**Design**

In order to implement the data-processing addressing modes, a shifter module was created that would process the immediate and register instruction encodings. The register-shifted register mode was not implemented due to the limitations of the single cycle architecture.

The shifter module was placed between the rd2 port of the register file and the first input of the SrcB multiplexer that feeds into the ALU. The shifter unit also takes as input the last 11 bits of the instruction word. The control logic causes the rd2 port to output the contents of register Rm during data processing instructions. The unmodified output of the second port, labeled WriteData, is still fed into an output port of the datapath module. A new 32-bit logic signal called Srcbmux0 was added to the datapath to connect the output of the shifter to the input of the ALU input multiplexer.

**Testing Strategy**

The team decided that testing each newly created or modified component individually would be too time consuming for this assignment. Instead, testing consisted of feeding various input programs into the whole simulation and verifying them for correctness. One program in particular that was used extensively was the provided Fibonacci calculator. Because the expected answer is known and the code was easy to run by hand, it was simple to debug the simulation by following its execution cycle by cycle until an incorrect value was found. After the Fibonacci program gave the expected output, the rest of the provided input files were run.

One challenge was that there was no way to dump the contents of the register file. This was overcome by instead using the waveform viewer in ModelSim to manually track the major logic signals, such as the output of the ALU and the read data ports of the register file. Additionally, programs that wrote to memory were verified by examining the memory dumps generated by the program.

**Evaluation**

The implemented ARM machine produced correct results for all provided input programs.

Several alternate designs were considered, and even partially implemented to initial success, but were ultimately turned down due to the complexity of implementing the necessary changes to the control logic. Originally, the team intended to implement a unique ALU control signal value for each data processing instruction. This became a problem when the team realized that the provided control signals and the structure of the datapath made it difficult to handle the shift instructions directly in the ALU. The solution to this problem lead to the implementation of the shifter module, which allowed the shift instructions to be handled as MOVs within the ALU.

A change related to the previous problem made some initial changes to the control logic to be obsolete. In order to implement extra logic for the shift instructions, the control module was modified to take in the entire instruction word rather than a small slice of it. While this alternate design still produced correct results, it was made obsolete and abandoned after the shifter module was implemented.

Overall, this lab served to both reinforce the ARM processor concepts explored in the previous labs and introduced several challenges involved in the physical implementation of ARM machines. Originally, the lab would have required the simulation to be implemented on an FPGA board. In addition to reviewing FPGA basics, this would reinforce the physical implementation aspect of the assignment. In the next lab these concepts will be explored further using a pipelined implementation of the ARM architecture.