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

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 2n 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.

Reg1, Reg2 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.)

Reg3 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 data1, data2 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 (Reg 3) 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 |

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 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

|  |
| --- |
| VCD info: dumpfile report/waveform.vcd opened for output.  0 [ID] inst = xxxxxxxxxxxxxxxxx, oper1 = xxxxx, oper2 = xxxxx, dest = xxxxx, mem\_read = 0, alu\_sig = xx  0 [MEM] r\_en = x, r\_addr1 = x, r\_addr2 = x, w\_addr = x, w\_en = x, data\_in = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  0 [MEM] r\_en = x, r\_addr1 = x, r\_addr2 = x, w\_addr = x, w\_en = 1, data\_in = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  0 [WB] dst\_addr\_i = x, data\_i = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, dst\_addr\_o = x, data\_o = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, w\_en = 1  5 [ID] inst = xxxxxxxxxxxxxxxxx, oper1 = xxxxx, oper2 = xxxxx, dest = xxxxx, mem\_read = 1, alu\_sig = xx  5 [MEM] r\_en = 1, r\_addr1 = x, r\_addr2 = x, w\_addr = x, w\_en = 0, data\_in = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  5 [WB] dst\_addr\_i = x, data\_i = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, dst\_addr\_o = x, data\_o = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, w\_en = 0  10 [ID] inst = xxxxxxxxxxxxxxxxx, oper1 = xxxxx, oper2 = xxxxx, dest = xxxxx, mem\_read = 0, alu\_sig = xx  10 [MEM] r\_en = 1, r\_addr1 = x, r\_addr2 = x, w\_addr = x, w\_en = 1, data\_in = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  10 [WB] dst\_addr\_i = x, data\_i = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, dst\_addr\_o = x, data\_o = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, w\_en = 1  15 [IF] inst[ 0]:11000110001100011  15 [ID] inst = xxxxxxxxxxxxxxxxx, oper1 = xxxxx, oper2 = xxxxx, dest = xxxxx, mem\_read = 1, alu\_sig = xx  15 [MEM] r\_en = 1, r\_addr1 = x, r\_addr2 = x, w\_addr = x, w\_en = 0, data\_in = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  15 [WB] dst\_addr\_i = x, data\_i = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, dst\_addr\_o = x, data\_o = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, w\_en = 0  20 [ID] inst = xxxxxxxxxxxxxxxxx, oper1 = xxxxx, oper2 = xxxxx, dest = xxxxx, mem\_read = 0, alu\_sig = xx  20 [MEM] r\_en = 1, r\_addr1 = x, r\_addr2 = x, w\_addr = x, w\_en = 1, data\_in = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  20 [WB] dst\_addr\_i = x, data\_i = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, dst\_addr\_o = x, data\_o = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, w\_en = 1  25 [IF] inst[ 1]:01000100001000010  25 [ID] inst = xxxxxxxxxxxxxxxxx, oper1 = xxxxx, oper2 = xxxxx, dest = xxxxx, mem\_read = 1, alu\_sig = xx  25 [ID] inst = 11000110001100011, oper1 = 00011, oper2 = 00011, dest = 00011, mem\_read = 1, alu\_sig = 11  25 [MEM] r\_en = 1, r\_addr1 = x, r\_addr2 = x, w\_addr = x, w\_en = 0, data\_in = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  25 [WB] dst\_addr\_i = x, data\_i = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, dst\_addr\_o = x, data\_o = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, w\_en = 0  30 [ID] inst = 11000110001100011, oper1 = 00011, oper2 = 00011, dest = 00011, mem\_read = 0, alu\_sig = 11  30 [MEM] r\_en = 1, r\_addr1 = x, r\_addr2 = x, w\_addr = x, w\_en = 1, data\_in = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  30 [WB] dst\_addr\_i = x, data\_i = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, dst\_addr\_o = x, data\_o = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, w\_en = 1  35 [IF] inst[ 2]:00000000000000000  35 [ID] inst = 11000110001100011, oper1 = 00011, oper2 = 00011, dest = 00011, mem\_read = 1, alu\_sig = 11  35 [ID] inst = 01000100001000010, oper1 = 00010, oper2 = 00010, dest = 00010, mem\_read = 1, alu\_sig = 01  35 [MEM] r\_en = 1, r\_addr1 = 3, r\_addr2 = 3, w\_addr = x, w\_en = 0, data\_in = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  35 [MEM] r\_en = 1, r\_addr1 = 3, r\_addr2 = 3, w\_addr = x, w\_en = 0, data\_in = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out1 = 00000000000000010000000000000001, data\_out2 = 00000000000000010000000000000001  35 [WB] dst\_addr\_i = x, data\_i = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, dst\_addr\_o = x, data\_o = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, w\_en = 0  40 [ID] inst = 01000100001000010, oper1 = 00010, oper2 = 00010, dest = 00010, mem\_read = 0, alu\_sig = 01  40 [MEM] r\_en = 1, r\_addr1 = 3, r\_addr2 = 3, w\_addr = x, w\_en = 1, data\_in = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out1 = 00000000000000010000000000000001, data\_out2 = 00000000000000010000000000000001  40 [WB] dst\_addr\_i = x, data\_i = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, dst\_addr\_o = x, data\_o = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, w\_en = 1  45 [IF] inst[ 3]:00000000000000000  45 [ID] inst = 01000100001000010, oper1 = 00010, oper2 = 00010, dest = 00010, mem\_read = 1, alu\_sig = 01  45 [ID] inst = 00000000000000000, oper1 = 00000, oper2 = 00000, dest = 00000, mem\_read = 1, alu\_sig = 00  45 [MEM] r\_en = 1, r\_addr1 = 2, r\_addr2 = 2, w\_addr = x, w\_en = 0, data\_in = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, data\_out1 = 00000000000000010000000000000001, data\_out2 = 00000000000000010000000000000001  45 [ALU] a = 00000000000000010000000000000001, b = 00000000000000010000000000000001, control = 11, res = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  45 [ALU] a = 00000000000000010000000000000001, b = 00000000000000010000000000000001, control = 11, res = 00000000000000100000000000000010  45 [MEM] r\_en = 1, r\_addr1 = 2, r\_addr2 = 2, w\_addr = 3, w\_en = 0, data\_in = 00000000000000100000000000000010, data\_out1 = 00000000000000100000000000000010, data\_out2 = 00000000000000100000000000000010  45 [WB] dst\_addr\_i = 3, data\_i = 00000000000000100000000000000010, dst\_addr\_o = 3, data\_o = 00000000000000100000000000000010, 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 = 2, r\_addr2 = 2, w\_addr = 3, w\_en = 1, data\_in = 00000000000000100000000000000010, data\_out1 = 00000000000000100000000000000010, data\_out2 = 00000000000000100000000000000010  50 [MEM] r\_en = 1, r\_addr1 = 2, r\_addr2 = 2, w\_addr = 3, w\_en = 1, data\_in = 00000000000000100000000000000010, data\_out1 = 00000000000000100000000000000010, data\_out2 = 00000000000000100000000000000010  50 [WB] dst\_addr\_i = 3, data\_i = 00000000000000100000000000000010, dst\_addr\_o = 3, data\_o = 00000000000000100000000000000010, w\_en = 1  55 [IF] inst[ 4]: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 = 00000000000000100000000000000010, data\_out1 = 00000000000000100000000000000010, data\_out2 = 00000000000000100000000000000010  55 [ALU] a = 00000000000000100000000000000010, b = 00000000000000100000000000000010, control = 01, res = 00000000000000100000000000000010  55 [ALU] a = 00000000000000100000000000000010, b = 00000000000000100000000000000010, control = 01, res = 00000000000000000000000000000010  55 [ALU] a = 00000000000000100000000000000010, b = 00000000000000100000000000000010, 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 = 00000000000000000000000000001000  65 [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  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 = 3, r\_addr2 = 1, w\_addr = 0, w\_en = 0, data\_in = 00000000000000000000000000000000, data\_out1 = 00000000000000000000000000000000, data\_out2 = 00000000000000000000000000000000  75 [MEM] r\_en = 1, r\_addr1 = 3, r\_addr2 = 1, w\_addr = 0, w\_en = 0, data\_in = 00000000000000000000000000000000, data\_out1 = 00000000000000100000000000000010, data\_out2 = 00000000000000010000000000000001  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 = 00000000000000100000000000000010, data\_out2 = 00000000000000010000000000000001  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 = 00000000000000100000000000000010, b = 00000000000000010000000000000001, 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 = 00000000000000100000000000000010, data\_out2 = 00000000000000010000000000000001  85 [ALU] a = 00000000000000100000000000000010, b = 00000000000000010000000000000001, control = 10, res = 00000000000000010000000000000001  85 [MEM] r\_en = 1, r\_addr1 = 3, r\_addr2 = 1, w\_addr = 7, w\_en = 0, data\_in = 00000000000000010000000000000001, data\_out1 = 00000000000000100000000000000010, data\_out2 = 00000000000000010000000000000001  85 [WB] dst\_addr\_i = 7, data\_i = 00000000000000010000000000000001, dst\_addr\_o = 7, data\_o = 00000000000000010000000000000001, 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 = 3, r\_addr2 = 1, w\_addr = 7, w\_en = 1, data\_in = 00000000000000010000000000000001, data\_out1 = 00000000000000100000000000000010, data\_out2 = 00000000000000010000000000000001  90 [MEM] r\_en = 1, r\_addr1 = 3, r\_addr2 = 1, w\_addr = 7, w\_en = 1, data\_in = 00000000000000010000000000000001, data\_out1 = 00000000000000100000000000000010, data\_out2 = 00000000000000010000000000000001  90 [WB] dst\_addr\_i = 7, data\_i = 00000000000000010000000000000001, dst\_addr\_o = 7, data\_o = 00000000000000010000000000000001, 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 = 00000000000000010000000000000001, data\_out1 = 00000000000000100000000000000010, data\_out2 = 00000000000000010000000000000001  95 [WB] dst\_addr\_i = 7, data\_i = 00000000000000010000000000000001, dst\_addr\_o = 7, data\_o = 00000000000000010000000000000001, 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 = 00000000000000010000000000000001, data\_out1 = 00000000000000100000000000000010, data\_out2 = 00000000000000010000000000000001  100 [WB] dst\_addr\_i = 7, data\_i = 00000000000000010000000000000001, dst\_addr\_o = 7, data\_o = 00000000000000010000000000000001, 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 = 00000000000000010000000000000001, data\_out1 = 00000000000000100000000000000010, data\_out2 = 00000000000000010000000000000001  105 [WB] dst\_addr\_i = 7, data\_i = 00000000000000010000000000000001, dst\_addr\_o = 7, data\_o = 00000000000000010000000000000001, w\_en = 0 WARNING: testbench.v:46: $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 = 00000000000000010000000000000001, data\_out1 = 00000000000000100000000000000010, data\_out2 = 00000000000000010000000000000001  110 [WB] dst\_addr\_i = 7, data\_i = 00000000000000010000000000000001, dst\_addr\_o = 7, data\_o = 00000000000000010000000000000001, w\_en = 1 |