CS224
4
1
Yağız Yaşar
21902951

Part 1)

a)

```
addi $2,$0,5
addi $3,$0,12
addi $7,$3,-9
or   $4,$7,$2
and  $5,$3,$4
add  $5,$5,$4
beq  $5,$7,0x80001044
slt  $4,$3,$4
beq  $4,$0,0x80001028
addi $5,$0,0
slt  $4,$7,$2
add  $7,$4,$5
sub  $7,$7,$2
sw   $7,68($3)
lw   $2,80($0)
j    0x80000040
add  $12,$0,$31
jal  0x80000048
sw   $2,84($0)
srl  $18,$3,0x1
jr   $31
```

d)

```
`timescale 1ns / 1ps
// Written by David_Harris@hmc.edu

// Top level system including MIPS and memories
module new_top(input   logic clk, reset,
                output  logic  memwrite,
        output logic alusrca, alusrcb, regwrite,
             output logic [1:0] jump,
                    output logic branch,
             output logic [1:0]  memtoreg, aluop, regdst,
        output logic [31:0] pc, instr, readdata, writedata, dataadr,
        output logic pcsrc, zero
                );


            logic [2:0]  alucontrol;
    mips mips (clk, reset, pc, instr, memwrite, dataadr, writedata,
    readdata, pcsrc, zero, alusrca, alusrcb, regwrite, jump, branch, memtoreg, aluop, regdst, alucontrol);
```

```
    imem imem (pc[7:2], instr);
    dmem dmem (clk, memwrite, dataadr, writedata, readdata);

endmodule

// External data memory used by MIPS single-cycle processor

module dmem (input  logic        clk, we,
             input  logic[31:0]  a, wd,
             output logic[31:0]  rd);

   logic  [31:0] RAM[63:0];

   assign rd = RAM[a[31:2]];    // word-aligned  read (for lw)

   always_ff @(posedge clk)
     if (we)
       RAM[a[31:2]] <= wd;      // word-aligned write (for sw)

endmodule
// External instruction memory used by MIPS single-cycle
// processor. It models instruction memory as a stored-program
// ROM, with address as input, and instruction as output

module imem ( input logic [5:0] addr, output logic [31:0] instr);

// imem is modeled as a lookup table, a stored-program byte-addressable ROM
             always_comb
               case ({addr,2'b00})                      // word-aligned fetch
//                       address             instruction
//                       -------             -----------
             8'h00: instr = 32'h20020005;      // disassemble, by hand
                       8'h04: instr = 32'h2003000c;      // or with a program,
                       8'h08: instr = 32'h2067fff7;      // to find out what
                       8'h0c: instr = 32'h00e22025;      // this program does!
                       8'h10: instr = 32'h00642824;
                       8'h14: instr = 32'h00a42820;
                       8'h18: instr = 32'h10a7000a;
                       8'h1c: instr = 32'h0064202a;
                       8'h20: instr = 32'h10800001;
                       8'h24: instr = 32'h20050000;
                       8'h28: instr = 32'h00e2202a;
                       8'h2c: instr = 32'h00853820;
                       8'h30: instr = 32'h00e23822;
                       8'h34: instr = 32'hac670044;
                       8'h38: instr = 32'h8c020050;
                       8'h3c: instr = 32'h088000010;
             8'h40: instr = 32'h001f6020;
             8'h44: instr = 32'h0c000012;
                       8'h48: instr = 32'hac020054;
             8'h4c: instr = 32'h00039042;
             8'h50: instr = 32'h03E00008;
                default:  instr = {32{1'bx}};   // unknown address
               endcase
```

```
endmodule

// single-cycle MIPS processor, with controller and datapath
module mips (input  logic        clk, reset,
             output logic[31:0]  pc,
             input  logic[31:0]  instr,
             output logic        memwrite,
             output logic[31:0]  aluout, writedata,
             input  logic[31:0]  readdata,
             output logic  pcsrc, zero, alusrca, alusrcb, regwrite,
                    output logic [1:0] jump,
                    output logic branch,
             output logic [1:0]  memtoreg, aluop, regdst,
             output logic [2:0]  alucontrol
              );
  controller c (instr[31:26], instr[5:0], zero, memtoreg, aluop, regdst, memwrite, pcsrc, alusrca, alusrcb,
regwrite, jump, branch, alucontrol);

  datapath dp (clk, reset, pcsrc, alusrca, alusrcb, regwrite, jump, memtoreg, regdst,
                    alucontrol, zero, pc, instr, aluout, writedata, readdata);

endmodule

module controller(input  logic[5:0] op, funct,
                  input  logic zero,
                  output logic[1:0]  memtoreg, aluop, regdst,
                  output logic memwrite,
                  output logic pcsrc, alusrca, alusrcb,
                  output logic regwrite,
                  output logic [1:0]jump, branch,
                  output logic[2:0] alucontrol);

   maindec md (op, memtoreg, memwrite, branch, alusrcb, regwrite, jump, aluop, regdst);

   aludec  ad (funct, aluop, alucontrol);

   assign pcsrc = (branch & zero);

   assign alusrca = (funct == 6'b000010); // If it is shift right logic

endmodule

// It will change
module maindec (input logic[5:0] op,
                output logic[1:0] memtoreg,
                       output logic memwrite, branch,
                       output logic alusrcb, regwrite,
               output logic[1:0] jump, aluop, regdst);
  logic [9:0] controls;

  assign {regwrite, regdst, alusrcb, branch, memwrite,
             memtoreg, aluop} = controls;

  always_comb
```

```
    case(op)
      6'b000000: begin controls <= 10'b1_01_0_0_0_00_10; jump <= 2'b00; end // R-type
      6'b100011: begin controls <= 10'b1_00_1_0_0_01_00; jump <= 2'b00; end // LW
      6'b101011: begin controls <= 10'b0_00_1_0_1_00_00; jump <= 2'b00; end // SW
      6'b000100: begin controls <= 10'b0_00_0_1_0_00_01; jump <= 2'b00; end // BEQ
      6'b001000: begin controls <= 10'b1_00_1_0_0_00_00; jump <= 2'b00; end // ADDI
      6'b000010: begin controls <= 10'b0_00_0_0_0_00_00; jump <= 2'b01; end // J
      6'b000011: begin controls <= 10'b1_10_0_0_0_10_00; jump <= 2'b01; end // JAL
          6'b000000: begin controls <= 10'b1_01_0_0_0_00_10; jump <= 2'b10; end // JR
      default:   begin controls <= 10'bx_xx_x_x_x_xx_xx; jump <= 2'bxx; end // illegal op
    endcase
endmodule

module aludec (input    logic[5:0] funct,
               input    logic[1:0] aluop,
               output   logic[2:0] alucontrol);
  always_comb
    case(aluop)
      2'b00: alucontrol  = 3'b010;  // add  (for lw/sw/addi)
      2'b01: alucontrol  = 3'b110;  // sub   (for beq)
      default: case(funct)          // R-TYPE instructions
          6'b100000: alucontrol  = 3'b010; // ADD
          6'b100010: alucontrol  = 3'b110; // SUB
          6'b100100: alucontrol  = 3'b000; // AND
          6'b100101: alucontrol  = 3'b001; // OR
          6'b101010: alucontrol  = 3'b111; // SLT
          6'b000010: alucontrol  = 3'b011; // SRL
          default:   alucontrol  = 3'bxxx; // ???
        endcase
    endcase
endmodule

module datapath (input  logic clk, reset, pcsrc, alusrca, alusrcb,
                 input  logic regwrite,
                 input  logic[1:0] jump, memtoreg, regdst,
                            input  logic[2:0]  alucontrol,
                 output logic zero,
                            output logic[31:0] pc,
                        input  logic[31:0] instr,
                 output logic[31:0] aluout, writedata,
                        input  logic[31:0] readdata);

  logic [4:0]  writereg;
  logic [31:0] pcnext, pcnextbr, pcplus4, pcbranch;
  logic [31:0] signimm, signimmsh, rd1, srca, srcb, result;

  // next PC logic
  flopr #(32) pcreg(clk, reset, pcnext, pc);
  adder       pcadd1(pc, 32'b100, pcplus4);
  sl2         immsh(signimm, signimmsh);
  adder       pcadd2(pcplus4, signimmsh, pcbranch);
  mux2 #(32)  pcbrmux(pcplus4, pcbranch, pcsrc,
                      pcnextbr);
  mux4 #(32)  pcmux(pcnextbr, {pcplus4[31:28],
```

```
                instr[25:0], 2'b00}, jump, readdata, 32'b0, pcnext);

  // register file logic
  regfile     rf (clk, regwrite, instr[25:21], instr[20:16], writereg,
                  result, rd1, writedata);

 mux4 #(5)    wrmux (instr[20:16], instr[15:11], 5'h0001F, 0, regdst, writereg);
  signext         se (instr[15:0], signimm);

  mux4 #(32)  resmux (aluout, readdata, pcplus4, 32'b0, memtoreg, result);

  // ALU logic
  mux2 #(32)  srcamux (rd1, {27'b0, instr[10:6]}, alusrca, srca);
  mux2 #(32)  srcbmux (writedata, signimm, alusrcb, srcb);
  alu         alu (srca, srcb, alucontrol, aluout, zero);

endmodule


module regfile (input    logic clk, we3,
                input    logic[4:0]  ra1, ra2, wa3,
                input    logic[31:0] wd3,
                output   logic[31:0] rd1, rd2);

 logic [31:0] rf [31:0];

 // three ported register file: read two ports combinationally
 // write third port on rising edge of clock. Register0 hardwired to 0.

 always_ff @(posedge clk) // !!!! I added posedge
   if (we3)
      rf [wa3] <= wd3;

 assign rd1 = (ra1 != 0) ? rf [ra1] : 0;
 assign rd2 = (ra2 != 0) ? rf[ ra2] : 0;

endmodule


module alu(input  logic [31:0] a, b,
           input  logic [2:0]  alucont,
           output logic [31:0] result,
           output logic zero);

 always_comb
 case(alucont)
   3'b010:
      begin
      result = a + b;
      zero = 0;
      end
   3'b110:
      begin
      result = a - b;
```

```
        if(result == 0)
            zero = 1;
        else
            zero = 0;
        end
    3'b000:
        begin
        result = a & b;
        zero = 0;
        end
    3'b001:
        begin
        result = a | b;
        zero = 0;
        end
    3'b111:
        begin
        if(a < b)
            result = 1;
        else
            result = 0;
        zero = 0;
        end
    3'b011:
            begin
          result = b >> a;
          zero = 0;
        end
  endcase
endmodule


module adder (input  logic[31:0] a, b,
              output logic[31:0] y);

     assign y = a + b;
endmodule

module sl2 (input  logic[31:0] a,
            output logic[31:0] y);

     assign y = {a[29:0], 2'b00}; // shifts left by 2
endmodule

module sl16 (input  logic[31:0] a,
            output logic[31:0] y);

     assign y = {a[15:0], 16'b0}; // shifts left by 16
endmodule

module signext (input  logic[15:0] a,
                output logic[31:0] y);

  assign y = {{16{a[15]}}, a};    // sign-extends 16-bit a
```

```
endmodule

// parameterized register
module flopr #(parameter WIDTH = 8)
              (input logic clk, reset,
                    input logic[WIDTH-1:0] d,
               output logic[WIDTH-1:0] q);

  always_ff@(posedge clk, posedge reset)
    if (reset) q <= 0;
    else       q <= d;
endmodule


// paramaterized 2-to-1 MUX
module mux2 #(parameter WIDTH = 8)
             (input  logic[WIDTH-1:0] d0, d1,
              input  logic s,
              output logic[WIDTH-1:0] y);

   assign y = s ? d1 : d0;
endmodule

// paramaterized 4-to-1 MUX
module mux4 #(parameter WIDTH = 8)
             (input  logic[WIDTH-1:0] d0, d1, d2, d3,
              input  logic[1:0] s,
              output logic[WIDTH-1:0] y);

   assign y = s[1] ? ( s[0] ? d3 : d2 ) : (s[0] ? d1 : d0);
endmodule


e)

module mips_tb();
  logic clk, reset;
  reg pcsrc, zero, alusrcb, regwrite, jump, branch;
  reg memwrite;
  reg [1:0] memtoreg, aluop, regdst;
  reg [31:0] pc, instr, readdata, writedata, dataadr;

  new_top mips_lite_extended(clk, reset, memwrite, alusrca, alusrcb,
                                  regwrite, jump, branch, memtoreg,
                        aluop, regdst, pc, instr, readdata,
                                  writedata, dataadr, pcsrc, zero);

   initial begin
    $dumpfile("dump.vcd");
     $dumpvars(1);
   end

  initial begin
    clk = 0;
```

```
    reset = 1; #5;
    reset = 0; #5;
  end
  initial begin
    for (int i = 0; i < 100; i++)
      begin
        #10; clk = ~clk;
      end
  end
endmodule
```

f)

i) RF[RT]
ii) Since it is not R-Type, it would be XXX,
iii) SW or LW is needed
iv) ALU Result
v) It is needed to be added with RA but RA is not defined

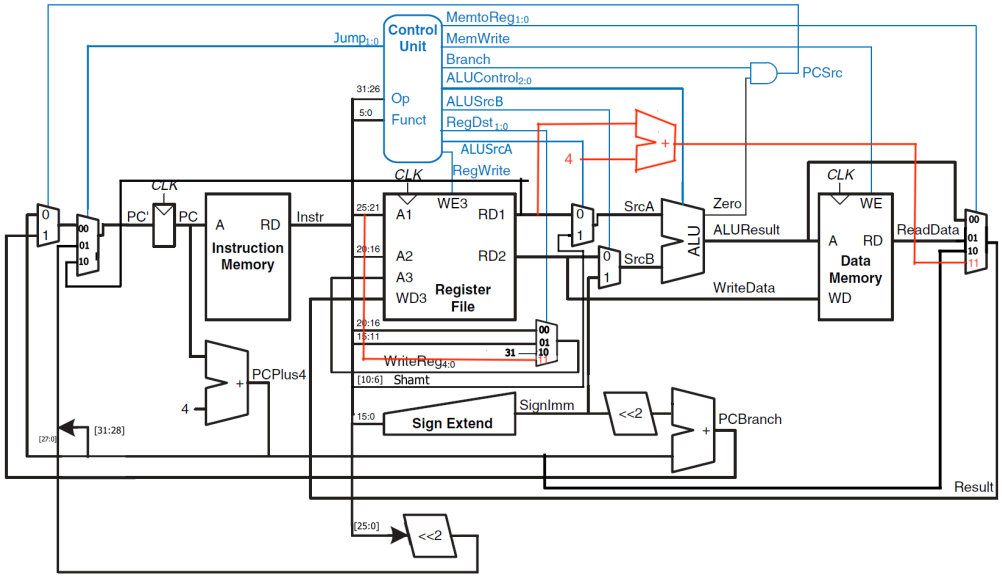
g)

i) Yes, funct code is needed to be changed to add srlv instruction.
ii) maindec module by adding a new opcode case.

Part 2)

a) IM[PC], DM[RF[rs] + signExtendImm]] <- RF[rt], RF[rs] <- RF[rs] + 4, PC <- PC + 4

b)



c)

Instruction	Opcode	RegWrite	RegDst	ALUSrcA	ALUSrcB	Branch	MemWrite	MemToReg	ALUOp	Jump
R-Type	000000	1	01	0	0	0	0	00	10	00
srl	000000	1	01	1	0	0	0	00	10	00
lw	100011	1	00	0	1	0	0	01	00	00
sw	101011	0	X	0	1	0	1	XX	00	00
beq	000100	0	X	0	0	1	0	01	01	00
addi	001000	1	00	0	1	0	0	00	00	00
j	000010	0	X	X	X	X	0	XX	XX	01
jal	000011	1	10	X	X	X	0	10	XX	01
sw+	101010	1	11	0	1	0	1	11	00	00
jr	000000	1	01	0	0	0	0	00	10	10