific instructions to the other components. The ALU could not function without enabling the correct register and without ensuring the correct operation is being called.

It was also critically important to track the correct register that will be used to read data. The complexity of the microprocessor is due to the variety of instructions that it can perform, which is the reason an instruction set and an FSM are used to provide specific instructions. For example, if a user desires to increment a number, that user first loads the number into a register. The user, wanting to increment this number, then needs to activate the ALU using the FSM. The FSM is designed to perform this specific operation by pre-assigning a state in the FSM that relates to certain conditions. The beauty of the microprocessor is that it is very flexible and in the case of the subject project, the microprocessor is also expandable and can be programmed to include additional arithmetic and bitwise functions.

A. Wiring/Block Diagram:

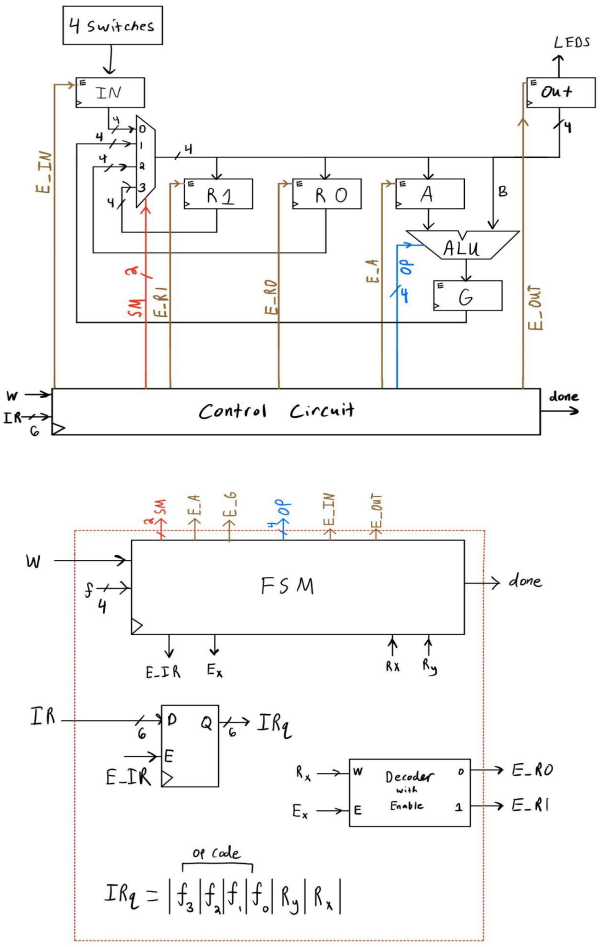

Figure 3: Block Diagram/Control Circuit

This figure shows the components of the simple microprocessor.

Upon review of the block diagram, one can see six registers, one ALU, and one MUX. The 6-bit input of IR is used to represent the set number of instructions that the microprocessor can implement. For specific instructions see Fig 4. The value of IR is inputted into the control circuit. Based on the value of IR, the Finite State Machine (FSM) is used to enable the specific registers that are required to execute the actions seen in Fig 7 . The 'w' input is a button that executes these instructions. Once a value is inputted into the IN register with the proper IR instruction, the user can enter any of the available commands to manipulate the entered value: whether that command is to add one, copy the value to a register, etc.

B. Instruction Set:

f(IR[5..2])	Operation	Function
0000	Load IN	IN←Switches
0001	Load R0, IN	R0←IN
0010	Copy R0, R1	R0←R1
0011	Add R0, R1	R1←R0+R1
0100	Add R0, IN	R0←R0+IN
0101	Xnor R0, R1	R1←R0 XNOR R1
0110	Subtract R1, R0	R0←R0-R1
0111	Subtract IN, R0	R0←IN - R0
1000	Xor R0,R1	R0←R1 Xor R0
1001	Inc R0	R0←R0+1
1010	LOAD OUT, R0	OUT←R0
1011	.....any op	Any function
1100	.....any op	Any function
1101	.....any op	Any function
1110	.....any op	Any function
1111	.....any op	Any function

Figure 4: Instruction Set/Assembly Code

This figure shows the instruction set of the microprocessor

As previously stated, the instruction set is one of the most essential components of this project. The Assembly code allows a user to load in data, copy that data, perform logic operations, and arithmetic operations etc. Upon review of Fig. 4, one will notice that the instruction set includes unassigned functions. This allows for an expansion of the microprocessor in the form of additional instruction sets.

The potential flexibility of the CPU is vast. A key tenet of this project was to create simple functionality that could be successfully implemented over an outright complex design that may have failed in its operation. As such, neither the use of signed numbers nor complex operations such as absolute value subtraction were considered. Admittedly, these operational instructions could have been easily created and applied to the project. However, even at present, these operations could be added without much difficulty. Put another way, every single operation and added operation in the instruction set could be executed and the results loaded to the board's LED display or to a seven segment display, should that output component be included to the project. That is, the microprocessor that was created for this project can be expanded and developed into an even more useful tool.

C. Rx and Ry Significance:

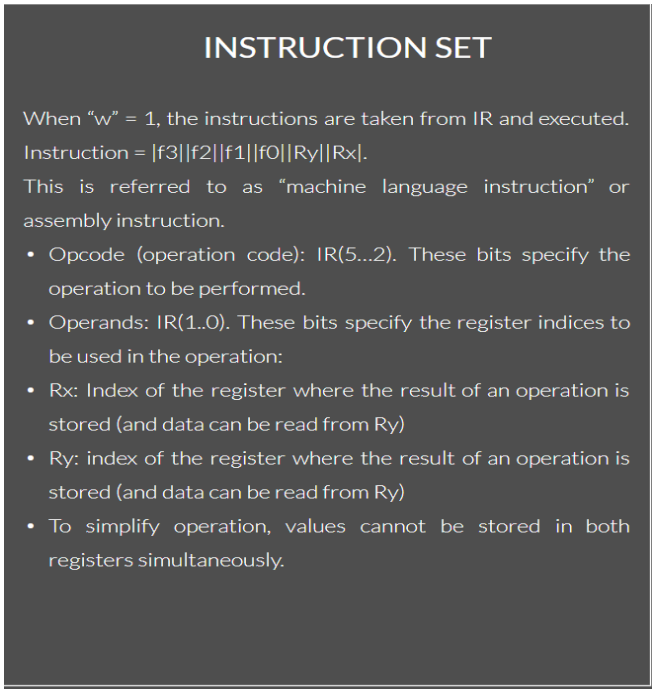

Figure 5: Rx, and Ry

This figure shows the jobs that Rx and Ry bits can accomplish.

To properly understand the FSM, a discussion of the inputs Rx and Ry are necessary. As stated earlier, instructions perform specific functions. This is easily understood. However, there are two other variables that are essential for this microprocessor to function as intended. Rx is the variable that dictates whether the R0 or R1 registers are enabled. Both cannot be enabled concurrently. An Rx value of '1' enables the R1 register. An Rx value of '0' enables the R0 register. Notwithstanding that function, Rx also serves an additional purpose: Rx determines where the processor reads the stored information and to which register it is written. The Rx value does this in conjunction with the select value of the multiplexor. Likewise, Ry also allows the device to recall and store data.

This seemingly simple aspect of the microprocessors is, in fact, the most complex part of the design owing to the fact that these two variables can be used in multiple ways. For example, given an Rx value of 1, data can be stored in the R1 register but it also allows the values in the register to be read. An Ry value of 0 allows for the reading of data from the R0 register and provides for the computation of an arithmetic or logic operation with values in the register that had been previously stored.

D. Arithmetic Logic Unit

The ALU is the component that handles all of the boolean and arithmetic operations of the Microprocessor. The ALU will take in two different numbers and a command that will determine which operation is done with the numbers. There are 3 inputs to the ALU. The first two inputs are "A" and "B," and these will be the two 4-bit numbers that the microprocessor will operate on. The third input is called "OP," which will represent the operation that the ALU needs to perform. "OP" comes from the control unit which will be described in more detail, *infra*. "OP" is a 4-bit number that will be fed into the ALUs internal multiplexor and will in turn select which operation will be performed. "OP" being a 4-bit number means that there are 16 potential operations that the ALU is able to accomplish. Some of the possible operations include incrementing A or B, adding or subtracting A and B, taking the NOT of A and B, and various other functions. The full list of operations is displayed in Fig. 6.

It is worth nothing that this ALU design has the potential to be expanded to a much larger scale; this will be done by increasing the number of bits in "OP." As stated earlier, "OP" currently has 16 possible values as a 4-bit number. By increasing the number of bits, more operations are possible. For example, increasing OP to 5-bits would allow for 32 possible operations that could be performed. The biggest benefit to adding more bits in this application would be to allow the ALU to handle negative numbers. Currently, the ALU is not able to manipulate negative numbers, and the "A" value inputted must always be the larger number. Adding more bits would allow the microprocessor to compute signed numbers, allowing a greater combination of numbers.. In general, the ALU could be expanded into a complete calculator if the application required it and this could be done by simply increasing the possible number of operations.

OP	Operation	Function
0000	y <= A	Transfer 'A'
0001	y <= A + 1	Increment 'A'
0010	y <= A - 1	Decrement 'A'
0011	y <= B	Transfer 'B'
0100	y <= B + 1	Increment 'B'
0101	y <= B - 1	Decrement 'B'
0110	y <= A + B	Add 'A' and 'B'
0111	y <= A - B	Subtract 'B' from 'A'
1000	y <= not A	Complement 'A'
1001	y <= not B	Complement 'B'
1010	y <= A AND B	AND
1011	y <= A OR B	OR
1100	y <= A NAND B	NAND
1101	y <= A NOR B	NOR
1110	y <= A XOR B	XOR
1111	y <= A XNOR B	XNOR

Figure 6: ALU Instruction Set
This figure shows the complete instruction set for the ALU. Based on the value of OP, the function corresponding to the value will be initiated.

D. Finite State Machine

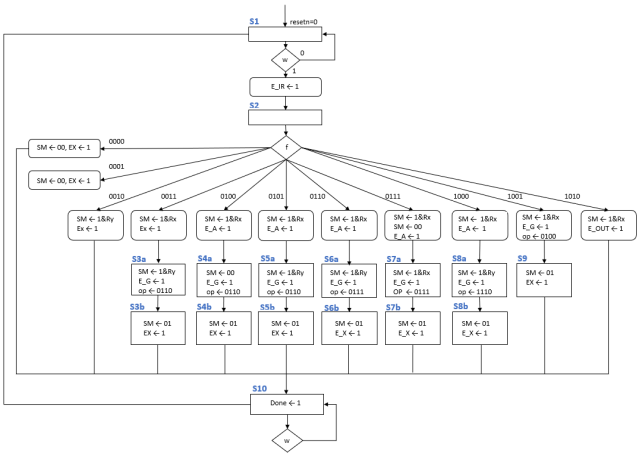

Figure 7: FSM (Please see link in appendix for full size)

The FSM requires that a certain condition be met before moving to the process of computation. This condition is based on the variable, "w" which needs to be equal to '1' for the FSM to move through the various states. When the FSM has cycled to the appropriate state, the programming logic determines which instructions are to be passed through. The

conditions will vary based on the instructions. For example, to load the register, "IN", this particular register needs to receive a signal to enable the register. An additional example would be performing the XNOR operation with data from registers R0 and R1. Fig. 1 is instructive in this explanation. To perform an XNOR operation the instruction needs to be inputted: here the instruction code is 0101 which allows the microprocessor to perform an XNOR operation with the values in register R0 and R1. The question now becomes: to which register are the results stored? Here, if Rx is '1' then the results are stored in R1 and if Rx is 0 the results are stored in R0. When this occurs, the Ry value must be 0. This is to avoid unnecessary redundancy in storing the values since both registers are needed to perform an XOR operation. The FSM is programmed so that the user needs to access data from the R0 register since the R1 register has been designated to both read and store the computational data. Conversely, if the R0 register has been set to read and store the data. Additional data will be read only from the R1 register.

III. Experimental Setup

To verify the results of this setup, simulations were used to test whether the microprocessor performed its intended function. The test bench feature in Vivado was used to test a series of instructions.

To create this test bench it was necessary to create the components of the circuit in VHDL

The VHDL code includes:

- Top File;
- FSM;
- ALU;
- Switch IN register;
- LEDout Register;
- R0 register;
- R1 register;
- RA register for the input of the ALU;
- RG register for the output of the ALU; and
- A control Circuit
    - FSM;
    - 6 bit register for Instructions; and
    - Decoder with enable.

The circuit is complex and involves many layers of code, any of which can hinder the operation of the CPU with a mere typographical error or missing semicolon. Therefore, Vivado's synthesis operation was frequently run to identify any obvious syntax errors. Even in the absence of syntax errors, some warnings were found to be detrimental to the entire project. The latch warning was a particular thorn and as a result, the team was wary of its presence. The latch warning denotes that outputs of the FSM lack initial values.

The test bench was programmed with multiple instructions to help track whether the microprocessor was performing as intended. Using the test bench with Vivado's behavioral analysis tool with a large number of instructions proved to be constructive at illustrating and demonstrating the breadth and effectiveness of this CPU Unit–especially while demonstrating the microprocessor to a group. The instructions are as follows:

- Load IN with 0110(6)
- Load R0 from IN
- copy R0 to R1
- increment R0 (store result in R0)
- Add R1 and R0 (store result in R1)
- XNOR R1 and R0 (store result in R1)
- Subtract IN and R0 (store result in R0)
- Subtract R1 and R0 (store result in R0)
- XNOR R0 and R1 (store result in R0)
- Load Out R0 to LEDs (store result in R0)

IV. Results

The results of the test bench and behavioral simulation can be reviewed in the appendix. Upon visual examination, one can see that the simulation worked effectively. Eleven instructions were used in the test bench resulting in a large and almost cumbersome amount of simulation data. For simplification, the first figure in the appendix demonstrates a value of 6 (six) being loaded in the first register, R0. The additional images depict the instructions that were input into the simulation such as copying the value into other registers, incrementing R0, etc. After processing multiple instructions, the results, both intended and produced, was an output value of '1'..

A simple video demonstration of the code and its application on a Basys 3 board is available via the link below. In this example, the following instructions were input into the board and executed by pressing the appropriate button on the board which corresponds to the 'w' input. The instructions are as follows:

- Load IN with 0110(6)
- Load R0 from IN
- Load out R0 to LEDs

This operation results in an output 0110 to the LEDs.

https://drive.google.com/file/d/1DClUGLph3xkf9GPyqZBBsjWjri1hhRxB/view?usp=share_link

Conclusions

At the onset of the project, it was advised to "simplify" the microprocessor. While the initial project was permeated with grand intentions, the advice proved to be sound. Despite the relatively basic nature of the microprocessor, the understanding and comprehension to implement the design did not come without suffering pains during the initial trials. When the basics of the project became functional additional challenges presented itself when the various additional instructions were introduced. Eleven instructions,

admittedly a modest amount, were difficult to debug and properly apply. Had the project been more complex, the truncated timetable of the project would have been a prohibitive factor to the successful completion of the project. As it stood, the project was operational with only a week to spare and was not completely debugged and fully operational until a day before the deadline, notwithstanding diligent and constant work from the beginning.

Additional features would have been beneficial. A seven-segment display would have been useful not only to the user during the process of testing and developing the processor. Additional instructions would have also been beneficial to allow the microprocessor to perform more impressive feats in the demonstration. The addition of a RAM component had been discussed but limited time did not allow this feature to be added.

At the conclusion of the project, what resulted was a functional microprocessor that instilled an appreciation and understanding for CPUs that are ever present in today's appliances: From a simple coffee maker to a cell phone, these processors represent a complex and likely underappreciated aspect in our daily lives. The project set the goal of making a microprocessor that was capable of basic arithmetic and logic functions. The project was successful in its scope and goal, and served the ultimate purpose of the project: to use the teachings of ECE 2700, knowledge of FPGA boards, microprocessors, and VHDL and to expand on these learned principles to understand and create a CPU using digital logic design.

References

List and number all bibliographical references in 9-point Times, single-spaced, at the end of your paper. When referenced in the text, enclose the citation number in square brackets, for example [1]. Where appropriate, include the name(s) of editors of referenced books. The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first . . ."

[1] Miller, Michael J. "The Microprocessor at 50: How the 4004 Changed The World." *PCMAG*, Ziff Davis, 15 Nov. 2021, www.pcmag.com/news/how-the-4004-changed-the-world. Accessed 3 Dec. 2022.

[2] Sack, Harold. "Intel 4004 – The World's First Microprocessor." *SciHi Blog*, SciHi, 15 Nov. 2021, scihi.org/intel-4004-microprocessor/. Accessed 3 Dec. 2022.

[3] D. Llamocca, *Reconfigurable Computing Research Laboratory.* [Online]. Available: http//www.secs.oakland.edu/~llamocca/index.html. [Accessed: 14-Nov-2022].

[4] Xilinx, "Vivado ML Overview," *Xilinx*. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html. [Accessed: 14-Nov-2022].

Appendix

1. Simulation Snapshots and FSM
https://drive.google.com/drive/folders/1V_jQyd0xcCKFhThhEq66CJoeQi55342u?usp=sharing