CS224

Lab 4

Section 3

Ahmet Eren Gökalp

22302136

16/03/2025

b)

Address	Machine Code	Instruction
0x00	0x20020005	addi $v0, $zero, 5
0x04	0x2003000C	addi $v1, $zero, 12
0x08	0x2067FFF7	addi $a3, $v1, -9
0x0C	0x00E22025	or $a0, $a3, $v0
0x10	0x00642824	and $a1, $v1, $a0
0x14	0x00A42820	add $a1, $a1, $a0
0x18	0x10A7000A	beq $a1, $a3, 0x28
0x1C	0x0064202A	slt $a0, $v1, $a0
0x20	0x10800001	beq $a0, $zero, 0x24
0x24	0x20050000	addi $a1, $zero, 0
0x28	0x00E2202A	slt $a0, $a3, $v0
0x2C	0x00853820	add $a3, $a0, $a1
0x30	0x00E23822	sub $a3, $a3, $v0
0x34	0xAC670044	sw $a3, 68($v1)
0x38	0x8C020050	lw $v0, 80($zero)
0x3C	0x08000011	j 0x44
0x40	0x20020001	addi $v0, $zero, 1
0x44	0xAC020054	sw $v0, 84($zero)
0x48	0x08000012	j 0x48

c)

RTL expression for “bncon”

if (R[rt] ≠ R[rs] + 1) then PC ← PC + 4 + (offset × 4)

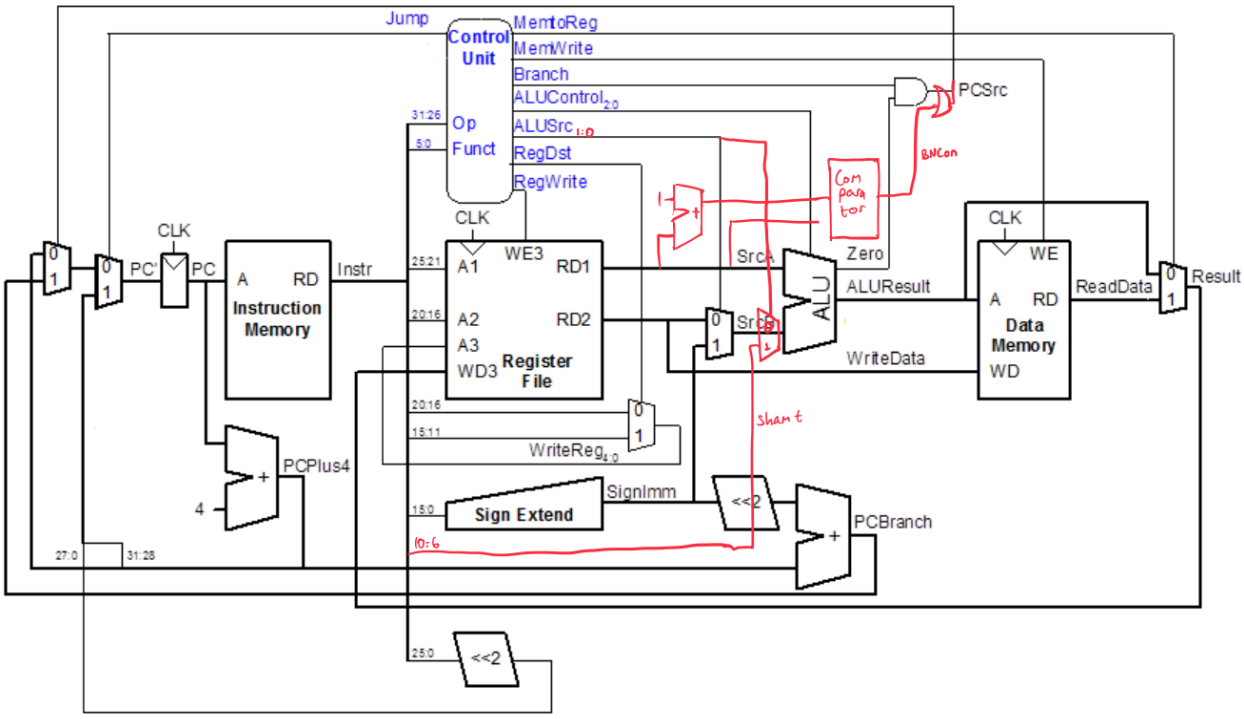
else PC ← PC + 4

RTL expression for sll

R[rd] ← R[rt] << shamt

d) Datapath changes

A barrel shifter has been added into the ALU



e) Updated Table 1

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemToReg	ALUOp	Jump
R-type	000000	1	1	00	0	0	0	10	0
lw	100011	1	0	01	0	0	1	00	0
sw	101011	0	X	01	0	1	X	00	0
beq	000100	0	X	00	1	0	X	01	0
addi	001000	1	0	01	0	0	0	00	0
j	000010	0	X	X	0	X	X	XX	1
sll	000000	1	1	10	0	0	0	11	0
bncon	011000	0	X	00	1	0	X	10	0

Updated Table 2

ALUOp	Funct	ALUControl
00	X	010 (add)
01	X	110 (subtract)
1X	100000	010 (add)
1X	100010	110 (subtract)
1X	100100	000 (and)
1X	100101	001 (or)
1X	101010	111 (set less than)
11	000000	011 (shift left logical)

f) Test program

```
 .data
array:  .word 5, 10, 15, 20
    .text
    .globl main
main:
    li $t0, 5
    li $t1, 10
    # test original 10
    add $t2, $t0, $t1
    sub $t3, $t2, $t0
    and $t4, $t0, $t1
    or  $t5, $t0, $t1
    slt $t6, $t0, $t1

    la $s0, array
    lw $s1, 0($s0)
    lw $s2, 4($s0)
    sw $s1, 8($s0)

    beq $t0, $t1, skip
    li $a0, 1

skip:
    beq $t0, $t0, bnconTest
# test for bncon
bnconTest:
    li $a1, 12
    li $a2, 8
    bncon $a2, $a1, shiftTest
    li $a0, 2
# test for sll
shiftTest:
    li $t7, 3
    sll $s3, $t7, 2

    j endTest
    li $a0, 3

endTest:
    li $v0, 10
    syscall
```

g) Updated System Verilog modules

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

   logic [10:0] controls;

   assign {regwrite, regdst, alusrc, branch, memwrite,
                    memtoreg,   aluop, jump} = {controls[10], controls[9], controls[8:7], controls[6],
controls[5], controls[4], controls[3:2], controls[1]};

  always_comb
    case(op)
      6'b000000: controls <= 11'b1100001000; // R-type
      6'b100011: controls <= 11'b1001001000; // LW
      6'b101011: controls <= 11'b0001010000; // SW
      6'b000100: controls <= 11'b0x0010x010; // BEQ
      6'b001000: controls <= 11'b1001000000; // ADDI
      6'b000010: controls <= 11'b0xxx0xxx01; // J
      6'b011000: controls <= 11'b0x0010x100; // BNCON
      6'b000011: controls <= 11'b1001001110; // SLL
      default:   controls <= 11'bxxxxxxxxxxx; // illegal op
    endcase
endmodule

// Updated aludec module
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
      2'b01: alucontrol  = 3'b110;  // sub
      2'b11: alucontrol  = 3'b011;  // sll
      default: case(funct)          // r-type
          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

// Updated controller
module controller(input  logic[5:0] op, funct,
                  input  logic     zero, bncon_flag,
                  output logic     memtoreg, memwrite,
                  output logic     pcsrc,
                  output logic [1:0] alusrc,
                  output logic     regdst, regwrite,
                  output logic     jump,
                  output logic[2:0] alucontrol);

  logic [1:0] aluop;
  logic       branch;

  maindec md (op, memtoreg, memwrite, branch, alusrc, regdst, regwrite, jump, aluop);
  aludec  ad (funct, aluop, alucontrol);
  assign pcsrc = (branch & zero) | bncon_flag;

endmodule

// ALU
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;
         3'b011: result = b << a[4:0]; // SLL
```

```
            default: result = {32{1'bx}};
        endcase

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

// Datapath
module datapath (input  logic clk, reset, memtoreg, pcsrc,
                 input  logic [1:0] alusrc,
                 input  logic regdst, 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, shamt_ext;

  assign shamt_ext = {27'b0, instr[10:6]}; // zero-extend shamt

  // 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)  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 with mux3
  mux3 #(32)  srcbmux (writedata, signimm, shamt_ext, alusrc, srcb);
```

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

endmodule


// MUX 3 for datapath
module mux3 #(parameter WIDTH = 32) (
    input logic [WIDTH-1:0] d0, d1, d2,
    input logic [1:0] s,
    output logic [WIDTH-1:0] y);

    always_comb begin
        case(s)
            2'b00: y = d0;
            2'b01: y = d1;
            2'b10: y = d2;
            default: y = {WIDTH{1'bx}};
        endcase
    end
endmodule

// Updated imem
module imem ( input logic [7:0] addr, output logic [31:0] instr);

 always_comb
    case (addr)
  // main
  8'h00: instr = 32'h20080005;
  8'h04: instr = 32'h2009000A;
  8'h08: instr = 32'h01095020;
  8'h0C: instr = 32'h014A5822;
  8'h10: instr = 32'h01095824;
  8'h14: instr = 32'h01096025;
  8'h18: instr = 32'h0109702A;
  // array setup
  8'h1C: instr = 32'h3C100100; // la $s0, array (pseudo)
  8'h20: instr = 32'h8E110000;
  8'h24: instr = 32'h8E120004;
  8'h28: instr = 32'hAE110008;
  // skip label
```

```
  8'h2C: instr = 32'h11090004; // beq $t0, $t1, skip
  8'h30: instr = 32'h20040001;
  // skip:
  8'h34: instr = 32'h11080003; // beq $t0, $t0, bnconTest
  // bnconTest:
  8'h38: instr = 32'h2005000C;
  8'h3C: instr = 32'h20060008;
  8'h40: instr = 32'h18C50002; // bncon $a2, $a1, shiftTest
  8'h44: instr = 32'h20040002;
  // shiftTest:
  8'h48: instr = 32'h200F0003;
  8'h4C: instr = 32'h000F1980;
  8'h50: instr = 32'h08000014;
  // endTest:
  8'h54: instr = 32'h2002000A;
  8'h58: instr = 32'h0000000C;
  default: instr = {32{1'bx}};
    endcase
endmodule
```