# CSCE 230 Project

Final Report

# Group #9

Johnathan Carlson, Tyler Zinsmaster, Jake Ediger

This project details the creation and implementation of a processor built in VHDL, implemented on a FPGA.

|  |  |  |
| --- | --- | --- |
| **Version** | **Changes/Additions** | **Date** |
| Phase II | Added documentation for B-Type, J-type, and D-type instructions | 11/2/17-11/8/17 |
| Phase III | Extending documentation for instruction types, as well as the initial assembler | 11/13/17-11/22/17 |
| Phase IV | Finishing the basic processor | 11/19/17 |
| Phase V and Extra Credit | Finished assembler, Optional Phase V, Demonstration | 11/27/17-11/3 |

# Overview

The project to design and implement a processor began with designing rudimentary parts in VHDL, a programming language that is used to implement circuits on boards like the Altera FPGA used in this project. Designing a processor on in VHDL involves making many individual parts, and putting them all together to perform tasks. 4 types of instructions were implemented separately- J-type, B-type, R-type, and D-type. After each part was created, it needed to be tested, integrated, and optimized for the processor to work. Each phase had a particular addition and/or modification. In a broad breakdown, each phase consisted of the following:

Parts added in each phase:

**Phase I-** Phase I was not a true phase, as all the parts were created prior to the beginning of the project. The parts included the 16x16 register file, the 16-bit ALU(Arithmetic Logic Unit) that can produce NZVC flags, and the barebones of the control unit. The control unit in this phase had minimal capabilities, only being able to execute R-type Instructions.

**Phase II-** Integrating the parts created in phase I to create a basic data-path compatible with R-type instructions. Additional parts needed included the IR(instruction register), immediate extender, as well other minor components.

**Phase III-** The remaining three types of instructions- J-type, D-type, and B-type, were implemented. A memory interface was added as well as the instruction address generator to the data-path. The control unit was updated to accommodate these new capabilities.

**Phase IV-**The final required phase, phase IV, was based in implementing I/O. With I/O, a memory interface was needed to be fully implemented, again updating the control unit.

**Phase V and Extra Features-** Many features that were not necessary were added to increase functionality and usefulness. Phase V was not required, but once implemented, significantly increases the capabilities of I/O. Although some of the LED’s and slider switches were already implemented in phase IV, much more detail was able to be implemented.

**Assembler-**The assembler was initially created to be a very basic two pass version to deal with instructions. As it was developed further, it became increasingly valuable for testing purposes. It allowed for efficient testing without re-compilation of the main program, as well catching small errors that would have gone undetected otherwise. The assembler was created in python.

# Detailed Processor Development and Implementation

## Control Unit

The control unit is arguably the most important part of the processor. It dictates all instructions and program flow. The control unit can be thought of as the component that ties everything together, and allows everything to flow seamlessly together. The control unit was by no means complete at the beginning of phase II, when it was first implemented and used. Each part added in the consecutive phases required additions to make each part compatible with the control unit. The control unit is also relied on to deal with logic. It also keeps track of an instruction as it is being executed, guiding it through the right.

## Instruction Types

* **R-type –** R-type instructions were the first instructions to be implemented on the processor. R-type instructions include Add, Sub, And, Or, *and* Xor.
* **D-type –** D-type instructions deal with data transfer between the memory and CPU. These instructions include load word, store word, add immediate, and sub immediate. Add immediate and sub immediate differ from the regular add and sub commands because they allow adding or subbing a numerical value, rather than the value of another register.
* **J-type-** J-type instructions involve jumping commands. Jump, jump and link, and load immediate are all J-type instructions.
* **B-type-** B-type instructions deal with branching, the main way loops and functions can be implemented in assembly. Instructions include Branch and Branch and link.

## I/O

Green LED’s, push buttons, and slider switches were all implemented in phase IV. The VHDL code determines which action to be performed based on the 4 bit code that corresponds to each part of I/O. A lot of difficulty occurred when moving the VHDL from phase III to the FPGA, which was necessary for phase IV I/O. Prior to implementing the program on the FPGA, a modeling software was used to test correctness. In order to transfer the functionality from the modeling software to the FPGA, the data-path needed to be drawn out so each stage could be understood. After creation of the data-path diagram, full implementation could be completed. A large step was made once we had functional I/O, but it was far from complete. All I/O initially had correctness issues, meaning although the I/O is responsive, the response were incorrect.

## Bonus: Assembler

The assembler had already been partially completed and had limiting functionality prior to phase IV. During phase IV, the rest of the assembler was completed. The assembler was designed in python, and proved to be a valuable tool in testing as well as implementation.

The assembler used is known as a two-pass assembler, which was necessary to be sure everything was correct. It outputs a .mif file, as well as an organized debugging interface. An example of a mif file is further down the report. The assembler was versatile in that any file could be run and tested, without the need to be redesigned for each new test. It stores the first 8 memory addresses in an array for use with I/O. A clean output provided easy debugging and testing. A variety of test files were used in the assembler, each testing a certain phase. Some phases required more than 1 test file to ensure correctness.

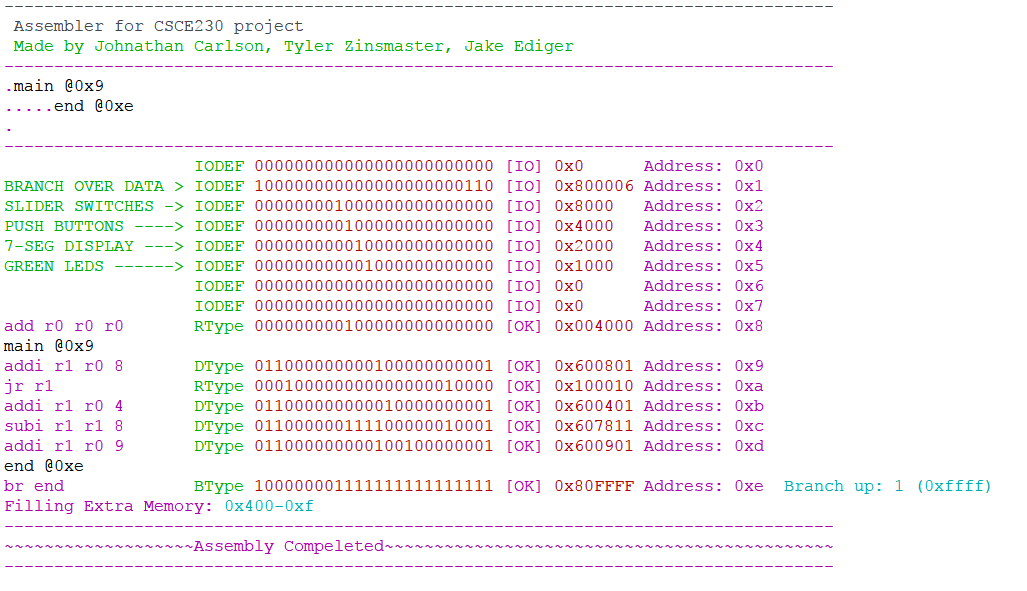


Figure 1: This image is a sample output of the assembler. The data is from one of the test files used in phase IV to test I/O with the FPGA, condensed to save room.

# Speed Overview

The speed of a processor is based on clock speed. 50Hz is the current clock speed that allows everything to run correctly. Fast adders were implemented to allow for higher clock speeds should the need arise. Although the clock speeds are relatively slow, higher clock speeds could be easily achieved with a few modifications. This would heighten the risk of unforeseen errors however, so the clock speed was kept slower. For any task we would want to perform, increased speed to be of minimal benefit.

# **MIF File**

An example .mif file, discussed above in the *Assembler* section.

WIDTH=24;

DEPTH=1024;

ADDRESS\_RADIX=UNS;

DATA\_RADIX=BIN;

CONTENT BEGIN

0: 000000000000000000000000;

1: 100000000000000000001100;

2: 000000001000000000000000;

3: 000000000100000000000000;

4: 000000000010000000000000;

5: 000000000101000000000000;

6: 000000000110000000000000;

7: 000000000111000000000000;

8: 000000000001000000000000;

9: 000000000011000000000000;

10: 000000001110000000000000;

11: 000000001111000000000000;

12: 000000000000000000000000;

13: 000000000000000000000000;

14: 010000000000101000000010;

15: 010000000000100000000011;

16: 011000000000000100000101;

17: 011000000000000000000110;

18: 010000000000000000100101;

19: 010100000000000000110101;

20: 100000001111111111111101;

[20..1023] : 000000000000000000000000;

# Group Experiences throughout project

Group #9 experienced a lot of trouble along the path to building a processor, but managed to get all checkoffs completed on time, and all parts functional. A full assembler was also created, which aided in progress throughout the project. The demonstration was well put together and made use of all the parts created.

# Conclusion

Building a processor is both easier and more difficult than it sounds when first presented. It is easier once you understand that it is just a set of parts working in conjunction to perform tasks, and each of these parts are relatively simple by themselves. It becomes much more difficult when a time constraint is introduced, making the workload quite immense at times. Making everything compatible proved to be an extremely complex task, fixing one thing always seemed to break another thing.