CS 224 - Lab 4 - Section 1 - Preliminary Report

Mehmet Akif Şahin - 22203673

18 March 2024

Part B

0x20020005	addi $v0 $zero 0x0005
0x2003000c	addi $v1 $zero 0x000c
0x2067fff7	addi $a3 $v1 0xfff7
0x00e22025	or $a0 $a3 $v0
0x00642824	and $a1 $v1 $a0
0x00a42820	add $a1 $a1 $a0
0x10a7000a	beq $a1 $a3 0x000A
0x0064202a	slt $a0 $v1 $a0
0x10800001	beq $a0 $zero 0x0001
0x20050000	addi $a1 $zero 0x0000
0x00e2202a	slt $a0 $a3 $v0
0x00853820	add $a3 $a0 $a1
0x00e23822	sub $a3 $a3 $v0
0xac670044	sw $a3 0x0044 ($v1)
0x8c020050	lw $v0 0x0050 ($zero)
0x08000011	j 0x0000011
0x20020001	addi $v0 $zero 0x0001
0xac020054	sw $v0 0x0054 ($zero)
0x08000012	j 0x0000012

Part C

bcon (Branch if consecutive): This I-type instruction branches to the target address only if the rt register has the value of a consecutive address value held in rs. Otherwise, branch is not taken.
Usage: bcon rs, rt, label. Example: bcon $a1, $a0, loopStart (when $a1 is 8 and $a0 is 12, branch is taken).

```
RTL
    IM[PC]
    IF ( RF[rs] + 4 - RF[rt] == 0 ) PC <- PC + 4 + 2 * signExtendedImmediate
    ELSE PC <- PC + 4
```

neg (Negate): This R-type instruction negates the right register's value and puts it into the left register.
Usage: neg rd, rs (rt and shamt are zero in this instruction). Example: neg $a0, $a1 (when $a1 is 9, $a0 will be -9).

```
RTL
    IM[PC]
    RF[rd] <- (-1) * RF[rs]
    PC <- PC + 4
```

Part D

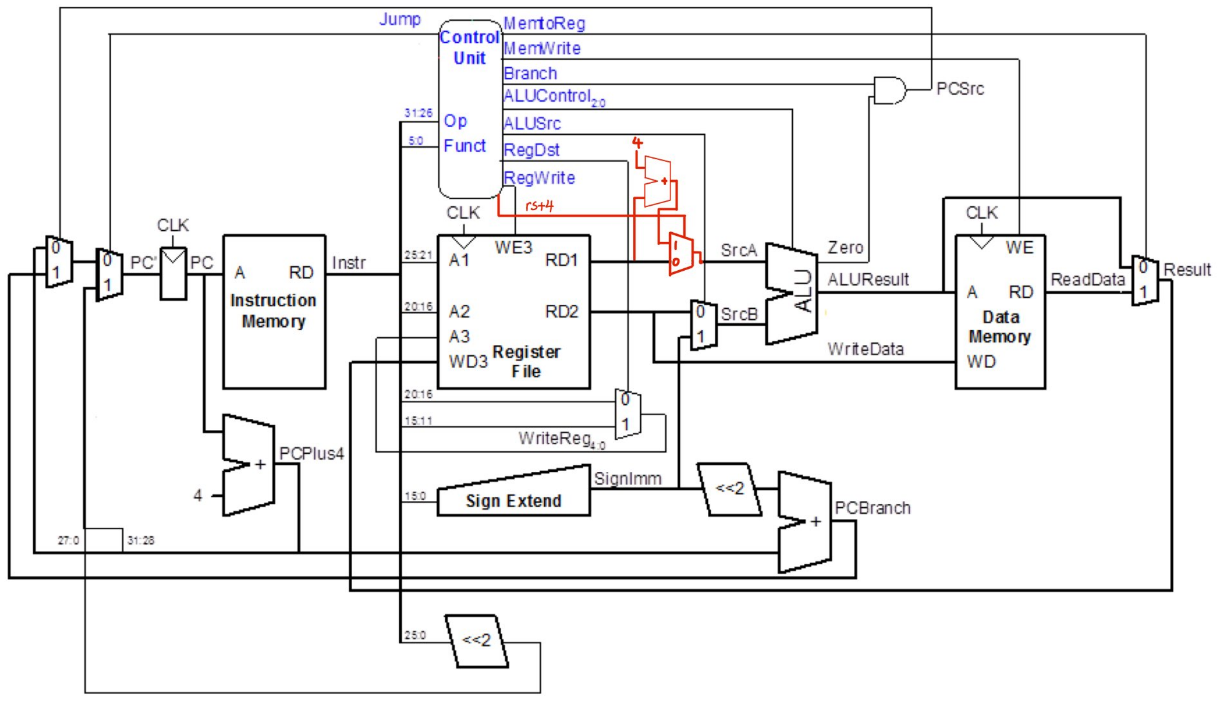

Figure 1: extended mips cpu schema

Part E

ALUOp	Funct	ALUControl
00	X	010 (add)
X1	X	110 (substract)
1X	100000 (add)	010 (add)
1X	100010 (sub)	110 (substract)
1X	100100 (and)	000 (and)
1X	100101 (or)	001 (or)
1X	101010 (slt)	111 (set less than)
1X	100001 (neg)	011 (negate)

Figure 2: New Alu Decoder Truth Table

Ins.	Opcode	Reg Write	RegDst	ALUSrc	Branch	Mem Write	Memto Reg	ALUOp	Jump	RSPlus4
R-type	000000	1	1	0	0	0	0	10	0	0
lw	100011	1	0	1	0	0	1	00	0	0
sw	101011	0	X	1	0	1	X	00	0	0
beq	000100	0	X	0	1	0	X	01	0	0
addi	001000	1	0	1	0	0	0	00	0	0
j	000010	0	X	X	X	0	X	XX	1	X
bcon	000101	0	X	0	1	0	X	01	0	1

Figure 3: New Main Decoder Truth Table

Part F

```
    .text

    # add, sub, and, or, slt, lw, sw, beq,
    # bcon, neg

    addi $t0, $0, 5
    addi $t1, $0, 9
    sw $t0, 0($0)
    sw $t1, 4($0)

    lw $s0, 0($0)
    lw $s1, 4($0)

    add $s3, $s0, $s1
    sub $s4, $s0, $s1
    and $s5, $s0, $s1
    or $s6, $s0, $s1
    slt $s7, $s0, $s1

    sub $t0, $s0, $s0
    beq $t0, $0, 1
    beq $0, $0, -2

    bcon $s0, $s1, 1
    beq $0, $0, -2

    neg $s0, $s0

    j 0x11
```

partftest.asm

Part G

Modules Requiring Changes

1. mips module
2. controller module
3. maindec module
4. aludec module
5. datapath module

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, rsplus4;
  logic [2:0]  alucontrol;

  controller c (instr[31:26], instr[5:0], zero, memtoreg, memwrite, pcsrc,
                         alusrc, regdst, regwrite, jump, rsplus4, alucontrol);

  datapath dp (clk, reset, memtoreg, pcsrc, alusrc, regdst, regwrite, jump, rsplus4
      ,
                                alucontrol, zero, pc, instr, aluout, writedata, readdata)
                                    ;

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      rsplus4,
                  output logic[2:0] alucontrol);

   logic [1:0] aluop;
   logic       branch;

   maindec md (op, memtoreg, memwrite, branch, alusrc, regdst, regwrite,
                 jump, rsplus4, aluop);

   aludec  ad (funct, aluop, alucontrol);

   assign pcsrc = branch & zero;

endmodule
```

maindec

```
module maindec (input logic[5:0] op,
                      output logic memtoreg, memwrite, branch,
                      output logic alusrc, regdst, regwrite, jump, rsplus4,
                      output logic[1:0] aluop );
    logic [9:0] controls;

    assign {regwrite, regdst, alusrc, branch, memwrite,
                memtoreg,  aluop, jump, rsplus4} = controls;

    always_comb
        case(op)
            6'b000000: controls <= 10'b1100001000; // R-type
            6'b100011: controls <= 10'b1010010000; // LW
            6'b101011: controls <= 10'b0010100000; // SW
            6'b000100: controls <= 10'b0001000100; // BEQ
            6'b001000: controls <= 10'b1010000000; // ADDI
            6'b000010: controls <= 10'b0000000010; // J
            6'b000101: controls <= 10'b0001000101; // BCON
            default:   controls <= 10'bxxxxxxxxxx; // illegal op
    endcase
endmodule
```

aludec

```
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  (for lw/sw/addi)
      2'b01: alucontrol  = 3'b110;  // sub   (for beq)
      default: case(funct)          // R-TYPE instructions
          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
          6'b100001: alucontrol  = 3'b011; // NEG
          default:   alucontrol  = 3'bxxx; // ???
        endcase
    endcase
endmodule
```

datapath

```
module datapath (input  logic clk, reset, memtoreg, pcsrc, alusrc, regdst,
                 input  logic regwrite, jump, rsplus4
                 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, rd1, rd1plus4;

  // 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, rd1, 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
   adder       rd1add4 (rd1, 32'b100, rd1plus4);
   mux2 #(32)  srcamux (rd1, rd1plus4, rtplus4, srca);
   mux2 #(32)  srcbmux (writedata, signimm, alusrc, srcb);
   alu         alu (srca, srcb, alucontrol, aluout, zero);

endmodule
```