CS 224

Lab04

Preliminary Design Report

Alper Kandemir

21703062

CS224-04

B)

Address:	Hex:	Instruction
8'h00	32'h20020005	addi $v0, $zero, 5
8'h04	32'h2003000c	addi $v1, $zero, 12
8'h08	32'h2067fff7	addi $a3, $v1, 65527
8'h0c	32'h00e22025	or $a0, $a3, $v0
8'h10	32'h00642824	and $a1, $v1, $a0
8'h14	32'h00a42820	add $a1, $a1, $a0
8'h18	32'h10a7000a	beq $a1, $a3, 10
8'h1c	32'h0064202a	slt $a0, $v1, $a0
8'h20	32'h10800001	beq $a0, $zero, 1
8'h24	32'h20050000	addi $a1, $zero, 0
8'h28	32'h00e2202a	slt $a0, $a3, $v0
8'h2c	32'h00853820	add $a3, $a0, $a1
8'h30	32'h00e23822	sub $a3, $a3, $v0
8'h34	32'hac670044	sw $a3, 68($v1)
8'h38	32'h8c020050	lw $v0, 80 ($zero)
8'h3c	32'h08000011	j 0x0000011
8'h40	32'h20020001	addi $v0, $zero, 1
8'h44	32'hac020054	sw $v0, 84($zero)
8'h48	32'h08000012	j 0x0000012

C)

jalm: : this I-type instruction is a jump to the address stored in the memory location indicated in the standard way. But it also puts the return address into the register specified.

Example: jalm $t5, 40($s3)

IM[PC]

RT[rt]<--- PC+4

PC<--- RF[rs] + SignExt(imm)

push: this I-type instruction does what you would expect—push a register value onto stack. Example: push $a3 {Note: the assembler automatically puts 29 in the rs field, and 0 into the immediate field of the machine code instruction.}

IM [PC]

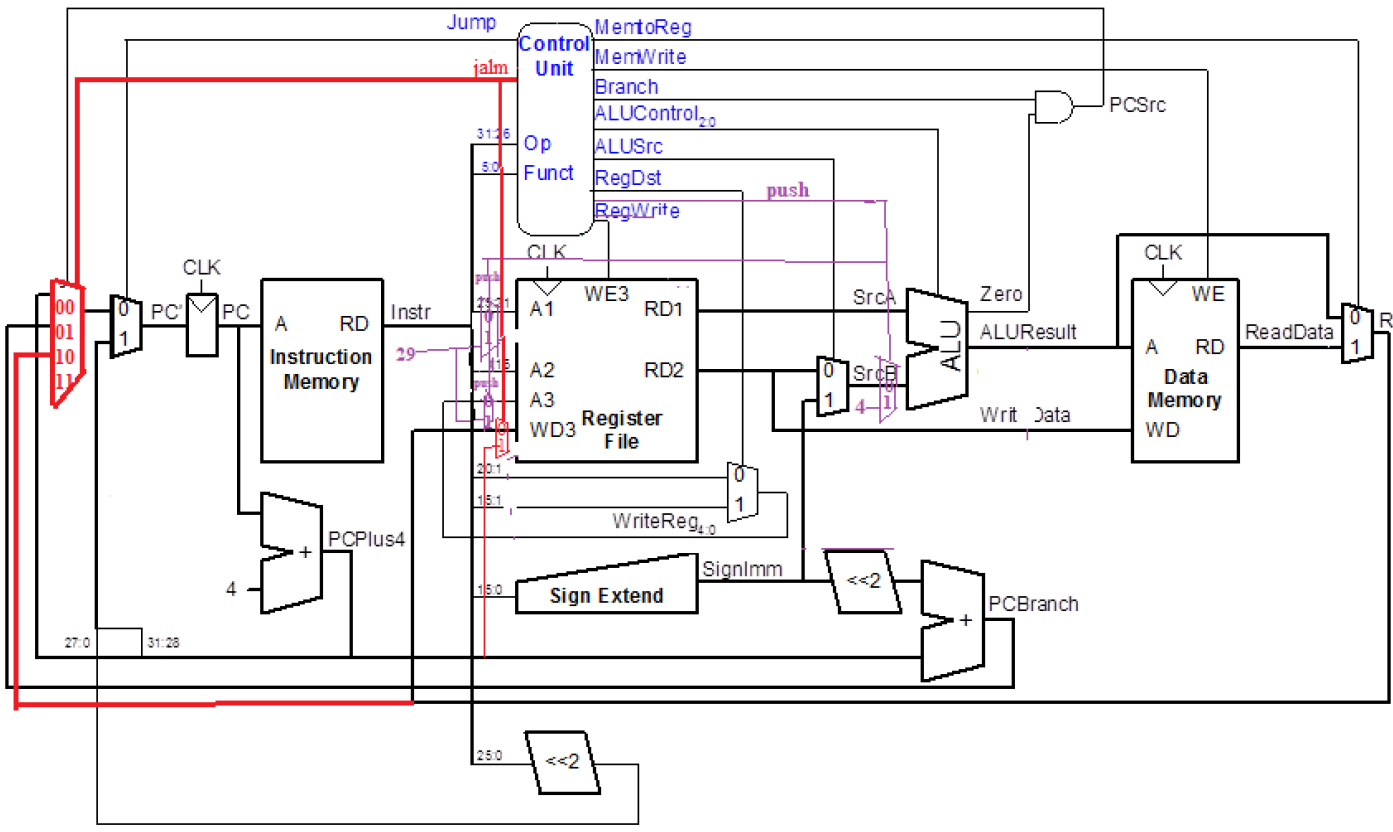
DM[RF[rs] – 4] <--- RF [rt]

RF [rs] <--- RF [rs] - 4

PC <--- PC + 4

D)

Added one mux4 and one mux2 to datapath for jalm instruction. Added three mux2 to datapath for push. Two of them are for choosing a1 and a3. The last one is for choosing 4 or srcb. Also, Jalm and push signals are added to control unit.



E)

Instrs	Op	RegWr	RegDest	AluSrc	Branch	MemWr	MemToReg	AluOp	jump	jalm	push
R-type	000000	1	1	0	0	0	0	10	0	0	0
lw	100011	1	0	1	0	0	1	00	0	0	0
sw	101011	0	x	1	0	1	x	00	0	0	0
beq	000100	0	x	0	1	0	x	01	0	0	0
j	000010	0	x	x	x	0	x	xx	1	0	0
addi	001000	1	0	1	0	0	0	00	0	0	0
jalm	000011	1	0	1	0	0	1	00	0	1	0
push	011010	1	x	x	0	1	0	01	0	0	1

F)

```
.text
start:
addi $v0, $zero, 5
addi $v1, $zero, 12
addi $a3, $v1, 65527
or $a0, $a3, $v0
and $a1, $v1, $a0
add $a1, $a1, $a0
beq $a1, $a3, flag1
slt $a0, $v1, $a0
beq $a0, $zero, flag2
addi $a1, $zero, 0
slt $a0, $a3, $v0
add $a3, $a0, $a1
sub $a3, $a3, $v0
```

sw $a3, 68 ($v1)

lw $v0, 80 ($zero)

j flag1

addi $v0, $zero, 1

flag1:

sw $v0, 84($zero)

j flag2

addi $v0, $zero, 1

flag2:

push $a3

jalm $t5, 40($s3)

exit2:

li $v0,10

syscall

.data

G)

```
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'h08000011;
8'h40: instr = 32'h20020001;
8'h44: instr = 32'hac020054;
8'h48: instr = 32'h08000012;  // j 48, so it will loop here
8'h4c: instr = 32'h6ba70000; // push a3
8'h50: instr = 32'h0e4d0028; // jalm t5, 40(s3)
     default:  instr = {32{1'bx}};    // unknown address
   endcase
endmodule

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);


  logic        memtoreg, pcsrc, zero, alusrc, regdst, regwrite, jump, jalm, push;
  logic [2:0]  alucontrol;


  controller c (instr[31:26], instr[5:0], zero, memtoreg, memwrite, pcsrc,
                        alusrc, regdst, regwrite, jump, jalm, push, alucontrol);


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


endmodule
```

```
module controller(input  logic[5:0] op, funct,
                  input  logic     zero,
                  output logic     memtoreg, memwrite,
                  output logic     pcsrc, alusrc,
                  output logic     regdst, regwrite,
                  output logic     jump,
   output logic     jalm,push,
                  output logic[2:0] alucontrol);


   logic [1:0] aluop;
   logic       branch;


   maindec md (op, memtoreg, memwrite, branch, alusrc, regdst, regwrite,
jump, jalm, push, aluop);


   aludec  ad (funct, aluop, alucontrol);


   assign pcsrc = branch & zero;

endmodule

module maindec (input logic[5:0] op,
              output logic memtoreg, memwrite, branch,
              output logic alusrc, regdst, regwrite, jump,
output logic jalm,push,
              output logic[1:0] aluop );
   logic [10:0] controls;


   assign {regwrite, regdst, alusrc, branch, memwrite,
               memtoreg,  aluop, jump, jalm, push} = controls;
```

```
  always_comb
    case(op)
      6'b000000: controls <= 11'b11000010000; // R-type
      6'b100011: controls <= 11'b10100100000; // LW
      6'b101011: controls <= 11'b00101000000; // SW
      6'b000100: controls <= 11'b00010001000; // BEQ
      6'b001000: controls <= 11'b10100000000; // ADDI
      6'b000010: controls <= 11'b00000000100; // J
      6'b000011: controls <= 11'b10100100010; // jalm
      6'b011010: controls <= 11'b1xx01001001; // push
      default:   controls <= 11'bxxxxxxxxxxx; // illegal op
    endcase
endmodule



module datapath (input  logic clk, reset, memtoreg, pcsrc, alusrc, regdst,
                 input  logic regwrite, jump,
                input  logic jalm, push,
                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, srca, srcb, result;
  logic [31:0] result2, srcb2,newA1,newA3;
  // 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)  pcmux(pcnextbr, {pcplus4[31:28],
                     instr[25:0], 2'b00}, jump, pcnext);
```

```
  //jalm
  logic[31:0] empty;
  mux4 jalm4Mux(pcplus4,pcbranch,result,empty,{jalm, pcsrc},pcnextbr);
  mux2 jalm2Mux(result, pcplus4,jalm,result2);

// register file logic
   regfile     rf (clk, regwrite, newA1, instr[20:16], writereg,
                   result2, srca, writedata);


   mux2 #(5)    wrmux (instr[20:16], instr[15:11], regdst, newA3);
   mux2 #(32)  resmux (aluout, readdata, memtoreg, result);
   signext         se (instr[15:0], signimm);


  // ALU logic
   mux2 #(32)  srcbmux (writedata, signimm, alusrc, srcb);

//push
   mux2        push4mux(srcb, 32'd4, push, srcb2);
   mux2        pushA1mux(instr[25:21], 5'd29, push, newA1);
   mux2        pushA3mux(newA3, 5'd29, push, writereg);

   alu         alu (srca, srcb2, alucontrol, aluout, zero);

endmodule
```

//module mux4

```
module mux4(input logic[31:0] a, b, c, d,
               input logic [1:0] s,
               output logic out );

assign out = s[1]? (s[0]? c:d):(s[0] ? b : a);
Endmodule

//

module alu(input  logic [31:0] a, b,
           input  logic [2:0]  alucont,
```

```
            output logic [31:0] result,
            output logic zero);
// details of the model need to be
  // filled in by you, the designer !

        always_comb
        case ( alucont)
                3'b000: result = a&b;
                3'b001: result = a|b;
                3'b010: result = a+b;
                3'b110: result = a-b;
                3'b111: result = (a<b) + 32'b0000;
                default: result = a+b;
        endcase
  assign zero = 0;


endmodule
```