Department of Electrical and Computer Engineering

*University of Wisconsin – Madison*

ECE 552 Introduction to Computer Architecture

Homework 3 (Due 02/18)

*For the first two questions below (verilog based), paste your entire verilog code in a single text/doc file and submit the text/doc file on Canvas. Note that these verilog questions need to follow the verilog rules specified in the 552 project/verilog rules document that can be found on Canvas.*

**1. Parallel Sub-Word Add**

In this homework, you will design a 16-bit parallel sub-word addition instruction: PSA.

Its operation is described below:

The PSA instruction performs four half-byte signed additions in parallel to realize sub-word parallelism. Specifically, each of the four half bytes (4-bits) will be treated as separate numbers, but stored in whole as a single vector. When PSA is performed, the four numbers will be added separately. To be more specific, let the contents in rs and rt be aaaa\_bbbb\_cccc\_dddd and eeee\_ffff\_gggg\_hhhh, respectively, where a, b, c, d, e, f, g and h are 4-bit signed integers. Then after execution of PSA, the contents of rd will be {(aaaa+eeee), (bbbb+ffff), (cccc+gggg), (dddd+hhhh)}. You will need to check for overflow in each of the additions. If any of the sub-word additions result in an overflow, the PSA error flag should be set.

Submit the verilog code and a testbench. The module template is provided below.

module PSA\_16bit (Sum, Error, A, B);

input [15:0] A, B; // Input data values

output [15:0] Sum; // Sum output

output Error; // To indicate overflows

…...

endmodule

**2. Shifter**

Design a combinational 16-bit shifter to implement the following two shift instructions: SLL (Shift Left Logical) and SRA (Shift Right Arithmetic). The 4-bit shift amount is an unsigned binary number.

You are required to provide a verilog design and testbench. The design interface to use is:

module Shifter (Shift\_Out, Shift\_In, Shift\_Val, Mode);

input [15:0] Shift\_In; // This is the input data to perform shift operation on

input [3:0] Shift\_Val; // Shift amount (used to shift the input data)

input Mode; // To indicate 0=SLL or 1=SRA

output [15:0] Shift\_Out; // Shifted output data

…….

endmodule

*The following question is non-verilog. Submit this as a text doc or a snapshot.*

**3.** **Single-Cycle Processor**

Refer to COD Figure 4.24.

1. Problems in this question refer to a clock cycle in which the processor fetches the following instruction word:
2. **10101100011000100000000000010100**

Assume that data memory is all zeros and that the processor's registers have the following values at the beginning of the cycle in which the above instruction word is fetched:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $1 | $2 | $3 | $4 | $5 | $6 | $8 | $12 | $31 | pc |
| -1 | 2 | 88 | 24 | 10 | -6 | 8 | 2 | -16 | 44 |

1. What are the outputs of the sign-extend and the jump "Shift left 2" unit (near the top of COD Figure 4.24; the simple control and datapath are extended to handle the jump instruction) for this instruction word?

|  |  |
| --- | --- |
| **Sign-extend** | **Shift left 2 unit** |
|  |  |

1. What are the values of the ALU control unit's inputs for this instruction?

|  |  |
| --- | --- |
| **ALUOp[1:0]** | **Instruction[5:0]** |
|  |  |

1. What is the new PC address after this instruction is executed? State the path through which this value is determined.

|  |  |
| --- | --- |
| **New PC** | **Path** |
|  |  |

1. For each Mux, show the values of its data output during the execution of this instruction (given the current values of registers).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **RegDst Mux** | **ALUSrc Mux** | **MemToReg Mux** | **Branch Mux** | **Jump Mux** |
|  |  |  |  |  |

1. For the ALU and the two add units, what are their data input values?

|  |  |  |
| --- | --- | --- |
| **ALU** | **Add (PC + 4)** | **Add (Branch)** |
|  |  |  |

1. What are the values of all inputs to the Register File?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Read Register 1** | **Read Register 2** | **Write Register** | **Write Data** | **RegWrite** |
|  |  |  |  |  |