Cs224 Lab04

Cs224,
Lab 4
Section 04
Altay İlker Yiğitel
22203024

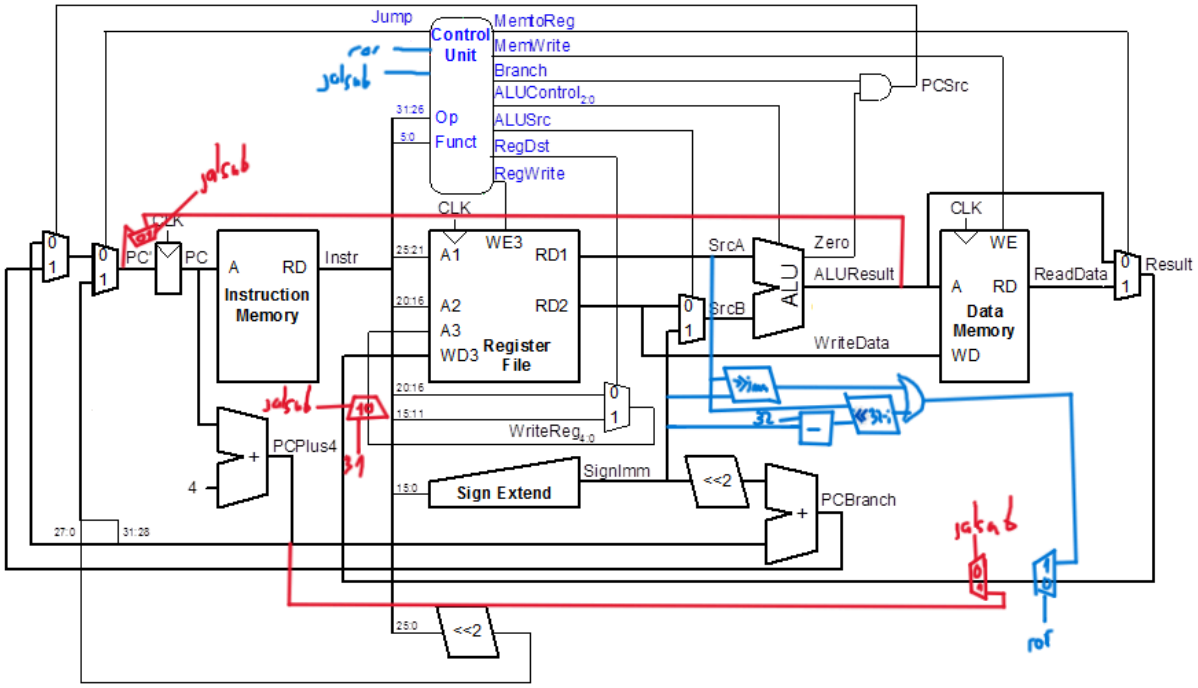
Location	Machine function	Assembly Equivalent
0x0000000000000000	20 02 00 05	addi $v0, $zero, 5
0x0000000000000004	20 03 00 0C	addi $v1, $zero, 0xC
0x0000000000000008	20 67 FF F7	addi $a3, $v1, -9
0x000000000000000c	00 E2 20 25	or $a0, $a3, $v0
0x0000000000000010	00 64 28 24	and $a1, $v1, $a0
0x0000000000000014	00 A4 28 20	add $a1, $a1, $a0
0x0000000000000018	10 A7 00 0A	beq $a1, $a3, 0x44
0x000000000000001c	00 64 20 2A	slt $a0, $v1, $a0
0x0000000000000020	10 80 00 01	beqz $a0, 0x28
0x0000000000000024	20 05 00 00	addi $a1, $zero, 0
0x0000000000000028	00 E2 20 2A	slt $a0, $a3, $v0
0x000000000000002c	00 85 38 20	add $a3, $a0, $a1
0x0000000000000030	00 E2 38 22	sub $a3, $a3, $v0
0x0000000000000034	AC 67 00 44	sw $a3, 0x44($v1)
0x0000000000000038	8C 02 00 50	lw $v0, 0x50($zero)
0x000000000000003c	08 00 00 11	j 0x44
0x0000000000000040	20 02 00 01	addi $v0, $zero, 1
0x0000000000000044	AC 02 00 54	sw $v0, 0x54($zero)
0x0000000000000048	08 00 00 12	j 0x48

jalsub

RTL:
IR ← MEM[PC]
RF[31] ← PC+4
PC ← RF[rs] - RF[rt]

ror

RTL:
IR ← MEM[PC]
RF[rd] ← (RF[rs] >> shamt) | (RF[rs] << (32 - shamt))
PC ← PC+4


Jump
Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
Op
Funct
31:26
5:0
PCSrc
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
WD
Data Memory
ReadData
Result
PCPlus4
4
WriteReg4:0
15:11
SignImm
Sign Extend
15:0
<<2
PCBranch
27:0
31:28
25:0
<<2

İnstruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp	Jump	jalsub	ror
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
addi	001000	1	0	1	0	0	0	00	0	0	0
j	000010	0	X	X	X	0	X	XX	1	0	0
jalsub	000111	1	X	X	0	0	X	01	0	1	0
ror	000110	1	1	1	0	X	X	XX	0	0	1

```
add $t5, $t0, $t1
sub $t6, $t0, $t1
and $t7, $t0, $t1
or $t8, $t0, $t1
slt $t9, $t0, $t1
sw $t0, 0($sp)
lw $s0, 0($sp)

jalsub:

addi $a0, $a0, 0x112345687
addi  $a1, $a1, 0xFFFF000F
jalsub $a0, $a1
addi $a0, $a0, 0x142345586
addi  $a1, $a1, 0xF00C000D
jalsub $a0, $a1
```

```
ror:
addi $a0, $a0, 8'b01011011
ror   $a1,$a0, 5
addi $a0, $a0, 8'b11111111
ror   $a1,$a0, 4
```

```
// Written by David_Harris@hmc.edu

// Top level system including MIPS and memories


module top  (input   logic         clk, reset,
             output  logic[31:0] writedata, dataadr,
             output  logic[31:0] readdata,
             output  logic       memwrite,
             output logic ror, output logic jalsub);

  logic [31:0] instr, pc;
  // instantiate processor and memories
  mips mips (clk, reset, pc, instr, memwrite, dataadr, writedata, readdata,ror,jalsub);
  imem imem (pc[7:0], 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 [7:0] addr, output logic [31:0] instr);

// imem is modeled as a lookup table, a stored-program byte-addressable ROM
          always_comb
```

```
            case (addr)                 // 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
              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 ror, output logic jalsub);

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

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

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

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[2:0] alucontrol,
                output logic     ror,
                output logic     jalsub);

  logic [1:0] aluop;
  logic       branch;

  maindec md (op, memtoreg, memwrite, branch, alusrc, regdst, regwrite,
                  jump, aluop, ror, jalsub);

  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[1:0] aluop , ror, jalsub);
  logic [8:0] controls;

  assign {regwrite, regdst, alusrc, branch, memwrite,
           memtoreg,  aluop, jump, jalsub, ror} = controls;

 always_comb
  case(op)
    6'b000000: controls <= 9'b110000100 00; // R-type
    6'b100011: controls <= 9'b101001000 00; // LW
    6'b101011: controls <= 9'b001010000 00; // SW
    6'b000100: controls <= 9'b000100010 00; // BEQ
    6'b001000: controls <= 9'b101000000 00; // ADDI
    6'b000010: controls <= 9'b000000001 00; // J
    6'b000111: controls  <= 9'b100000010 10; //jalsub
    6'b000110: controls  <= 9'b111000000 01; //ror
    default:   controls <= 9'bxxxxxxxxx; // 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
          default:   alucontrol  = 3'bxxx; // ???
        endcase
    endcase
endmodule

module datapath (input  logic clk, reset, memtoreg, pcsrc, alusrc, regdst,
                 input  logic regwrite, jump,
                        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,
                      output logic ror,
                      output logic jalsub);

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

  // 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);
  ror         rorlog(srca,signimm,rorres,ror);
  mux2 #(32)  pcbrmux(pcplus4, pcbranch, pcsrc,
                      pcnextbr);
  mux2 #(32)  pcmux(pcnextbr, {pcplus4[31:28],
                    instr[25:0], 2'b00}, jump, pcnext);
 mux2 #(32)   pcjalmux(pcnext, aluout, jalsub, pcnext);
// register file logic
   regfile     rf (clk, regwrite, instr[25:21], instr[20:16], writereg,
                   result, srca, writedata);

   mux2 #(5)    wrmux (instr[20:16], instr[15:11], regdst, writereg);
   mux2 #(5)    wramux(writereg,5'd31,jalsub,writereg);
   mux2 #(32)  resmux (aluout, readdata, memtoreg, result);
   mux2 #(32)  lastresmux(result,pcplus4,jalsub,result);
```

```
  mux2 #(32)  lastrorresmux(result,rorres,ror, result);
  signext         se (instr[15:0], signimm);
  jalsub          subj(pc, instr[25:21], instr[20:16],jalsub,5'd31,outlink);


 // ALU logic
  mux2 #(32)  srcbmux (writedata, signimm, alusrc, 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)
   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: result = a + b;
         3'b110: result = a - b;
         3'b000: result = a & b;
         3'b001: result = a | b;
         3'b111: result = (a < b) ? 1 : 0;
         default: result = {32{1'bx}};
      endcase

   assign zero = (result == 0) ? 1'b1 : 1'b0;
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 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

module ror ( input logic [31:0] a,
                    input logic [4:0] shamt,
                    output logic [31:0] b,
                    input logic ror);

assign b= (a >> shamt) | (a << (32 - shamt));
```

```
endmodule



module jalsub ( input logic [31:0] pc,
                input logic[4:0] a,
                input logic[4:0]b,
                input logic jalsub,
                input logic [4:0]ra,
                output logic [31:0] link_address, // address to store in $ra
                );

                assign link_address = pc + 4; // store return address in $ra
                rf[ra]<=link_address;
                assign pc = a-b; // jump to target address

        endmodule
```