Project Part III

CSCE 230 Honors

Professor Reidesel

11/20/2015

Will Preachuk

Levi Amen

Cameron Johnson

Overview:

During Stage III, the processor being designed truly became a processor. While in Stage II, instructions were forced to specific values in our simulation, the Stage III Processor had a fully functional and implemented memory and program counter. This addition allowed instructions to be pulled from and stored in memory, in turn allowing for D type and B type instructions to be implemented. These D and B type instructions follow the patterns created and defined in the ISA (Instruction Set Architecture) which had been previously completed. A 32 bit by 1024 word memory bank, memory interface, multiple multiplexers, and temporary buffer registers were added into the previously existing VHDL Code. Signals to facilitate the implementation of the new instructions, and functionality involving memory and the handling of immediate values were also added.

Implemented Instructions:

In this edition of the processor, 6 new instructions were added to the processor. Unlike the 5 already implemented R type instructions, the 7 instructions implemented (addi,lw,sw,cmp,jr,b and bal) all have differing opcodes. The lw and sw command interacted with the newly implemented memory interface. sw took a value in a specific register and placed it into a defined address in memory (which may have an immediate offset added). lw took a word (32 bits long) from a specific memory location (which may also have an immediate offset) and stored it in a specified register. B and BAL will move the PC forward a specific integer value, the difference being that BAL will store the previous PC value into the Link register. The JR command takes the value in a register and sets the program counter to the value in the specified

register. CMP takes two integers, subtracts them and then stores the ALU flags into the CMP register this can be used to determine whether two values are the same or differ. Finally, addi takes the sum of a register value and an immediate value and then stores it in a register.

Components Added:

The largest component added to the processor was a main memory component. This composed of a 32 bit by 1024 word memory bank to store and load values from. The addition made necessary a 2-port-mux to choose between PC and RZ for a memory address input. The memory output goes to the already existing MuxY. A 3-port-mux was added as the RegD input to the register file to accommodate for the different bit locations for the RegD address in R-Type, D-Type and bal instructions. Most of the new components were added to the instruction address generator, including an adder to both increment the PC and add branch offsets. A 2-port-mux was added to the PC input to select between the link register (R30) and the PC adder circuit. Another 2-port-mux was added to the input of the PC adder to select between a value of '1' for normal PC incrementing and an immediate value for branching.

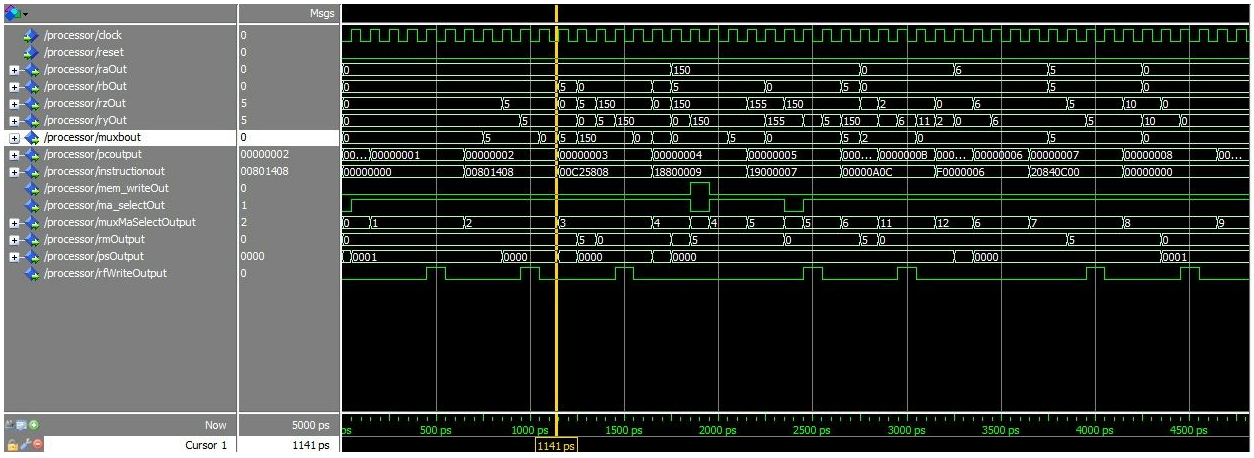
Processor Speed:

After completing the third stage of processor implementations, the processor was tested to see how fast the clock could pulse while still resulting in correct instructions and results being fed into the processor. During the initial debugging stage, a clock speed of .1 megahertz was used. After debugging and finalized results, the processor was safely able to calculate values and access memory without issues at 10 megahertz. This is a significant drop from the 500 megahertz

speed the processor was able to operate at during stage two. This slowdown is due to the fact that instructions are being pulled from memory, rather than being forced to specific values in ModelSim, as well as time needed to potentially access memory during lw and sw operations.

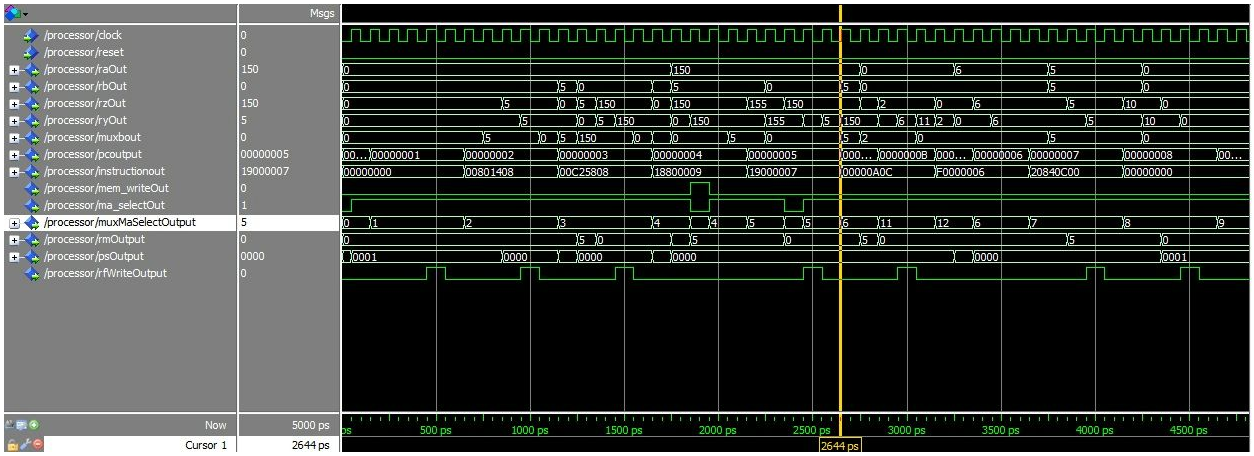
Testing:

Testing was primarily done using Altera Modelsim, a program which simulates pin inputs and clock speed, while showing the calculated results of the defined output pins. In order to test the newly implemented memory interface, testing was done to make sure that the instructions were being loaded from memory correctly, this was done by loading information into the Memory Initialization File (.MIF) allowing the program counter to implement and seeing if the value is loaded into the new IR register which is done as seen below.

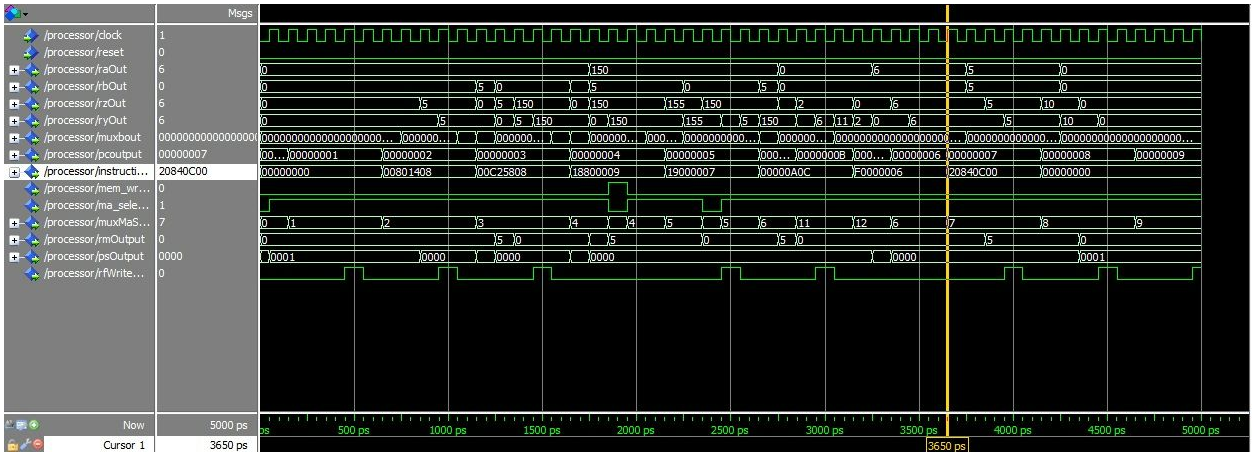


As can be seen, not only is the instruction loaded in from memory, but the first operation an ADDI of R0 and 5 into R2 is completed successfully. Finally in order to test the capabilities of

our processor to store and load from memory, the value 5 was stored into address 150 and the restored into register 5



Finally, the bal and jr commands were tested by branching to a different space in memory, then placing a jr at the branch location and jumping back to the value in the link register.



as can be seen, the program counter changes to the location specified in the branch, and then changes back after the jump, allowing for the final add command to take place.

Group Experiences:

While making the normal changes and additions necessary to complete part III was exciting, as the processor was being fully realized, this quickly turned into frustration. There was, and in a sense still is, and demonic bug in our processor which causes inexplicable timing issues with our lw instruction. This bug took a total of around 40 man hours to “fix” between the group members and TAs. The “fix” constituted of a hack which required an additional MUX to be added to the circuit which picked the value coming directly from the memory output port a clock cycle early instead of waiting for it to propagate to RY. Overall, the experience on part III for the group was a negative one filled with frustration, but it’s done now.