CS224

Lab No: 04

Section No: 04

Zeynep Doğa Dellal

22002572

15/11/2023

b)

|  |  |  |
| --- | --- | --- |
| Location | Machine Instruction | Assembly Language Equivalent |
| 0x00 | 0x20020005 | addi $v0 $zero 0x0005 |
| 0x04 | 0x2003000C | addi $v1 $zero 0x000C |
| 0x08 | 0x2067FFF7 | addi $a3 $v1 0xFFF7 |
| 0x0C | 0x00E22025 | or $a0 $a3 $v0 |
| 0x10 | 0x00642824 | and $a1 $v1 $a0 |
| 0x14 | 0x00A42820 | add $a1 $a1 $a0 |
| 0x18 | 0x10A7000A | beq $a1 $a3 0x000A |
| 0x1C | 0x0064202A | slt $a0 $v1 $a0 |
| 0x20 | 0x10800001 | beq $a0 $zero 0x0001 |
| 0x24 | 0x20050000 | addi $a1 $zero 0x0000 |
| 0x28 | 0x00E2202A | slt $a0 $a3 $v0 |
| 0x2C | 0x00853820 | add $a3 $a0 $a1 |
| 0x30 | 0x00E23822 | sub $a3 $a3 $v0 |
| 0x34 | 0xAC670044 | sw $a3 0x0044 $v1 |
| 0x38 | 0x8C020050 | lw $v0 0x0050 $zero |
| 0x3C | 0x08000011 | j 0x0000011 |
| 0x40 | 0x20020001 | addi $v0 $zero 0x0001 |
| 0x44 | 0xAC020054 | sw $v0 0x0054 $zero |
| 0x48 | 0x08000012 | J 0x0000012 |

c)

For jalm

IM[PC]

RF[rt] 🡨 PC + 4

PC🡨 DM[RF[rs] + SignExt(immed)]

For push

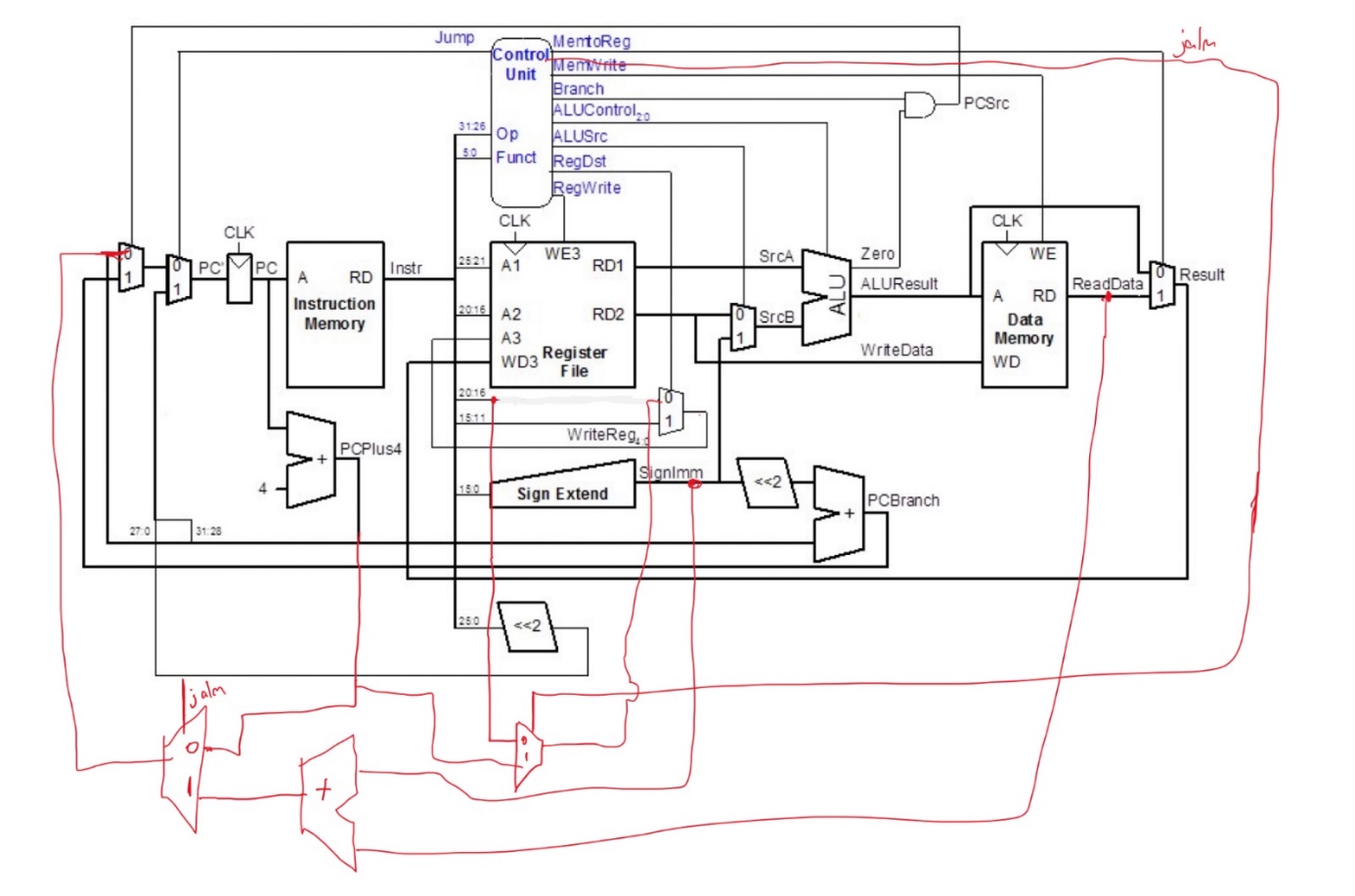
IM[PC]

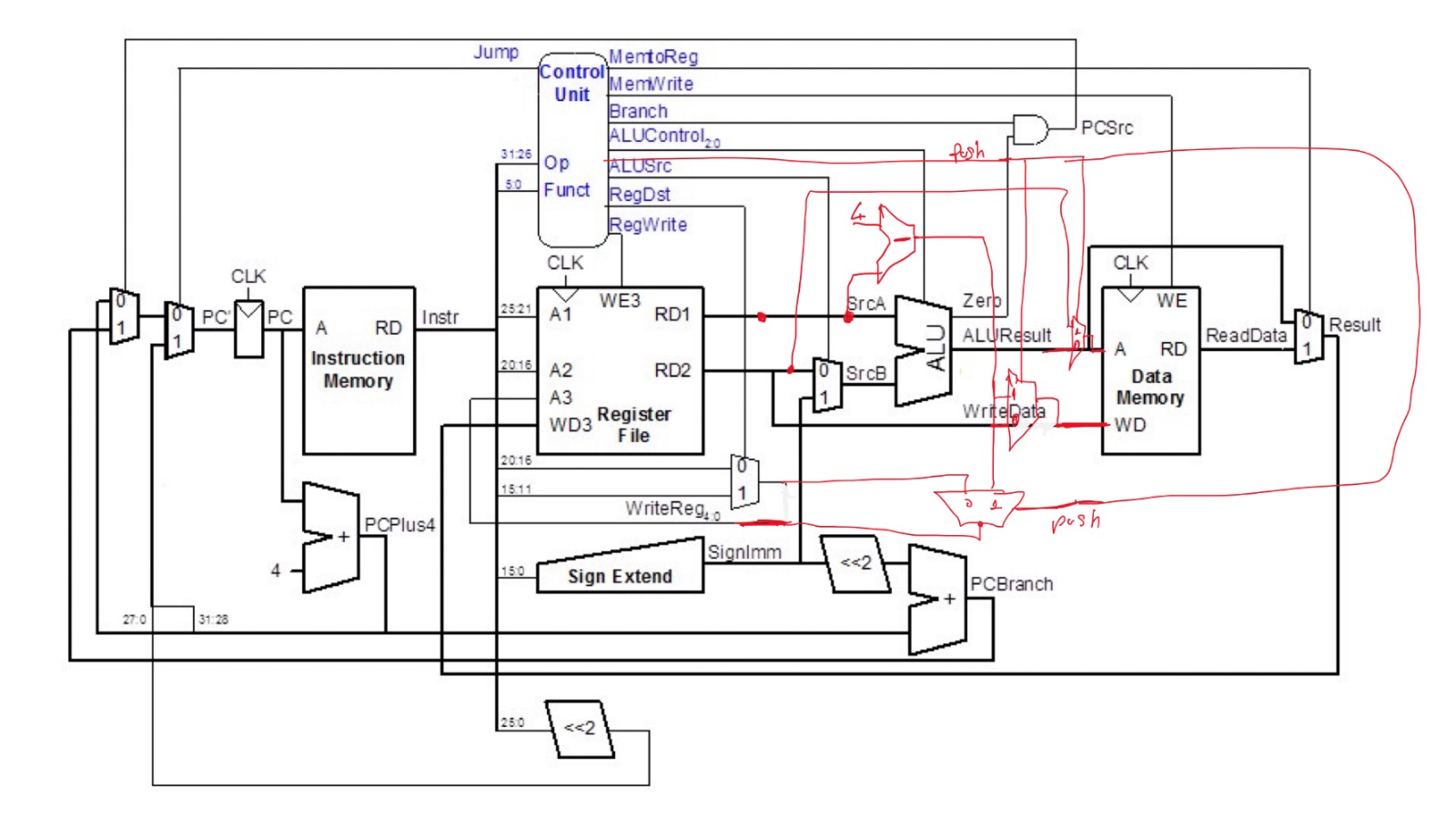
DM[RF[rs] - 4]🡨 RF[rt]

RF[rs] 🡨 RF[rs] – 4

PC 🡨 PC + 4

d)

For jalm

For push

e)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Opcode** | **RegWrite** | **RegDst** | **ALUSrc** | **Branch** | **MemWrite** | **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 |
| addi | 001000 | 1 | 0 | 1 | 0 | 0 | 0 | 00 | 0 | 0 | 0 |
| j | 000010 | 0 | X | X | X | 0 | X | XX | 1 | 0 | 0 |
| jalm | 100101 | 1 | 0 | x | 0 | 0 | 1 | XX | 0 | 1 | 0 |
| push | 000000 | 0 | 0 | x | 0 | 1 | x | 01 | 0 | 0 | 1 |

f)

.data

array: .word 0, 1, 2, 3, 4

.text

main:

# Test ADD

add $t0, $zero, $zero # Initialize $t0 to 0

add $t0, $t0, 5 # $t0 = 0 + 5

# Test OR

or $t1, $t0, $zero # $t1 = $t0 | $zero

# Test AND

and $t2, $t1, $t0 # $t2 = $t1 & $t0

# Test ADDI

addi $t3, $zero, 10 # $t3 = 0 + 10

# Test SLT

slt $t4, $t3, $t0 # $t4 = 1 if $t3 < $t0, otherwise 0

# Test SUB

sub $t5, $t3, $t0 # $t5 = $t3 - $t0

# Test SW and LW

sw $a3 0x0044 $v1

lw $v0 0x0050 $zero

# Test JALM

jalm $ra, 40($s3) # Jump and link, store return address in $ra

# Test PUSH

push $t0 # Push the value in $t0 onto the stack

# Exit program

li $v0, 10

syscall

g)

**1. Add new instructions to the imem module:**

Update the imem to include the machine code for the new instructions (jalm and push)

**2. Update the controller module:**

Modify maindec to decode the opcode for the new instructions and generate the appropriate control signals. This enables us to add new control signals for jalm and push.

**3. Update the datapath module:**

Extend the datapath module to handle the new instructions. Specify the RTL expressions for each of the new instructions in the datapath module, including the fetch and updating of the PC.

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;

8'h04: instr = 32'h2003000c;

8'h08: instr = 32'h2067fff7;

8'h0c: instr = 32'h00e22025;

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'hA1C00000; // Example JALM machine code

8'h50: instr = 32'h92000000; // Example PUSH machine code

default: instr = {32{1'bx}}; // unknown address

endcase

endmodule

---------------------------------------------------------maindec--------------------------------------------------------

module maindec (input logic[5:0] op,

output logic memtoreg, memwrite, branch,

output logic alusrc, regdst, regwrite, jump,

output logic[1:0] aluop );

logic [8:0] controls;

assign {regwrite, regdst, alusrc, branch, memwrite,

memtoreg, aluop, jump} = controls;

always\_comb

case(op)

6'b000000: controls <= 9'b110000100; // R-type

6'b100011: controls <= 9'b101001000; // LW

6'b101011: controls <= 9'b001010000; // SW

6'b000100: controls <= 9'b000100010; // BEQ

6'b001000: controls <= 9'b101000000; // ADDI

6'b000010: controls <= 9'b000000001; // J

6'b100101: controls = 9'b100001000; // JALM

6'b100100: controls = 9'b001010000; // PUSH

default: controls <= 9'bxxxxxxxxx; // illegal op

endcase

endmodule

-----------------------------------------------------datapath------------------------------------------------------------

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);

logic [4:0] writereg;

logic [31:0] pcnext, pcnextbr, pcplus4, pcbranch;

logic [31:0] signimm, signimmsh, srca, srcb, result;

// next PC logic

flopr #(32) pcreg(clk, reset, pcnext, pc);

flopr #(32) jalm\_reg(clk, reset, pcnext, pc); // Add a register for the JALM instruction

adder pcadd1(pc, 32'b100, pcplus4);

sl2 immsh(signimm, signimmsh);

adder pcadd2(pcplus4, signimmsh, pcbranch);

mux2 #(32) pcbrmux(pcplus4, pcbranch, pcsrc,

pcnextbr);

mux2 #(32) pcmux(pcnextbr, {pcplus4[31:28],

instr[25:0], 2'b00}, jump, 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 #(32) resmux (aluout, readdata, memtoreg, result);

signext se (instr[15:0], signimm);

// ALU logic

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

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

endmodule