Benjamin Newcomer

2-23-2015

CS 3220

Project 2

The objective of this project is to create a multi cycle processor and accompanying assembler that implements a particular ISA. Unfortunately the processor was not completed. However, many lessons were learned in the design and debug process. The processor, written in verilog, was written mainly in one large module split into blocks by function: Parameters and Timing; Bus and PC; Memory and I/O; Instructions and IR; Register File; ALU and Z; and Microcontroller. The first block defined parameters such as memory width, opcodes, microstate codes, and also defined a pll and reset signal. The rest of the blocks defined components of the processor.

SignalTap was used extensively to debug the processor. Registers like IR, PC, A, B, state, MAR, and MDR were monitored to follow program execution. Data collection was usually triggered on the falling edge of the reset signal, to catch the processor from its initial state. One challenge was that registers were frequently optimized out of the design and could not be monitored. To fix this, registers were output to pins (LEDG for example) to ensure they would not be optimized out. Another major challenge is that it is not possible to view the real time contents of memory or the register file, which made complicated the debugging process. In the future, more extensive use of SignalTap and more incremental testing will decrease overall debug time.

When testing the processor with the provided Test.a32 file, the processor displayed *Fd6b* on the seven segment display and *23* on the green LEDs. The cause of the test failure is attributed to the fp register accumulating an incorrect value, which caused a jump to go to

the incorrect destination. Other problems that were successfully debugged include asserting the correct ALUfunc when taking a branch, correctly reading memory (takes two cycles) and splitting instructions up differently depending on instruction type.

The assembler, written in python, leverages dictionaries with opcodes, instruction types, memory parameters, and more. Most of the challenges in creating the assembler surfaced in dealing with different encoding formats for different instruction types, dealing with .WORD directives, and handling two’s compliment signed numbers (shortening and extending).