Objective:

In this assignment, you will make a working ALU unit. ALU is one of the fundamental units of a processor as it is used most of the time for doing basic Arithmetic and Logical operations. Learning how to design a good ALU is necessary for all computer architects.

From each problem follow the hierarchical approach, i.e.:

1. Break down your design into sub-modules.
2. Define interfaces between these modules.
3. Draw paper and pencil schematics for these modules.
4. Then start writing Verilog.

Before you start please take a look at the files mentioned in hw1.pdf section "Before you start" if you have not seen them yet. They include all the basic rules and regulations, as well as how to guide for ModelSim and Verilog.
Also, note that we do not have any specified commenting guidelines for this assignment (we do have other restrictions like naming convention, etc, see hw1.pdf) but please comment your code appropriately. It will help you debug things easily and you may reuse a few of these modules in your course project, so making it modular and commented will help you in future.

Problem 1: (40 Points)

Design a 16-bit barrel shifter called `shifter` in `shifter.v` with the following interface. Consult lecture 3: Arithmetic Circuits for shifter design.

Inputs:

```
[15:0] In - 16 bit input, value to be shifted
[3:0] Cnt - 4 bit amount to shift (number of bit positions to shift)
[1:0] Op  - shift type, see encoding in table below
```

Output:

```
[15:0] Out - 16 bit output
```

Opcode	Operation
00	Shift Right Arithmetic
01	Shift Right Logical
10	Rotate Left
11	Shift Left Logical

Following Link will explain details regarding logical shift and arithmetic shift: Logical Vs. Arithmetic Shift - Open4Tech

Also, note that in Shift Left both arithmetic and logical are the same so we do not have Shift Left Arithmetic. In its place, we have “rotate left” which basically means a cyclic rotation of bits.

Constraints:

1. You cannot use verilog logical shift operators (i.e. <<, <<<, >>, >>>, etc) in your code
2. You have to design your solution using muxes. (You may need to design a 2-to-1 mux, 4-to-1 mux, etc for your solution based on your implementation)

Lastly, verify your solution using the testbench provided. If there are any incorrect answers in the testbench will print an `ERRORCHECK: ...` message.

What to submit:

```
hw2 ->
    hw2_1 ->
        All Verilog files
        All Verilog rules checking output files
```

Submit all the files needed to compile the program and run the testbenches.

If you have designed any layout, diagrams, or personal testbenches feel free to add them in your submission, but note that they are not compulsory and you will not be graded on them. Also, make sure that the provided testbench is not interrupted due to any of the additional files.

Problem 2: (60 Points)

Design a simple 16-bit ALU called `alu` in `alu.v`. Operations to be performed are 2's Complement ADD, bitwise-OR, bitwise-XOR, bitwise-AND, and the shift unit from problem 1. In addition, it must have the ability to invert either of its data inputs before performing the operation and have a Cin input (to enable subtraction). Another input line also determines whether the arithmetic to be performed is signed or unsigned.

Use a carry look-ahead adder (CLA) in your design. (Hint: First design a 4-bit CLA. Then use blocks of this CLA for designing the 16-bit CLA.) For all the shift and rotate operations, assume the number to shift is input A to ALU and the shift/rotate amount is bits [3:0] of input B.

Opcode	Function	Result
000	ADD	A+B
001	OR	A OR B
010	XOR	A XOR B
011	AND	A AND B

100	sra	Shift Right Arithmetic
101	srl	Shift Right Logical
110	rll	Rotate Left
111	sll	Shift Left Logical

ALU's I/O interface:

Inputs:

- **[15:0] A, [15:0] B** - Data input lines A and B (16 bits each)
- **Cin** - A carry-in for the LSB of the adder
- **[2:0] Op** - The OP code (3 bits). It determines the operation to be performed. The opcodes are shown in the table above
- **invA** - An invert-A input (active high) that causes the A input to be inverted before the operation is performed
- **invB** - An invert-B input (active high) that causes the B input to be inverted before the operation is performed
- **sign** - A signed-or-unsigned input (active high for signed) that indicates whether signed or unsigned arithmetic to be performed for ADD function on the data lines. This affects the Ofl output

Outputs:

- **[15:0] Out** - Data out (16 bits)
- **Ofl** - This bit is high if an overflow occurred
- **Zero** - This indicates that the result is exactly zero

Other assumptions:

- You can assume 2's complement numbers for signed numbers.
- In case of logic functions, Ofl is not set.

Simulate and verify your own design using the provided testbench. Make your own testbench to simulate and verify the designs of the submodules. You must reuse the shifter unit designed in problem 1.

What to submit:

```
hw2 ->
    hw2_1 ->
        ...........
    hw2_2 ->
        All Verilog files
```

```
        All Verilog rules checking output files
```

Submit all the files needed to compile the program and run the testbenches.

If you have designed any layout, diagrams, or personal testbenches feel free to add them in your submission, but note that they are not compulsory and you will not be graded on them. Also, make sure that the provided testbench is not interrupted due to any of the additional files.

Final Submission:

- Tar the `hw2` folder which will contain 2 subdirectories, i.e. `hw2_1 and hw2_2.`
- Upload the `hw2.tar` file on canvas to submit the assignment.