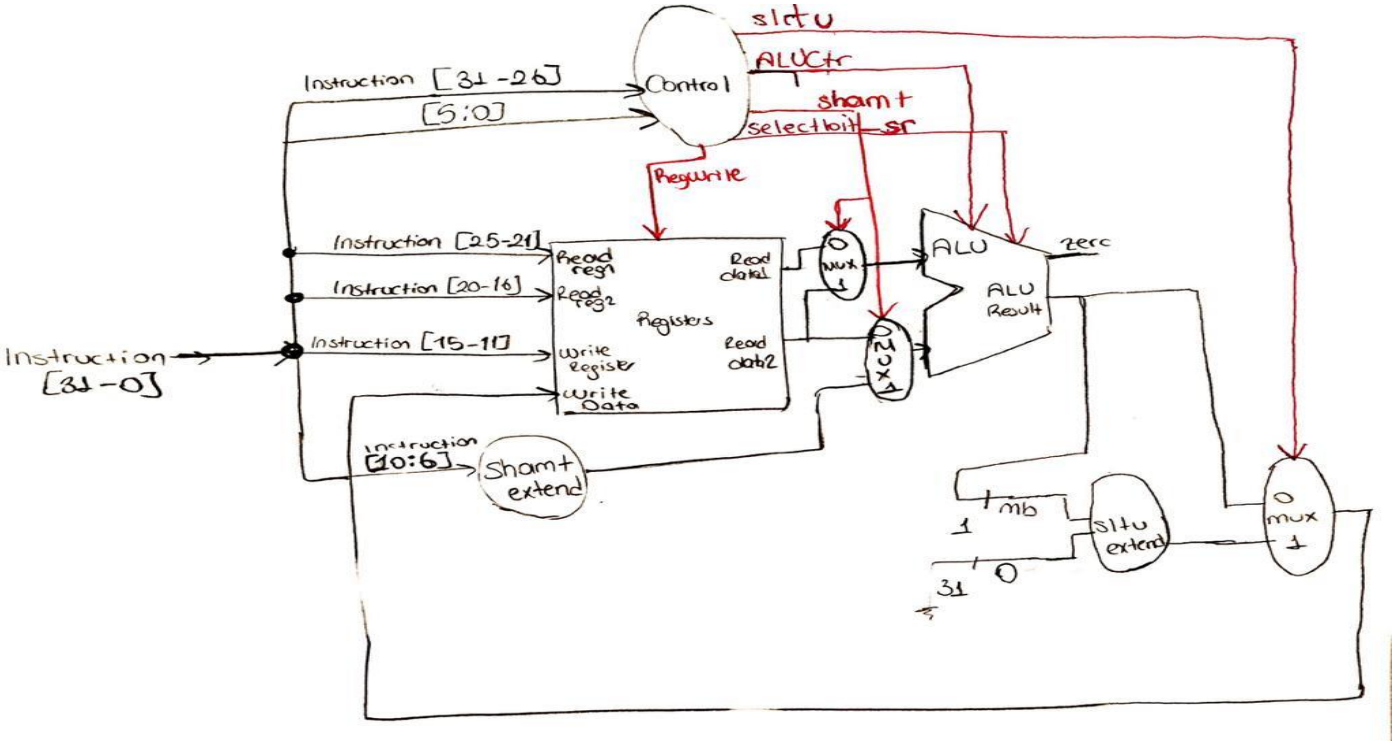
CSE 331 Computer Organization
Project 3 – R-type Single cycle MIPS with Structural Verilog

SEVGİ BAYANSALDUZ – 151044076

Firstly, I design the mips processor on the paper, the schematic design of the mips is the below.



The module mips32 and its description:

workspace/mips32.v*

```
module mips32(Instruction,clk, Result);
input [31:0] Instruction;
input clk;
output [31:0] Result;
//these wires is used for reading content from the register file
wire [31:0] read_data1[0:1];
wire [31:0] read_data2[0:1];
//write_data holds the function's result
//readData1 and readData2 holds the datas for alu32.
//s_extend holds the shamt as 32 bits.
//aluRes holds the result of alu32
//sltuRes holds the result of sltu.
wire [31:0] write_data,readData1,readData2,s_extend,aluRes,sltuRes;
//AluCtr holds the alu function's code.
wire [2:0]AluCtr;
wire signal_reg_write,slct_shamt,slct_sr,slct_sltu,Zero,C,V;

////////////Register file
//read_data1/2 is seted.
mips_registers mr1(read_data1[0],read_data2[0],Result,Instruction[25:21],Instruction[20:16],Instruction[15:11],signal_reg_write,clk);
//////////////////////////////////////////////////////////////
//control signals are set up
control_unit c_u1(AluCtr,signal_reg_write,slct_shamt,slct_sr,slct_sltu,Instruction[5:0]);
//////// datas for alu32 are seted.
//if slct_shamt is 1,the contents of the Rt will be selected as readData1.Otherwise the contents of the Rs will be selected.
mux_2_1_32bit mux(readData1,read_data1[0],read_data2[0],slct_shamt);
//shamt is extended
shamt_extend s_ex(s_extend,Instruction[10:6]);
//if slct_shamt is 1,s_extend will be selected as readData2.Otherwise the contents of the Rt will be selected.
mux_2_1_32bit mux1(readData2,read_data2[0],s_extend,slct_shamt);
////////////alu32
//aluRes holds the result of the alu.
alu32 alu(aluRes,C,V,Zero,readData1,readData2,AluCtr,slct_sr);
////////sltu
sltu_extend slt_mips(sltuRes,aluRes[31],AluCtr[0]);
/////////select the write data and result
//if slct_sltu is 1,Result will be selected as sltuRes.Otherwise alures will be selected.
mux_2_1_32bit mux2(Result,aluRes,sltuRes,slct_sltu);
endmodule
```

This module first calls the module mips_register to read datas from register. (clk is zero until Result is calculated. Then clk will be 1 and Result will be writed to write_reg.)
It calls control_unit to calculate signals according to function code.
Then it selects readData1 and ReadData 2 according to the signal shamt.
(Before ReadData2`s selection, shamt value will be extend as 32 bits.)
After ReadData`s selection, the module alu32 will be called.And the result of the alu32 will be kept.
Then sltu_exten function will be called to calculate sltu. Its result will be also kept.
After the alu`s result and sltu result calculation, Result will be selected according to the signal sltu.

The module mips_registers and its description:

workspace/mips32_testbench.v workspace/mips_registers.v

```
module mips_registers
( read_data_1, read_data_2, write_data, read_reg_1, read_reg_2, write_reg, signal_reg_write, clk );

   output [31:0] read_data_1, read_data_2;
   input [31:0] write_data;
   input [4:0] read_reg_1, read_reg_2, write_reg;
   input signal_reg_write, clk;

   reg [31:0] registers [31:0];

   //Datas from regiters are read.
   assign read_data_1 = registers[read_reg_1];
   assign read_data_2 = registers[read_reg_2];

   //write_data is written to the write_reg, when clk and signal_reg_write are 1 and the write register is not $zero.
   always @( posedge clk )
   if ( signal_reg_write && (write_reg != 5'b0))begin
      registers[write_reg] <= write_data;
      end
   //end

endmodule
```

This modul always read datas from registers.

If clk and signal_reg_write are 1 and write_reg is not $zero,this module will write the write_data to the write_reg. Otherwise write operation will not occur.

(clk is zero until Result is calculated. Then clk will be 1 and Result will be writed to write_reg.)

The schematic design of the control_unit:

	Function	Function Code	ALUcode
1)	add	10 0000 (20)hex	0 1 0
2)	addu	10 0001 (21)hex	0 1 0
3)	nor	10 0111 (27)hex	1 1 1
4)	or	10 0101 (25)hex	0 0 1
5)	sltu	10 1011 (2b)hex	1 0 0
6)	sll	00 0000 (00)hex	1 1 0
7)	srl	00 0010 (02)hex	1 0 1
8)	sub	10 0010 (22)hex	1 0 0
9)	subu	10 0011 (23)hex	1 0 0
10)	and	10 0100 (24)hex	0 0 0

$$ALUcode[2] = \overline{f5} + f1$$

$$ALUcode[1] = \overline{f1}.\overline{f2} + f1.f2$$

$$ALUcode[0] = f0.f2 + \overline{f5}.f1$$

$$shamt = \overline{f5}$$

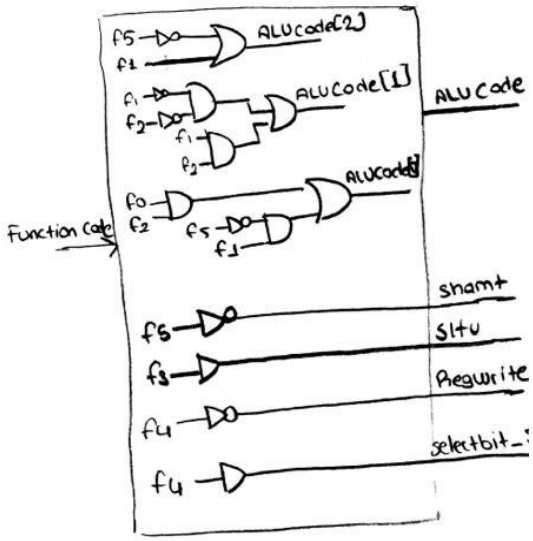
$$sltu = f3$$

Regwrite is always zero for these instructions so;

$$Regwrite = \overline{f4}$$

Selectbit_sr is always zero for this assignment so;

$$Selectbit\_sr = f4$$



The module control_unit and its description:

workspace/mips32_testbench.v workspace/control_unit.v

```
module control_unit(select_bits_ALU,regWrite,selectbit_shamt,selectbit_sr,selectbit_sltu, functi

input [5:0] function_code;
output [2:0] select_bits_ALU;
output regWrite,selectbit_shamt,selectbit_sr,selectbit_sltu;

wire [6:0]temp;

/////////////select_bits_ALU
//select_bits_ALU[2]=~f5+f1
not n1(temp[0],function_code[5]);//~f5
or o1(select_bits_ALU[2],temp[0],function_code[1]);
//select_bits_ALU[1]=~f2.~f1+f1.f2
not n2(temp[1],function_code[1]);//~f1
not n3(temp[2],function_code[2]);//~f2
and a2(temp[3],temp[1],temp[2]);//~f2.~f1

and aa2(temp[4],function_code[1],function_code[2]);//f1.f2

or o2(select_bits_ALU[1],temp[3],temp[4]);//~f2.~f1+f1.f2

//select_bits_ALU[0]=f0.f2+~f5.f1
and a3(temp[5],function_code[0],function_code[2]);//f0.f2
and a4(temp[6],temp[0],function_code[1]);//~f5.f1

or o3(select_bits_ALU[0],temp[5],temp[6]);//f0.f2+~f5.f1

//////////////////selectbit_sr
//selectbit_sr is always 0
buf b1(selectbit_sr,function_code[4]);
//////////////////selectbit_shamt
//// when function is sll or srl selectbit_shamt will be 1, otherwise 0.
not n4(selectbit_shamt,function_code[5]);
/////////////////regWrite
//regWrite is always 1 for R-type instructions.
not n5(regWrite,function_code[4]);
//////////////////////selectbit_sltu
///when function is sltu selectbit_sltu will be 1, otherwise 0.
//selectbit_sltu = f3
and a5(selectbit_sltu,function_code[3]);
endmodule
```

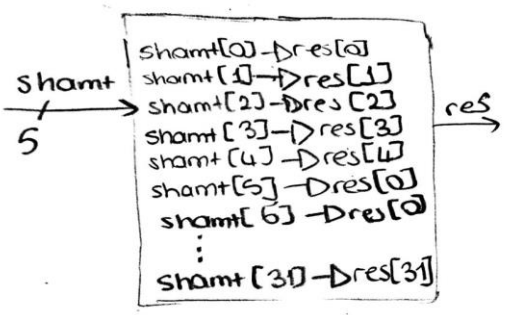
This module calculates control signals according to its schematic design.(The schematic design is above.)

The schematic design of the shamt extend, the shamt_extend module and its descripstion:

```
module shamt_extend(output[31:0] shamt_ext,input [4:0] shamt);

//set the first 5 bits with the shamt
//Then extends it with a random bit
buf b0(shamt_ext[0],shamt[0]);
buf b1(shamt_ext[1],shamt[1]);
buf b2(shamt_ext[2],shamt[2]);
buf b3(shamt_ext[3],shamt[3]);
buf b4(shamt_ext[4],shamt[4]);

//random bit, shamt[0].
buf b5(shamt_ext[5],shamt[0]);
buf b6(shamt_ext[6],shamt[0]);
buf b7(shamt_ext[7],shamt[0]);
buf b8(shamt_ext[8],shamt[0]);
buf b9(shamt_ext[9],shamt[0]);
buf b10(shamt_ext[10],shamt[0]);
buf b11(shamt_ext[11],shamt[0]);
buf b12(shamt_ext[12],shamt[0]);
buf b13(shamt_ext[13],shamt[0]);
buf b14(shamt_ext[14],shamt[0]);
```



This module takes the 5 bits shamt and extends it with a random bit as 32 bits.

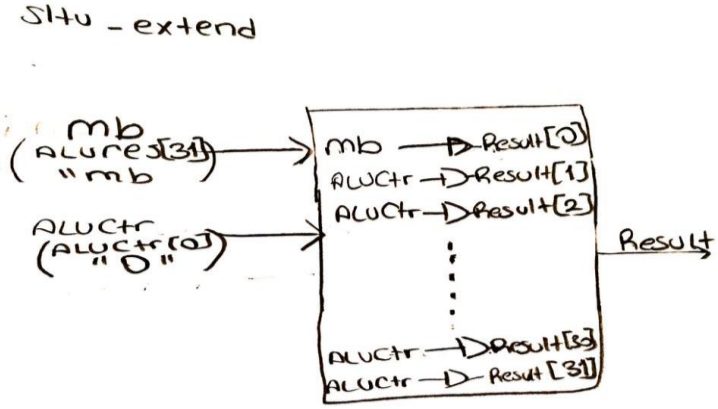
PS: Schematics design of the alu32 module is in the previous homework`s report. I just added one 2:1 mux for shift right operations.

The schematic design of the sltu extend, the sltu_extend module and its descripstion:

```
module sltu_extend(output [31:0] Result,input mb,input ALUCTR);
//AluCTR is equal to 0 when functon is sltu.

//Takes the most significant bit and extend it with zero.
buf b0(Result[0],mb);//Result[1]=mb
buf b1(Result[1],ALUCTR);//Result[1]=0
buf b2(Result[2],ALUCTR);//Result[2]=0 .....
buf b3(Result[3],ALUCTR);
buf b4(Result[4],ALUCTR);
buf b5(Result[5],ALUCTR);
buf b6(Result[6],ALUCTR);
buf b7(Result[7],ALUCTR);
buf b8(Result[8],ALUCTR);
buf b9(Result[9],ALUCTR);
buf b10(Result[10],ALUCTR);
buf b11(Result[11],ALUCTR);
```

This module takes a bit (most significant bit) and extends it with zero bit as 32 bits.



TESTBENCH

workspace/mips32_testbench.v

```
begin
 #15 clk2=~clk2;
end

initial begin
    clk=0;
    clk2=0;
    $readmemb("test_input.tv",testVectors);
    $readmemb("registers.mem",i0.mr1.registers);
    instr =32'd0;
    index=0;
end
always @(posedge clk2)
begin
instr<=testVectors[index];
end

always  @(posedge clk2)
begin
   if(index!=0 && index!=8'd11)begin
          $display("opcode = %b, rs = %b, rt = %b, rd = %b,
instr[20:16],instr[15:11],instr[10:6],instr[5:0],index);
          $display("result = %b", R);
          $display("$rs     = %b\n$rt     = %b",i0.readData1,
      end
   index <= index + 1;
   if (index==8'd11)
      begin
          $writememb("reg_chng.mem",i0.mr1.registers);
          $display("%d tests completed.",index-1);
          $finish;
      end
```

The file “test_inpu.tv” holds 10 instructions. (See the red arrow).
If you want to test your instructions you can change the file name int the specific line which is showed with the red arrow or you can change the content of the file “test_inpu.tv””.(PS: your number of instruction should be 10 for this test bench.)

If you want to change the content of registers you can change the file “register.mem” with your file or you can change the content of the file “register.mem”. (See the green arrow).

After instructions are called,contents of some registers will be change and these changes will be wrote to the file “reg_chng.mem”.

Test Result:

```
VSIM 6> step -current
# opcode = 000000, rs = 10000, rt = 10001, rd = 10001, shamt = 00000, funct = 100000 ,index==  1
# result = 00000000000000000000000000100001
# $rs    = 00000000000000000000000000010000
# $rt    = 00000000000000000000000000010001
# opcode = 000000, rs = 10100, rt = 10101, rd = 10011, shamt = 00000, funct = 100001 ,index==  2
# result = 00000000000000000000000000101001
# $rs    = 00000000000000000000000000010100
# $rt    = 00000000000000000000000000010101
# opcode = 000000, rs = 01001, rt = 01010, rd = 01101, shamt = 00000, funct = 100111 ,index==  3
# result = 11111111111111111111000000001111
# $rs    = 00000000000000000000000011110000
# $rt    = 00000000000000000000111100000000
# opcode = 000000, rs = 01000, rt = 01001, rd = 01010, shamt = 00000, funct = 100101 ,index==  4
# result = 00000000000000000000000011111111
# $rs    = 00000000000000000000000000001111
# $rt    = 00000000000000000000000011110000
# opcode = 000000, rs = 10110, rt = 10111, rd = 01111, shamt = 00000, funct = 101011 ,index==  5
# result = 00000000000000000000000000000001
# $rs    = 00000000000000000000000000010110
# $rt    = 00000000000000000000000000010111
# opcode = 000000, rs = 00000, rt = 10100, rd = 10000, shamt = 00010, funct = 000000 ,index==  6
# result = 00000000000000000000000001010000
# $rs    = 00000000000000000000000000010100
# $rt    = 00000000000000000000000000000010
# opcode = 000000, rs = 00000, rt = 10000, rd = 10000, shamt = 00011, funct = 000010 ,index==  7
# result = 00000000000000000000000000001010
# $rs    = 00000000000000000000000001010000
# $rt    = 11111111111111111111111111100011
# opcode = 000000, rs = 10000, rt = 10001, rd = 11000, shamt = 00000, funct = 100010 ,index==  8
# result = 11111111111111111111111111101001
# $rs    = 00000000000000000000000000001010
# $rt    = 00000000000000000000000000100001
# opcode = 000000, rs = 10110, rt = 01000, rd = 11001, shamt = 00000, funct = 100011 ,index==  9
# result = 00000000000000000000000000000111
# $rs    = 00000000000000000000000000010110
# $rt    = 00000000000000000000000000001111
# opcode = 000000, rs = 10101, rt = 01110, rd = 01000, shamt = 00000, funct = 100100 ,index== 10
# result = 00000000000000000000000000000100
# $rs    = 00000000000000000000000000010101
# $rt    = 00000000000000000000000000001100
#          10 tests completed.
# ** Note: $finish    : C:/Users/seyqi/Desktop/151044076/workspace/mips32_testbench.v(52)
```

reg_chng.mem

simulation/modelsim/reg_chng.mem

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips32_testbench/i0/mrl/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
00000000000000000000000000000100  10.
00000000000000000000000011110000
00000000000000000000000000111111  4.
00000000000000000000100000000000
00000000000000000000000000100000
11111111111111111111000000001111  3.
00000000000000000000000000001100
00000000000000000000000000000001  5.
00000000000000000000000000001010  7.
00000000000000000000000000100001  1.
00000000000000000000000000010010
00000000000000000000000000101001  2.
00000000000000000000000000010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000010111
11111111111111111111111111101001  8.
00000000000000000000000000000111  9.
00000000000000000000000000011010
00000000000000000000000000011011
00000000000000000000000000011100
00000000000000000000000000011101
00000000000000000000000000011110
00000000000000000000000000011111
```

PS: The destination register of the 6. instruction and the 7. instruction is the same,So This register holds the result of the 7. instruction because the 7. instruciton executes after the 6. instruciton.