

ASSIGNMENT 4 REPORT

The Modules I Designed

1) 1 bit FULL ADDER and 32 bit Adder

I design 32 bit Adder with full adders.

32 bit Adder Testbench:

1. Test = $15 + 13 = 28$

2. Test $6 + 7 = 13$

```
Transcript
#       adder_32_bit
# vlog -vlog01compat -work work +incdir+D:/Dersler/CSE331-Computer_Organization/HW4/HW4 (D:/Dersler/CSE331-Computer_Organization/HW4/HW4/adder_tb.v)
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module adder_tb
#
# Top level modules:
#       adder_tb
#
ModelSim> vsim work.adder_tb
# vsim work.adder_tb
# Loading work.adder_tb
# Loading work.adder_32_bit
# Loading work.full_adder
VSI3> run
# input1=000000000000000000000000000000001111=      15
# input2=000000000000000000000000000000001101 =      13
# out=0000000000000000000000000000000011100 =      28
#
# input1=000000000000000000000000000000000110=       6
# input2=000000000000000000000000000000000111 =       7
# out=0000000000000000000000000000000001101 =      13
#
VSI4>
```

2) 32 bit Subtractor

Design With Xor and 1 bit Full Adder Gates

Subtractor Testbench:

1. Test = $19 - 13 = 6$

2. Test = $9 - 7 = 2$

```
Transcript
#       sub_32_bit
# vlog -vlog01compat -work work +incdir+D:/Dersler/CSE331-Computer_Organization/HW4/HW4 (D:/Dersler/CSE331-Computer_Organization/HW4/HW4/sub_tb.v)
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module sub_tb
#
# Top level modules:
#       sub_tb
#
ModelSim> vsim work.sub_tb
# vsim work.sub_tb
# Loading work.sub_tb
# Loading work.sub_32_bit
# Loading work.full_adder
VSI3> run
# input1=0000000000000000000000000000000010011=      19
# input2=0000000000000000000000000000000001101 =      13
# out=00000000000000000000000000000000000110 =       6
#
# input1=0000000000000000000000000000000001001=       9
# input2=000000000000000000000000000000000111 =       7
# out=00000000000000000000000000000000000010 =       2
#
```

3)32 bit XOR

Design with 32 - 1 bit XOR Gates

Testbench:

```
Transcript
# Top level modules:
#   xor_32_bit
# vlog -vlog01compat -work work +incdir+D:/Dersler/CSE331-Computer_Organization/HW4/HW4 {D:/Dersler/CSE331-Computer_Organization/HW4/HW4/xor_tb.v}
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module xor_tb
#
# Top level modules:
#   xor_tb
#
ModelSim> vsim work.xor_tb
# vsim work.xor_tb
# Loading work.xor_tb
# Loading work.xor_32_bit
VSI3> run
# input1=00000000000000000000000010001100000
# input2=00000000000000000000000010000000101010
#   out=00000000000000000000000010010001001010
#
# input1=0000000000000000000000001010010001
# input2=0000000000000000000000000111001000
#   out=0000000000000000000000001101011001
#
```

4)32 bit AND

Design with 32 - 1 bit AND Gates

Testbench:

```
Transcript
# Top level modules:
#   and_32_bit
# vlog -vlog01compat -work work +incdir+D:/Dersler/CSE331-Computer_Organization/HW4/HW4 {D:/Dersler/CSE331-Computer_Organization/HW4/HW4/and_tb.v}
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module and_tb
#
# Top level modules:
#   and_tb
#
ModelSim> vsim work.and_tb
# vsim work.and_tb
# Loading work.and_tb
# Loading work.and_32_bit
SIM3> run
# input1=00000000000000000000000010001100000
# input2=00000000000000000000000010000000101010
#   out=000000000000000000000000000000000000
#
# input1=0000000000000000000000001010010001
# input2=0000000000000000000000000111001000
#   out=000000000000000000000000000000000000
```

5)32 bit OR

Design with 32 - 1 bit OR Gates

Testbench:

```
Transcript
# Top level modules:
#   or_32_bit
# vlog -vlog01compat -work work +incdir+D:/Dersler/CSE331-Computer_Organization/HW4/HW4 {D:/Dersler/CSE331-Computer_Organization/HW4/HW4/or_tb.v}
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module or_tb
#
# Top level modules:
#   or_tb
#
ModelSim> vsim work.or_tb
# vsim work.or_tb
# Loading work.or_tb
# Loading work.or_32_bit
VSI3> run
# input1=00000000000000000000000010001100000
# input2=00000000000000000000000010000000101010
#   out=00000000000000000000000010010001101010
#
# input1=0000000000000000000000001010010001
# input2=0000000000000000000000000111001000
#   out=0000000000000000000000001111011001
#
```

6) Extender 16 bit to 32 bit

Testbench:

```
Transcript
# Updated modelsim.ini.
#
# vlog -vlog01compat -work work +incdir+D:/Dersler/CSE331-Computer_Organization/HW4/HW4 {D:/Dersler/CSE331-Com
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module extend_16_to_32
#
# Top level modules:
#   extend_16_to_32
# vlog -vlog01compat -work work +incdir+D:/Dersler/CSE331-Computer_Organization/HW4/HW4 {D:/Dersler/CSE331-Com
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module extend_tb
#
# Top level modules:
#   extend_tb
#
ModelSim> vsim work.extend_tb
# vsim work.extend_tb
# Loading work.extend_tb
# Loading work.extend_16_to_32
VSIM 3> run
# Unsigned : 0010100010101010 == 00000000000000000010100010101010
# Signed  : 1110100010001010 == 111111111111111110100010001010
```

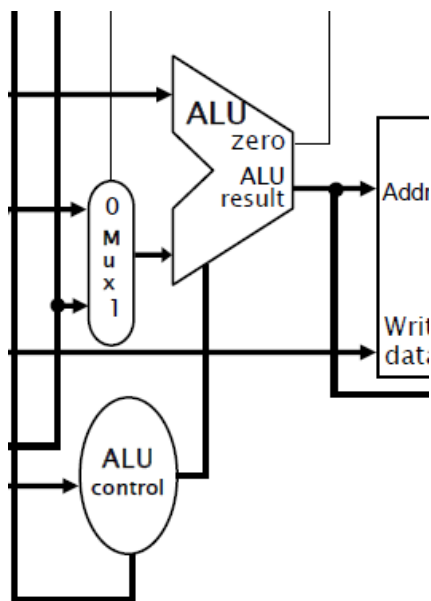
7) 32 Bit Multiplexer

Firstly I design 1 bit Multiplexer with gates and design 32 bit multiplexer with 1 bit multiplexer.

Testbench:

```
Transcript
#
# Top level modules:
#   mux_1_bit_2_1
# vlog -vlog01compat -work work +incdir+D:/Dersler/CSE331-Computer_Organization/HW4/HW4 {D:/Dersler/CSE331-Comput
# Model Technology ModelSim ALTERA vlog 10.1d Compiler 2012.11 Nov  2 2012
# -- Compiling module mux_32_tb
#
# Top level modules:
#   mux_32_tb
#
ModelSim> vsim work.mux_32_tb
# vsim work.mux_32_tb
# Loading work.mux_32_tb
# Loading work.mux_32_bit_2_1
# Loading work.mux_1_bit_2_1
VSIM 3> run
# input1=00000100010000010000000000000000
# input2=00000000000000000000000000000000 Select Bit :0
# out=00000100010000010000000000000000
# input1=00000100010000010000000000000000
# input2=00000000000000000000000000000000 Select Bit :1
# out=00000000000000000000000000000000
```

8)ALU CONTROL UNIT



```
Transcript
# Top level modules:
#   alu_cu_tb
#
ModelSim> vsim work.alu_cu_tb
# vsim work.alu_cu_tb
# Loading work.alu_cu_tb
# Loading work.alu_control_unit
VSIM 3> run
# Alu Op : 00 Function Field:000100
# Alu Control Unit Out=010
# Alu Op : 01 Function Field:000100
# Alu Control Unit Out=110
# Alu Op : 10 Function Field:100000
# Alu Control Unit Out=010
# Alu Op : 10 Function Field:100010
# Alu Control Unit Out=110
# Alu Op : 10 Function Field:100100
# Alu Control Unit Out=000
# Alu Op : 10 Function Field:100101
# Alu Control Unit Out=001
# Alu Op : 10 Function Field:101010
# Alu Control Unit Out=111
```

This ALU Control Unit take 2 bit AluOp and function field from control unit then send 3 bit signal to ALU for operations.

010 = Add

110 = Substraction

000 = And

001 = Or

111 = Xor

9)32 Bit ALU

Alu 32 bit take 32 bit two inputs and 3 bit alu operation from alu control unit then make this operations:

010 = Add

110 = Substraction

000 = And

001 = Or

111 = Xor

Testbench

```
#           Region: /alu_32_tb/alu1
VSIM 3> run
# input1=000000000000000000000000000000001001 =          9
# input2=000000000000000000000000000000000011 =          3 Operation:010
#   out=0000000000000000000000000000000001100 =         12
#
# input1=000000000000000000000000000000001001 =          9
# input2=000000000000000000000000000000000011 =          3 Operation:110
#   out=0000000000000000000000000000000000110 =          6
#
# input1=000000000000000000000000000000001001 =          9
# input2=000000000000000000000000000000000011 =          3 Operation:000
#   out=0000000000000000000000000000000000001 =          1
#
# input1=000000000000000000000000000000001001 =          9
# input2=000000000000000000000000000000000011 =          3 Operation:001
#   out=00000000000000000000000000000000001011 =         11
#
# input1=000000000000000000000000000000001001 =          9
# input2=000000000000000000000000000000000011 =          3 Operation:111
#   out=00000000000000000000000000000000001010 =         10
#
```

10)Control Unit

It generates the signals required for mips32.

Opcode = 000000 = R-type

Opcode = 100011 = lw

Opcode = 101011 = sw

Opcode = 000100 = beq

```
ModelSim> vsim work.cu_tb
# vsim work.cu_tb
# Loading work.cu_tb
# Loading work.control_unit
VSIM 3> run
# opcode :000000
# regDst=1 branch=0 MemRead=0 ,MemToReg=0 AluOp=10 MemWrite=0 ALUSrc=0 RegWrite=1
# opcode :100011
# regDst=0 branch=0 MemRead=1 ,MemToReg=1 AluOp=00 MemWrite=0 ALUSrc=1 RegWrite=1
# opcode :101011
# regDst=0 branch=0 MemRead=0 ,MemToReg=0 AluOp=00 MemWrite=1 ALUSrc=1 RegWrite=0
# opcode :000100
# regDst=0 branch=1 MemRead=0 ,MemToReg=0 AluOp=01 MemWrite=0 ALUSrc=0 RegWrite=0
VSIM 4>
```

11)Program Counter,Data Memory,Register Block

```
module data_memory_block(address, write_data, mem_write_sg, read_data, clock);

    input [31:0] address, write_data;
    input mem_write_sg, clock;
    output [31:0] read_data;

    reg [31:0] memory [255:0];

    assign read_data = memory[address[7:0]];
    always @(posedge clock)
    begin
        if (mem_write_sg == 1'b1)
            begin
                memory[address[7:0]] <= write_data;
                $writememb("/Dersler/CSE331-Computer_Organization/HW4/HW4/memory_data.txt", memory);
            end
        end
    end
endmodule

module register_block(clock, read_reg_1, read_reg_2, write_register1, write_register2, read_data_1, read_data_2,
    signal_reg_write, write_data1, write_data2);

    reg [31:0] registers[31:0];
    input [31:0] write_data1, write_data2;
    input [4:0] read_reg_1, read_reg_2, write_register1, write_register2;
    input clock;
    input signal_reg_write;
    output [31:0] read_data_1, read_data_2;
    assign read_data_1 = registers[read_reg_1];
    assign read_data_2 = registers[read_reg_2];

    always @(posedge clock)
    begin
        if (signal_reg_write == 1'b1)
            begin
                registers[write_register1] <= write_data1;
                registers[write_register2] <= write_data2;
                $writememb("/Dersler/CSE331-Computer_Organization/HW4/HW4/new_registers.mem", registers);
            end
        end
    end
endmodule
```

```

module program_counter_block(clock,new_instruction);
input clock;
output [31:0] new_instruction;
reg [31:0] n_instruction;
reg [5:0] program_c;
reg [31:0] all_instruction[27:0];

initial
begin
    program_c = 6'b000000;
end
assign new_instruction = n_instruction;

always @(posedge clock) begin
    n_instruction = all_instruction[program_c];
    #10;
    program_c = program_c + 1'b1;
end

endmodule

```

I designed these 3 modules but they don't have their own testbenches, I tried to test them in mips 32.

11)Mips32 Tests

My Mips32 processor is not working properly. I could not do things like jump and brach.

LW TEST

Lw \$1, 0(\$0) = 10001100000000010000000000000000

I take data_memory[0] from memory_data txt and this is = 000....00011

Then put this variable to \$1 register. Changed register 1 in the right side of image.

The screenshot shows the Quartus II software interface. The 'Transcript' window on the left displays the loading of work blocks and the execution of a Verilog simulation. The 'Waveform' window on the right shows the values of the 'mips32_tb/mips/register/registers' block over time, with a red arrow pointing to the 'mips32_tb/mips/register/registers(1)' signal.

SW TEST

Sw \$2 7(\$2) = 10101100010001110000000000000010

Take register \$9 value it is = 000...0001100 and put data_memory[2]

The screenshot displays a MIPS simulator interface. The main window shows assembly code with comments and binary data. A specific instruction is highlighted: `# Opcode:2b,Rs:02 Rt:07 Rd:00 immediate:0002 regWrite:0 Alu Control Unit Out:010 RegDst:1 Branch:0 MemWrite:1 ALUSrc:1`. Below this, the state of registers and memory is shown. To the right, a memory dump is visible, showing the contents of memory locations [0] through [28]. An arrow points from the memory dump to a specific location, indicating the value stored in `data_memory[2]`.

```
# Read_data_1:00000000000000000000000000000000
# data[0]:00000000000000000000000000000000011
# data[1]:000000000000000000000000000000000100
# data[2]:000000000000000000000000000000000000
# Reg[2]=000000000000000000000000000000000011
# Data Read Data:00000000000000000000000000000000
# Final Data:00000000000000000000000000000000
#
# Opcode:2b,Rs:02 Rt:07 Rd:00 immediate:0002 regWrite:0 Alu Control Unit Out:010 RegDst:1 Branch:0 MemWrite:1 ALUSrc:1
MemToReg:0 AluOp:00
# Alu Result:000000000000000000000000000000010 Read_data_2:000000000000000000000000000001100
# Read_data_1:000000000000000000000000000000000000
# data[0]:000000000000000000000000000000000011
# data[1]:000000000000000000000000000000000100
# data[2]:000000000000000000000000000000000000
# Reg[2]=000000000000000000000000000000000011
# Data Read Data:00000000000000000000000000000000
# Final Data:000000000000000000000000000000010
#
# Opcode:00,Rs:08 Rt:09 Rd:03 immediate:1820 regWrite:1 Alu Control Unit Out:010 RegDst:1 Branch:0 MemWrite:0 ALUSrc:0
MemToReg:0 AluOp:10
# Alu Result:0000000000000000000000000000000100011 Read_data_2:0000000000000000000000000000010000
# Read_data_1:000000000000000000000000000000000011
# data[0]:000000000000000000000000000000000011
# data[1]:000000000000000000000000000000000100
# data[2]:0000000000000000000000000000000001100
# Reg[2]=000000000000000000000000000000000011
# Data Read Data:000000000000000000000000000000000000
# Final Data:0000000000000000000000000000000100011
#
# Opcode:00,Rs:03 Rt:09 Rd:05 immediate:2960 regWrite:1 Alu Control Unit Out:010 RegDst:1 Branch:0 MemWrite:0 ALUSrc:0
MemToReg:0 AluOp:10
# Alu Result:0000000000000000000000000000000100011 Read_data_2:0000000000000000000000000000010000
# Read_data_1:000000000000000000000000000000000011
```

Memory dump (right side):

```
[28] = 00000000000000000000000000000000...
[27] = 00000000000000000000000000000000...
[26] = 00000000000000000000000000000000...
[25] = 00000000000000000000000000000000...
[24] = 00000000000000000000000000000000...
[23] = 00000000000000000000000000000000...
[22] = 00000000000000000000000000000000...
[21] = 00000000000000000000000000000000...
[20] = 00000000000000000000000000000000...
[19] = 00000000000000000000000000000000...
[18] = 00000000000000000000000000000000...
[17] = 00000000000000000000000000000000...
[16] = 00000000000000000000000000000000...
[15] = 00000000000000000000000000000000...
[14] = 00000000000000000000000000000000...
[13] = 00000000000000000000000000000000...
[12] = 00000000000000000000000000000000...
[11] = 00000000000000000000000000000000...
[10] = 00000000000000000000000000000000...
[9] = 00000000000000000000000000000000...
[8] = 00000000000000000000000000000001...
[7] = 00000000000000000000000000000000...
[6] = 00000000000000000000000000000000...
[5] = 00000000000000000000000000000000...
[4] = 00000000000000000000000000000000...
[3] = 00000000000000000000000000000000...
[2] = 00000000000000000000000000000000...
[1] = 00000000000000000000000000000000...
[0] = 00000000000000000000000000000000...
```

Memory address and value (bottom right):

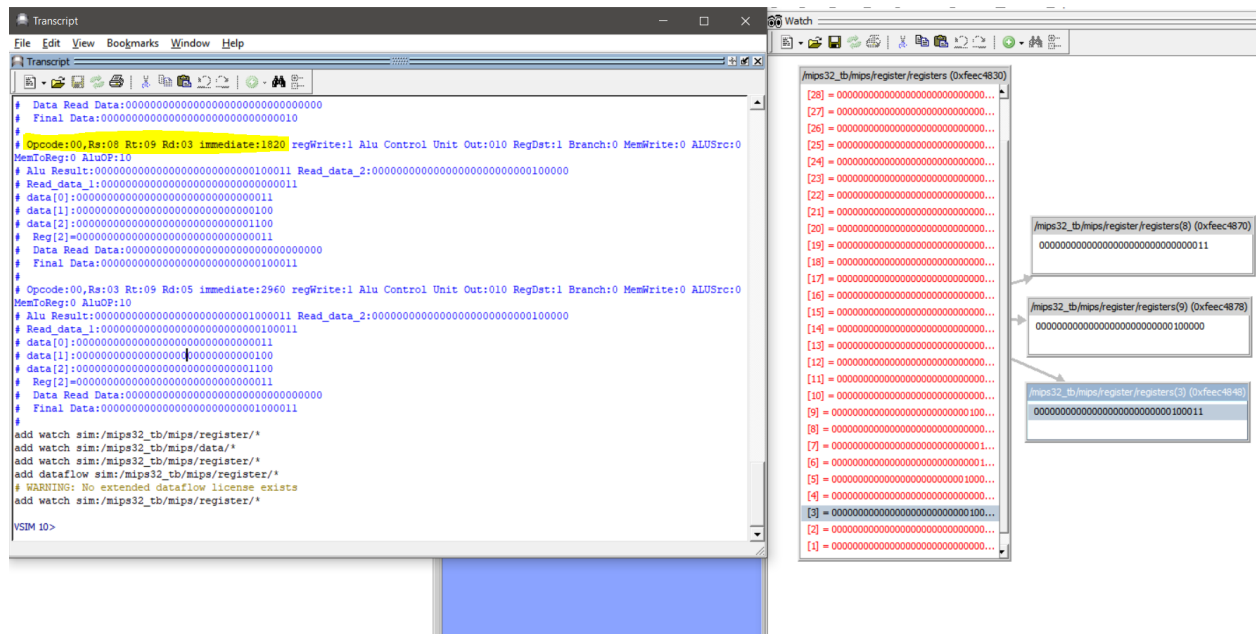
```
/mips32_tb/mips/data/memory(2) (0xfeecabe0)
000000000000000000000000000000001100
```

I could not design exactly addn,xorn,subn,orn because could not design 32 bit comparator I designed only 1 bit comparator but I did the add,xor,sub,or instructions at least.

Add test

addn \$3, \$8, \$9 = 00000001000010010001100000100000

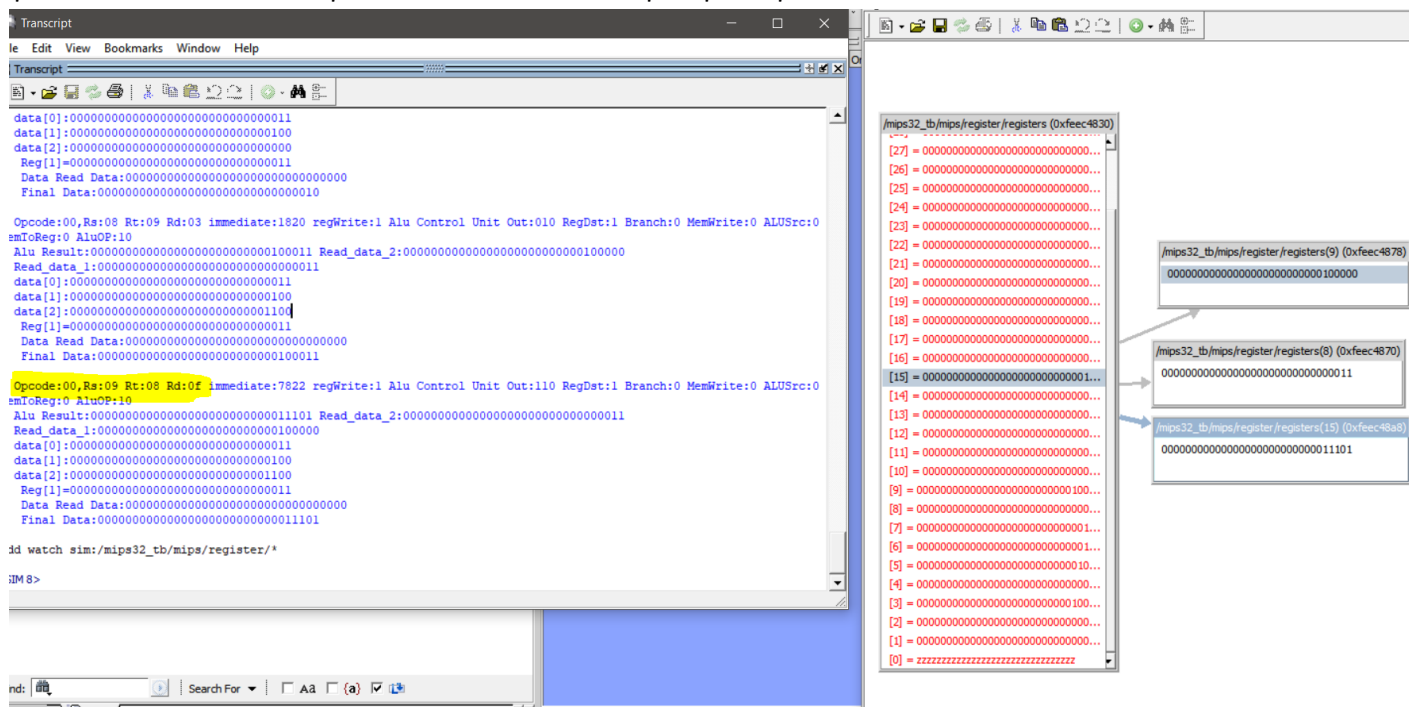
\$8 = 000...00000011 \$9 = 000...01000000 \$8 + \$9 = \$3 = 0000...001000011



Sub Test

subn \$15, \$9, \$8 = 00000001001010000111100000100010

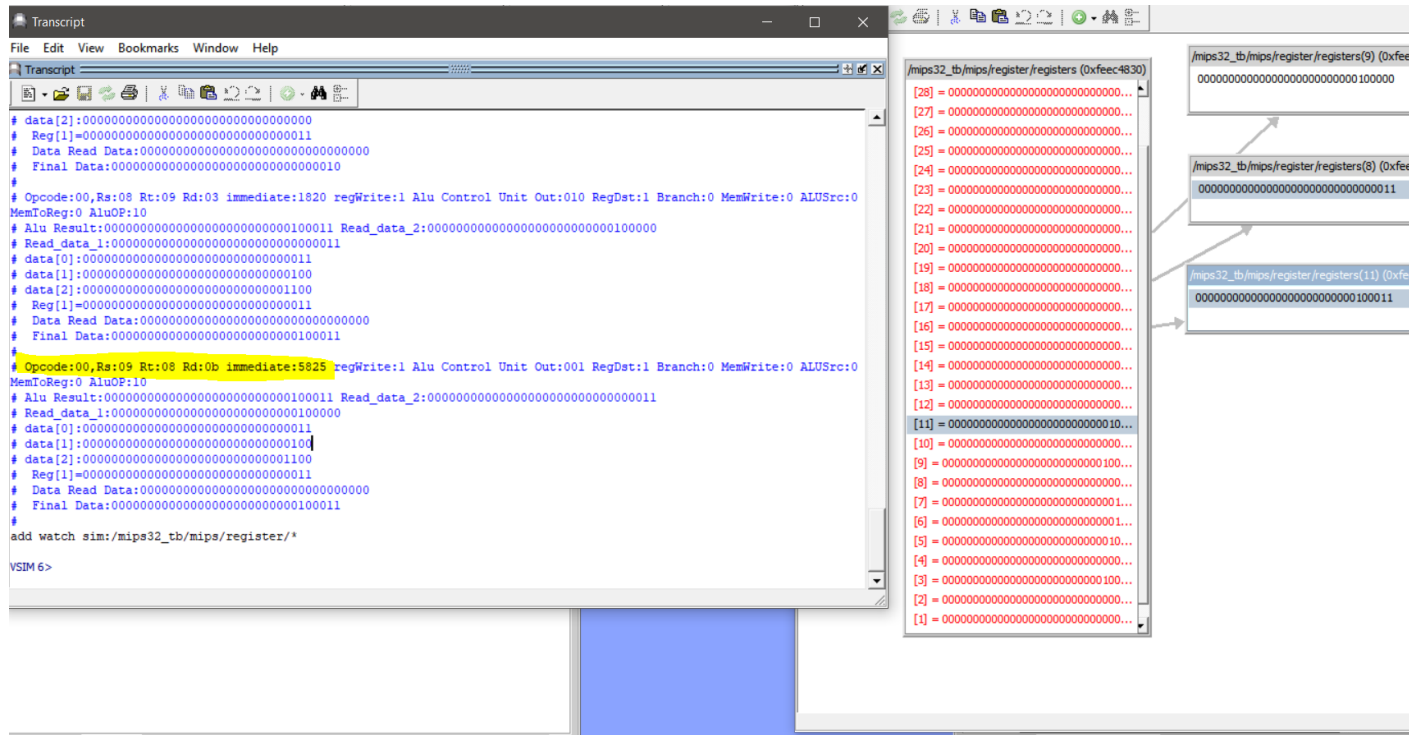
\$8 = 000...00000011 \$9 = 000...01000000 \$9 - \$8 = \$15 = 000..00011101



OR Test

orn \$11, \$9,\$8 = 00000001001010000111100000100010

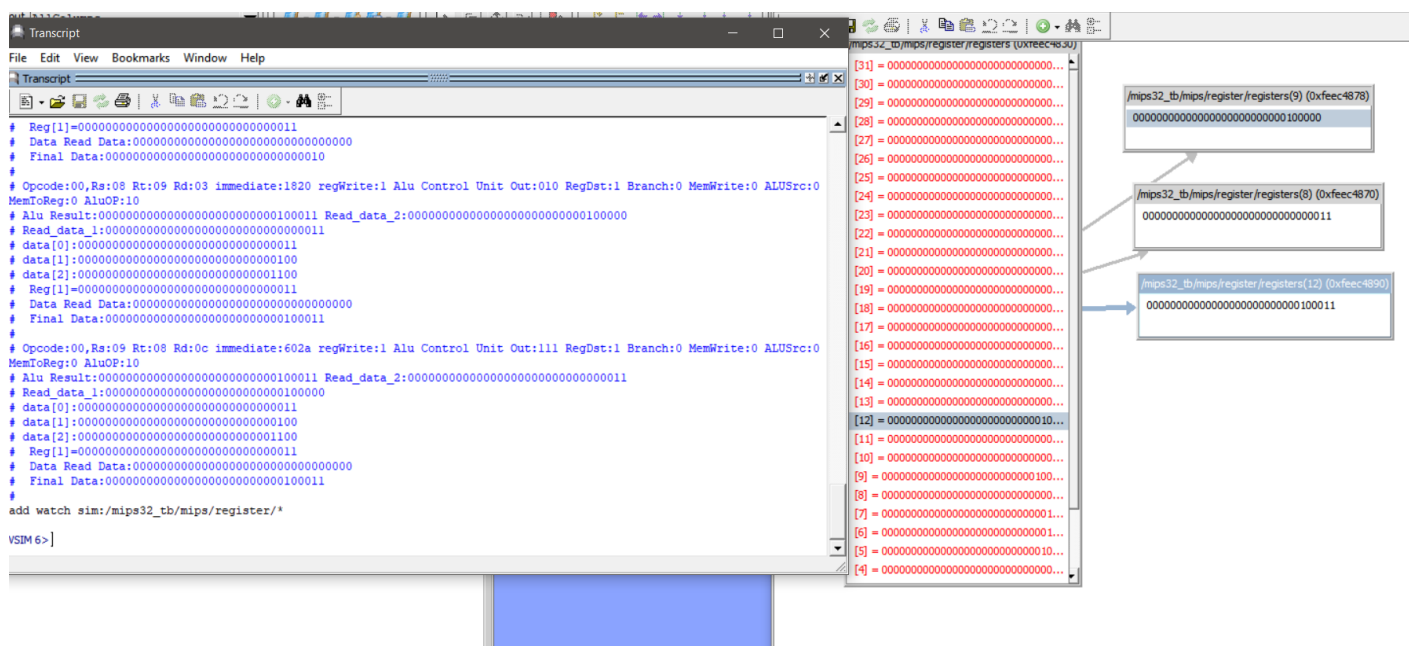
\$8 = 000...00000011 \$9 = 000...01000000 \$9 orn \$8 = 000...001000011



Xor Test

xorn \$12,\$9,\$8

\$8 = 000...00000011 \$9 = 000...01000000 \$9 xorn \$8 = 000...001000011



$\$8 = 000 \dots 00000011$ $\$9 = 000 \dots 01000000$ $\$9 \text{ xorn } \$8 = 000 \dots 00000000$

