Project 2: Single Cycle Computer (SCC)

Objectives:

1. Design a SCC using the concepts of 5 main parts of the processor: IF, ID, EXE, MEM, WB
2. Create and use a test bench to validate the design
3. Run ARM8 opcodes against the design

**Groups:**

The project is done in groups and graded as a group. Grading will be done by the groups as well.

**The Project Overview:**

Using the CIA (Class Instruction Architecture) as your target, design a single cycle computer in Verilog to support the Opcodes in the CIA. A single cycle computer runs one instruction at a time. There is no pipeline. Instruction two does not start until instruction one completes. The assumption for a SCC is that the time required to execute any and all instructions can be completed in one clock cycle.

You will reference the CIA document. It contains explanation and examples for each supported instruction. You may also use the ARM reference documentation as well. If the instruction is listed as (optional) in the CIA document, you do not need to implement in Verilog for this project.

You should support all of the instructions.

Even thought it is a SCC design, it is still static. It is clocked. Registers/flops, should only change when they are required to change based on the instruction.

Think about design choices for this project because this project will be the starting point for the “pipeline the SCC” project.

**Assumptions for the Project:**

1. You only need to support 32bit data widths. Of course all instructions are 32bits wide since it is ARM RISC architecture. Any 64bit wide registers or instructions are not supported.
2. If the instruction sets a flag (carry flag, for example), support the flag.
3. You do not need to use optimized architectural design for parts of the ALU. For example, you can just add two registers using the “+”. You do not need to instantiate carry look ahead adders as we did on the first project.
4. This is your code. Comment it. Do not cut and paste other code.
5. Try to use meaningful names. Use variable names. Use parameters where appropriate.

**Wrapper**

All designs must use the same I/O wrapper so that we have consistency on the I/O names for debug/test.

**Test Bench**

See Wrapper.

You will also need to think about a testbench memory for read/write to simulate memory store and fetch.

Final functional testing will use the same instruction opcode stream for all groups.

**Functional Test Code**

We will have a contest to create an optimal and easy to use functional test code program. Each group will write assembly and compile by hand or with compiler valid opcodes for a program which the class decides the function choosing from:

1. A number sorting algorithm given a starting and ending address range for the sort
   1. Assume base 16 numbers 2 bytes wide (a word)
   2. Data is 4 bytes wide so a 4 byes of data has two words (A double word).
   3. Sort at the word level
2. A pattern search algorithm given a 1-4 byte pattern, a starting and ending address.
3. A Hex to decimal convertor for 4byte data supports 2’s complement format for hex.
4. A countdown timer. Set starting value. Timer counts down to zero. Each count is a length of time (number of clocks) which is changeable.

The “best” design receives a $100 Amazon gift card for the group to split.