DEPARTMENT OF COMPUTER ENGINEERING

COMPUTER ORGANIZATION CS 224

SPRING 2018

PRELIMINARY DESIGN REPORT

SECTION 1 / LAB 5

ELİF KEVSER ARSLAN

21400763

YASİN BALCANCI

21501109

1. RTL EXPRESSIONS:

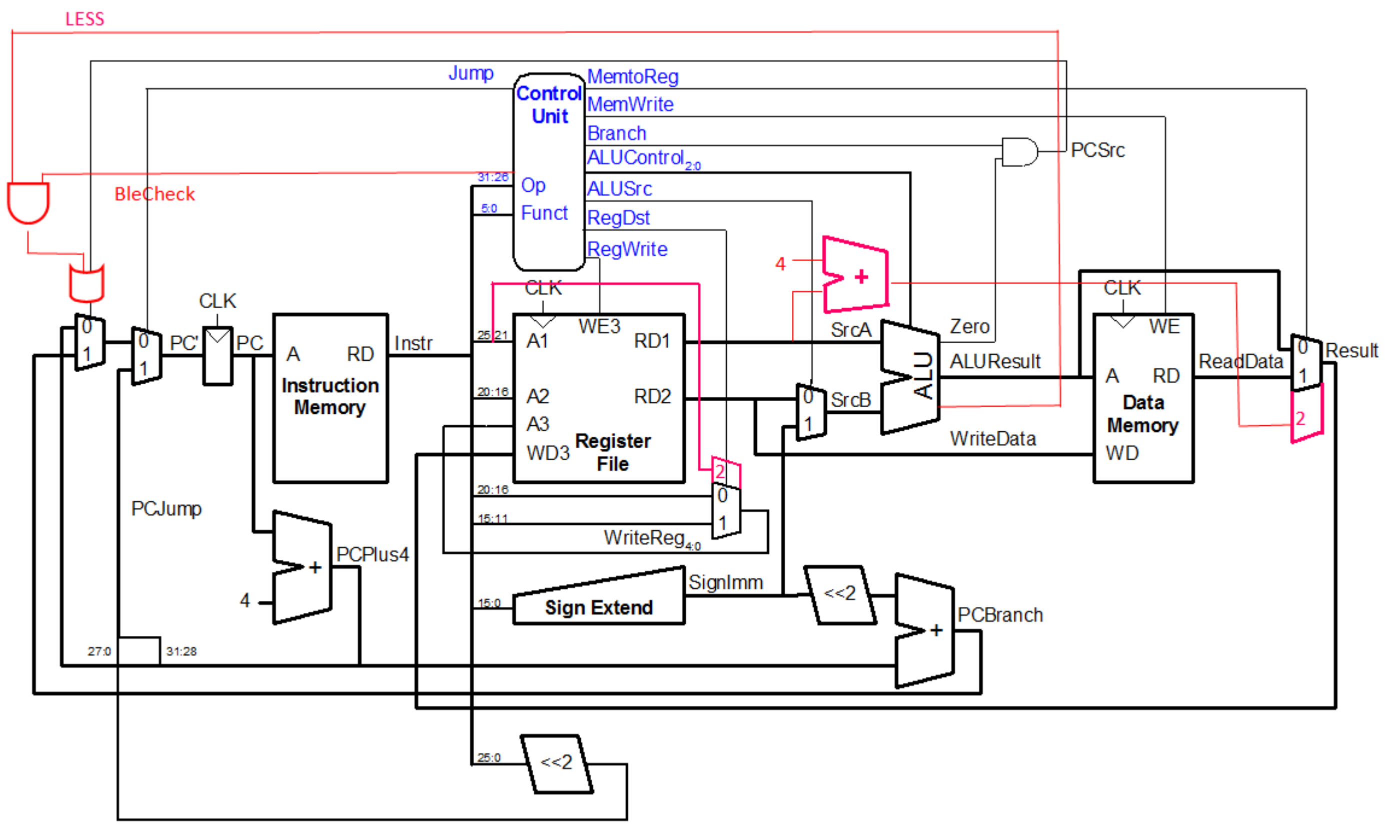
BLE:

```
IM[PC]

If(RF[rs] ←RF[rt])

        PC ←PC + 4;+ SignExt(immed)*4

else

        PC ←PC + 4;                         Calculate next address
```

SW+:

```
IM[PC];                             Fetch the instruction from memory
DM[RF[rs]+ SignExt(imm)] ← RF[rt]   Compute memory address
                                    and store data into memory
RF[rs] ← RF[rs] + 4;                increment by 4 (base address)
PC ←PC + 4;                         Calculate next address
```

2. DATAPATH



Explanation to datapath change:

ble:If BleCheck is 1, and LESS is 1 (coming from ALU), PC will assigned through Pcbranch by mux, also if BleCheck is 1 and most significant bit of ALUResult != 1 but zero flag is zero, datapath will work like ordinary beq instruction.

sw+: For this instruction, base address should be incremented by 4, that's why 1 adder is added. Then, to do update rs's new value, 2 muxes are expanded.

3. CONTROL TABLE

Instruction	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp	Jump	BleCheck
R type	1	1	0	0	0	0	10	0	0
Lw	1	0	1	0	0	1	00	0	0
Sw	0	x	1	0	1	x	00	0	0
Beq	0	x	0	1	0	x	01	0	0
Addi	1	0	1	0	0	0	00	0	0
J	0	x	x	x	0	x	xx	1	0
sw+	1	10	1	0	1	10	00	0	0
Ble	0	x	0	1	0	x	01	0	1

4.TEST PROGRAM IN MIPS

4.1 Test Program For BLE

```
.text

        #to check BLE, initialize registers

        #check ble for 2 cases (equality and being less)

        move $t0, $2

        move $t1, $2

        move $t2, $7


        #check first case

        #if they are equal, j to firstCheckPoint

        ble $t0, $t1, firstCheckPoint


        #else print error message

        la $a0, err

        li $v0, 4

        syscall




        firstCheckPoint:

                #check second case

                #if t1 is less than t2 (2<7), j to secondCheckPoint

                ble $t1, $t2, secondCheckPoint


                #else print error message

                la $a0, err

                li $v0, 4

                syscall
        secondCheckPoint:

                #print BLE is working correctly

                la $a0, bleWorks

                li $v0, 4

                syscall



.data

        bleWorks: .asciiz "BLE is working correctly!"

        err: .asciiz "It is not working!"
```

4.2 Test Program For SW+

```
#Test program for SW+

.text

        # put address of an array into t2 register

        #and call the sw+ function for testing

        la $t2, list

        move $t1, $5

        j StoreWordPlus


        StoreWordPlus:

                #store t1 contents into memory address t2

                sw $t1, ($t2)

                move $t3, $t2

                addi $t2, $t2, 4

        endOfFunc:
```

```
	#if t3 + 4 = t2 then it is correct

	#else wrong

	addi $t3, $t3, 4

	beq $t2, $t3, correct

	la $a0, err

	li $v0, 4

	syscall


	j end


	correct:

		la $a0, swWorks

		li $v0, 4

		syscall




	end:

		li $v0, 10

		syscall


.data

	swWorks: .asciiz "SW+ is working correctly!"

	err: .asciiz "It is not working!"

	list:	.word 3, 0, 1, 2
```

5. SYSTEMVERILOG MODULES

```
//*****************************

// CHANGED MODULES

//*****************************


/*

   memtoreg is now 2 bits.

   blecheck is a new 1 bit output that will be used only with ble.

   2 new control signal cases are added for sw+ and ble.

*/

module maindec (input logic[5:0] op,

                      output logic memtoreg,

                      output logic memwrite, branch,

                      output logic alusrc, regdst, regwrite, jump,

                      output logic[1:0] aluop,

                      output logic blecheck);


  logic [9:0] controls;


  assign {regwrite, regdst, alusrc, branch, memwrite,

            memtoreg,  aluop, jump, blecheck} = controls;



  always_comb

   case(op)

     6'b000000: controls <= 10'b1100001000; // R-type

     6'b100011: controls <= 10'b1010010000; // lw

     6'b101011: controls <= 10'b0x101x0000; // sw

     6'b000100: controls <= 10'b0x010x0100; // beq
```

```
      6'b001000: controls <= 10'b1010000000; // addi
      6'b000010: controls <= 10'b0xxx0xxx10; // j
      6'b111110: controls <= 10'b0x101x0000; // sw+
      6'b111111: controls <= 10'b0x010x0101; // ble
      default:   controls <= 10'bxxxxxxxxxx; // illegal op
    endcase
endmodule

/*
    less is a new 1 bit output that will indicate whether
    srcA is less then srcB or not.
*/
module alu(input  logic [31:0] a, b,
           input  logic [2:0]  alucont,
           output logic [31:0] result,
           output logic zero, less);

    always_comb
    begin
        case(alucont)
            3'b000: result = a & b;
            3'b001: result = a | b;
            3'b010: result = a + b;
            3'b110: result = a - b;
            3'b111: result = a < b;
            default: result = 0;
        endcase

        if (result == 0) zero = 1;
```

```
              else zero = 0;
        if (a < b) less = 1;
               else less = 0;
     end
endmodule
```