CS 224
Section No.: 2
Spring 2019
Lab 04
Berrak Taşkınsu / 21602054

CS 224 – Spring 2019 – Lab #4

Extending Single-Cycle MIPS Processor & Experiments on SystemVerilog and BASYS3 Board

Preliminary Design Report

Berrak Taşkınsu / 21602054

Section No.: 2

CS 224
Section No.: 2
Spring 2019
Lab 04
Berrak Taşkınsu / 21602054

b) Instructions in machine language given in the imem module, their locations and their assembly language equivalents are as follows:

Location (Last 8 bits)	Instruction in Machine Language	Instruction in Assembly Language
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, 0x000a
0x1c	0x0064202a	slt $a0, $v1, $a0
0x20	0x10800001	beq $a0, $zero, 1
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 $a3, 44($v1)
0x38	0x8c020050	lw $v0, 50($zero)

CS 224
Section No.: 2
Spring 2019
Lab 04
Berrak Taşkınsu / 21602054

0x3c	0x08000011	j 0x00000044
0x40	0x20020001	addi $v0, $zero, 1
0x44	0xac020054	sw $v0, 54($zero)
0x48	0x08000012	j 0x00000048

c) RTL expression of the “nop” and “subi” instruction is as follows:

“nop” is an I-type instruction which does nothing, changes no values, takes one clock cycle. An example is simply “nop”. Its RTL expression is as follows:

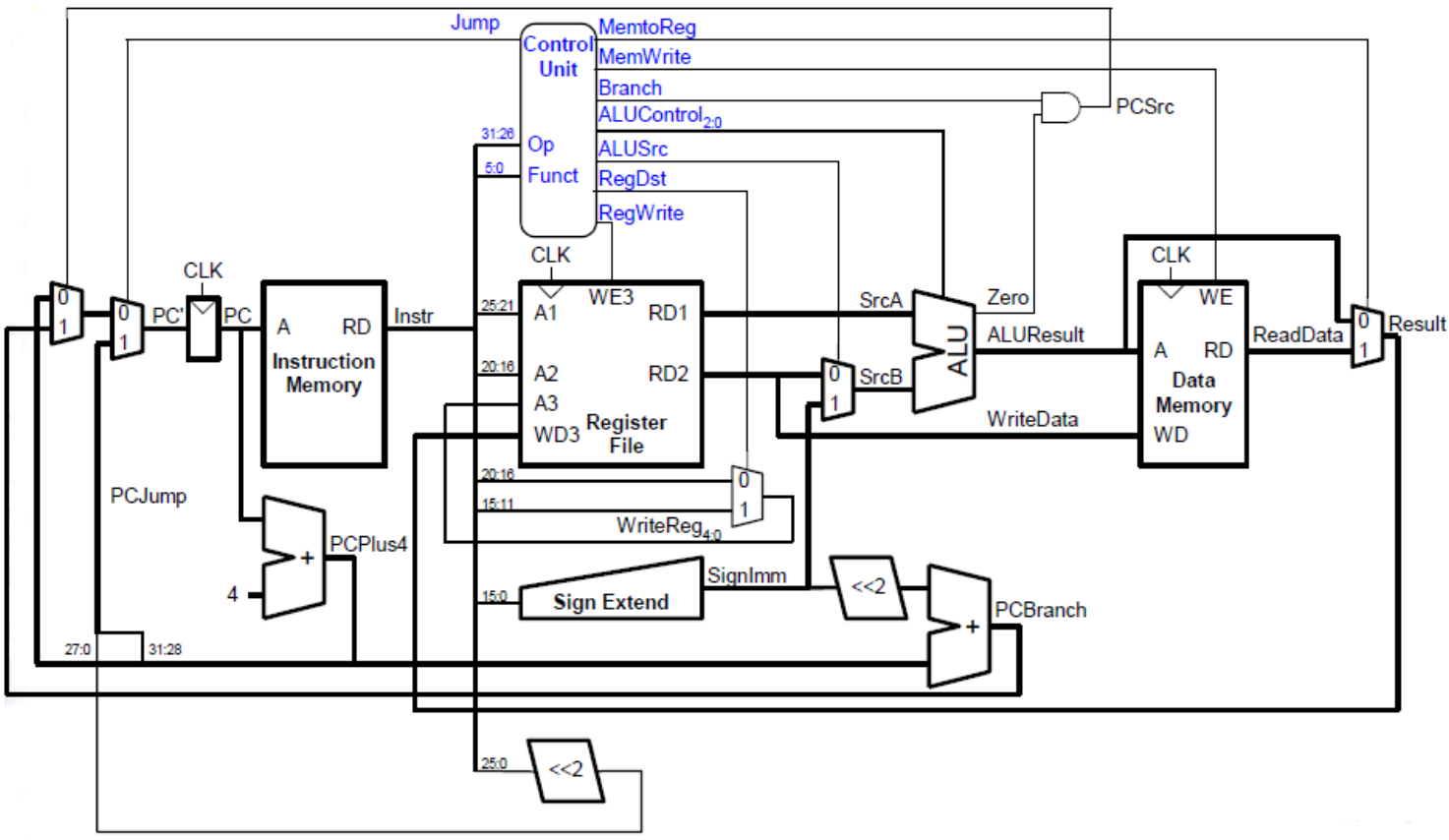
```
IM[pc];
pc <- pc + 4;
```

“subi” is an I-type instruction which substracts a sign-extended immediate value from a register ($rs) and stores the result in another ($rt). Its RTL expression is as follows:

```
IM[pc];
RF[rt] <- RF[rs] – SignExt(imm);
pc <- pc + 4;
```

CS 224
Section No.: 2
Spring 2019
Lab 04
Berrak Taşkınsu / 21602054

d) No changes are made to the datapath. The base datapath given below is used.



CS 224
Section No.: 2
Spring 2019
Lab 04
Berrak Taşkınsu / 21602054

e) Main control table must be modified to add the instructions nop and subi.

Instruction	**Opcode**	**RegWrite**	**RegDst**	**ALUSrc**	**Branch**	**MemWrite**	**MemtoReg**	**ALUOp$_{1:0}$**	**Jump**
R-type	000000	1	1	0	0	0	0	1X	0
lw	100010	1	0	1	0	0	1	00	0
sw	101011	0	X	1	0	1	X	00	0
beq	000100	0	X	0	1	0	X	01	0
addi	001000	1	0	1	0	0	0	00	0
j	000010	0	X	X	X	0	X	XX	1
nop	**000001**	**0**	**0**	**1**	**0**	**0**	**0**	**00**	**0**
subi	**000011**	**1**	**0**	**1**	**0**	**0**	**0**	**01**	**0**

The control signals ALUOp$_{1:0}$ for nop and subi are 00 and 01, respectively. Since both are I-type instructions, they have a non-zero opcode. Therefore, corresponding ALUControl$_{2:0}$ signals are 010 (addition) and 110 (subtraction), respectively. No changes need to be done in ALU table.

CS 224
Section No.: 2
Spring 2019
Lab 04
Berrak Taşkınsu / 21602054

f)

```
0x00:  addi   $v0, $zero, 5;
0x04:  addi   $v1, $zero, 12;
0x08:  nop
0x0c:  addi   $a3, $v1, -9;
0x10:  or     $a0, $a3, $v0;
0x14:  nop
0x18:  and    $a1, $v1, $a0;
0x1c:  add    $a1, $a1, $a0;
0x20:  subi   $a2, $a1, 1;
0x24:  beq    $a2, $a3, 0x000a
       ...
0x4c:  j      0x00000050;
0x50:  sw     $v0, 54($zero); // If subi works, it will jump here.
0x54:  j      0x00000054;
// If nop works, no value will differ.
```

Therefore the new imem module to test the new functions will be as follows:

```
module imem ( input logic [5:0] addr, output logic [31:0] instr);

// imem is modeled as a lookup table, a stored-program byte-addressable ROM
        always_comb
          case ({addr,2'b00})                   // word-aligned fetch
//              address                 instruction
//              -------             -----------
                8'h00: instr = 32'h20020005;
                8'h04: instr = 32'h2003000c;
                8'h08: instr = 32'h04000000;  // nop
                8'h0c: instr = 32'h2067fff7;
                8'h10: instr = 32'h00e22025;
                8'h14: instr = 32'h04000000;  // nop
                8'h18: instr = 32'h00642824;
                8'h1c: instr = 32'h00a42820;
                8'h20: instr = 32'h0ca60001; // subi $a2, $a1, 1
                8'h24: instr = 32'h10c7000a; // beq $a2, $a3, 0x000a
                8'h28: instr = 32'h0064202a;
                8'h2c: instr = 32'h10800001;
                8'h30: instr = 32'h20050000;
                8'h34: instr = 32'h00e2202a;
                8'h38: instr = 32'h00853820;
                8'h3c: instr = 32'h00e23822;
                8'h40: instr = 32'hac670044;
                8'h44: instr = 32'h8c020050;
                8'h48: instr = 32'h08000014; // j 50
                8'h4c: instr = 32'h20020001;
                8'h50: instr = 32'hac020054;
```

```
            8'h54: instr = 32'h08000015; // j 54, so it will loop here
        default:  instr = {32{1'bx}};        // unknown address
      endcase
endmodule
```

g) Datapath doesn't need to be modified. Changes will be made only in maindec module to add the new instructions.

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

    logic [8:0] controls;

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

    always_comb
        case(op)
            6'b000000: controls <= 9'b110000100; // R-type
            6'b100011: controls <= 9'b101001000; // LW
            6'b101011: controls <= 9'b001010000; // SW
            6'b000100: controls <= 9'b000100010; // BEQ
            6'b001000: controls <= 9'b101000000; // ADDI
            6'b000010: controls <= 9'b000000001; // SLT
            6'b000001: controls <= 9'b001000000; // NOP
            6'b000011: controls <= 9'b101000010; // SUBI
            default:   controls <= 9'bxxxxxxxxx; // illegal op
        endcase
endmodule
```