CS224 PRELIMINARY REPORT

Spring 2018

Section NO: 1

Lab NO: 5

Student Name/ID: Doğa Oruç, 21602022

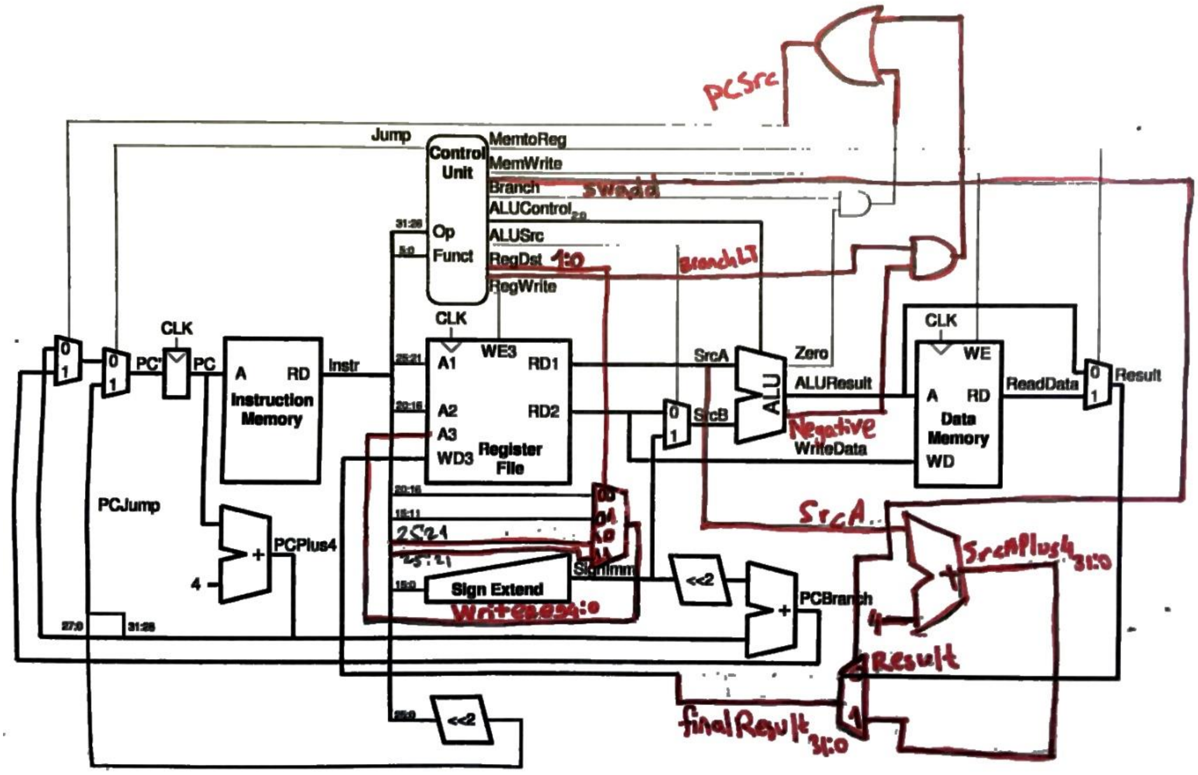
Student Name/ID: Deniz Yüksel, 21600880

1. Write the Register Transfer Level (RTL) expressions for the "ble" and "sw+" instructions that you are adding, including the fetch and the updating of the PC.

```
RTL For "ble":
    IM[PC];
    if(RF[rs] <= RF[rt])
        PC <- PC + 4 +[SignExt(imm) << 2];
    else
        PC <- PC + 4;

RTL For "sw+":
    IM[PC];
    DM[RF[rs] + SignExt(imm)] <- RF[rt];
    RF[rs] <- RF[rs] + 4;
    PC <- PC + 4;
```

2. Make any additions or changes to the datapath which are needed in order to make the RTLs for the instructions possible. The base datapath should be in black, with changes marked in red or other colors (one color per new instruction). You can take a photo/scan your modified datapath and add it to your Preliminary Report, as long as the changes you have done are clear and understandable.



3. Make a new row in the main control table for "ble" and "sw+", and if necessary add new columns for any new control signals that are needed (input or output). Be sure to completely fill in the table—all values must be specified. If any changes are needed in the ALU decoder table, give this table in its new form (with new rows, columns, etc). The base table should be in black, with changes marked in red or other colors.

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp	Jump	BLE	SwAdd
R-type	000000	1	1	0	0	0	0	10	0	0	0
lw	100011	1	0	1	0	0	1	00	0	0	0
sw	101011	0	X	1	0	1	X	00	0	0	0
beq	000100	0	X	0	1	0	X	01	00	0	0
addi	001000	1	0	1	0	0	0	00	0	0	0
j	000010	0	X	X	X	0	X	XX	1	0	0
ble	011111	0	X	0	1	0	X	01	0	1	0
sw+	111111	1	0	1	0	1	X	00	0	0	1

4. Write a test program in MIPS assembly language, that will show whether the new instructions are working or not , and that will confirm that all existing old instructions still continue to work. Don't use any pseudo-instructions; use only real MIPS instructions that will be recognized by the new control unit.

As written in the Instruction Memory(IMEM):

```
32'h00000000: instr = 32'h20020005;
32'h00000004: instr = 32'h2003000c;
32'h00000008: instr = 32'h2067fff7;
32'h0000000c: instr = 32'h00e22025;
32'h00000010: instr = 32'h00642824;
32'h00000014: instr = 32'h00a42820;
32'h00000018: instr = 32'h10a7000a;
32'h0000001c: instr = 32'h0064202a;
32'h00000020: instr = 32'h10800001;
32'h00000024: instr = 32'h20050000;
32'h00000028: instr = 32'h00e2202a;
32'h0000002c: instr = 32'h00853820;
32'h00000030: instr = 32'h00e23822;
```

```
32'h00000034: instr = 32'hac670044;
32'h00000038: instr = 32'h8c020050;
32'h0000003c: instr = 32'h08000011;
32'h00000040: instr = 32'h20020001;
32'h00000044: instr = 32'hac020054;
32'h00000048: instr = 32'h7c600000;//ble fail
32'h0000004c: instr = 32'hfc030000;//swadd
32'h00000050: instr = 32'h7c030001;//ble success
32'h00000054: instr = 32'h20020005;//unreachable add
32'h00000058: instr = 32'h08000012;//loopz
```

Converted to MIPS Assembly:

```
1.   addi $2,$0,5       #$2 is now equal to 5.
2.   addi $3,$0,12      #$3 is now equal to 12.
3.   addi $7,$3,-9      #$7 is now equal to 12 – 9( = 3).
4.   or   $4,$7,$2      #$4 is now equal to 3 | 5 ( = 7).
5.   and  $5,$3,$4      #$5 is now equal to 12 & 7 ( = f).
6.   add  $5,$5,$4      #$5 is now equal to f + 7 ( = 16).
7.   beq  $5,$7,0x44    #Do not branch.
8.   slt  $4,$3,$4      #$4 is now equal to 0, for $3 > $4 (12 > 3).
9.   beq  $4,$0,0x28    #Branch to the address of 0x28(To line 11).
10.  addi $5,$0,0       #Unreachable statement.
11.  slt  $4,$7,$2      #$4 is now equals to 1, for $7 < $2 (3 < 5).
12.  add  $7,$4,$5      #$7 is now equal to 1 + 5 ( = 6).
13.  sub  $7,$7,$2      #$7 is now equal to 6 – 5 ( = 1).
14.  sw   $7,68($3)     #68 ($3) [ or  just $20] is now equal to 1.
15.  lw   $2,80($0)     #$7 is now equals to 80 ($0) [ or just $20] ( = 1).
16.  j    0x44          #Jump to the address of 0x44(To line 18).
17.  addi $2,$0,1       #Unreachable statement.
18.  sw   $2,84($0)     #84 ($0) [ or just $21] is now equal to 5.
19.  ble  $3 $0 0x48    #Do not branch. (12 > 0)
20.  sw+  $3 0($0)      #$0 is now equal to 12, $3 is now 16.
21.  ble  $0 $3 0x48    #Branch to the address of 0x58. (12 < 16)
22.  addi $2,$0,5       #Unreachable due to the branch operation in line 21.
23.  j    0x48          #Jump to the address of 0x48(To line 19)(Inf. loop).
```

5. Write a list of the System Verilog modules that will need changes in order to make these new instructions part of the single-cycle MIPS processor's instruction set. For each module in the list, determine the new System Verilog model that will be needed in order for the instructions to be added. Give the System Verilog code for each module that needs to be changed.

This part is a little messy, because we didn't want to include all the modules and all of their contents, intead we only included the new parts. So please don't mind.

New modules in the extended MIPS- Lite Processor:

```
module alu(input logic[31:0] a, b, input logic[2:0] alucont, output
logic[31:0] result, output logic zero, negative);
      always_comb begin
            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;
                  default: result <= a;
            endcase
                  zero <= (result == 0) ? 1 : 0;
                  negative <= a < b;
      end
endmodule

module mux4 #(parameter WIDTH = 8) (input logic[WIDTH-1:0] d0, d1, d2,
d3, 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;
                  2'b11: y <= d3;
            endcase
      end
endmodule
```

// In the datapath, multiplexer to determine register written (before it was rd or rd) changes. Now it has another input because regdst is 2 bits.

```
module datapath(input  logic clk, reset, memtoreg, pcsrc, alusrc, input
logic [1:0] regdst, input  logic regwrite, jump, blt, swplus, logic[2:0]
alucontrol, output logic zero, negative, output logic[31:0] pc, input
logic[31:0]  instr,  output  logic[31:0]  aluout,  writedata,  input
logic[31:0] readdata);

      …
      mux4 #(5) wrmux (instr[20:16], instr[15:11], instr[25:21],
      instr[25:21], regdst, writereg);
      // For sw+, an adder is needed for rs:
        adder swplusadder(srca, 32'b100, srcaplus4);
        …

endmodule

// Main decoder for the control unit has changed. ble and sw+ are added.
module maindec(input logic[5:0] op, output logic memtoreg, memwrite,
branch, output logic alusrc, output logic[1:0]regdst, output logic
regwrite, jump, swplus, output logic[1:0] aluop );
        logic[11:0] controls;
        assign {regwrite, regdst, alusrc, branch, memwrite, memtoreg,
        aluop, jump, blt, swplus} = controls;
        always_comb begin
                case(op)
                        6'b000000: controls <= 12'b101000010000;// R-type
                        6'b100011: controls <= 12'b100100100000;//lw
                        6'b101011: controls <= 12'b000101000000;//sw
                        6'b000100: controls <= 12'b000010001000;//beq
                        6'b001000: controls <= 12'b100100000000;//addi
                        6'b000010: controls <= 12'b000000000100;//jump
                        6'b011111: controls <= 12'b000010001010;//ble
                        6'b111111: controls <= 12'b110101000001;//sw+
                        default:   controls <= 12'bxxxxxxxxxxxx;//illegal op
                endcase
        end
endmodule


// The controller has changed:
module controller(input logic[5:0] op, funct, input  logic zero,
input logic negative, output logic memtoreg, memwrite, logic pcsrc,
alusrc, output logic[1:0] regdst, output logic regwrite, output logic
jump, output logic blt, swplus, output logic[2:0] alucontrol, output
logic branch);
        logic [1:0] aluop;
        maindec md(op, memtoreg, memwrite, branch, alusrc, regdst,
        regwrite, jump, blt, swplus, aluop);
        aludec ad(funct, aluop, alucontrol);
        always_comb begin
                pcsrc = branch & zero | blt & negative;
        end
endmodule
```