CS224

LAB 4

PRELIMINARY WORK

SECTION 3

MUSTAFA MERT GÜLHAN

22201895

b)

Location	Machine Ins.	Assembly Equivalent
00	0x20020005	addi $v0,$zero,5
04	0x2003000c	addi $v1,$zero,12
08	0x2067fff7	addi $a3,$v1,-9
0C	0x00e22025	or $a0,$a3,$v0
10	0x00642824	and $a1,$v1,$a0
14	0x00a42820	add $a1,$a1,$a0
18	0x10a7000a	beq $a1,$a3,0x44
1C	0x0064202a	slt $a0,$v1,$a0
20	0x10800001	beq $a0,$zero,0x28
24	0x20050000	addi $a1,$zero,0
28	0x00e2202a	slt $a0,$a3,$v0
2C	0x00853820	add $a3,$a0,$a1
30	0x00e23822	sub $a3,$a3,$v0
34	0xac670044	sw $a3,68($v1)
38	0x8c020050	lw $v0,80($zero)
3C	0x08000011	j 0x44
40	0x20020001	addi $v0,$zero,1
44	0xac020054	sw $v0,84($zero)
48	0x08000012	j 0x48

c)

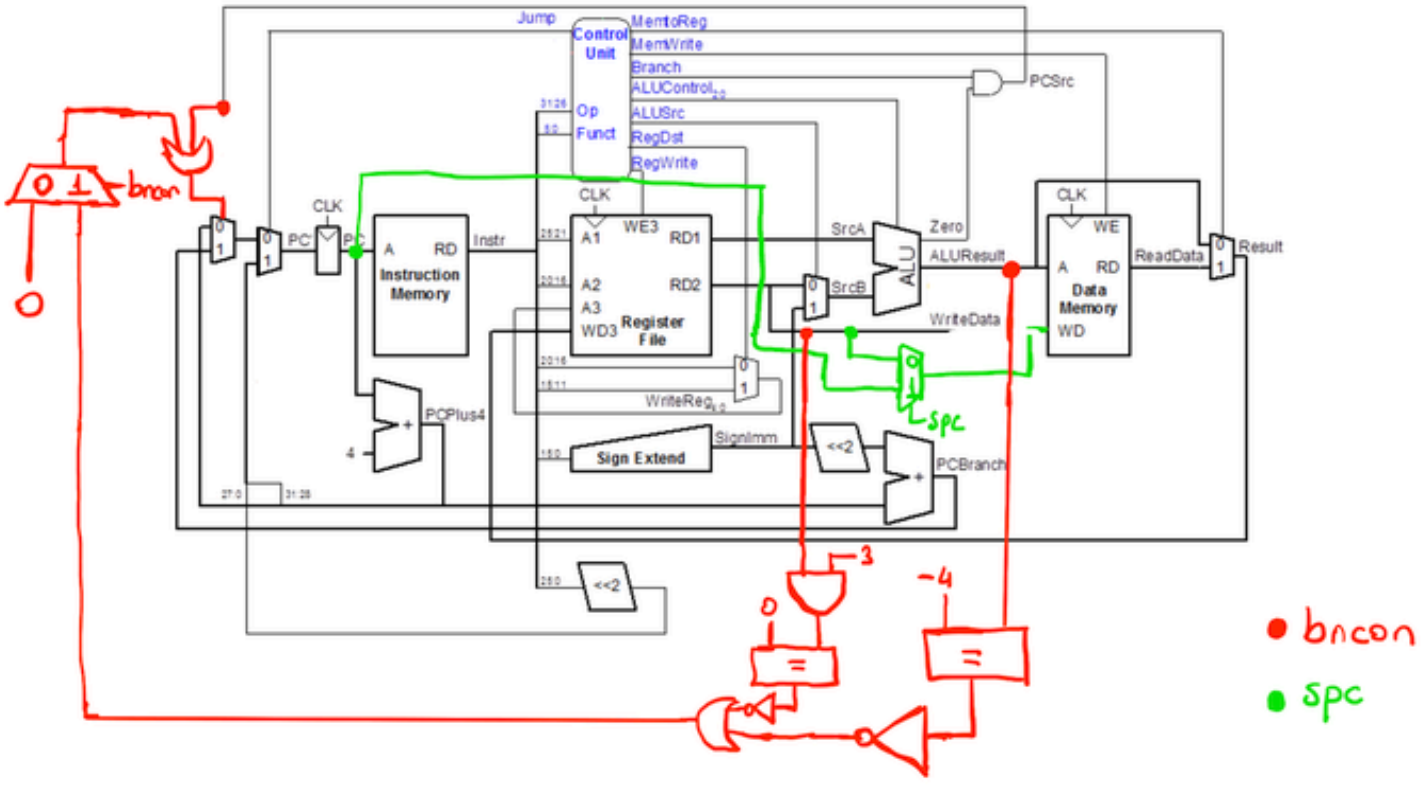
bncon:

```
IM[PC]
if(RF[rs] - RF[rt] != -4)
    PC <- BTA
else
    PC <- PC + 4
```

spc:

```
IM[PC]
DM[RF[rs]+SignExt(Imm)] <- PC
PC <- PC + 4
```

d)



e)

Instruction	OPcode	Regwrite	RegDST	ALUSRC	Branch	MemWrite	MemToReg	ALUop	Jump	bncon	spc
R-type	000000	1	1	0	0	0	0	10	0	0	X
lw	100011	1	0	1	0	0	1	00	0	0	X
sw	101011	0	X	1	0	1	X	00	0	0	0
beq	000100	0	X	0	1	0	X	01	0	0	X
addi	001000	1	0	1	0	0	0	00	0	0	X
j	000010	0	X	X	X	0	X	XX	1	0	X
bncon	100000	0	X	0	0	0	X	01	0	1	X
spc	110000	0	X	1	0	1	X	00	0	0	1

f)

bncon:

```
addi $a0, $zero, 1
addi $a1, $zero, 3
bncon $a0, $a1, 2  #branch bcz a1 is not word-align
addi $a1, $zero, 8 #skip
addi $a0, $a1, 4   #skip
addi $a0, $zero, 16
addi $a1, $zero, 20
bncon $a0, $a1, -2 #not branch
addi $a0, $zero, 24
bncon $a0, $a1, 2  #branch no consecutive
addi $a0, $zero, 1 #skip
addi $a1, $zero, 2 #skip
addi $a2, $zero, 5
```

```
spc:
    spc 60($zero)
    lw $a0, 60($zero)
    addi $a1, $zero, 4
    addi $a0, $a0, 12 # a0 = PC
    spc 64($zero)
    lw $a1, 64($zero)
    addi $a1, $a1, 8 # a1 = PC
```

g) I need to change maindec, controller, datapath, mips and new module bnconcheck

//maindec

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

   assign {regwrite, regdst, alusrc, branch, memwrite,
                memtoreg,  aluop, jump, bncon, spc} = 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'b100000: controls <= 11'b00000001010; // bncon
      6'b110000: controls <= 11'b00101000001; // spc
      default:   controls <= 11'bxxxxxxxxxxx; // illegal op
```

```
    endcase
endmodule
```

//controller

```
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 bncon, spc,
                  output logic[2:0] alucontrol);

   logic [1:0] aluop;
   logic       branch;

   maindec md (op, memtoreg, memwrite, branch, alusrc, regdst,
regwrite,
            jump, bncon, spc, aluop);

   aludec  ad (funct, aluop, alucontrol);

   assign pcsrc = branch & zero;

endmodule
```

//datapath

```
module datapath (input  logic clk, reset, memtoreg, pcsrc, alusrc,
regdst,
                 input  logic regwrite, jump, bncon, spc,
            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;

  //bncon var
  logic bnconresult;
  logic bnconbranch;
  logic newpcsrc;

// 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);

   //bncon logic
   bnconcheck  bc(aluout,srcb,bnconresult);
   mux2 #(1) bnconmux(0,bnconresult,bncon,bnconbranch);
   assign newpcsrc = bnconbranch | pcsrc;

   //spc logic
   mux2 #(32) wdmux(writedata, pc, spc, writedata);


  // 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, newpcsrc,
                      pcnextbr);
  mux2 #(32)  pcmux(pcnextbr, {pcplus4[31:28],
                     instr[25:0], 2'b00}, jump, pcnext);

endmodule
```

//mips

```
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, bncon, spc;
  logic [2:0]  alucontrol;

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

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

endmodule
```

//new module bnconcheck

```
module bnconcheck(input logic[31:0] aluresult, input logic[31:0]
srcb, output logic bnconresult );
    assign bnconresult = (aluresult != -4) | (srcb[1:0] & 2'b11 != 0
);
endmodule
```