Course No: CS224

Lab No.: 4

Section No.: 1

Full Name: Ahmet Faruk Ulutaş

Bilkent ID: 21803717

B)

Location (hex)	Machine Instr. (hex)	Assemby Lang. Eqv.
00	20020005	addi $v0, $0, 0x0005
04	2003000c	addi $v1, $0, 0x000c
08	2067fff7	addi $a3, $0, 0xfff7
0C	00e22025	or $a0, $a3, $v0
10	00642824	and $a1, $v1, $a0
14	00a42820	add $a1, $a1, $a0
18	10a7000a	beq $a1, $a1, $a0
1C	0064202a	slt $a0, $v1, $a0
20	10800001	beq $a0, $0, 0x0001
24	20050000	addi $a1, $0, 0x0000
28	00e2202a	slt $a0, $a3, $v0
2C	00853820	add $a3, $a0, $a1
30	00e23822	sub $a3, $a3, $v0
34	ac670044	sw $a3, 0x0044, $v1
38	8c020050	lw $v0, 0x0050, $0
3C	08000011	j 0x00000011
40	20020001	addi $v0, $0, 0x0001
44	ac020054	sw $v0, 0x0054, $0
48	08000012	j 0x00000012

C)

RTL Expression of “ble”:

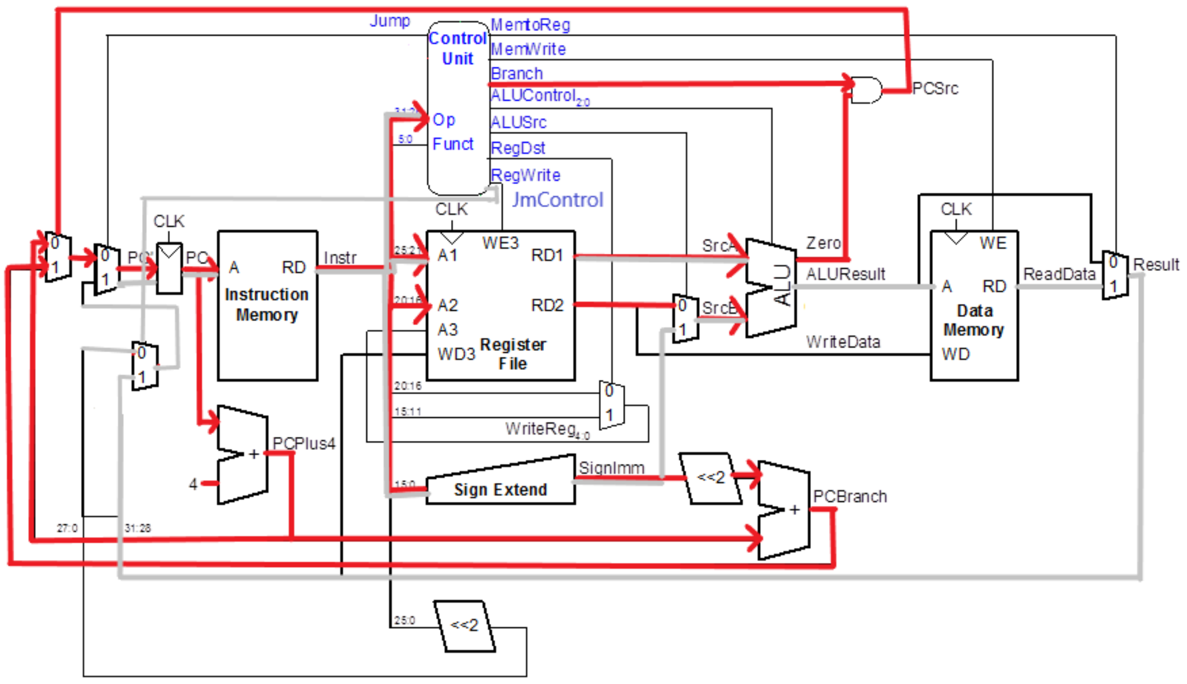
```
IM[PC]

if(R[rs]<=R[rt])

PC = Label

PC = PC + 4
```

RTL Expression of “jm”:

IM[PC]

PC = mem[R[rs] + ZeroExt(imm)]

D)



E)

Instr.	JmControl	Op(5:0)	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALU Op(1:0)	Jump	AluControl
ble	X	110010	0	X	0	1	0	X	00	0	010
jm	1	010101	0	X	1	X	0	1	00	1	010

F)

Jm:

```
.globl start
.text
start:
```

```
addi $a0, $zero, 0x00400014
jr $a0 # Jumps to address 0x00400014
addi $a1, $zero, 3      # Not executed
            addi $a2, $zero, 6      # Jumped to here
```

Ble:

```
.data
msgCorrect: .asciiz "Correct!"
msgIncorrect: .asciiz "Incorrect!"
.globl start
.text
start:
addi $t0, $0, 1
addi $t1, $0, 2
ble $t0, $t1, correct # case less than
equal:
ble $t1, $t0, incorrect # case equal
exit:
# exit the program
li $v0, 10
syscall
correct:
li $v0, 4
la $a0, msgCorrect
syscall
j equal
incorrect:
li $v0, 4
la $a0, msgIncorrect
syscall
j exit
```

G)

No special modifications are required for the blem instruction to work. Just add opcode and controls to maindec. However, in the jm instruction, we added a mux because two wires go to where the PC is updated, and this mux controls the controller output called jmcontroller. Added opcode and controller in maindec to ensure this. Later all controller bits were increased to 10 and new jmcontroller was added. Later, jmcontroller was added to controller and mips function respectively. In the latest datapath two results entered muxUpdate and with the help of jmcontroller the result is determined. The result of mux is given to pcmux.

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

        logic [9:0] controls;

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

        memtoreg, aluop, jump} = controls;

        always_comb

                case(op)

                        6'b000000: controls <= 10'b0110000100; // R-type

                        6'b100011: controls <= 10'b0101001000; // LW

                        6'b101011: controls <= 10'b0001010000; // SW

                        6'b000100: controls <= 10'b0000100010; // BEQ

                        6'b001000: controls <= 10'b0101000000; // ADDI

                        6'b000010: controls <= 10'b0000000001; // J

                        6'b110010: controls <= 10'b0000100000; // BLE

                        6'b010101: controls <= 10'b1001001001; // JM

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

                endcase

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 jmcontroller, jump, output
logic[2:0] alucontrol);

        logic [1:0] aluop;

        logic branch;
```

```
	maindec md (op, jmcontroller, memtoreg, memwrite, branch, alusrc, regdst, regwrite,
	jump, aluop);
	aludec ad (funct, aluop, alucontrol);
	assign pcsrc = branch & zero;
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, jmcontroller;
	logic [2:0] alucontrol;
	controller c (instr[31:26], instr[5:0], zero, memtoreg, memwrite, pcsrc,
	alusrc, regdst, regwrite, jmcontroller, jump, alucontrol);
	datapath dp (clk, reset, memtoreg, pcsrc, alusrc, regdst, regwrite, jmcontroller, jump,
	alucontrol, zero, pc, instr, aluout, writedata, readdata);
endmodule

module datapath (input logic clk, reset, memtoreg, pcsrc, alusrc, regdst, input logic regwrite,
jmcontroller, 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);
	adder pcadd1(pc, 32'b100, pcplus4);
	sl2 immsh(signimm, signimmsh);
	adder pcadd2(pcplus4, signimmsh, pcbranch);
	mux2 #(32) pcbrmux(pcplus4, pcbranch, pcsrc, pcnextbr);
	mux2 #(32) muxUpdate( jmcontroller, result, {pcplus4[31:28], instr[25:0], 2'b00},
	muxUpdateResult);
	mux2 #(32) pcmux(pcnextbr, muxUpdateResult , 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
```