Brown University
School of Engineering
EN164 Design of Computing Systems
Professor Sherief Reda
LAB 06 (200 points)
Final report due on Friday May 9 2014 (or before)

In this lab you are going to augment the design of your processor with memory and I/O capabilities. You can either use your single-cycle design or pipeline design as the baseline.

a) [100 points] For memory, it is required to interface your processor to the external asynchronous 256 KB ×16 SRAM available on the FPGA board. You should use this external SRAM to substitute your internal M4K used for your data memory. The interfacing will be done by designing a memory controller that takes as inputs the original inputs to the internal memory (e.g., address bus, data bus, memory read/write control signals) and outputting a 32 bit data and potential other new control signals. Since the SRAM has a 16-bit data bus, writing or reading from the SRAM requires two bus cycles to read/write 32 bits. Implement your design and check it correctness by running the factorial program and reading the content of the external SRAM. To read the content of the external SRAM, there is a tool (DE2 control panel) that can be used to check the content of the SRAM; however, before using it, you need to download its .sof file in the FPGA. Note that this downloading does not affect the content of the SRAM because it is external to the FPGA. Report the frequency of operation, the average cycles per instruction, and the design area required for the memory controller. Contrast your design performance to the one that uses the internal M4K memory. If you were to improve your design by using the internal memory as a direct-mapped cache to the external SRAM, only explain with sketches and words (no need for implementations), how would you do the cache-based design.

b) [100 points] For I/O, we are going to interface the input switches and the 7-segment displays (four of them) to your processor using the memory I/O technique. We can reserve, say, word address 100 in the memory for the input switches and word address 101 in the memory for the 7 segment displays. Any read to location 100 should get the status of the switches, and any write to location 101 should write to the pins of the 7 segment displays (first display uses least seven bits of location 101, second display uses next seven bits, and so forth). First, you need to write a SW routine that acts as a device driver that will take an 8-bit signed number (using argument register $a0) and display it on the four 7 segment displays. Second, you will write a program that reads the status of 16 switches (two 8-bit numbers) and the operation code (specified using two switches) and outputs the 8-bit result of the operation on the 7-segment displays. The required operations are: add, sub, multiply, and bitwise AND. Ignore overflows. Note the program is implementing in SW what was implemented in HW in Question 3 of Lab 1. Include your source code in the report.