_3bit_mux_2x1

This module selects one of the 2 3-bit inputs as an output according to the select bit.

testbench

```
`define DELAY 20
module _3bit_mux_2x1_testbench();
reg [2:0] a, b;
reg s;
wire [2:0] r;

_3bit_mux_2x1 mux0(r, s, a, b);

initial begin
a = 3'b000; b = 3'b001; s = 1'b0;
#`DELAY;
a = 3'b000; b = 3'b001; s = 1'b1;
end

initial
begin
$monitor("time = %2d\na = %3b\nb = %3b\ns = %1b\nr = %3b", $time, a, b, s, r);
end
endmodule
```

testbench result

```
# time =  0
# a = 000
# b = 001
# s = 0
# r = 000
# time = 20
# a = 000
# b = 001
# s = 1
# r = 001
```

alu_control

This module decides the operation alu will do according to the ALUop that main_control sent.

testbench

```
initial begin
ALUop = 3'b000;
func = 3'b000;
#`DELAY;
ALUop = 3'b001;
#`DELAY;
ALUop = 3'b010;
#`DELAY;
ALUop = 3'b011;
#`DELAY;
ALUop = 3'b100;
#`DELAY;
ALUop = 3'b101;
#`DELAY;
ALUop = 3'b110;
func = 3'b000;
#`DELAY;
ALUop = 3'b110;
func = 3'b001;
#`DELAY;
ALUop = 3'b110;
func = 3'b010;
#`DELAY;
ALUop = 3'b110;
func = 3'b011;
#`DELAY;
ALUop = 3'b110;
func = 3'b100;
#`DELAY;
ALUop = 3'b110;
func = 3'b101;
end
```

testbench result

```
# time =  0
# ALUop = 000
# func = 000
# control = 000
#
# time = 20
# ALUop = 001
# func = 000
# control = 110
#
# time = 40
# ALUop = 010
# func = 000
# control = 111
#
# time = 60
# ALUop = 011
# func = 000
# control = 101
#
# time = 80
# ALUop = 100
# func = 000
# control = 010
#
# time = 100
# ALUop = 101
# func = 000
# control = 100
#
# time = 120
# ALUop = 110
# func = 000
# control = 110
#
# time = 140
# ALUop = 110
# func = 001
# control = 000
#
# time = 160
# ALUop = 110
# func = 010
# control = 010
#
# time = 180
# ALUop = 110
# func = 011
# control = 001
```

data_memory

This module is the main memory of the MiniMIPS.

testbench

```
initial begin
 $readmemb("data.mem",dm.data);
 clk = 1;
 MemWrite = 0;
 Address = 32'd0;
 WriteData = 32'd20;
 end

always begin
 #10 clk= ~clk;
 end

initial begin
 #`DELAY;
 Address = 32'd1;
 #`DELAY;
 Address = 32'd2;
 #`DELAY;
 Address = 32'd3;
 #`DELAY;
 Address = 32'd4;
 #`DELAY;
 Address = 32'd5;
 #`DELAY;
 Address = 32'd6;
 #`DELAY;
 Address = 32'd7;
 #`DELAY;
 MemWrite = 1;
 Address = 32'd8;
 #`DELAY;
 $writememb("data_updated.mem",dm.data);
 $stop;
 end
```

testbench result

```
# time = 20
# Address = 00000000000000000000000000000001
# ReadData = 00000000000000000000000000000001
# MemWrite = 0
# WriteData = 00000000000000000000000000010100
#
# time = 40
# Address = 00000000000000000000000000000010
# ReadData = 00000000000000000000000000000010
# MemWrite = 0
# WriteData = 00000000000000000000000000010100
#
# time = 60
# Address = 00000000000000000000000000000011
# ReadData = 00000000000000000000000000000011
# MemWrite = 0
# WriteData = 00000000000000000000000000010100
#
# time = 80
# Address = 00000000000000000000000000000100
# ReadData = 00000000000000000000000000000100
# MemWrite = 0
# WriteData = 00000000000000000000000000010100
#
# time = 100
# Address = 00000000000000000000000000000101
# ReadData = 00000000000000000000000000000101
# MemWrite = 0
# WriteData = 00000000000000000000000000010100
#
# time = 120
# Address = 00000000000000000000000000000110
# ReadData = 00000000000000000000000000000110
# MemWrite = 0
# WriteData = 00000000000000000000000000010100
#
# time = 140
# Address = 00000000000000000000000000000111
# ReadData = 00000000000000000000000000000111
# MemWrite = 0
# WriteData = 00000000000000000000000000010100
#
# time = 160
# Address = 00000000000000000000000000001000
# ReadData = 00000000000000000000000000010100
# MemWrite = 1
# WriteData = 00000000000000000000000000010100
```

instruction_memory

This module holds all the instructions of the MiniMIPS.

testbench

```
`define DELAY 20
module instruction_memory_testbench();
reg [31:0] address;
reg clk;
wire [15:0] instruction;

instruction_memory im(instruction, address, clk);

initial begin
$readmemb("instructions.mem",im.instructions);
clk = 1;
address = 32'd0;
end

always begin
#10 clk = ~clk;
end

initial begin
#`DELAY;
address = 32'd1;
#`DELAY;
address = 32'd2;
#`DELAY;
address = 32'd3;
#`DELAY;
address = 32'd4;
#`DELAY;
address = 32'd5;
#`DELAY;
$stop;
end

initial begin
$monitor("time = %2d\ninstruction = %16b\n", $time, instruction);
end
```

testbench result

```
# time =  0
# instruction = 0000000001010000
#
# time = 20
# instruction = 0000011100101000
#
# time = 40
# instruction = 0000011101101001
#
# time = 60
# instruction = 0000110111100001
#
# time = 80
# instruction = 0000101111000010
#
# time = 100
# instruction = 0000010100111010
#
```

main_control

This module is the main control unit of the MiniMIPS that generates all the control signals.

testbench

```
`define DELAY 20
module main_control_testbench();
reg [3:0] opcode;
wire [2:0] ALUop;
wire RegWrite, ALUSrc, RegDst, MemtoReg, MemRead, MemWrite, Branch;

main_control mc(RegWrite, ALUSrc, RegDst, MemtoReg, MemRead, MemWrite, Branch, ALUop, opcode);

initial begin
opcode = 4'b0000;
#`DELAY;
opcode = 4'b0001;
#`DELAY;
opcode = 4'b0010;
#`DELAY;
opcode = 4'b0011;
#`DELAY;
opcode = 4'b0100;
#`DELAY;
opcode = 4'b0101;
#`DELAY;
opcode = 4'b0110;
#`DELAY;
opcode = 4'b0111;
#`DELAY;
opcode = 4'b1000;
#`DELAY;
opcode = 4'b1001;
end

initial
begin
$monitor("time = %2d\nopcode = %4b\nRegWrite = %1b\nALUSrc = %1b\nRegDst = %1b\nMemtoReg = %1b\nMemRead = %1b\nMemWrite = %1b\nBranch = %1b\nALUop =
end
endmodule
```

testbench result

```
# time =  0
# opcode = 0000
# RegWrite = 1
# ALUSrc = 0
# RegDst = 1
# MemtoReg = 0
# MemRead = 0
# MemWrite = 0
# Branch = 0
# ALUop = 110
#
# time = 20
# opcode = 0001
# RegWrite = 1
# ALUSrc = 1
# RegDst = 0
# MemtoReg = 0
# MemRead = 0
# MemWrite = 0
# Branch = 0
# ALUop = 000
#
# time = 40
# opcode = 0010
# RegWrite = 1
# ALUSrc = 1
# RegDst = 0
# MemtoReg = 0
# MemRead = 0
# MemWrite = 0
# Branch = 0
# ALUop = 001
#
# time = 60
# opcode = 0011
# RegWrite = 1
# ALUSrc = 1
# RegDst = 0
# MemtoReg = 0
# MemRead = 0
# MemWrite = 0
# Branch = 0
# ALUop = 010
#
```

MiniMIPS

This module is the mips processor.

testbench

```
`define DELAY 10
module MiniMIPS_testbench();
reg clk;
reg [31:0] Counter;
wire [31:0] NewCounter;

MiniMIPS mm(NewCounter, Counter, clk);

initial begin
   $readmemb("registers.mem",mm.rb.registers);
   $readmemb("data.mem",mm.dm.data);
   $readmemb("instructions.mem",mm.im.instructions);
   clk = 1;
   Counter = 32'd0;
end

always begin
   #5 clk= ~clk;
end

always begin
   #`DELAY;
   Counter = NewCounter;
   if(Counter > 32'd29) begin
      #`DELAY;
      $writememb("registers_updated.mem",mm.rb.registers);
      $writememb("data_updated.mem",mm.dm.data);
      $stop;
   end
end

initial begin
   $monitor("time = %2d\nCounter = %2d\nInstruction = %16b\nReadData1 = %32b\nReadData2 = %32b\nResult = %32b\nNewCounter = %2d\n", $time, Counter, m
end
endmodule
```

testbench result

```
# time =  0
# Counter =  0
# Instruction = 0000000001010000
# ReadData1 = 00000000000000000000000000000000
# ReadData2 = 00000000000000000000000000000001
# Result = 00000000000000000000000000000000
# NewCounter =  1
#
# time = 10
# Counter =  1
# Instruction = 0000011100101000
# ReadData1 = 00000000000000000000000000000011
# ReadData2 = 00000000000000000000000000000100
# Result = 00000000000000000000000000000000
# NewCounter =  2
#
# time = 20
# Counter =  2
# Instruction = 0000011101101001
# ReadData1 = 00000000000000000000000000000011
# ReadData2 = 00000000000000000000000000000000
# Result = 00000000000000000000000000000011
# NewCounter =  3
#
# time = 30
# Counter =  3
# Instruction = 0000110111100001
# ReadData1 = 00000000000000000000000000000110
# ReadData2 = 00000000000000000000000000000111
# Result = 00000000000000000000000000001101
# NewCounter =  4
#
# time = 40
# Counter =  4
# Instruction = 0000101111000010
# ReadData1 = 00000000000000000000000000000011
# ReadData2 = 00000000000000000000000000000111
# Result = 11111111111111111111111111111100
# NewCounter =  5
#
# time = 50
# Counter =  5
# Instruction = 0000010100111010
# ReadData1 = 00000000000000000000000000000000
# ReadData2 = 00000000000000000000000000001101
# Result = 11111111111111111111111111110011
# NewCounter =  6
```

register_block

This module holds all the registers of the MiniMIPS processor.

testbench

```
initial begin
$readmemb("registers.mem",rb.registers);
clk = 1;
RegWrite = 0;
ReadReg1 = 3'd0;
ReadReg2 = 3'd1;
WriteReg = 3'd0;
WriteData = 32'd30;
end

always begin
#10 clk= ~clk;
end

initial begin
#`DELAY;
ReadReg1 = 3'd2;
ReadReg2 = 3'd3;
#`DELAY;
ReadReg1 = 3'd4;
ReadReg2 = 3'd5;
#`DELAY;
RegWrite = 1;
ReadReg1 = 3'd6;
ReadReg2 = 3'd7;
#`DELAY;
WriteReg = 3'd1;
#`DELAY;
$writememb("registers_updated.mem",rb.registers);
$stop;
end

initial begin
$monitor("time = %2d\nReadReg1 = %3b\nReadReg2 = %3b\nReadData1 = %32b\nReadData2 = %32b\nRegWrite = %1b\nWriteReg = %3b\nWriteData = %32b\n", $time,
end
```

testbench result

```
# time =  0
# ReadReg1 = 000
# ReadReg2 = 001
# ReadData1 = 00000000000000000000000000000000
# ReadData2 = 00000000000000000000000000000001
# RegWrite = 0
# WriteReg = 000
# WriteData = 00000000000000000000000000011110
#
# time = 20
# ReadReg1 = 010
# ReadReg2 = 011
# ReadData1 = 00000000000000000000000000000010
# ReadData2 = 00000000000000000000000000000011
# RegWrite = 0
# WriteReg = 000
# WriteData = 00000000000000000000000000011110
#
# time = 40
# ReadReg1 = 100
# ReadReg2 = 101
# ReadData1 = 00000000000000000000000000000100
# ReadData2 = 00000000000000000000000000000101
# RegWrite = 0
# WriteReg = 000
# WriteData = 00000000000000000000000000011110
#
# time = 60
# ReadReg1 = 110
# ReadReg2 = 111
# ReadData1 = 00000000000000000000000000000110
# ReadData2 = 00000000000000000000000000000111
# RegWrite = 1
# WriteReg = 000
# WriteData = 00000000000000000000000000011110
#
# time = 80
# ReadReg1 = 110
# ReadReg2 = 111
# ReadData1 = 00000000000000000000000000000110
# ReadData2 = 00000000000000000000000000000111
# RegWrite = 1
# WriteReg = 001
# WriteData = 00000000000000000000000000011110
#
```

_32bit_shift_left

This module shifts the given 32-bit input to the left as the given shift amount.

testbench

```
`define DELAY 20
module _32bit_shift_left_testbench();
reg [31:0] i;
reg [4:0] shamt;
wire [31:0] r;

_32bit_shift_left sl(r, i, shamt);

initial begin
i = 32'hFFFF_FFFF; shamt = 5'd5;
#`DELAY;
i = 32'hFFFF_FFFF; shamt = 5'd1;
#`DELAY;
i = 32'hFFFF_FFFF; shamt = 5'd10;
#`DELAY;
i = 32'hFFFF_FFFF; shamt = 5'd31;
#`DELAY;
i = 32'hFFFF_FFFF; shamt = 5'd16;
end

initial
begin
$monitor("time = %2d\ni = %32b\nshamt = %5b\nresult = %32b\n", $time, i, shamt, r);
end
endmodule
```

testbench result

```
# time =  0
# i = 11111111111111111111111111111111
# shamt = 00101
# result = 11111111111111111111111111100000
#
# time = 20
# i = 11111111111111111111111111111111
# shamt = 00001
# result = 11111111111111111111111111111110
#
# time = 40
# i = 11111111111111111111111111111111
# shamt = 01010
# result = 11111111111111111111110000000000
#
# time = 60
# i = 11111111111111111111111111111111
# shamt = 11111
# result = 10000000000000000000000000000000
#
# time = 80
# i = 11111111111111111111111111111111
# shamt = 10000
# result = 11111111111111110000000000000000
```

sign_extend

This module sign extends the given 6-bit input to 32 bits.

testbench

```
`define DELAY 20
module sign_extend_testbench();
reg [5:0] I;
wire [31:0] R;

sign_extend se(R, I);

initial begin
I=6'b000001;
#`DELAY;
I=6'b101010;
#`DELAY;
I=6'b111111;
#`DELAY;
I=6'b000000;
#`DELAY;
I=6'b100000;
end

initial begin
$monitor("time = %2d\ninitial=%6b\nextended = %32b\n", $time, I, R);
end
endmodule
```

testbench result

```
# time =  0
# initial=000001
# extended = 00000000000000000000000000000001
#
# time = 20
# initial=101010
# extended = 11111111111111111111111111101010
#
# time = 40
# initial=111111
# extended = 11111111111111111111111111111111
#
# time = 60
# initial=000000
# extended = 00000000000000000000000000000000
#
# time = 80
# initial=100000
# extended = 11111111111111111111111111100000
#
```

zero_check

This module checks if the given 32-bit input is zero.

testbench

```
`define DELAY 20
module zero_check_testbench();
reg [31:0] I;
wire R;

zero_check zc(R, I);

initial begin
I = 32'd0;
#`DELAY;
I = 32'd1;
#`DELAY;
I = 32'h8FFF_FFFF;
#`DELAY;
I = 32'hFFFF_FFFF;
end

initial begin
$monitor("time = %2d\nI = %32b\nIs zero = %1b\n", $time, I, R);
end
endmodule
```

testbench result

```
# time =  0
# I = 00000000000000000000000000000000
# Is zero = 1
#
# time = 20
# I = 00000000000000000000000000000001
# Is zero = 0
#
# time = 40
# I = 10001111111111111111111111111111
# Is zero = 0
#
# time = 60
# I = 11111111111111111111111111111111
# Is zero = 0
#
```

P.S. In MiniMIPS testbench, at the end monitor prints an instruction with all X because I designed the register block and data memory such that they can be updated at the posedge of the cloak, so MiniMIPS requires essentially a nop at the end to make sure all the registers and data is updated correctly.

Burak Yıldırım
1901042609