CSE 331/503

Computer Organization

Final Project – MiniMIPS Design

1901042705

Zeynep Çiğdem Parlatan

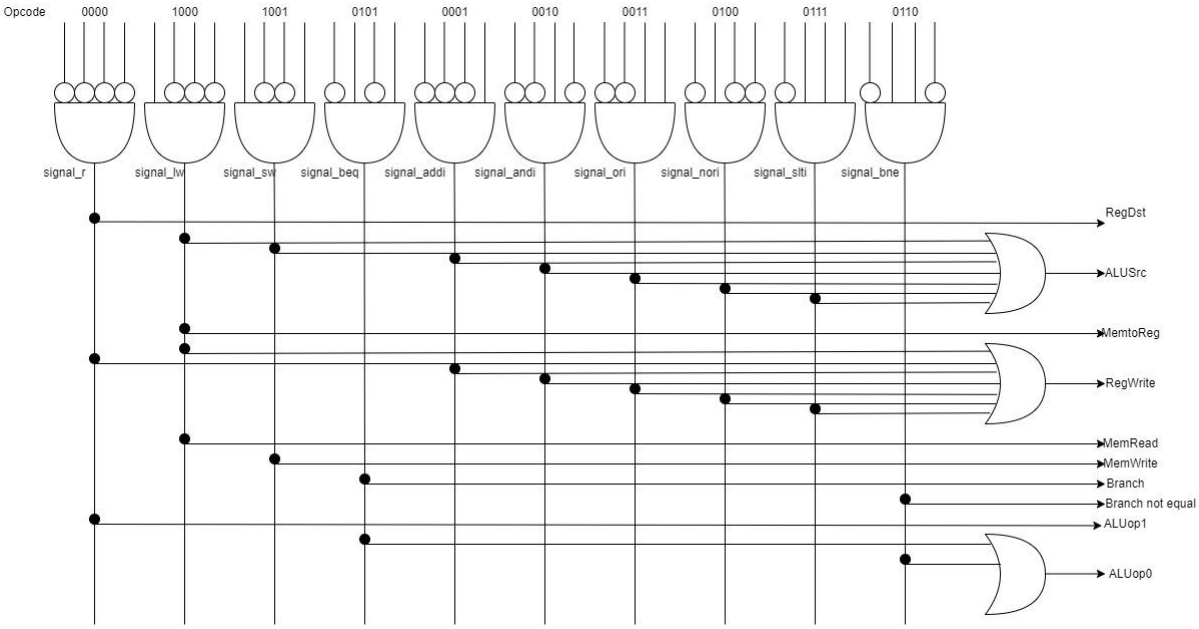
Control Unit module

Main Control

- In the instructions I specified which outputs were 1 and which signals were zero.

opcode	0000	1000	1001	0101	0001	0010	0011	0100	0111	0110
	R-type	lw	sw	beq	addi	andi	ori	nori	slti	bne
RegDst	1	0	x	x	0	0	0	0	0	X
ALUSrc	0	1	1	0	1	1	1	1	1	0
MemtoReg	0	1	x	x	0	0	0	0	0	X
RegWrite	1	1	0	0	1	1	1	1	1	0
MemRead	0	1	0	0	0	0	0	0	0	0
MemWrite	0	0	1	0	0	0	0	0	0	0
Branch	0	0	0	1	0	0	0	0	0	0
Branch not equal	0	0	0	0	0	0	0	0	0	1
ALUop(symbolic)	R-type	Add	Add	Subtract	Add	And	Or	Nor	Slti	Subtract
ALUop1	1	0	0	0	0	0	0	0	0	0
Aluop0	0	0	0	1	0	0	0	0	0	1

- I produced a signal that was 1 each time a different type of instruction came in. And while designing each output, I applied an or gate to the signals of that instruction, which should be 1 in each instruction. And accordingly I designed the main control unit.



RegDst=signal_r

ALUSrc = signal_lw + signal_sw + signal_addi + signal_andi + signal_ori + signal_nori + signal_slti

MemtoReg = signal_lw

RegWrite = signal_r + signal_lw + signal_addi + signal_andi + signal_ori + signal_nori + signal_slti

MemRead = signal_lw

MemWrite = signal_sw

Branch = signal_beq

Branch not equal = signal_bne

ALUop1 = signal_r

ALUop0=signal_beq + signal_bne

- **Main control test**

```
initial
  begin

      opcode = 4'b0000;        //R-type
      #10
         opcode = 4'b0001;     //addi
      #10
         opcode = 4'b0010;     //andi
      #10
         opcode = 4'b0011;     //ori
      #10
         opcode = 4'b0100;     //nori
      #10
         opcode = 4'b0101;     //beq
      #10
         opcode = 4'b0110;     //bne
      #10
         opcode = 4'b0111;     //slti
      #10
         opcode = 4'b1000;     //lw
      #10
         opcode = 4'b1001;     //sw

   end
```

```
Transcript
VSIM 5> step -current
# Testing the main control
#  opcode=0000 => time =  0, RegDst =1, ALUSrc=0, MemtoReg=0, RegWrite=1, MemRead=0, MemWrite=0, Branch=0, Branchne=0, ALUop1=1, ALUop0=0
# Testing the main control
#  opcode=0001 => time = 10, RegDst =0, ALUSrc=1, MemtoReg=0, RegWrite=1, MemRead=0, MemWrite=0, Branch=0, Branchne=0, ALUop1=0, ALUop0=0
# Testing the main control
#  opcode=0010 => time = 20, RegDst =0, ALUSrc=1, MemtoReg=0, RegWrite=1, MemRead=0, MemWrite=0, Branch=0, Branchne=0, ALUop1=0, ALUop0=0
# Testing the main control
#  opcode=0011 => time = 30, RegDst =0, ALUSrc=1, MemtoReg=0, RegWrite=1, MemRead=0, MemWrite=0, Branch=0, Branchne=0, ALUop1=0, ALUop0=0
# Testing the main control
#  opcode=0100 => time = 40, RegDst =0, ALUSrc=1, MemtoReg=0, RegWrite=1, MemRead=0, MemWrite=0, Branch=0, Branchne=0, ALUop1=0, ALUop0=0
# Testing the main control
#  opcode=0101 => time = 50, RegDst =0, ALUSrc=0, MemtoReg=0, RegWrite=0, MemRead=0, MemWrite=0, Branch=1, Branchne=0, ALUop1=0, ALUop0=1
# Testing the main control
#  opcode=0110 => time = 60, RegDst =0, ALUSrc=0, MemtoReg=0, RegWrite=0, MemRead=0, MemWrite=0, Branch=0, Branchne=1, ALUop1=0, ALUop0=1
# Testing the main control
#  opcode=0111 => time = 70, RegDst =0, ALUSrc=1, MemtoReg=0, RegWrite=1, MemRead=0, MemWrite=0, Branch=0, Branchne=0, ALUop1=0, ALUop0=0
# Testing the main control
#  opcode=1000 => time = 80, RegDst =0, ALUSrc=1, MemtoReg=1, RegWrite=1, MemRead=1, MemWrite=0, Branch=0, Branchne=0, ALUop1=0, ALUop0=0
# Testing the main control
#  opcode=1001 => time = 90, RegDst =0, ALUSrc=1, MemtoReg=0, RegWrite=0, MemRead=0, MemWrite=1, Branch=0, Branchne=0, ALUop1=0, ALUop0=0
```

ALU Control

- Using the truth table, I found separate boolean expressions for the ALU control bits and designed the ALU control block accordingly.(While finding the control bits, I could not set the I types accordingly.)

Inst. opcode	P1 P0 ALUop	Function F2F1F0	Desired Alu Action	C2 C1 C0 ALU control
LW	00	XXX	add	000
SW	00	XXX	add	000
Beq	01	XXX	subtract	010
Bne	01	XXX	subtract	010
R-type(and)	10	000	and	110
R-type(add)	10	001	add	000
R-type(sub)	10	010	subtract	010
R-type(xor)	10	011	xor	001
R-type(nor)	10	100	nor	101
R-type(or)	10	101	or	111
I-type(addi)	00	XXX	add	000
I-type(andi)	00	XXX	and	110
I-type(ori)	00	XXX	or	111
I-type(nori)	00	XXX	nor	101
I-type(slti)	00	XXX	Set on less than	100

C2 = P1P0’F2’F1’F0’ + P1P0’F2F1’F0’ + P1P0’F2F1’F0

Inst. opcode	P1 P0 ALUop	Function F2F1F0	Desired Alu Action	C2 C1 C0 ALU control
LW	00	XXX	add	000
SW	00	XXX	add	000
Beq	01	XXX	subtract	010
Bne	01	XXX	subtract	010
R-type(and)	10	000	and	110
R-type(add)	10	001	add	000
R-type(sub)	10	010	subtract	010
R-type(xor)	10	011	xor	001
R-type(nor)	10	100	nor	101
R-type(or)	10	101	or	111
I-type(addi)	00	XXX	add	000
I-type(andi)	00	XXX	and	110
I-type(ori)	00	XXX	or	111
I-type(nori)	00	XXX	nor	101
I-type(slti)	00	XXX	Set on less than	100

C1 = P0 + P1P0’F2’F1’F0’ + P1P0’F2’F1F0’ + P1P0’F2F1’F0

Inst. opcode	P1 P0 ALUop	Function F2F1F0	Desired Alu Action	C2 C1 C0 ALU control
LW	00	XXX	add	000
SW	00	XXX	add	000
Beq	01	XXX	subtract	010
Bne	01	XXX	subtract	010
R-type(and)	10	000	and	110
R-type(add)	10	001	add	000
R-type(sub)	10	010	subtract	010
R-type(xor)	10	011	xor	001
R-type(nor)	10	100	nor	101
R-type(or)	10	101	or	111
I-type(addi)	00	XXX	add	000
I-type(andi)	00	XXX	and	110
I-type(ori)	00	XXX	or	111
I-type(nori)	00	XXX	nor	101
I-type(slti)	00	XXX	Set on less than	100

C0 = P1P0'F2'F1F0 + P1P0'F2F1'F0' + P1P0'F2F1'F0

- **ALU control test**

```
initial
  begin

      ALUop = 2'b00;              //lw-sw
      func = 3'b000;
      #10
         ALUop = 2'b01;           //beq-bne
         func = 3'b000;
      #10
         ALUop = 2'b10;           //and
         func = 3'b000;
      #10
         ALUop = 2'b10;           //add
         func = 3'b001;
      #10
         ALUop = 2'b10;           //sub
         func = 3'b010;
      #10
         ALUop = 2'b10;           //xor
         func = 3'b011;
      #10
         ALUop = 2'b10;           //nor
         func = 3'b100;
      #10
         ALUop = 2'b10;           //or
         func = 3'b101;

  end
```

```
sim:/ALU_control_testbench/ALUctr
VSIM 5> step -current
# Testing the ALU control
#  ALUop=00 func=000 => time =  0, ALUctr =000
# Testing the ALU control
#  ALUop=01 func=000 => time = 10, ALUctr =010
# Testing the ALU control
#  ALUop=10 func=000 => time = 20, ALUctr =110
# Testing the ALU control
#  ALUop=10 func=001 => time = 30, ALUctr =000
# Testing the ALU control
#  ALUop=10 func=010 => time = 40, ALUctr =010
# Testing the ALU control
#  ALUop=10 func=011 => time = 50, ALUctr =001
# Testing the ALU control
#  ALUop=10 func=100 => time = 60, ALUctr =101
# Testing the ALU control
#  ALUop=10 func=101 => time = 70, ALUctr =111
```

Register module

- While reading the registers Read_data_1(Rs) and Read_data_2(Rt), I did not do any signal checking, they are read every cycle. While writing to the register, I checked the signal_reg_write signal and the number of the register to be written (since we cannot write it into the zero register).
- In testbench, I checked the register read and write operations by writing the number of registers to be read and written, the write signal, and the data to be written.

- **Register test**

Register addresses to be read and write

Register contents

```
initial
   begin
      read_reg_1 = 3'b010;        //register[2]
      read_reg_2 = 3'b011;        //register[3]
      write_reg = 3'b110;         //register[6]
      signal_reg_write = 1'b1;
      write_data = 32'b00000000000000000000000000000011;
      $writememb("registers_out.txt", try.mips_registers.registers);

      #5
      read_reg_1 = 3'b001;        //register[1]
      read_reg_2 = 3'b010;        //register[2]
      write_reg = 3'b011;         //register[3]
      signal_reg_write = 1'b1;
      write_data = 32'b00000000000000000000000000000000;
      $writememb("registers_out.txt", try.mips_registers.registers);

      #5
      read_reg_1 = 3'b111;        //register[7]
      read_reg_2 = 3'b100;        //register[4]
      write_reg = 3'b001;         //register[1]
      signal_reg_write = 1'b1;
      write_data = 32'b00000000000000000000000000000111;
      $writememb("registers_out.txt", try.mips_registers.registers);

   end
```

simulation/modelsim/registers.txt | mips_registers.v

```
1  00000000000000000000000000000000
2  00000000000000000000000000000001
3  00000000000000000000000000000010
4  00000000000000000000000000000011
5  00000000000000000000000000000100
6  00000000000000000000000000000101
7  00000000000000000000000000000110
8  00000000000000000000000000000111
```

```
sim:/mips_registers_testbench/read_data_2
VSIM 5> step -current
# Testing the mips registers
#  Rs and Rt: time =  0, read_data_1 =00000000000000000000000000000010, read_data_2=00000000000000000000000000000011
# Testing the mips registers
#  Rs and Rt: time =  5, read_data_1 =00000000000000000000000000000001, read_data_2=00000000000000000000000000000010
# Testing the mips registers
#  Rs and Rt: time = 10, read_data_1 =00000000000000000000000000000111, read_data_2=00000000000000000000000000000100
```

Writing the data to a different file(register_out) according to the clk being in posedge

simulation/modelsim/registers.txt | simulation/modelsim/registers_out.txt

```
1  // memory data file (do not edit the following lin
2  // instance=/mips_registers_testbench/try/registe
3  // format=bin addressradix=h dataradix=b version=
4  00000000000000000000000000000000
5  00000000000000000000000000000001
6  00000000000000000000000000000010
7  00000000000000000000000000000000
8  00000000000000000000000000000100
9  00000000000000000000000000000101
10 00000000000000000000000000000011
11 00000000000000000000000000000111
12
```

Data memory module

- If the signal_mem_read signal is 1, I did a read operation from memory, and if the signal_mem_write signal is one, I did a write operation to memory.

  - **Data memory test**

When the signal_mem_read signal is 1, the numbers at the 8th and 14th addresses are read from the memory.

```
initial
  begin

    signal_mem_write = 1'b0;
    signal_mem_read = 1'b1;
    address = 32'b00000000000000000000000000001000;
    write_data = 32'b00000000000000000000000000000110;
    $writememb("data_out.txt", tryData.mips_data_memory.memregisters);

    #5
    signal_mem_write = 1'b1;
    signal_mem_read = 1'b0;
    address = 32'b00000000000000000000000000001100;
    write_data = 32'b00000000000000000000000000001110;
    $writememb("data_out.txt", tryData.mips_data_memory.memregisters);

    #5
    signal_mem_write = 1'b0;
    signal_mem_read = 1'b1;
    address = 32'b00000000000000000000000000001110;
    write_data = 32'b00000000000000000000000000000110;
    $writememb("data_out.txt", tryData.mips_data_memory.memregisters);

  end
```

simulation/modelsim/data.txt | simulation/modelsim/data_out.txt

```
1  00000000000000000000000000000000
2  00000000000000000000000000010100
3  00000000000000000000000000000010
4  00000000000000000000000000000011
5  00000000000000000000000000000100
6  00000000000000000000000000000101
7  00000000000000000000000000000110
8  00000000000000000000000000000111
9  00000000000000000000000000001000
10 00000000000000000000000000001001
11 00000000000000000000000000001010
12 00000000000000000000000000001011
13 00000000000000000000000000001100
14 00000000000000000000000000001101
15 00000000000000000000000000001110
16 00000000000000000000000000001111
17 00000000000000000000000000010000
18 00000000000000000000000000010001
19 00000000000000000000000000010010
```

```
sim:/mips_data_memory_testbench/read_data
VSIM 5> step -current
# Testing the data memory
#  Read_data: time =  0, read_data =00000000000000000000000000001000
# Testing the data memory
#  Read_data: time = 10, read_data =00000000000000000000000000001110
```

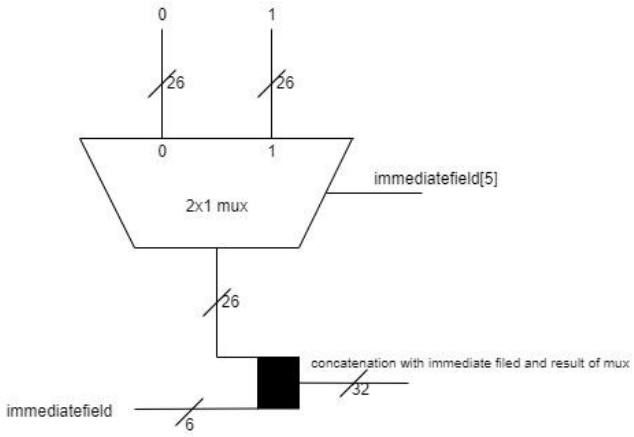
When the signal_mem_write signal is 1, a new value is written to the 12th address in the memory.

simulation/modelsim/data.txt | simulation/modelsim/data_out.txt

```
1  // memory data file (do not edit the follo
2  // instance=/mips_data_memory_testbench/tr
3  // format=bin addressradix=h dataradix=b v
4  00000000000000000000000000000000
5  00000000000000000000000000010100
6  00000000000000000000000000000010
7  00000000000000000000000000000011
8  00000000000000000000000000000100
9  00000000000000000000000000000101
10 00000000000000000000000000000110
11 00000000000000000000000000000111
12 00000000000000000000000000001000
13 00000000000000000000000000001001
14 00000000000000000000000000001010
15 00000000000000000000000000001011
16 00000000000000000000000000000001
17 00000000000000000000000000001101
18 00000000000000000000000000001110
19 00000000000000000000000000001111
20 00000000000000000000000000010000
21 00000000000000000000000000010001
22 00000000000000000000000000010010
```

Sign Extend module

- While designing Sign extended block, I did it as follows. I looked at the most significant bit of Immediatefield, that is whether the number is positive or negative, and according to that, I selected with mux whether to put zero or one.
- To concatenate the multiplexer's result and immediatefield, using the or gate I first put the immediatefield on the first 6 bits of the number, then put the multiplexer's result on the rest.



- **Sign extend test**

```
# Top level modules:
#       my_signExtend_testbench
ModelSim> vsim work.my_signExtend_testbench
# vsim work.my_signExtend_testbench
# Loading work.my_signExtend_testbench
# Loading work.my_signExtend
add wave -position insertpoint  \
sim:/my_signExtend_testbench/immediatefield \
sim:/my_signExtend_testbench/num
VSIM 5> step -current
# Testing the sign extend module
#  sign extend : time =  0, 6 bit immediatefield =010001, 32 bit num=00000000000000000000000000010001
# Testing the sign extend module
#  sign extend : time =  5, 6 bit immediatefield =100011, 32 bit num=11111111111111111111111111100011
```

ALU module

- I used the ALU I designed in the last assignment. I designed each operation in the ALU in a separate module and then created the ALU using an 8x1 mux.

- **ALU test**

```
initial
  begin
    i1= 32'h10001001;
    i2= 32'h00101000;
    select=3'b000;                //add
    carri=3'b000;
    #5 i1= 32'h10001001;
      i2= 32'h00101000;
       select=3'b001;             //xor

    #5 i1= 32'h10001001;
      i2= 32'h00101000;
       select=3'b010;             //sub

     #5 i1= 32'h10001001;
       i2= 32'h00101000;
        select= 3'b100;           //slt

     #5 i1= 32'h10001001;
       i2= 32'h00101000;
        select = 3'b101;          //nor
     #5 i1= 32'h10001001;
       i2= 32'h00101000;
        select = 3'b110;          //and

     #5 i1= 32'h10001001;
       i2= 32'h00101000;
        select = 3'b111;          //or
```

```
sim:/my_alu_testbench/out
VSIM 5> step -current
# Testing the 32-bit alu
#  32-bit alu: time =  0, i1 =10001001, i2=101000, select=000, out=10102001
# Testing the 32-bit alu
#  32-bit alu: time =  5, i1 =10001001, i2=101000, select=001, out=10100001
# Testing the 32-bit alu
#  32-bit alu: time = 10, i1 =10001001, i2=101000, select=010, out=ff00001
# Testing the 32-bit alu
#  32-bit alu: time = 15, i1 =10001001, i2=101000, select=100, out=0
# Testing the 32-bit alu
#  32-bit alu: time = 20, i1 =10001001, i2=101000, select=101, out=efefeffe
# Testing the 32-bit alu
#  32-bit alu: time = 25, i1 =10001001, i2=101000, select=110, out=1000
# Testing the 32-bit alu
#  32-bit alu: time = 30, i1 =10001001, i2=101000, select=111, out=10101041
```

MiniMIPS

- I tried to design miniMIPS by combining all necessary modules. My miniMips calculates the result correctly according to the instruction I gave. But I had a problem writing the result to the register.(It finds the write_data correct, but there is a problem while writing to the register.) I guess I didn't set the clk properly. That's why I couldn't write the correct results to the register_out file in miniMips_module testbench. (But writing result to register works correctly in mips_registers_module and mips_registers_testbench.)

- **MiniMIPS Test**

İnside Register

simulation/mod.../registers.txt | Min...S.v

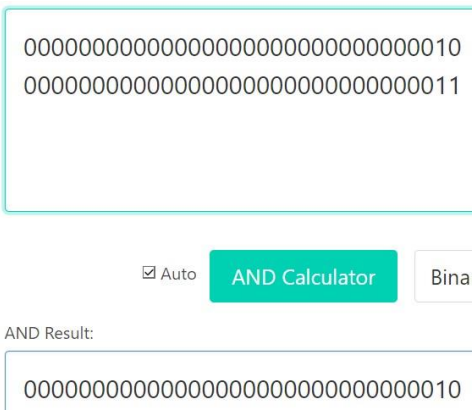
```
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
```

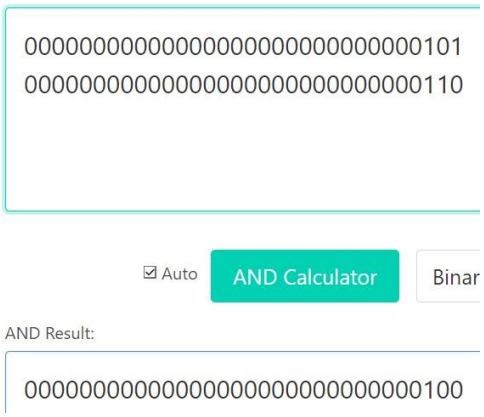
✓ **And**

```
//****************************AND*****************************//

instruction = 16'b0000010011001000;        //and $R1, $R2, $R3
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

#5
instruction = 16'b0000101110111000  ;      //and $R7, $R5, R$6
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

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

```
sim:/MiniMIPS_testbench/result
VSIM 5> step -current
# Testing the miniMIPS
#  time= 0,ALUctr=110, Result=00000000000000000000000000000010, writedata=00000000000000000000000000000010,
# Testing the miniMIPS
#  time= 5,ALUctr=110, Result=00000000000000000000000000000100, writedata=00000000000000000000000000000100,
```

I showed the accuracy of the results in the online calculator.



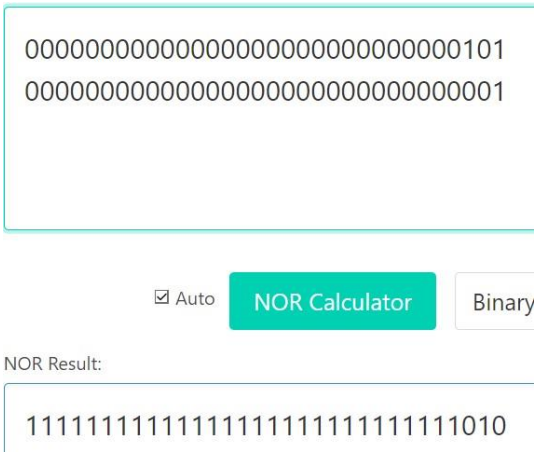


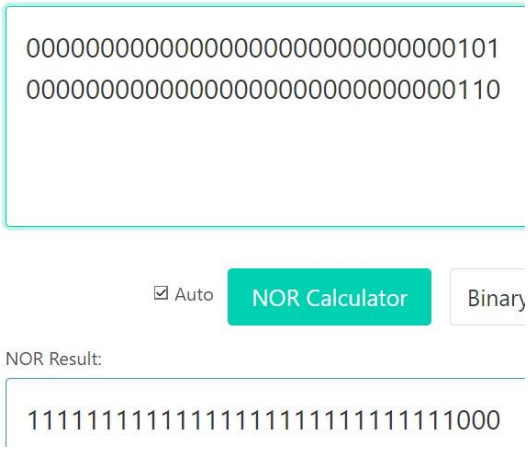
✓ Nor

```
//****************************NOR*****************************//
#5
instruction = 16'b0000101001010100;        //nor $R2, $R5, $R1
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

#5
instruction = 16'b0000101110111100;     //nor $R7, $R5, R$6
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

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

```
# Testing the miniMIPS
#  time=10,ALUctr=101, Result=11111111111111111111111111111010, writedata=11111111111111111111111111111010,
# Testing the miniMIPS
#  time=15,ALUctr=101, Result=11111111111111111111111111111000, writedata=11111111111111111111111111111000,
```

I showed the accuracy of the results in the online calculator.

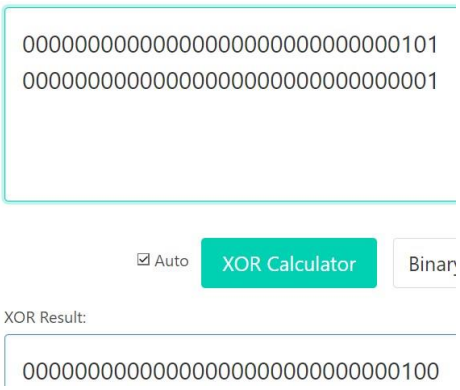


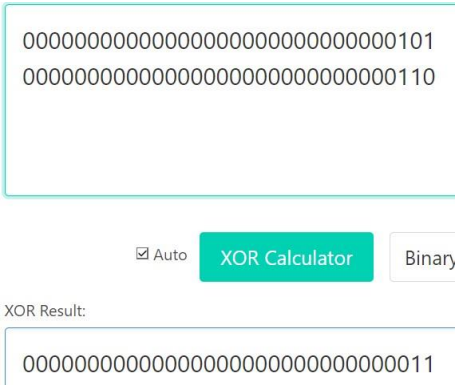


✓ XOR

```
//****************************XOR*****************************//
#5
instruction = 16'b0000101001010011;        //xor $R2, $R5, $R1
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

#5
instruction = 16'b0000101110111011;     //xor $R7, $R5, R$6
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

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

```
# Testing the miniMIPS
#  time=20,ALUctr=001, Result=00000000000000000000000000000100, writedata=00000000000000000000000000000100,
# Testing the miniMIPS
#  time=25,ALUctr=001, Result=00000000000000000000000000000011, writedata=00000000000000000000000000000011,
```

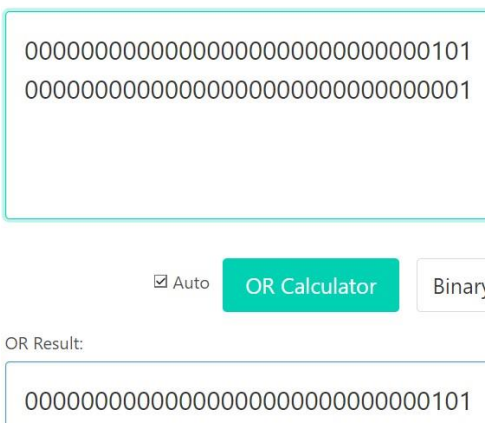


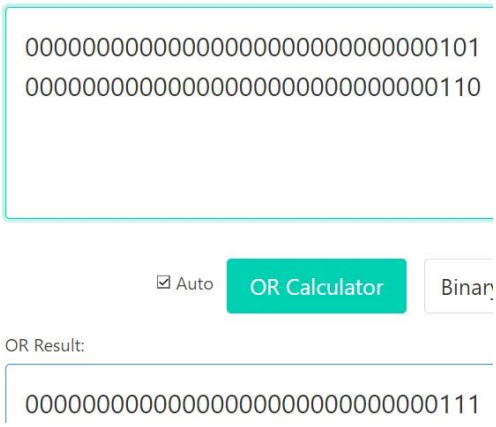


✓ OR

```
//****************************OR*****************************//
#5
instruction = 16'b0000101001010101;        //or $R2, $R5, $R1
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

#5
instruction = 16'b0000101110111101;     //or $R7, $R5, R$6
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

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

```
# Testing the miniMIPS
#  time=30,ALUctr=111, Result=00000000000000000000000000000101, writedata=00000000000000000000000000000101,
# Testing the miniMIPS
#  time=35,ALUctr=111, Result=00000000000000000000000000000111, writedata=00000000000000000000000000000111,
```





✓ ADD

```
//****************************ADD*****************************//
#5
instruction = 16'b0000101001010001;        //add $R2, $R5, $R1
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

#5
instruction = 16'b0000101110111001;     //add $R7, $R5, R$6
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

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

```
# Testing the miniMIPS
#  time=40,ALUctr=000, Result=00000000000000000000000000000110, writedata=00000000000000000000000000000110,
# Testing the miniMIPS
#  time=45,ALUctr=000, Result=00000000000000000000000000001011, writedata=00000000000000000000000000001011,
```

Result

Binary value:

00000000000000000000000000000101 + 00000000000000000000000000000001
= **0110**

Result

Binary value:

00000000000000000000000000000101 + 00000000000000000000000000000110
= **01011**

✓ SUB

```
//*****************************SUB*****************************//
#5
instruction = 16'b0000101001010010;       //sub $R2, $R5, $R1
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

#5
instruction = 16'b0000101110111010;    //sub $R7, $R5, R$6
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

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

```
# Testing the miniMIPS
#  time=50,ALUctr=010, Result=00000000000000000000000000000100, writedata=00000000000000000000000000000100,
# Testing the miniMIPS
#  time=55,ALUctr=010, Result=11111111111111111111111111111111, writedata=11111111111111111111111111111111,
```

Result

Binary value:

00000000000000000000000000000101 – 00000000000000000000000000000001
= **100**

Result

Binary value:

00000000000000000000000000000101 – 00000000000000000000000000000110
= **-1**

✓ SW

```
//*****************************SW*****************************//
#5
instruction = 16'b1001010001000010;       //sw $R1,2($R2)
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.mips_data_memory.memregisters);

#5
instruction = 16'b1001010110000001;    //sw $R6, 1($R2)
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.mips_data_memory.memregisters);

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

```
# Testing the miniMIPS
#  time=70,ALUctr=000, Result=00000000000000000000000000000100, writedata=00000000000000000000000000000100,
# Testing the miniMIPS
#  time=75,ALUctr=000, Result=00000000000000000000000000000011, writedata=00000000000000000000000000000011,
```

✓ LW

```
//****************************LW*****************************//
#5
instruction = 16'b1000101001000010;        //lw $R2, $R5, 2($R5)
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

#5
instruction = 16'b1000101110000110;     //lw $R7, $R5,6($R5)
//$readmemb("registers.txt", try_mips.register.mips_registers.registers);
$writememb("registers_out.txt", try_mips.MiniMIPS.register.mips_registers.registers);
$writememb("data_out.txt", try_mips.Datamem.memregisters);

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

```
# Testing the miniMIPS
#  time=60,ALUctr=000, Result=00000000000000000000000000000111, writedata=00000000000000000000000000000111,
# Testing the miniMIPS
#  time=65,ALUctr=000, Result=00000000000000000000000000001011, writedata=00000000000000000000000000001011,
```

Data memory

simulation...isters.txt | simulat...ata.txt | M...v

```
1  00000000000000000000000000000000
2  00000000000000000000000000010100
3  00000000000000000000000000000010
4  00000000000000000000000000000011
5  00000000000000000000000000000100
6  00000000000000000000000000000101
7  00000000000000000000000000000110
8  00000000000000000000000000000111
9  00000000000000000000000000001000
10 00000000000000000000000000001001
11 00000000000000000000000000001010
12 00000000000000000000000000001011
13 00000000000000000000000000001100
14 00000000000000000000000000001101
15 00000000000000000000000000001110
16 00000000000000000000000000001111
17 00000000000000000000000000010000
18 00000000000000000000000000010001
19 00000000000000000000000000010010
```

---

miniMIPS

```
sim:/MiniMIPS_testbench/instruction \
sim:/MiniMIPS_testbench/result
VSIM 5> step -current
# Testing the miniMIPS
#  time= 0,ALUctr=110, Result=00000000000000000000000000000010, writedata=00000000000000000000000000000010,
# Testing the miniMIPS
#  time= 5,ALUctr=110, Result=00000000000000000000000000000100, writedata=00000000000000000000000000000100,
# Testing the miniMIPS
#  time=10,ALUctr=101, Result=11111111111111111111111111111010, writedata=11111111111111111111111111111010,
# Testing the miniMIPS
#  time=15,ALUctr=101, Result=11111111111111111111111111111000, writedata=11111111111111111111111111111000,
# Testing the miniMIPS
#  time=20,ALUctr=001, Result=00000000000000000000000000000100, writedata=00000000000000000000000000000100,
# Testing the miniMIPS
#  time=25,ALUctr=001, Result=00000000000000000000000000000011, writedata=00000000000000000000000000000011,
# Testing the miniMIPS
#  time=30,ALUctr=111, Result=00000000000000000000000000000101, writedata=00000000000000000000000000000101,
# Testing the miniMIPS
#  time=35,ALUctr=111, Result=00000000000000000000000000000111, writedata=00000000000000000000000000000111,
# Testing the miniMIPS
#  time=40,ALUctr=000, Result=00000000000000000000000000000110, writedata=00000000000000000000000000000110,
# Testing the miniMIPS
#  time=45,ALUctr=000, Result=00000000000000000000000000001011, writedata=00000000000000000000000000001011,
# Testing the miniMIPS
#  time=50,ALUctr=010, Result=00000000000000000000000000000100, writedata=00000000000000000000000000000100,
# Testing the miniMIPS
#  time=55,ALUctr=010, Result=11111111111111111111111111111111, writedata=11111111111111111111111111111111,
# Testing the miniMIPS
#  time=60,ALUctr=000, Result=00000000000000000000000000000111, writedata=00000000000000000000000000000111,
# Testing the miniMIPS
#  time=65,ALUctr=000, Result=00000000000000000000000000001011, writedata=00000000000000000000000000001011,
# Testing the miniMIPS
#  time=70,ALUctr=000, Result=00000000000000000000000000000100, writedata=00000000000000000000000000000100,
# Testing the miniMIPS
#  time=75,ALUctr=000, Result=00000000000000000000000000000011, writedata=00000000000000000000000000000011,
```