Home Work 9. Immediate Extension Unit

MIPS Processor is a 32-bit processor, which means that its components, like ALU, operate on 32-bit operands, and the addresses to the next instruction are specified in 32 bits as well.

There are cases when data or addresses are written in less than 32 bits. For example: ADDI $1 $2 1999. The immediate 1999 is written in 16 bits. Since the ALU operates on 32-bit operands, we need to have a unit to extend the bits.

Assignment 8 asks you to write the **Immediate Extension Unit**.

We suggest having the following inputs:

U – if 1 we have zero extension. If 0 we have sign extension.

immediateIN – N bit immediate that need to be extended.

immediateOUT – M bit immediate, the result of extension.

Also you are advised to use concatenation operator. For syntax see Hint #2.

1) **Sign Extension**: The most significant bit is the sign bit. When we do sign extension, we keep the arithmetic value and sign of the original immediate by adding the sign bit at the front of immediate. For example, if we have 4 bit immediate – 1010 (sign bit is 1) and want to sign extend it to 8 bits, we add 4 1s at the front: 11111010. If immediate is 0100 (sign bit is 0), we add 4 0 and get 00000100s. Your Verilog Module Should be able to perform sign extension.
2) **Zero Extension**: When we do zero extension, we add 0s at the front of the immediate. For example, if we have 4 bit immediate – 1010 and want to sign extend it to 8 bits, we add 4 0s at the front: 00001010. Your Verilog Module Should be able to perform zero extension.
3) **Parametrized Module**: Parameters are very useful in Verilog. Parameters can be compared to the arguments of a function in software programming languages. Parameters allow us to change logic of a module during instantiation. For example, if we want to extend 16-bit immediate to 32 bit, we can specify size of original immediate and desired immediate with parameters. If we want to extend 12 bit immediate to 26 bit immediate, we can use the same module, we just need to change parameters during instantiation (See HINT #1). You should write immediate Extension Unit with two parameters: N – number of bits before the extension, M – desired number of bits after the extension.

4) **Test Your design in Testbench:** Write testbench for your design. Generate Waveforms and explain in your reports why do you think your design works correctly. Write Report In Template.

Hint #1:

The bellow you see parametrized adder. Parameters N and M specify number of bits of operand A and B. Default values for N is 9 and for M is 8.

```
module adder #(parameter N = 9, M = 8)(
    input  [N-1 : 0] operandA,  // operand A has N bits
    input  [M-1 : 0] operandB,  // operand B has M bits
    output [M   : 0] sum        // sum has M + 1 bits

);

assign sum = operandA + operandB;

endmodule
```

We should specify values for parameters during instantiation, otherwise, parameters would have default values:

```
module testbench;

reg  [21 : 0] operandA;   // operand A has 22 bits - N = 22
reg  [22 : 0] operandB;   // operand B has 23 bits - M = 23
wire [23 : 0] sum;        // sum       has 24 bits


adder #(.N(22),.M(23)) uut(
    .operandA(operandA),  // operand A has N bits
    .operandB(operandB),  // operand B has M bits
    .sum(sum)             // sum has M + 1 bits

);

initial begin
operandA = 22'd1556789;
operandB = 23'd1678987;

end

endmodule
```

Hint #2:

Concatenation operator – {} - combines two or more operands to form a larger signal.

For example, let's say A = 4'b0011 and B = 4'b1010. If we concatenate B and A and store the result in C, C would become 8'b1010 0011

assign C = {B,A};