BILKENT UNIVERSITY
COMPUTER ENGINEERING
CS 224
COMPUTER ORGANIZATION

PRELIMINARY DESIGN REPORT
LAB 04

BERK YILDIZ
21502040
SECTION 4

29.11.2018

CS224
Section 4
Fall 2018
Lab 04
Berk Yıldız / 21502040

b)

0	ADDI $v0, $zero, 0x0005	32'h20020005
4	ADDI $v1, $zero, 0x000c	32'h2003000c
8	ADDI $a3, $v1, 0xfff7	32'h2067fff7
12	OR $a0, $a3, $v0	32'h00e22025
16	AND $a1, $v1, $a0	32'h00642824
20	ADD $a1, $a1, $a0	32'h00a42820
24	BEQ $a1, $a3, 0x000A	32'h10a7000a
28	SLT $a0, $v1, $a0	32'h0064202a
32	BEQ $a0, $zero, 0x0001	32'h10800001
36	ADDI $a1, $zero, 0x0000	32'h20050000
40	SLT $a0, $a3, $v0	32'h00e2202a
44	ADD $a3, $a0, $a1	32'h00853820
48	SUB $a3, $a3, $v0	32'h00e23822
52	SW $a3, 0x0044, $v1	32'hac670044
56	LW $v0, 0x0050, $zero	32'h8c020050
60	J 0x0000011	32'h08000011
64	ADDI $v0, $zero, 0x0001	32'h20020001
68	SW $v0, 0x0054, $zero	32'hac020054
72	J 0x0000012	32'h08000012

c)

push: IM[PC]

RF[29] <--- RF[29] - 4

DM[RF[29]] <--- RF[rt]

jalm: IM[PC]

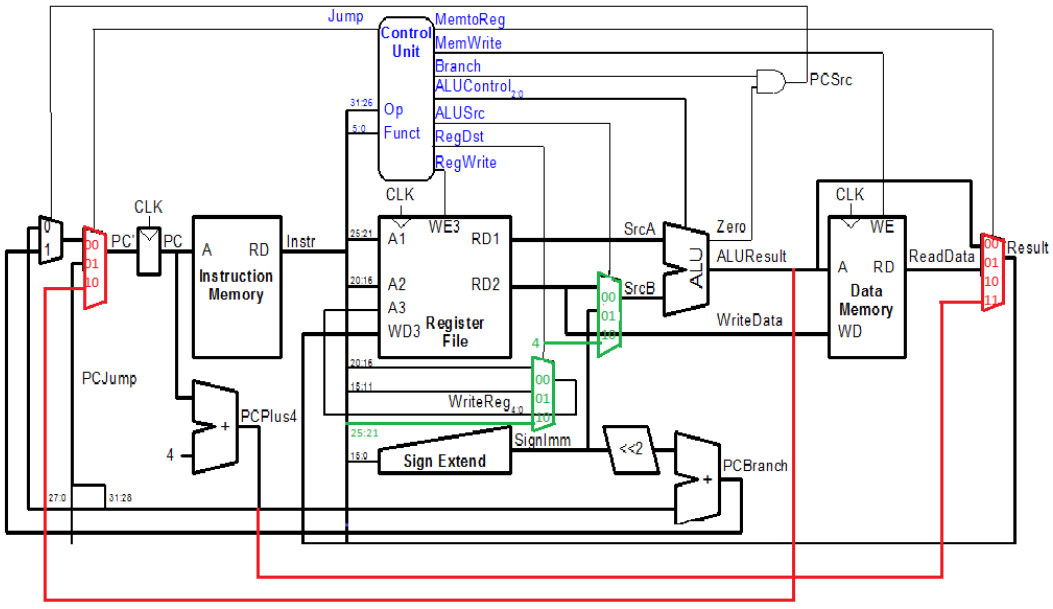
RF[rt] <- PC+4

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

CS224
Section 4
Fall 2018
Lab 04
Berk Yıldız / 21502040

d-e)

Instruc-tion	$Op_{5:0}$	Reg Write	Reg Dst	Alu Src	Branch	Mem Write	Mem toReg	ALU $Op_{1:0}$	Jump
R-type	000000	1	01	00	0	0	00	1x	00
lw	100011	1	00	01	0	0	01	00	00
sw	101011	0	XX	01	0	1	XX	00	00
beq	000100	0	XX	00	1	0	XX	01	00
j	000010	0	XX	XX	X	0	XX	XX	01
addi	001000	1	00	01	0	0	00	00	00
push	111110	1	10	10	X	0	10	XX	10
jalm	010101	1	00	01	X	0	10	00	10



f)

```
push $ra
lw $ra, 0($sp)
addi $t0, $zero, 52
sw $t0, 20($0) //store 52 in DM[20]
O: jalm $ra, 20($0) // go to DM[20], store O+4 in $ra
...
52: sw $ra, 20($0)// it passes test if it reaches here and $ra = O+4
56:j 56// looping
```

g)

```
module maindec (input logic[5:0] op,
output logic memtoreg, memwrite, branch,
output logic alusrc, regdst, regwrite, jump,
output logic[1:0] aluop,
output logic push , jalmC );
logic [10:0] controls;
assign {regwrite, regdst, alusrc, branch, memwrite,
memtoreg, aluop, jump, pushC, jalmC} = controls;
always_comb
case(op)
6&#39;b000000: controls &lt;= 11&#39;b1010000001000 // R-type
6&#39;b100011: controls &lt;= 11&#39;b1000100010000; // LW
6&#39;b101011: controls &lt;= 11&#39;b0000101000000; // SW
6&#39;b000100: controls &lt;= 11&#39;b0000010000100; // BEQ
6&#39;b001000: controls &lt;= 11&#39;b1000100000000; // ADDI
6&#39;b000010: controls &lt;= 11&#39;b0000000000001; // J
6&#39;b111110: controls &lt;= 11&#39;b1101000100010; //push
```

```
6&#39;b010101: controls &lt;= 11&#39;b1000100100010; // Jalm
default: controls &lt;= 11&#39;bxxxxxxxxxxx; // illegal op
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,
input logic jalmC);
logic [4:0] writereg;
logic [31:0] pcnext, pcnextbr, pcplus4, pcbranch, pcformemory,
pccomingmemory;
logic [31:0] signimm, signimmsh, srca, srcb, result, signimmshJalm,
resultmux1, resultforreg;
// next PC logic in i i o
flopr #(32) pcreg(clk, reset, pcnext, pc);
// i i o
adder pcadd1(pc, 32&#39;b100, pcplus4);
// i o
sl2 immsh(signimm, signimmsh);
// i need sign extented value for that
signext seJalm (instr[15:0], signimmshJalm);
adder pcAddforJalm(pc, signimmshJalm, pccomingmemory); // lets say it
read from memory // i I o
adder pcadd2(pcplus4, signimmsh, pcbranch); // i i i 0
```

```
mux2 #(32) pcbrmux(pcplus4, pcbranch, pcsrc,pcnextbr);// i i
mux2 #(32) pcmux(pcnextbr, {pcplus4[31:28],
// i i o
instr[25:0], 2&#39;b00}, jump, pcnext);
mux3 #(32) pcmuxJ(pcnext,pccomingmemory, jalmC, pcformemory);
// register file logic
regfile rf (clk, regwrite, instr[25:21], instr[20:16], writereg,
result, srca, writedata);
//out out
mux2 #(5) wrmux (instr[20:16], instr[15:11], regdst, writereg);
mux2 #(32) resmux (aluout, readdata, memtoreg, result);
sl16 immsh16(signimm, signimmsh);
mux3 #(32) jalmmux (resultmux1, pcplus4, jalmC, resultforreg);
signext se (instr[15:0], signimm);
// ALU logic
mux2 #(32) srcbmux (writedata, signimm, alusrc, srcb);
alu alu (srca, srcb, alucontrol, aluout, zero);
endmodule
```