شماره آزمایش: **8**	موضوع: ALU اعداد مختلط	تاریخ آزمایش: 17 اردیبهشت 1400
عرشیا اخوان **97110422**	محمدحسین عبدی **97110285**	علیرضا ایلامی **97101286**

مقدمه:

در این آزمایش یک واحد محاسبه (ALU) برای اعداد مختلط طراحی میکنیم.

شرح آزمایش:

این آزمایش از سه بخش تشکیل شده است.

- بخش اول: جمع و تفریق اعداد مختلط
- بخش دوم: ضرب اعداد مختلط
- بخش سوم: یک واحد پایپ لاین

برای دو بخش اول:

اگر عدد ما n2 بیت است، فرض کردیم که n بیت برای بخش Real عدد داریم و n بیت هم برای بخش Imaginary عدد.

برای جمع و تفریق لازم بود برای هر کدام از این دو بخش، جمع و تفریق را جدا محاسبه کنیم. یعنی بخش حقیقی آنها با هم و بخش موهومی هم با هم جمع/تفریق می شود.

برای ضرب هم باز همین کار را میکنیم. دو تا پرانتز داریم:

$$(a + bi) * (c + di) = (ac-bd) + (ad-bc)i$$

و با عملیاتی ساده آنها را در هم ضرب می کنیم.

قسمت اصلی این آزمایش، بخش سوم آن، یعنی طراحی پایپ لاین است.

5 تا Stage داریم:

IF: Instruction Fetch

ID: Instruction Decode
WB: Write Back
ALU
Memory Unit

که هر یک از بخش ها را جداگانه تشریح میکنیم:

IF.v

```
module IF (clk,
           rstn,
           inst);

    parameter INST_CAP = 5;
    parameter INST_LEN = 17;

    input clk, rstn;
    output reg[INST_LEN-1:0] inst;

    reg [INST_LEN-1:0] inst_mem [INST_CAP-1:0];
    reg[$clog2(INST_CAP):0] pc;

    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            pc <= 0;
        end
        else begin
            if (pc < INST_CAP) begin
                $display($time, "\t [IF] inst[%d]:%b", pc, inst_mem[pc]);
                inst <= inst_mem[pc];
                pc   <= pc + 1;
            end
        end
    end

endmodule
```

توضیحات IF

در instruction fetch، در هر مرحله یکی از خانه های حافظه را از Instruction Memory میخوانیم.

سپس Program Counter را یک واحد زیاد میکنیم.

و دستور مربوطه را به Instruction Decode می دهد.

ID.v

```
le ID (clk,
           inst,
           rstn,
           oper1,
           oper2,
           dest,
           alu_sig,
           mem_read);

    parameter INST_LEN  = 17;
    parameter WORD_SIZE = 32;
    parameter MEM_SIZE  = 32;
    parameter ADDR_LEN  = 5;

    input clk, rstn;
    input [INST_LEN-1:0] inst;
    output mem_read;
    output [ADDR_LEN-1:0] oper1, oper2, dest;
    output [1:0] alu_sig;

    assign alu_sig  = inst[INST_LEN-1 : INST_LEN-2];
    assign oper1    = inst[3*ADDR_LEN-1 : 2*ADDR_LEN];
    assign oper2    = inst[2*ADDR_LEN-1 : ADDR_LEN];
    assign dest     = inst[ADDR_LEN-1 : 0];
    assign mem_read = clk;

    always @(*) begin
        $display($time, "\t [ID] inst = %b, oper1 = %b, oper2 = %b, dest = %b,
mem_read = %b, alu_sig = %b",inst, oper1, oper2, dest, mem_read, alu_sig);
    end

endmodule
```

توضیحات ID:

می دانیم که هر دستور از بخش های مختلفی تشکیل شده است. در اینجا ابتدا بخش های Reg1, Reg2, Reg3, Opcode را از یکدیگر جدا میکنیم (برای انجام عملیات)
و Opcode را به واحد ALU میدهیم.
2Reg1, Reg را به Memory Unit میدهیم.
enable را ست میکنیم. وقتی این enable ست شد، Memory Unit آن دو آدرس رجیسترها را میخواند. و به ALU میدهد. (و ورودی های ALU آپدیت میشوند.)
3Reg هم که مقصد محاسبه است. به Write Back داده میشود تا خروجی ALU در آن نوشته و ذخیره شود.

Mem.v

```
module MEMORY_UNIT (r_addr1,
                    r_addr2,
                    w_addr,
                    r_en,
                    w_en,
                    data_out1,
                    data_out2,
                    data_in,
                    clk);

    parameter ADDR_LEN  = 5;
    parameter WORD_SIZE = 32;
    parameter MEM_SIZE  = 32;


    input [ADDR_LEN-1:0]r_addr1,r_addr2,w_addr;
    input r_en,w_en,clk;

    input [WORD_SIZE-1:0]data_in;
    output reg [WORD_SIZE-1:0]data_out1,data_out2;

    reg [WORD_SIZE-1:0] mem [MEM_SIZE-1:0];

    reg [WORD_SIZE-1:0] data_o1,data_o2;

    always @(*)
```

```
    begin
        if (r_en) begin
            data_out1 <= mem[r_addr1];
            data_out2 <= mem[r_addr2];
        end
        if (w_en) begin
```

توضیحات Memory Unit

با توجه به اینکه ما میتوانیم همزمان دو جای مختلف از Memory را بخوانیم، باید دو تا Read Address داشته باشیم. و همچنین یک Write Address برای نوشتن در حافظه.

اگر در حال نوشتن باشیم، یک data in و یک write address میگیریم. که در خانه مربوطه، دیتا را بنویسیم. و اگر در حال خواندن باشیم، دو تا خروجی data ,2data1 داریم.)

پاین حافظه read و write را با هم ساپورت میکند، اما ما به گونه ای از آن استفاده میکنیم که وقتی clk=1 است، بخواند و وقتی در clk=0 است، بنویسید.

ALU.v

```
`define OPP_ASN 1
`define OPP_MUL 0

module ALU (clk,
            a,
            b,
            res,
            control_sig);

    parameter PART_LEN = 8;

    input [2*PART_LEN-1:0] a, b;
    output [2*PART_LEN-1:0] res;
    input clk;

    input [1:0]control_sig;

    wire asn = control_sig[0];
    wire opp = control_sig[1];
```

```
    wire [2*PART_LEN-1:0]res_as,res_mul;

    MUL_C #(.PART_LEN(PART_LEN))mulc(
    .a(a),
    .b(b),
    .res(res_as)
    );

    AS_C #(.PART_LEN(PART_LEN)) asc(
    .a(a),
    .b(b),
    .asn(asn),
    .res(res_mul));

    assign res = (opp != `OPP_ASN) ? res_as: res_mul;

    always @(*)
        $display($time, "\t [ALU] a = %b, b = %b, control = %b, res = %b",
a, b, control_sig, res);

endmodule
```

توضیحات ALU:

واحد محاسبه ما ترکیبی است. و نه ترتیبی. به این معنا که به ازای ورودی های داده شده هم جمع هم تفریق و هم ضرب را محاسبه میکند ولی به ازای سیگنال Operand ای که ما داده ایم، خروجی مربوطه را به ما نمایش میدهد و دو تای دیگر را نشان نمیدهد. (هرچند که محاسبه شده است)

خروجی ALU هم به ماژول Write Back داده میشود.

WB.v

```
module WB (dst_addr_i,
           data_i,
           clk,
           dst_addr_o,
           data_o,
```

```
            w_en);

    parameter ADDR_LEN  = 5;
    parameter WORD_SIZE = 32;

    input [ADDR_LEN-1:0]dst_addr_i;
    input [WORD_SIZE-1:0]data_i;
    input clk;

    output [ADDR_LEN-1:0]dst_addr_o;
    output [WORD_SIZE-1:0]data_o;
    output w_en;

    reg [ADDR_LEN-1:0]dst_addr_out;
    reg [WORD_SIZE-1:0]data_out;
    reg memory_write_mod;

    assign dst_addr_o = dst_addr_out;
    assign data_o     = data_out;
    assign w_en       = memory_write_mod;

    always @(*)
    begin
        dst_addr_out     <= dst_addr_i;
        data_out         <= data_i;
        memory_write_mod <= !clk;
    end

    initial
        $monitor($time, "\t [WB] dst_addr_i = %d, data_i = %b, dst_addr_o =
%d, data_o = %b, w_en = %b", dst_addr_i, data_i, dst_addr_o, data_o, w_en);

endmodule
```

توضیحات Write Back:

این ماژول کارش این است که نتیجه نهایی را از ماژول ALU گرفته و در آدرسی که ما میخواهیم بنویسد و در حافظه ذخیره کند.

ورودی های WB خروجی ALU و آدرس رجیستر سوم (3 Reg) که از ID بدست آمده است، میباشد.

علت وجود ماژول Write Back برای جلوگیری از ایجاد مخاطرات (Hazard) است.

(اگر داده در همان کلاکی که میخواهد خوانده شود برسد، بدون WB دچار مخاطره میشویم.)

ماژول های داخلی:

Add_sub_comp.v

```
module AS_C (a,
             b,
             asn,
             res);

    parameter PART_LEN = 8;
    input asn;
    input signed [2*PART_LEN-1:0] a,b;
    output signed [2*PART_LEN-1:0]res;


    wire signed [PART_LEN-1:0]ra = a[2*PART_LEN-1:PART_LEN];
    wire signed [PART_LEN-1:0]rb = b[2*PART_LEN-1:PART_LEN];

    wire signed [PART_LEN-1:0]ia = a[PART_LEN-1:0];
    wire signed [PART_LEN-1:0]ib = b[PART_LEN-1:0];

    wire signed [PART_LEN:0]rr = (asn) ? ra + rb : ra - rb;
    wire signed [PART_LEN:0]ir = (asn) ? ia + ib : ia - ib;

    assign res = { rr[PART_LEN-1:0],ir[PART_LEN-1:0] };


endmodule
```

Mul.v

```
module MUL_C(a,
             b,
             res);
```

```
    parameter PART_LEN = 8;
    input signed [2*PART_LEN-1:0] a,b;
    output signed [2*PART_LEN-1:0]res;


    wire signed [PART_LEN-1:0]ra = a[2*PART_LEN-1:PART_LEN];
    wire signed [PART_LEN-1:0]rb = b[2*PART_LEN-1:PART_LEN];

    wire signed [PART_LEN-1:0]ia = a[PART_LEN-1:0];
    wire signed [PART_LEN-1:0]ib = b[PART_LEN-1:0];

    wire signed [PART_LEN:0]rr = ra*rb - ia*ib;
    wire signed [PART_LEN:0]ir = ra*ib + rb*ia;

    assign res = {rr[PART_LEN-1:0], ir[PART_LEN-1:0]};

endmodule
```

Pipeline_processor.v

```
module PIPELINE(clk,
                rstn);

    parameter PART_LEN  = 16;
    parameter WORD_SIZE = 32;
    parameter ADDR_LEN  = 5;
    parameter MEM_SIZE  = 32;
    parameter INST_LEN  = 17;
    parameter INST_CAP  = 5;

    input clk, rstn;
    wire w_en, r_en;
    wire [INST_LEN-1:0] inst;
    wire [ADDR_LEN-1:0] r_addr1, r_addr2, w_addr, id_dest_addr;
    wire [1:0] alu_sig;
    wire [WORD_SIZE-1:0] res, wb_data_out, data_out1, data_out2;

    reg id_mem_r_en;
```

```
    reg [1:0] id_alu_sig, alu_sig_buff;
    reg [ADDR_LEN-1:0] dst_addr_buff, dst_addr, id_mem_r_addr1, id_mem_r_addr2;
    reg [WORD_SIZE-1:0] mem_alu_data_out1, mem_alu_data_out2;
    reg [INST_LEN-1:0] if_id_buff_inst;

    always @(posedge clk)
    begin
        dst_addr_buff <= id_dest_addr;
        dst_addr      <= dst_addr_buff;

        if_id_buff_inst   <= inst;
        id_mem_r_addr1    <= r_addr1;
        id_mem_r_addr2    <= r_addr2;
        id_mem_r_en       <= r_en;
        alu_sig_buff      <= alu_sig;
        id_alu_sig        <= alu_sig_buff;
        mem_alu_data_out1 <= data_out1;
        mem_alu_data_out2 <= data_out2;
    end

    IF #(.INST_LEN(INST_LEN), .INST_CAP(INST_CAP)) ifm (
    .clk(clk),
    .rstn(rstn),
    .inst(inst)
    );

    ID #(.INST_LEN(INST_LEN), .MEM_SIZE(MEM_SIZE), .WORD_SIZE(WORD_SIZE),
.ADDR_LEN(ADDR_LEN)) idm (
    .clk(clk),
    .inst(if_id_buff_inst),
    .rstn(rstn),
    .oper1(r_addr1),
    .oper2(r_addr2),
    .dest(id_dest_addr),
    .mem_read(r_en),
    .alu_sig(alu_sig)
    );

    MEMORY_UNIT
#(.ADDR_LEN(ADDR_LEN),.WORD_SIZE(WORD_SIZE),.MEM_SIZE(MEM_SIZE))memory(
    .r_addr1(id_mem_r_addr1),
    .r_addr2(id_mem_r_addr2),
```

```
    .w_addr(w_addr),
    .w_en(w_en),
    .r_en(id_mem_r_en),
    .data_in(wb_data_out),
    .data_out1(data_out1),
    .data_out2(data_out2),
    .clk(clk)
    );

    ALU #(.PART_LEN(PART_LEN))alu (
    .clk(clk),
    .a(mem_alu_data_out1),
    .b(mem_alu_data_out2),
    .control_sig(id_alu_sig),
    .res(res)
    );

    WB #(.ADDR_LEN(ADDR_LEN),.WORD_SIZE(WORD_SIZE))wbm(
    .dst_addr_i(dst_addr),
    .data_i(res),
    .clk(clk),
    .dst_addr_o(w_addr),
    .data_o(wb_data_out),
    .w_en(w_en)
    );

endmodule
```

توضیحات: همانطور که میدانیم، لازم است در معماری پایپ لاین، بین Stage ها ، برای ذخیره اطلاعات، آن ها را در Register ها ذخیره کنیم.

توضیح هرکدام از انها:

IF, ID: `if_id_buff_inst`
ID, MU: `id_mem_r_addr1, id_mem_r_addr2, id_mem_r_en`
MU. ALU: `mem_alu_data_out1, mem_alu_data_out2`
ID, WB: `Id_dest_addr`

نکته مهم:

برای انتقال `alu_sig` از ID به ALU نمی توانیم مستقیم عمل کنیم چون آن وسط MU را داریم. پس آن را باید در یک بافر موقتی به نام `alu_sig_buff` بریزیم و در نهایت آن را به `id_alu_sig` انتقال دهیم.

این مساله را برای نیز `Id_dest_addr` داریم.

چون بعد از مازول WB دیگر هیچ ماژولی نداریم و پایپ لاین تمام میشود، نیازی نیست برای انتقال از WB به MU، از رجیستر استفاده کنیم.

Testbench.v

```
module testbench();

    parameter PART_LEN  = 16;
    parameter WORD_SIZE = 32;
    parameter ADDR_LEN  = 5;
    parameter MEM_SIZE  = 32;
    parameter INST_LEN  = 17;
    parameter INST_CAP  = 5;
    parameter clk_c     = 10;

    reg clk, rstn;

    PIPELINE #(
    .PART_LEN(PART_LEN),
    .WORD_SIZE(WORD_SIZE),
    .ADDR_LEN(ADDR_LEN),
    .MEM_SIZE(MEM_SIZE),
    .INST_LEN(INST_LEN),
    .INST_CAP(INST_CAP)
    ) pipe (
    .clk(clk),
    .rstn(rstn)
    );
```

```
    initial begin
        $dumpfile("report/waveform.vcd");
        $dumpvars(0,pipe);
    end

    initial begin
        clk         = 0;
        forever clk = #(clk_c/2) ~clk;
    end

    initial begin
        $readmemb("report/memory.mem", pipe.memory.mem, 0, MEM_SIZE-1);
        $readmemb("report/is.mem", pipe.ifm.inst_mem, 0, INST_CAP-1);
    end

    initial begin
        rstn = 0;
        #clk_c
        rstn = 1;
        #(10*clk_c);
        $writememb("report/result.mem", pipe.memory.mem);
        $finish;
    end

endmodule
```

توضیحات تست بنچ:

برای خواندن از مموری از دستور readmemb استفاده کرده ایم. به این علت که از بیت های باینری استفاده کرده ایم.

برای ضرب و جمع از اعداد رندوم استفاده کرده ایم.

Testbench_add_sub.v

```
`define NULL 0
module AS_C_TESTBENCH();

    parameter PART_LEN = 8;
    parameter clk_c    = 10;
    reg clk, rstn;
```

```
initial begin
    clk         = 0;
    forever clk = #(clk_c/2) ~clk;
end

reg asn;
reg signed [2*PART_LEN-1:0] a,b;
wire signed [2*PART_LEN-1:0] res;

AS_C #(.PART_LEN(PART_LEN)) as_c (a,b,asn,res);

integer data_file;
integer scan_file;
integer seed;
initial begin
    data_file = $fopen("seed.dat", "r");
    if (data_file == `NULL) begin
        $display("data_file handle was NULL");
        $finish;
    end
    scan_file = $fscanf(data_file, "%d", seed);
    if (scan_file == `NULL) begin
        $display("integer read error");
        $finish;
    end
end

integer n;
integer i;
reg [PART_LEN-3:0]ra_tmp,rb_tmp,ia_tmp,ib_tmp;
initial begin
    n = 10;
    $display("testing add");
    asn = 1;
    for (i = 0; i < n; i++) begin
        ra_tmp = {(PART_LEN-2){$random(seed)}};
        rb_tmp = {(PART_LEN-2){$random(seed)}};
        ia_tmp = {(PART_LEN-2){$random(seed)}};
        ib_tmp = {(PART_LEN-2){$random(seed)}};

        a = {2'b00,ra_tmp,2'b00,ia_tmp};
```

```
            b = {2'b00,rb_tmp,2'b00,ib_tmp};
            #clk_c;
        end

        #clk_c;
        $display("testing sub");
        asn = 0;
        for (i = 0; i < n; i++) begin
            ra_tmp = {(PART_LEN-2){$random(seed)}};
            rb_tmp = {(PART_LEN-2){$random(seed)}};
            ia_tmp = {(PART_LEN-2){$random(seed)}};
            ib_tmp = {(PART_LEN-2){$random(seed)}};

            a = {2'b00,ra_tmp,2'b00,ia_tmp};
            b = {2'b00,rb_tmp,2'b00,ib_tmp};
            #clk_c;
        end
        $finish;
    end

    initial
        $monitor($time, "\t(%d, %di) (+/-) (%d, %di) = (%d, %di)",
a[2*PART_LEN-1:PART_LEN], a[PART_LEN-1:0], b[2*PART_LEN-1:PART_LEN],
b[PART_LEN-1:0], res[2*PART_LEN-1:PART_LEN], res[PART_LEN-1:0]);

endmodule
```

Testbench_mul.v

```
`define NULL 0
module MUL_C_TESTBENCH();

    parameter PART_LEN = 8;
    parameter clk_c    = 10;

    reg clk;
    reg signed [2*PART_LEN-1:0] a, b;
    wire signed [2*PART_LEN-1:0] c;

    MUL_C #(.PART_LEN(PART_LEN)) mulc(
    .a(a),
```

```
    .b(b),
    .res(c)
    );

    integer data_file;
    integer scan_file;
    integer seed;
    initial begin
        data_file = $fopen("seed.dat", "r");
        if (data_file == `NULL) begin
            $display("data_file handle was NULL");
            $finish;
        end
        scan_file = $fscanf(data_file, "%d", seed);
        if (scan_file == `NULL) begin
            $display("integer read error");
            $finish;
        end
    end

    initial begin
        clk         = 0;
        forever clk = #(clk_c/2) ~clk;
    end

    integer n;
    integer i;
    reg signed [PART_LEN-3:0]ra_tmp,rb_tmp,ia_tmp,ib_tmp;
    initial begin
        n = 10;
        $display("testing mul");
        for (i = 0; i < n; i++) begin
            ra_tmp = {(PART_LEN-2){$random(seed)}};
            rb_tmp = {(PART_LEN-2){$random(seed)}};
            ia_tmp = {(PART_LEN-2){$random(seed)}};
            ib_tmp = {(PART_LEN-2){$random(seed)}};

            a = {2'b00,ra_tmp,2'b00,ia_tmp};
            b = {2'b00,rb_tmp,2'b00,ib_tmp};
            #clk_c;
        end
        $finish;
```

```
    end

    initial
        $monitor($time, "\t(%d, %di) * (%d, %di) = (%d, %di)",
a[2*PART_LEN-1:PART_LEN], a[PART_LEN-1:0], b[2*PART_LEN-1:PART_LEN],
b[PART_LEN-1:0], c[2*PART_LEN-1:PART_LEN], c[PART_LEN-1:0]);

endmodule
```

نتایج:

Is.mem

```
11_00011_00011_00011  // ADD
01_00010_00010_00010  // MUL v
00_00000_00000_00000  // STALL
00_00000_00000_00000  // STALL
10_00011_00001_00111  // SUB


/*
IF, ID, ALU, WR
4 register -> size(2*BIT_PART), A, B, C, D;
word -> 32bit;
mem -> 32*32 bit;
opcode: 2bit, 5, 5, 5 -> (7bit)
ADD(mem1, mem2, mem3) -> 11
SUB(mem1, mem2, mem3) -> 10
MUL(mem1, mem2, mem3) -> 01
*/
```

Memory.mem

```
00000000_00000000_00000000_00000000
00000000_00000001_00000000_00000001
00000000_00000010_00000000_00000010
00000000_00000001_00000000_00000001
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
```

```
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
00000000_00000000_00000000_00000000
```

این خانه های حافظه قبل از انجام عملیات است.

Result.mem

```
// 0x00000000
00000000000000000000000000000000
00000000000000010000000000000001
00000000000000000000000000001000
00000000000000100000000000000010
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000010000000000000001
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
// 0x00000010
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
```

این خانه های حافظه بعد از انجام عملیات است.

فایل نتایج:

Result_asc.txt

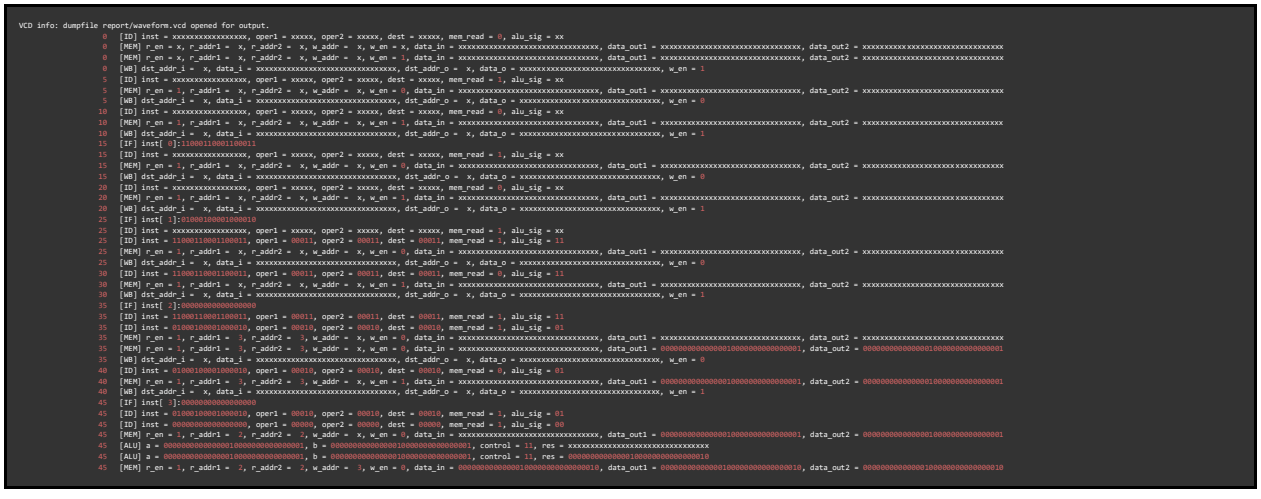
```
testing add
                  0    ( 14,  39i) (+/-) ( 27,   4i) = ( 41,  43i)
                 10    ( 16,  39i) (+/-) (  4,  23i) = ( 20,  62i)
                 20    ( 41,  60i) (+/-) ( 41,  39i) = ( 82,  99i)
                 30    ( 14,  26i) (+/-) ( 48,  14i) = ( 62,  40i)
                 40    ( 22,  20i) (+/-) ( 15,  62i) = ( 37,  82i)
                 50    ( 57,  52i) (+/-) ( 62,  11i) = (119,  63i)
                 60    (  8,  10i) (+/-) ( 12,  40i) = ( 20,  50i)
                 70    (  6,  40i) (+/-) ( 27,  40i) = ( 33,  80i)
                 80    ( 21,   7i) (+/-) ( 30,  38i) = ( 51,  45i)
                 90    (  0,  13i) (+/-) ( 62,  36i) = ( 62,  49i)
testing sub
```

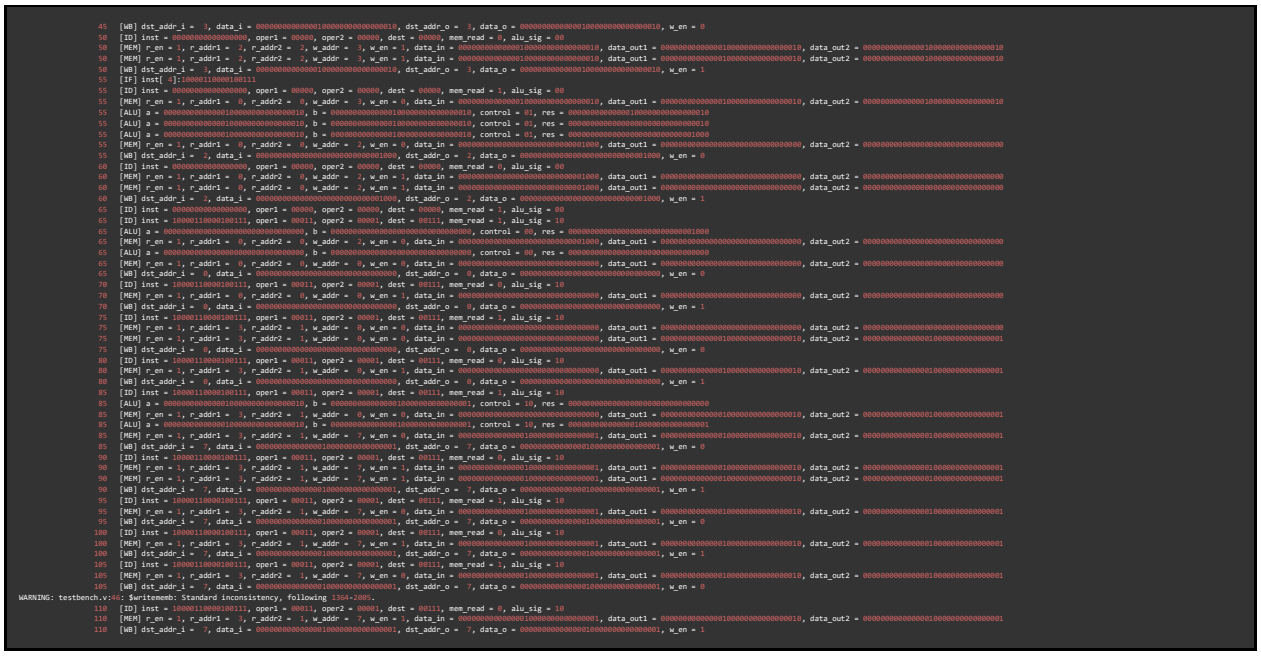
```
110    ( 10,  42i) (+/-) ( 43,  59i) = (223, 239i)
120    (  7,   1i) (+/-) (  1,  56i) = (  6, 201i)
130    ( 38,  30i) (+/-) ( 15,  51i) = ( 23, 235i)
140    ( 60,  38i) (+/-) ( 29,   9i) = ( 31,  29i)
150    ( 47,   4i) (+/-) (  6,   0i) = ( 41,   4i)
160    (  9,   1i) (+/-) (  3,  59i) = (  6, 198i)
170    ( 41,   5i) (+/-) ( 55,   5i) = (242,   0i)
180    (  2,  58i) (+/-) ( 26,  36i) = (232,  22i)
190    ( 44,   1i) (+/-) ( 30,  49i) = ( 14, 208i)
200    ( 50,  60i) (+/-) ( 23,  32i) = ( 27,  28i)
```

Result_mulc.txt

```
testing mul
  0    ( 14,  39i) * ( 27,   4i) = (222,  85i)
 10    ( 16,  39i) * (  4,  23i) = (191,  12i)
 20    ( 41,  60i) * ( 41,  39i) = (109, 219i)
 30    ( 14,  26i) * ( 48,  14i) = ( 52, 164i)
 40    ( 22,  20i) * ( 15,  62i) = (114, 128i)
 50    ( 57,  52i) * ( 62,  11i) = (146,  11i)
 60    (  8,  10i) * ( 12,  40i) = (208, 184i)
 70    (  6,  40i) * ( 27,  40i) = ( 98,  40i)
 80    ( 21,   7i) * ( 30,  38i) = (108, 240i)
 90    (  0,  13i) * ( 62,  36i) = ( 44,  38i)
```

Result.txt




45 [WB] dst_addr_i = 3, data_i = 00000000000001000000000000000010, dst_addr_o = 3, data_o = 00000000000001000000000000000010, w_en = 0
50 [ID] inst = 00000000000000000, oper1 = 00000, oper2 = 00000, dest = 00000, mem_read = 0, alu_sig = 00
50 [MEM] r_en = 1, r_addr1 = 7, r_addr2 = 7, w_addr = 3, w_en = 1, data_in = 00000000000001000000000000000010, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000010
50 [MEM] r_en = 1, r_addr1 = 2, r_addr2 = 2, w_addr = 3, w_en = 1, data_in = 00000000000001000000000000000010, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000010
50 [WB] dst_addr_i = 3, data_i = 00000000000001000000000000000010, dst_addr_o = 3, data_o = 00000000000001000000000000000010, w_en = 1
55 [IF] inst[0]:10000110000100111
55 [ID] inst = 00000000000000000, oper1 = 00000, oper2 = 00000, dest = 00000, mem_read = 1, alu_sig = 00
55 [MEM] r_en = 1, r_addr1 = 0, r_addr2 = 0, w_addr = 3, w_en = 0, data_in = 00000000000001000000000000000010, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000010
55 [ALU] a = 00000000000001000000000000000010, b = 00000000000001000000000000000010, control = 01, res = 00000000000010000000000000000010
55 [ALU] a = 00000000000001000000000000000010, b = 00000000000001000000000000000010, control = 01, res = 00000000000000000000000000000010
55 [ALU] a = 00000000000001000000000000000010, b = 00000000000001000000000000000010, control = 01, res = 00000000000000000000000000001000
55 [MEM] r_en = 1, r_addr1 = 0, r_addr2 = 0, w_addr = 2, w_en = 0, data_in = 00000000000000000000000000001000, data_out1 = 00000000000000000000000000000000, data_out2 = 00000000000000000000000000000000
55 [WB] dst_addr_i = 2, data_i = 00000000000000000000000000001000, dst_addr_o = 2, data_o = 00000000000000000000000000001000, w_en = 0
60 [ID] inst = 00000000000000000, oper1 = 00000, oper2 = 00000, dest = 00000, mem_read = 0, alu_sig = 00
60 [MEM] r_en = 1, r_addr1 = 0, r_addr2 = 0, w_addr = 2, w_en = 1, data_in = 00000000000000000000000000001000, data_out1 = 00000000000000000000000000000000, data_out2 = 00000000000000000000000000000000
60 [MEM] r_en = 1, r_addr1 = 0, r_addr2 = 0, w_addr = 2, w_en = 1, data_in = 00000000000000000000000000001000, data_out1 = 00000000000000000000000000000000, data_out2 = 00000000000000000000000000000000
60 [WB] dst_addr_i = 2, data_i = 00000000000000000000000000001000, dst_addr_o = 2, data_o = 00000000000000000000000000001000, w_en = 1
65 [ID] inst = 00000000000000000, oper1 = 00000, oper2 = 00000, dest = 00000, mem_read = 1, alu_sig = 00
65 [ID] inst = 10000110000100111, oper1 = 00011, oper2 = 00001, dest = 00111, mem_read = 1, alu_sig = 10
65 [ALU] a = 00000000000000000000000000000000, b = 00000000000000000000000000000000, control = 00, res = 00000000000000000000000001000
65 [MEM] r_en = 1, r_addr1 = 0, r_addr2 = 0, w_addr = 2, w_en = 0, data_in = 00000000000000000000000001000, data_out1 = 00000000000000000000000000000000, data_out2 = 00000000000000000000000000000000
65 [ALU] a = 00000000000000000000000000000000, b = 00000000000000000000000000000000, control = 00, res = 00000000000000000000000000000000
65 [MEM] r_en = 1, r_addr1 = 0, r_addr2 = 0, w_addr = 0, w_en = 0, data_in = 00000000000000000000000000000000, data_out1 = 00000000000000000000000000000000, data_out2 = 00000000000000000000000000000000
65 [WB] dst_addr_i = 0, data_i = 00000000000000000000000000000000, dst_addr_o = 0, data_o = 00000000000000000000000000000000, w_en = 0
70 [ID] inst = 10000110000100111, oper1 = 00011, oper2 = 00001, dest = 00111, mem_read = 0, alu_sig = 10
70 [MEM] r_en = 1, r_addr1 = 0, r_addr2 = 0, w_addr = 0, w_en = 1, data_in = 00000000000000000000000000000000, data_out1 = 00000000000000000000000000000000, data_out2 = 00000000000000000000000000000000
70 [WB] dst_addr_i = 0, data_i = 00000000000000000000000000000000, dst_addr_o = 0, data_o = 00000000000000000000000000000000, w_en = 1
75 [ID] inst = 10000110000100111, oper1 = 00011, oper2 = 00001, dest = 00111, mem_read = 1, alu_sig = 10
75 [MEM] r_en = 1, r_addr1 = 7, r_addr2 = 1, w_addr = 0, w_en = 0, data_in = 00000000000000000000000000000000, data_out1 = 00000000000000000000000000000000, data_out2 = 00000000000010000000000000000001
75 [MEM] r_en = 1, r_addr1 = 3, r_addr2 = 1, w_addr = 0, w_en = 0, data_in = 00000000000000000000000000000000, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000001
75 [WB] dst_addr_i = 0, data_i = 00000000000000000000000000000000, dst_addr_o = 0, data_o = 00000000000000000000000000000000, w_en = 0
80 [ID] inst = 10000110000100111, oper1 = 00011, oper2 = 00001, dest = 00111, mem_read = 0, alu_sig = 10
80 [MEM] r_en = 1, r_addr1 = 3, r_addr2 = 1, w_addr = 0, w_en = 1, data_in = 00000000000000000000000000000000, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000001
80 [WB] dst_addr_i = 0, data_i = 00000000000000000000000000000000, dst_addr_o = 0, data_o = 00000000000000000000000000000000, w_en = 1
85 [ID] inst = 10000110000100111, oper1 = 00011, oper2 = 00001, dest = 00111, mem_read = 1, alu_sig = 10
85 [ALU] a = 00000000000001000000000000000010, b = 00000000000010000000000000000001, control = 10, res = 00000000000000000000000000000000
85 [MEM] r_en = 1, r_addr1 = 3, r_addr2 = 1, w_addr = 0, w_en = 0, data_in = 00000000000000000000000000000000, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000001
85 [ALU] a = 00000000000001000000000000000010, b = 00000000000010000000000000000001, control = 10, res = 00000000000001000000000000000001
85 [MEM] r_en = 1, r_addr1 = 3, r_addr2 = 1, w_addr = 7, w_en = 0, data_in = 00000000000001000000000000000001, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000001
85 [WB] dst_addr_i = 7, data_i = 00000000000001000000000000000001, dst_addr_o = 7, data_o = 00000000000001000000000000000001, w_en = 0
90 [ID] inst = 10000110000100111, oper1 = 00011, oper2 = 00001, dest = 00111, mem_read = 0, alu_sig = 10
90 [MEM] r_en = 1, r_addr1 = 7, r_addr2 = 1, w_addr = 7, w_en = 1, data_in = 00000000000001000000000000000001, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000001
90 [MEM] r_en = 1, r_addr1 = 3, r_addr2 = 1, w_addr = 7, w_en = 1, data_in = 00000000000001000000000000000001, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000001
90 [WB] dst_addr_i = 7, data_i = 00000000000001000000000000000001, dst_addr_o = 7, data_o = 00000000000001000000000000000001, w_en = 1
95 [ID] inst = 10000110000100111, oper1 = 00011, oper2 = 00001, dest = 00111, mem_read = 1, alu_sig = 10
95 [MEM] r_en = 1, r_addr1 = 3, r_addr2 = 1, w_addr = 7, w_en = 0, data_in = 00000000000001000000000000000001, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000001
95 [WB] dst_addr_i = 7, data_i = 00000000000001000000000000000001, dst_addr_o = 7, data_o = 00000000000001000000000000000001, w_en = 0
100 [ID] inst = 10000110000100111, oper1 = 00011, oper2 = 00001, dest = 00111, mem_read = 0, alu_sig = 10
100 [MEM] r_en = 1, r_addr1 = 3, r_addr2 = 1, w_addr = 7, w_en = 1, data_in = 00000000000001000000000000000001, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000001
100 [WB] dst_addr_i = 7, data_i = 00000000000001000000000000000001, dst_addr_o = 7, data_o = 00000000000001000000000000000001, w_en = 1
105 [ID] inst = 10000110000100111, oper1 = 00011, oper2 = 00001, dest = 00111, mem_read = 1, alu_sig = 10
105 [MEM] r_en = 1, r_addr1 = 3, r_addr2 = 1, w_addr = 7, w_en = 0, data_in = 00000000000001000000000000000001, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000001
105 [WB] dst_addr_i = 7, data_i = 00000000000001000000000000000001, dst_addr_o = 7, data_o = 00000000000001000000000000000001, w_en = 0
WARNING: testbench.v:66: $writememb: Standard inconsistency, following 1364-2005.
110 [ID] inst = 10000110000100111, oper1 = 00011, oper2 = 00001, dest = 00111, mem_read = 0, alu_sig = 10
110 [MEM] r_en = 1, r_addr1 = 3, r_addr2 = 1, w_addr = 7, w_en = 1, data_in = 00000000000001000000000000000001, data_out1 = 00000000000001000000000000000010, data_out2 = 00000000000010000000000000000001
110 [WB] dst_addr_i = 7, data_i = 00000000000001000000000000000001, dst_addr_o = 7, data_o = 00000000000001000000000000000001, w_en = 1