Instructions:
1. The assignment is split into 2 sections: Section 1 submission should be hardcopy in-class and Section 2 should be submitted in class (hardcopy)
2. Only Verilog assignment to be done in teams or individually – Part (a) of the assignment to be done individually.
3. Each Verilog assignment should be accompanied by waveforms (output) and verilog (code).

Section 1: HARDCOPY SUBMISSION (50 Points) – 4a

Problem 1 (25 Points):
We wish to implement the instruction sw (store word) in the attached figure.

Problem 2 (25 Points):
We wish to add the instruction jr (jump register) to the single cycle-datapath described in class. Add any necessary datapaths and control signals to the single-cycle datapath and show the additions in the attached figure.

Problem 3 (25 Points):
This question is similar to Problem 1, except that we wish to add the instruction sll (shift left logical) and show the additions in the attached figure.

Problem 4 (BONUS, 25 Points):
MIPS choose to simplify the structure of its instructions. The way we implement complex instructions through the use of MIPS instructions is to decompose such complex instructions into multiple simpler MIPS ones. Show how MIPS can implement the instruction swap $rs, $rt which swaps the contents of registers $rs and $rt. Consider the case in which there is an available register that may be destroyed as well as the case in which no such register exists.

If the implementation of this instruction in hardware will increase the clock period of a single-cycle instruction implementation by 10%, what percentage of swap operations in the instruction mix would recommend implementing it in hardware?
Section 2: VERILOG SUBMISSION (100 Points) – 4b
(a) (30 Points) Design a 16-bit ALU from the 4-bit ALU designed above. The 16-bit ALU is as shown below. Submit only the verilog code. Convince yourself that your model works correctly.

The verilog model must have the following format.

```
module ALU16Bit(a, b, cin, less, op, result, cout, set, zero, g, p, overflow);
input [15:0] a, b;
input cin, less;
output [15:0] result;
output cout, set, zero, g, p, overflow;
 // set is the result of the most-significant
 // ADDER unit. zero is 1 if the result is 0x0000.
 // Otherwise, it is 0.
```
(b) **(30 Points)** Design a 32-bit ALU as shown below. This is the 32-bit ALU model that you will use in your processor design finally.

The verilog model must have the following format.

```verilog
module ALU32Bit(a, b, op, result, set, zero, overflow);
input [31:0] a, b;
output [31:0] result;
output set; // set is the result of the most-significant ADDER unit.
output zero; // zero is 1 if the result is 0x0000. Otherwise, it is 0.
output overflow; // Overflow is 1 if the output is 0x00000000.
```

Submit the verilog code for the 32-bit ALU. Use test inputs to show that your ALU is performing as required. Explain the waveform in your final submission.

(c) **(20 Points)** Design a 16 to 32-bit sign extension unit. Use "switch" or "if-then-else" statement. (You can use "assign" statements within always block.)

The module must have the following format.

```verilog
module SignExtension(a, result);
input [15:0] a; // 16-bit input
output [31:0] result; // 32-bit output
```
(d) (20 Points) Design a 2-input 32-bit multiplexor and a 2-input 5-bit multiplexor. Use "switch" or "if-then-else" statement. Submit your verilog code and explain the waveform used for testing. (You can use "assign" statements within always block.)

The module must have the following format.

```verilog
module Mux32Bit2To1(a, b, op, result);
input [31:0] a, b;  // 32-bit inputs
input op;  // one-bit selection input
output [31:0] result;  // 32-bit output

module Mux5Bit2To1(a, b, op, result);
input [4:0] a, b;  // 5-bit inputs
input op;  // one-bit selection input
output [4:0] result;  // 5-bit output
```
