**Alex Carter & Jordan Richardson**

**CPU Design Project part 5**

1. What did you learn from this project?

This project taught us the basics of designing, and implementing larger scale projects using VHDL, specifically a processor. It also forced us to become acquired with the Altera Quartus II IDE and working with the Altera version of ModelSim to verify our designs. In designing a processor, we learned to deal with timing issues, invalid results, and improper designs in simulation, specifically dealing with synchronous and asynchronous pieces of the processor, such as ALU’s, registers, and Control Units. We also applied our previously learned skills in designing a finite state machine using VHDL to create a Control Unit, and an ALU Control Unit that implement the basic functionality of the Processor. Finally we learned what was involved with implementing the design on the DE2 FPGA board.

1. What would you do differently next time?

Coming into this project we really weren’t prepared for the amount of testing, and correcting our code. I would be mentally prepared for the amount of corrections I would have to make in all parts of the design process. Also I would ideally be implementing a pipelined processor as opposed to a multicycle processor. Had we had more time then pipelined would have seemed a better, more efficient design. However for the time period for our project I would recommend aiming for simplicity as much as possible, specifically in the instruction format. Initially we intended to mimic the RISC architecture completely in format of our assembly instructions vs. our binary machine code instructions, however we finally decided on an instruction set in which the order of the operands in assembly language represented the binary language was easier to handle effectively.

1. What is your advice to someone working on a similar project?

Remember there will always be unaccounted for issues, specifically in timing. Throughout the project we kept finding that we had clock data races, or that our data that we needed in one location was no longer valid. In the design of the ISA I would highly recommend getting exactly what you want the first time. Going back and making changes is time consuming, and requires way more work than doing it correctly in the first place. In Simulation, to save time I would recommend you prepare very thoroughly. So the sooner you write a test bench or do file, the sooner you won’t have to completely do it over again. Also generics are very important in VHDL code, so if you have your files in a formatted with generics in the first place You’ll save time with the mistakes you do make

**TEST PROGRAM**

LUI 00000001, $3

LLI 00101100, $3 # Register 3 now contains decimal 300

LUI 00000001, $4

LLI 11110100, $4 # Register 4 now contains decimal value 500

LUI 11111111, $5

LLI 00111000, $5 # Register 5 now contains decimal -200

LUI 11111101, $6

LLI 10101000, $6 # Register 6 now contains decimal value -600

ADD $3, $4, $7 # Register 7 should now contain 300+500=800

SUB $5, $6, $8 # Register 8 should now contain -200-(-600)=-800

AND $7, $8, $9 # Register 9 should now contain 00000000 00100000

OR $7, $8, $10 # Register 10 should now contain 1111111111100000

LUI 00000000, $12

LLI 00011110, $12 # Load register 12 with decimal 30, the location of HALT

BEQ $9, $10, $12 # Should not branch. If branch occurs, program will HALT

SW $3, $0, $9 # Store value of register 9 at address 300

SW $4, $0, $10 # Store value of register 10 at address 500

LUI 00000000, $12

LLI 00011010, $12 # Load register 12 with decimal 26, the location of the subroutine

JAL $12, $0, $15 # Store PC+1 in register 15 and jump to address in register 12

LUI 00000000, $12

LLI 00011110, $12 # Load register 12 with decimal 30, the location of HALT

BEQ $1, $0, $12 # Register 1 should contain 0. If the branch occurs, the program is finished.

SW $7, $0, $1 # If branch does not occur, store value of $1 at address 800 and HALT

HALT

LW $3, $0, $10

LW $4, $0, $11

ADD $10, $11, $1 #Register 1 should now contain 0

JAL $15, $0, $0 # Return to calling address

HALT