Problem Set # 7 (Assigned 6 March, Due 16 March)

In this assignment, you are to implement the datapath, to the register level, and a Moore Machine controller for a very simple processor. Your machine has separate instruction and data memories (Harvard architecture), each of which is 256 by 8 bits. The machine is a zero-address expression stack architecture, that is, most instructions reference the top elements of a stack organized in the data memory. The stack grows from data memory location 0 towards the top of the data memory. Program variables are allocated from the top of the data memory downwards. It is up to the compiler or the assembly language programmer to make sure that the expression stack and the variable memory don’t overlap. No special hardware support is provided. Instructions are executed from instruction memory, starting from location 0 on reset.

Instruction Set

Your instruction set includes the following instructions (we do not specify every possible instruction here, just a representative subset: PUSH/POP, ADD/SUB, Conditional Branches, and Jump Subroutine/Return from Subroutine). Note that TOS stands for “Top of Stack”, a special processor register that holds the address of the current top of the stack in the data memory. On reset, it is initialized to 255, just as the PC is initialized to 0.

PUSH <addr>  
TOS ← TOS + 1; Data Memory[TOS] ← Data Memory[addr];

POP <addr>  
Data Memory[addr] ← Data Memory[TOS]; TOS ← TOS – 1;

ADD  
Data Memory[TOS – 1] ← Data Memory[TOS] + Data Memory[TOS – 1];  
TOS ← TOS – 1;

SUB  
Data Memory[TOS – 1] ← Data Memory[TOS] - Data Memory[TOS – 1];  
TOS ← TOS – 1;

BRZ <addr>  
IF Data Memory[TOS] = 0 THEN PC ← addr;

BRN <addr>  
IF Data Memory[TOS]<?> = 1 THEN PC ← addr;

JSR <addr>  
TOS ← TOS + 1; Data Memory[TOS] ← PC + 1; PC ← addr;

SRR  
PC ← Data Memory[TOS]; TOS ← TOS – 1;

These instructions are encoded in 8 bit parcels as follows. The first parcel is the opcode. PUSH is encoded as 10000000₂, POP as 01000000₂, and so on to SRR as 00000001₂. For those instructions with an address field, like PUSH/POP and the branch instructions above, a second 8 bit parcel holds the target address. PUSH/POP refers to the data memory while BRZ/BRN and JSR refer to the instruction memory.

Memory Interface

The memory interface is kept simple. It is synchronous, running on the same clock as the processor. Assert Read and an address following the positive clock edge and data will be available for latching on the following positive edge. Assert Write and an address following the positive clock edge and the data will be written at the following positive edge.
What You Are To Do

1. Given the partial instruction set specification given above, design a datapath to the level of registers, arithmetic units (adders/subtractors), and busses to support the instruction set. Identify the control signals associated with each element, such as Load, Multiplexer/Select, Arithmetic Operations, Tri-state enables, and so on.

2. Write the controller Verilog for a “high level” register-transfer description of the instruction set subset described above. Your Verilog should describe the high level processing of instruction fetch, instruction decode, and instruction execution. Treat instruction and data memory as nothing more than arrays of 256 8-bit words.

***Verilog coming soon***

3. Show the Moore Machine controller state diagram for Instruction Fetch, Instruction Decode, and Instruction Execution for the portion of the instruction set specified above. For the purposes of simplifying the decode step, assume that there are no other instructions in the instruction set. For each state, show the control signals that are to be asserted in that state. Remember to include an initial state, entered when an external Reset signal is asserted, that sets the PC to 0, TOS to 255 (11111111\_2), and initializes any other registers to their initial values.
4. Write the controller Verilog that corresponds to the state diagram of your solution to Problem 3.

```
module InstructionDecoderFSM(instructionMemOut, StoredData, PCMuxSel, DataMuxSel, AddressMuxSel, TOSMuxSel, PCEn, StoredDataEn, TOSEn, PCRst, TOSRst, DataRead, DataWrite, Clock, Reset);

input [7:0] instructionMemOut, StoredData;
input Clock, Reset;
output [1:0] PCMuxSel, DataMuxSel, AddressMuxSel;
output TOSMuxSel, PCEn, StoredDataEn, TOSEn, DataRead, DataWrite;
wire [7:0] instruction;

Register instructionReg(.D(instructionMemOut), .Q(instruction), .En(CS == InstructionFetch), .Clock(Clock), .Reset(Reset));

parameter S_Init = 3'b000,
                S_InstructionFetch = 3'b001,
                S_InstructionDecode = 3'b010,
                S_InstructionExecute = 3'b011;

parameter PUSH = 8'b10000000,
                POP  = 8'b01000000,
                ADD  = 8'b00100000,
                SUB  = 8'b00010000,
                BRZ  = 8'b00001000,
                BRN  = 8'b00000100,
                JSR  = 8'b00000010,
                SRR  = 8'b00000001;

reg [2:0] CS, NS;
always @ (posedge Clock) begin
  if (Reset) CS <= S_Init;
  else CS <= NS;
end

always @ (instructionMemOut or CS) begin
  //default values
```
case (CS)
    S_Init: begin
        PCRst = 1;
        TOSRst = 1;
        NS = S_InstructionFetch;
    end
S_InstructionFetch: begin
    //get next value (in case you need the address)
    PCEn = 1;
    PCMuxSel = 0;
    //get DataMem[TOS]
    DataRead = 1;
    AddressMuxSel = 0;
    NS = S_InstructionDecode;
end
S_InstructionDecode: begin
    PCEn = 0;
    NS = S_InstructionExecute;
    case (Instruction)
        PUSH: begin
            //getDataMem[address]
            DataRead = 1;
            AddressMuxSel = 1;
        end
        POP : begin
            //write DataMem[address] with value stored from DataMem[TOS]
            DataWrite = 1;
            AddressMuxSel = 1;
            StoredDataEn = 1;
            //decrement TOS
            TOSEn = 1;
            TOSSel = 1;
            // Don't need another state, go straight back to InstructionFetch
            NS = S_InstructionFetch;
        end
        ADD : begin
            //Read DataMem[TOS -1]
            DataRead = 1;
            AddressMuxSel = 2;
            StoredDataEn = 1;
        end
        SUB : begin
            //Read DataMem[TOS -1]
            DataRead = 1;
            AddressMuxSel = 2;
            StoredDataEn = 1;
        end
        BRZ : begin
            //check if ReadDataMem[TOS] == 0, if so use the output of the instrMem for the PC
            StoredDataEn = 1;
            PCEn = 1;
            if (StoredData == 0)
                PCSel = 1;
            else
                PCSel = 0;
        end
    end
end
// Don't need another state, go straight back to InstructionFetch
NS = S_InstructionFetch;
end
BRN : begin
//check if ReadDataMem[TOS] != 0, if so use the output of the instrMem for the PC
  StoredDataEn = 1;
  PCEn = 1;
  if (StoredData[7] == 1)
    PCSel = 1;
  else
    PCSel = 0;

// Don't need another state, go straight back to InstructionFetch
NS = S_InstructionFetch;
end
JSR : begin
//write PC to DataMem[TOS]
  DataWrite = 1;
  AddressMuxSel = 0;
  DataMuxSel = 3;
end
SRR : begin
//put DataMem[TOS] in the PC
  StoredDataEn = 1;
  PCEn = 1;
  PCSel = 2;
  //decrement TOS
  TOSEn = 1;
  TOSSel = 1;

// Don't need another state, go straight back to InstructionFetch
NS = S_InstructionFetch;
end
default: NS = S_InstructionFetch; //default, shouldn't get here
endcase
S_InstructionExecute: begin
  PCEn = 0;
  NS = S_InstructionExecute;
  case (Instruction)
    PUSH: begin
      //write DataMem[TOS] with value stored from DataMem[address]
      DataWrite = 1;
      AddressMuxSel = 2;
      DataMuxSel = 2;
      StoredDataEn = 1;

      //increment TOS
      TOSEn = 1;
      TOSSel = 0;

    end
    POP : begin
      //do nothing, shouldn't get here
    end
    ADD : begin
      //write the value stored in the storedDataReg (DataMem[TOS]) + value on dataLine (DataMem[TOS-1]) to DataMem[TOS-1]
      DataWrite = 1;
      AddressMuxSel = 2;
      DataMuxSel = 1;

      //decrement TOS
      TOSEn = 1;
      TOSSel = 1;
    end
    SUB : begin
      //write the value stored in the storedDataReg (DataMem[TOS]) - value on dataLine (DataMem[TOS-1]) to DataMem[TOS-1]
      DataWrite = 1;
    end
  endcase
AddressMuxSel = 2;
DataMuxSel = 1;

//decrement TOS
TOSEn = 1;
TOSSel = 1;
end
BRZ : begin
  //do nothing, shouldn't get here
end
BRN : begin
  //do nothing, shouldn't get here
end
JSR : begin
  //increment TOS
  TOSEn = 1;
  TOSSel = 0;
  //put address in PC
  PCEn = 1;
  if (StoredData != 0)
    PCSel = 1;
end
SRR : begin
  //do nothing, shouldn't get here
end
default: NS = S/InstructionFetch; //default, shouldn't get here
endcase
default: begin
  NS = S/Init;
end
endcase
default
endmodule

5. Go back to working on your project!