پیش گزارش آزمایش هشتم

اعضای گروه:

صادق محمدیان:401109477

متین محمدی:401110329

امیرحسین ملک محمدی:401106577

شرح آزمایش:

هدف از این آزمایش، طراحی تعدادی پیمانه مختلف در چند بخش و سپس در کنار هم قرار دادن آن ها و ساخت یک کامپیوتر پایه است.

حافظه این ماشین را 32 کلمه ای در نظر می‌گیریم.

حال ماژول های مختلف را مورد بررسی قرار می‌دهیم:

1) ماژول جمع و تفریق:

در این ماژول، جمع و تفریق اعداد مختلط با توجه به op مشخص شده صورت می‌گیرد. 8 بیت اول را بخش حقیقی و 8 بیت دوم را بخش موهومی در نظر می‌گیریم. (a[15:8] = Real part و a[7:0] = Imaginary part)

a[15:8] + b[15:8] = s[15:8], a[7:0]+b[7:0] = s[7:0]

```
module add_sub (
    input [15:0] a,
    input [15:0] b,
    input        op,
    output [15:0] s
);
wire signed [7:0] real_a = $signed(a[15:8]);
wire signed [7:0] imag_a = $signed(a[7:0]);
wire signed [7:0] real_b = $signed(b[15:8]);
wire signed [7:0] imag_b = $signed(b[7:0]);
assign s[15:8] = real_a + (op ? -real_b : real_b);
assign s[7:0] = imag_a + (op ? -imag_b : imag_b);
endmodule
```

در اینجا، 8 بیت حقیقی خروجی را جمع (تفریق) 8 بیت حقیقی دو ورودی و 8 بیت موهومی آن را نیز همینطور درنظر می‌گیریم.

جمع با تفریق برحسب بیت op تعیین می‌شود.

در زیر چند حالت را تست کرده و نتیجه را گزارش می‌کنیم:

```
module add_sub_TB();
    reg [15:0] a;
    reg [15:0] b;
    reg op;
    wire [15:0] s;
    add_sub addsub (a, b, op, s);
    initial begin
        a = 16'b00001111_00001111;
        b = 16'b00000001_00000001;
```

```
        op = 0;
        #10;
        $display("Addition: (%d, %d) + (%d, %d) = (%d, %d)", $signed(a[15:8]),
$signed(a[7:0]), $signed(b[15:8]), $signed(b[7:0]), $signed(s[15:8]),
$signed(s[7:0]));
        op = 1;
        #10;
        $display("Subtraction: (%d, %d) - (%d, %d) = (%d, %d)", $signed(a[15:8]),
$signed(a[7:0]), $signed(b[15:8]), $signed(b[7:0]), $signed(s[15:8]),
$signed(s[7:0]));
        a = 16'b11110000_11110000;
        b = 16'b00000001_00000001;
        op = 0;
        #10;
        $display("Addition: (%d, %d) + (%d, %d) = (%d, %d)", $signed(a[15:8]),
$signed(a[7:0]), $signed(b[15:8]), $signed(b[7:0]), $signed(s[15:8]),
$signed(s[7:0]));
        op = 1;
        #10;
        $display("Subtraction: (%d, %d) - (%d, %d) = (%d, %d)", $signed(a[15:8]),
$signed(a[7:0]), $signed(b[15:8]), $signed(b[7:0]), $signed(s[15:8]),
$signed(s[7:0]));
        a = 16'b00000000_11110000;
        b = 16'b00000001_00000000;
        op = 0;
        #10;
        $display("Addition: (%d, %d) + (%d, %d) = (%d, %d)", $signed(a[15:8]),
$signed(a[7:0]), $signed(b[15:8]), $signed(b[7:0]), $signed(s[15:8]),
$signed(s[7:0]));
        op = 1;
        #10;
        $display("Subtraction: (%d, %d) - (%d, %d) = (%d, %d)", $signed(a[15:8]),
$signed(a[7:0]), $signed(b[15:8]), $signed(b[7:0]), $signed(s[15:8]),
$signed(s[7:0]));
    end
endmodule
```

که خروجی همانطور که انتظار می‌رود به شکل زیر است:

```
# Addition: (  15,   15) + (   1,    1) = (  16,   16)
# Subtraction: (  15,   15) - (   1,    1) = (  14,   14)
# Addition: ( -16,  -16) + (   1,    1) = ( -15,  -15)
# Subtraction: ( -16,  -16) - (   1,    1) = ( -17,  -17)
# Addition: (   0,  -16) + (   1,    0) = (   1,  -16)
# Subtraction: (   0,  -16) - (   1,    0) = (  -1,  -16)

/STM 20>
```

2) ماژول ضرب کننده:

در این ماژول، 2 عدد مختلط ورودی داده شده، با توجه به قواعد ضرب اعداد مختلط، در هم ضرب شده و حاصل در p ریخته می‌شود:

a[15:8]*b[15:8] – a[7:0]*b[7:0] = p[15:8] (8 least significant bits of answer in the Real part)

a[15:8]*b[7:0] + a[7:0]*b[15:8] = p[7:0] (8 least significant bits of answer in the imaginary part)

```
module mul (
    input [15:0] a,
    input [15:0] b,
    output [15:0] p
);
wire signed [7:0] real_a = $signed(a[15:8]);
wire signed [7:0] imag_a = $signed(a[7:0]);
wire signed [7:0] real_b = $signed(b[15:8]);
wire signed [7:0] imag_b = $signed(b[7:0]);
wire signed [15:0] real_part = real_a * real_b - imag_a * imag_b;
wire signed [15:0] imag_part = real_a * imag_b + imag_a * real_b;
assign p[15:8] = real_part[7:0]; // Truncate to 8 bits
assign p[7:0] = imag_part[7:0]; // Truncate to 8 bits
endmodule
```

در این بخش دقیقا همان قاعده ای که ذکر شد صورت گرفته و خروجی p را تولید کرده ایم.

در زیر تعدادی از حالات را بررسی می‌کنیم:

```
module mul_TB;
    reg [15:0] a;
    reg [15:0] b;
```

```
    wire [15:0] p;
    mul mul (a, b, p);
    initial begin
        a = 16'b00001111_11111100;
        b = 16'b00000010_00000111;
        #20;
        $display("Multiplication: (%d, %d) * (%d, %d) = (%d, %d)",
                 $signed(a[15:8]), $signed(a[7:0]),
                 $signed(b[15:8]), $signed(b[7:0]),
                 $signed(p[15:8]), $signed(p[7:0]));
        a = 16'b11111101_11110010;
        b = 16'b00010101_11110011;
        #20;
        $display("Multiplication: (%d, %d) * (%d, %d) = (%d, %d)",
                 $signed(a[15:8]), $signed(a[7:0]),
                 $signed(b[15:8]), $signed(b[7:0]),
                 $signed(p[15:8]), $signed(p[7:0]));
        a = 16'b00000000_00000000;
        b = 16'b10000001_00000001;
        #20;
        $display("Multiplication: (%d, %d) * (%d, %d) = (%d, %d)",
                 $signed(a[15:8]), $signed(a[7:0]),
                 $signed(b[15:8]), $signed(b[7:0]),
                 $signed(p[15:8]), $signed(p[7:0]));
        a = 16'b0000111_00001101;
        b = 16'b11111111_00001110;
        #20;
        $display("Multiplication: (%d, %d) * (%d, %d) = (%d, %d)",
$signed(a[15:8]), $signed(a[7:0]),
                 $signed(b[15:8]), $signed(b[7:0]),
                 $signed(p[15:8]), $signed(p[7:0]));
        a = 16'b0000111_00001101;
        b = 16'b0000000_00001110;
        #20;
        $display("Multiplication: (%d, %d) * (%d, %d) = (%d, %d)",
$signed(a[15:8]), $signed(a[7:0]),
                 $signed(b[15:8]), $signed(b[7:0]),
                 $signed(p[15:8]), $signed(p[7:0]));
        a = 16'b0000101_00000001;
        b = 16'b0001111_00101110;
        #20;
        $display("Multiplication: (%d, %d) * (%d, %d) = (%d, %d)",
$signed(a[15:8]), $signed(a[7:0]),
                 $signed(b[15:8]), $signed(b[7:0]),
                 $signed(p[15:8]), $signed(p[7:0]));
```

```
        $stop();
    end
endmodule
```

که همانطور که انتظار می‌رود، خروجی پاسخ درست را به ما می‌دهد.

```
# Multiplication: (  15,   -4) * (   2,    7) = (  58,   97)
# Multiplication: (  -3,  -14) * (  21,  -13) = (  11,    1)
# Multiplication: (   0,    0) * (-127,    1) = (   0,    0)
# Multiplication: (   7,   13) * (  -1,   14) = (  67,   85)
# Multiplication: (   7,   13) * (   0,   14) = (  74,   98)
# Multiplication: (   5,    1) * (  15,   46) = (  29,  -11)
```

نکته: در انتها خروجی 11- به این دلیل است که 8 بیت را به صورت علامتدار در اختیار داریم. اگر فضا برای 16 بیت کامل داشتیم خروجی به طور کامل درست نشان داده می‌شد.

3) ماژول ALU:

این ماژول که واحد محاسباتی است، با توجه به 2 بیت عملیات مشخص می‌کند که نیاز داریم چه عملیاتی را انجام دهیم.

```
module alu (
    input [15:0] a,
    input [15:0] b,
    input [1:0] op,
    output reg [15:0] s
);
wire [15:0] addsub_res, mul_res;
add_sub adder_sub (a, b, op[0], addsub_res);
mul multiplier (a, b, mul_res);
always @(*) begin
    case (op[1])
        1'b0: s = addsub_res;
        1'b1: s = mul_res;
    endcase
end
endmodule
```

op = 00 => add op = 01 => sub op = 10 or 11 => mul

4) ماژول data_mem:

```
module data_mem (
    input [4:0] raddr1,
```

```
    input [4:0] raddr2,
    input [15:0] wdata,
    input [4:0] waddr,
    output reg [15:0] rdata1,
    output reg [15:0] rdata2
);
    reg [15:0] mem [31:0];
    always @(*) begin
        rdata1 = mem[raddr1];
        rdata2 = mem[raddr2];
    end
    always @(wdata or waddr) begin
        mem[waddr] = wdata;
    end
endmodule
```

حافظه را به صورت 32 کلمه 16 بیتی در نظر می‌گیریم. در دو بلاک always جداگانه، به کمک آدرس های خواندن، دو کلمه را می‌خوانیم و به کمک آدرس نوشتن، در یک آدرس از حافظه می‌نویسیم. (اعداد خوانده شده همان اعداد مختلط هستند)

5) ماژول پایپلاین:

سپس به معرفی ماژول پایپلاین می‌پردازیم. در این ماژول ابتدا حافظه ای 32 کلمه ای برای دستورات طراحی می‌کنیم (دستورات را 17 بیتی در نظر گرفته ایم. از این 17 بیت، 2 بیت برای عملیات، 5 بیت برای آدرس کلمه اولی که می‌خوانیم، 5 بیت برای آدرس کلمه دومی که می‌خوانیم و 5 بیت نیز برای آدرس کلمه ای که بر آن می‌نویسیم در نظر گرفته شده است.)

سپس یک سری اتصالات میان این ها را مشخص می‌کنیم و پس از آن از دو ماژول بالا (data_mem و alu) یک instance می‌گیریم تا پایپلاین را بسازیم.

در بلاک always که با لبه بالارونده کلاک یا لبه پایین رونده reset فعال می‌شود، اگر ریست صفر باشد، به ابتدای کار می‌رویم و در غیر اینصورت، باید سیگنال های کنترلی را مقدار دهی کنیم.

ابتدای کار، در 4 خط، عملیات، آدرس نوشتن و آدرس های خواندن بافر را مقدار دهی می‌کنیم (به کمک دستوری که در مموری قرار دارد) پس از آن، مقادیر خواندن و عملیات را مشخص می‌کنیم و حاصل خروجی alu را بعنوان دیتای نوشتن مشخص می‌کنیم. آدرس نوشتن را نیز داریم که data_mem کار نوشتن را انجام می‌دهد. در ادامه نیز پیش از اضافه کردن pc (تا به دستور بعدی برویم)، در 3 خط مشاهده می‌کنیم که وضعیت کامپیوتر ما در چه حال است.

کد بخش پایپلاین:

```
module pipeline (
    input clk,
    input rstN
);
    wire [1:0] i_op;
    wire [4:0] i_waddr, i_raddr1, i_raddr2;
    reg [16:0] inst_mem [31:0];
    reg [4:0] pc;
```

```
    assign {i_op, i_waddr, i_raddr1, i_raddr2} = inst_mem[pc];
    wire [15:0] m_rdata1, m_rdata2;
    reg [1:0] buff_op, op;
    reg [4:0] buff_waddr, buff2_waddr, raddr1, raddr2, waddr;
    reg [15:0] rdata1, rdata2, wdata;
    data_mem data_mem(raddr1, raddr2, wdata, waddr, m_rdata1, m_rdata2);
    wire [15:0] alu_out;
    alu ALU(rdata1, rdata2, op, alu_out);
    always @(posedge clk or negedge rstN) begin
        if (!rstN) begin
            pc <= 0;
        end else begin
            buff_op <= i_op;
            buff_waddr <= i_waddr;
            raddr1 <= i_raddr1;
            raddr2 <= i_raddr2;
            rdata1 <= m_rdata1;
            rdata2 <= m_rdata2;
            op <= buff_op;
            buff2_waddr <= buff_waddr;
            waddr <= buff2_waddr;
            wdata <= alu_out;
            $display("Time: #%d", $time);
            $display("IF: op=%b, waddr=%d, raddr1=%d, raddr2=%d",
                    buff_op, buff_waddr, raddr1, raddr2);
            $display("MEM: op=%b, waddr=%d, rdata1=(%d, %d), rdata2=(%d, %d)",
                    op, buff2_waddr, $signed(rdata1[15:8]), $signed(rdata1[7:0]),
$signed(rdata2[15:8]), $signed(rdata2[7:0]));
            $display("ALU: waddr=%d, wdata=(%d, %d)\n",
                    waddr, $signed(wdata[15:8]), $signed(wdata[7:0]));
            pc <= pc + 1;
        end
    end
endmodule
```

نکته: دلیل استفاده از non-blocking assignment این است که آپکد و دستورات جدید که وارد پایپلاین می‌شوند باعث از دست رفتن محتوا نشوند و در صحت عملکرد مدار خلل ایجاد نکنند.

در زیر پایپلاین را تست می‌کنیم:

```
module pipeline_TB ();
reg rstN = 0, clk = 1;
```

```
pipeline PIPELINE(clk, rstN);
always #10 clk = ~clk;
initial begin
    $readmemb("data/inst_mem.txt", PIPELINE.inst_mem, 0, 32);
    $readmemb("data/initial_mem.txt", PIPELINE.data_mem.mem, 0, 32);
    #40 rstN = 1;
    wait(PIPELINE.pc == 18);
    $writememb("data/final_mem.txt", PIPELINE.data_mem.mem);
    $stop;
end
endmodule
```

در اینجا از دو فایل تکستی که در فولدر لوکال داریم دستورات و مموری اولیه خوانده می‌شوند و سپس حاصل در فایل خروجی نوشته می‌شود.

در زیر نتیجه را مشاهده می‌کنیم:

```
#    Time: 0 ps  Iteration: 0  Instance: /pipeline_TB
# Time: #                   40
# IF: op=xx, waddr= x, raddr1= x, raddr2= x
# MEM: op=xx, waddr= x, rdata1=(   x,    x), rdata2=(   x,    x)
# ALU: waddr= x, wdata=(   x,    x)
#
# Time: #                   60
# IF: op=10, waddr= 7, raddr1=10, raddr2=23
# MEM: op=xx, waddr= x, rdata1=(   x,    x), rdata2=(   x,    x)
# ALU: waddr= x, wdata=(   x,    x)
#
# Time: #                   80
# IF: op=00, waddr=14, raddr1=21, raddr2= 9
# MEM: op=10, waddr= 7, rdata1=(   2,    1), rdata2=(  45,   17)
# ALU: waddr= x, wdata=(   x,    x)
#
# Time: #                  100
# IF: op=00, waddr=28, raddr1= 9, raddr2=19
# MEM: op=00, waddr=14, rdata1=(   3,    5), rdata2=(  -4,   -5)
# ALU: waddr= 7, wdata=(  73,   79)
#
# Time: #                  120
# IF: op=00, waddr=17, raddr1=28, raddr2= 3
# MEM: op=00, waddr=28, rdata1=(  -4,   -5), rdata2=( -27,   43)
# ALU: waddr=14, wdata=(  -1,    0)
#
# Time: #                  140
# IF: op=00, waddr=10, raddr1=18, raddr2=27
# MEM: op=00, waddr=17, rdata1=(  41,  -25), rdata2=(  12,   10)
# ALU: waddr=28, wdata=( -31,   38)
```

```
# Time: #                  160
# IF: op=01, waddr=20, raddr1=12, raddr2=29
# MEM: op=00, waddr=10, rdata1=(  -6,  -28), rdata2=(   9,    5)
# ALU: waddr=17, wdata=(  53,  -15)
#
# Time: #                  180
# IF: op=01, waddr=30, raddr1= 9, raddr2=18
# MEM: op=01, waddr=20, rdata1=(  37,   17), rdata2=(  78,   22)
# ALU: waddr=10, wdata=(   3,  -23)
#
# Time: #                  200
# IF: op=01, waddr= 5, raddr1=26, raddr2=15
# MEM: op=01, waddr=30, rdata1=(  -4,   -5), rdata2=(  -6,  -28)
# ALU: waddr=20, wdata=( -41,   -5)
#
# Time: #                  220
# IF: op=01, waddr=24, raddr1=14, raddr2=17
# MEM: op=01, waddr= 5, rdata1=(  -6,   21), rdata2=(  24,   13)
# ALU: waddr=30, wdata=(   2,   23)
#
# Time: #                  240
# IF: op=10, waddr= 2, raddr1=21, raddr2=24
# MEM: op=01, waddr=24, rdata1=(  -1,    0), rdata2=(  53,  -15)
# ALU: waddr= 5, wdata=( -30,    8)
#
# Time: #                  260
# IF: op=10, waddr=13, raddr1=27, raddr2=10
# MEM: op=10, waddr= 2, rdata1=(   3,    5), rdata2=( -12,   11)
# ALU: waddr=24, wdata=( -54,   15)
#
# Time: #                  280
# IF: op=10, waddr=19, raddr1= 4, raddr2=21
# MEM: op=10, waddr=13, rdata1=(   9,    5), rdata2=(   3,  -23)
# ALU: waddr= 2, wdata=( -91,  -27)
#
```

```
# Time: #                  280
# IF: op=10, waddr=19, raddr1= 4, raddr2=21
# MEM: op=10, waddr=13, rdata1=(   9,    5), rdata2=(   3,  -23)
# ALU: waddr= 2, wdata=( -91,  -27)
#
# Time: #                  300
# IF: op=10, waddr=28, raddr1=10, raddr2= 7
# MEM: op=10, waddr=19, rdata1=(  -8,    7), rdata2=(   3,    5)
# ALU: waddr=13, wdata=(-114,   64)
#
# Time: #                  320
# IF: op=01, waddr=15, raddr1=17, raddr2= 8
# MEM: op=10, waddr=28, rdata1=(   3,  -23), rdata2=(  73,   79)
# ALU: waddr=19, wdata=( -59,  -19)
#
# Time: #                  340
# IF: op=01, waddr= 5, raddr1=20, raddr2= 5
# MEM: op=01, waddr=15, rdata1=(  53,  -15), rdata2=( -12,   13)
# ALU: waddr=28, wdata=( -12,   94)
#
# Time: #                  360
# IF: op=xx, waddr= x, raddr1= x, raddr2= x
# MEM: op=01, waddr= 5, rdata1=( -41,   -5), rdata2=( -30,    8)
# ALU: waddr=15, wdata=(  65,  -28)
#
# Time: #                  380
# IF: op=xx, waddr= x, raddr1= x, raddr2= x
# MEM: op=xx, waddr= x, rdata1=(   x,    x), rdata2=(   x,    x)
# ALU: waddr= 5, wdata=( -11,  -13)
```

که حاصل مطابق انتظار صحیح است. (در هر کلاک، ورودی های آن مرحله و خروجی های مرحله قبلی نمایش داده می‌شود مثلا در لحظه 220، ورودی ها (21 ,6-) و (13 ,24) هستند و op = 01 (عملیات تفریق) که در لحظه 240، حاصل نمایش داده شده است. (8 ,30-))

بدین ترتیب یک پایپلاین طراحی و تست کردیم و فعالیتمان به پایان رسید. (تمامی فایل ها در فایل زیپ پیوست شده اند.)