گزارش کار آزمایشگاه DSD

آزمایش شماره 10

11 خرداد 1400

عرشیا اخوان
محمدرضا عبدی
علیرضا ایلامی

شماره آزمایش: **10**	موضوع: پیاده‌سازی یک پردازنده ساده	تاریخ آزمایش: 11 خرداد 1400
عرشیا اخوان **97110422**	محمدرضا عبدی **97110285**	علیرضا ایلامی **97101286**

مقدمه:

یک پردازنده با معماری پشته ای دارای یک پشته با 8 ثبات 8 بیتی است.این پردازنده دارای 8 دستور در مجموعه دستورالعمل های خود است. همچنین دارای حافظه به اندازه 256 خانه 8 بیتی است که 8 خانه آخر آن، به صورت Memory Mapped I/O هستند.

شرح آزمایش:

کلا دو تا استیج داریم: IFIDC و EXEC

در استیج IFIDC، دستور را Fetch میکنیم، آن را Decode میکنیم و در نهایت آن را کنترل میکنیم و به ماژول Exec میدهیم تا کار آن دستور را انجام دهد.

در استیج EXEC، به فراخور نوع دستور، از ماژولهای مختلفی استفاده میکنیم که در زیر توضیح داده شده اند:

برای دستورهای ADD, SUB از ماژول ALU استفاده میکنیم.

برای دستورهای JUMP, JUMPZ, JUMPS از ماژول PC استفاده میکنیم.

برای دستورهای PUSHC, PUSHM, POPM از ماژول PP استفاده میکنیم.

ابتدا استیج IFIDC:

```
`define INIT    2'b00
`define IF      2'b01
`define ID      2'b10
`define OP_STL  4'b1000

module IFIDC (clk,
```

```
            en,
            rstn,
            pc,
            IS_ready,
            control_bus,
            data);

    parameter INST_CAP = 20;
    parameter INST_LEN = 12;
    parameter DATA_LEN = 8;

    input clk, rstn, en;
    input [$clog2(INST_CAP):0] pc;
    output reg [3:0]control_bus;
    output reg [DATA_LEN-1:0]data;
    output reg IS_ready;

    reg [INST_LEN-1:0] inst_mem [INST_CAP-1:0];
    reg[INST_LEN-1:0] inst;

    reg [1:0]state;

    always @(posedge clk or negedge rstn) begin
        if (!rstn && !en) begin
            control_bus <= `OP_STL;
            data        <= 8'bx;
            state       <= `INIT;
            IS_ready    <= 0;
        end
        else begin
            case(state)
                `INIT:
                begin
                    if (en) begin
                        state <= `IF;
                    end
                    IS_ready <= 0;
                end
                `IF:
                begin
                    inst  <= inst_mem[pc];
                    state <= `ID;
                end
                `ID:
                begin
                    control_bus[3:0] <= inst[INST_LEN-1:DATA_LEN];
                    data             <= inst[DATA_LEN-1:0];
                    state            <= `INIT;
                    IS_ready         <= 1;
                end
            endcase
        end
```

```
    end

    //debugging
    always @(*)
        $display($time, "\t [IFIDC::%d] pc = %d control_bus = %b addr_imm = %d IS_ready = %b
rstn = %b en = %b", state, pc, control_bus, data,IS_ready,rstn, en);

endmodule
```

ابتدا در case: INIT هستیم. وقتی enable = 1 شد وارد فاز IF: Instruction Fetch میشویم. و دستور را از حافظه میخوانیم و اصطلاحا fetch میکنیم.

مرحله بعدی ID: Instruction Decode است. که در آن بخشهای مختلف دستور را جدا میکنیم.

در پایان مجددا به فاز INIT برمیگردیم و سیگنال is_ready = 1 میکنیم. (برای اینکه ماژول های داخلی EXEC بدانند کار استیج قبلی برای یک دستور تمام شده و می توانند آن را اجرا نمایند.)

استیج EXEC:

```
`define NXTI 2'b00
`define EXIT 2'b11

module EXEC (clk,
             en,
             rstn,
             pc,
             control_bus,
             addr_const,
             stk_data_in,
             stk_push,
             stk_pop,
             stk_data_out,
             mem_data_in,
             mem_addr,
             mem_r_en,
             mem_w_en,
             mem_data_out,
             fin_sig);

    parameter DATA_LEN = 8;
```

```
    parameter ADDR_LEN = 8;
    parameter INST_CAP = 20;

    input clk, en, rstn;
    input [3:0] control_bus;
    input [DATA_LEN-1:0] stk_data_out, mem_data_out, addr_const;
    output wire [DATA_LEN-1:0] stk_data_in, mem_data_in;
    output wire [ADDR_LEN-1:0] mem_addr;
    output wire stk_push, stk_pop, mem_r_en, mem_w_en, fin_sig;
    output wire [$clog2(INST_CAP):0] pc;
    wire alu_z_flag, alu_s_flag, alu_fin_sig, wbpb_fin_sig, pc_fin_sig;

    assign fin_sig = (alu_fin_sig | wbpb_fin_sig | pc_fin_sig);

    ALU  #(
    .DATA_LEN(DATA_LEN)
    ) alu0 (
    .clk(clk),
    .rstn(rstn),
    .en(en),
    .control_bus(control_bus),
    .z_flag(alu_z_flag),
    .s_flag(alu_s_flag),
    .stk_data_in(stk_data_in),
    .stk_push(stk_push),
    .stk_pop(stk_pop),
    .stk_data_out(stk_data_out),
    .fin_sig(alu_fin_sig)
    );

    PP  #(
    .DATA_LEN(DATA_LEN),
    .ADDR_LEN(ADDR_LEN)
    ) pp0 (
    .clk(clk),
    .rstn(rstn),
    .en(en),
    .control_bus(control_bus),
    .addr_const(addr_const),
    .stk_data_in(stk_data_in),
    .stk_push(stk_push),
    .stk_pop(stk_pop),
    .stk_data_out(stk_data_out),
    .mem_data_in(mem_data_in),
    .mem_addr(mem_addr),
    .mem_r_en(mem_r_en),
    .mem_w_en(mem_w_en),
    .mem_data_out(mem_data_out),
    .fin_sig(wbpb_fin_sig)
    );

    PC #(
```

```
    .DATA_LEN(DATA_LEN),
    .INST_CAP(INST_CAP)
    ) pc0 (
    .clk(clk),
    .rstn(rstn),
    .en(en),
    .control_bus(control_bus),
    .pc(pc),
    .z_flag(alu_z_flag),
    .s_flag(alu_s_flag),
    .stk_pop(stk_pop),
    .stk_push(stk_push),
    .stk_data_out(stk_data_out),
    .fin_sig(pc_fin_sig)
    );

endmodule
```

در ماژول EXEC صرفا سیم کشی ها انجام میشود و از سه submodule آن، instantiate میکنیم.

حال به سراغ ماژول های داخلی استیج EXEC میرویم:

ماژول ALU:

```
`define INIT    4'b0000
`define OP1_POP 4'b0001
`define OP1_RCV 4'b0010
`define OP1_STR 4'b0011
`define OP2_POP 4'b0100
`define OP2_RCV 4'b0101
`define OP2_STR 4'b0110
`define ALU     4'b0111
`define PUSH    4'b1000
`define PUSH_W  4'b1001

module ALU (control_bus,
            clk,
            en,
            rstn,
            z_flag,
            s_flag,
            stk_data_in,
```

```
        stk_push,
        stk_pop,
        stk_data_out,
        fin_sig);

parameter DATA_LEN = 8;

input clk, rstn, en;
input [3:0] control_bus;
input [DATA_LEN-1:0] stk_data_out;
output reg [DATA_LEN-1:0] stk_data_in;
output reg stk_push, stk_pop;
output reg z_flag, s_flag, fin_sig;

wire asn    = !control_bus[0];
wire alu_en = (en && control_bus[1] && control_bus[2]);

reg [3:0]state;

reg signed [DATA_LEN-1:0] op1, op2, result;

always @(posedge clk, negedge rstn) begin
    if (!rstn && !alu_en) begin
        state       <= `INIT;
        stk_data_in <= {DATA_LEN{1'bz}};
        stk_push    <= 1'bz;
        stk_pop     <= 1'bz;
        s_flag      <= 0;
        z_flag      <= 0;
        fin_sig     <= 0;
    end
    else begin
        case(state)
            `INIT:
            begin
                if (alu_en) begin
                    state <= `OP1_POP;
                end
                fin_sig     <= 0;
                stk_data_in <= {DATA_LEN{1'bz}};
                stk_push    <= 1'bz;
                stk_pop     <= 1'bz;
            end
            `OP1_POP:
            begin
                stk_push<= 0;
                stk_pop <= 1;
                state   <= `OP1_RCV;
            end
            `OP1_RCV:
            begin
                stk_pop <= 0;
```

```
                state   <= `OP1_STR;
            end
            `OP1_STR:
            begin
                op1     <= stk_data_out;
                state   <= `OP2_POP;
            end
            `OP2_POP:
            begin
                stk_pop <= 1;
                state   <= `OP2_RCV;
            end
            `OP2_RCV:
            begin
                stk_pop <= 0;
                state   <= `OP2_STR;
            end
            `OP2_STR:
            begin
                op2     <= stk_data_out;
                state   <= `ALU;
            end
            `ALU:
            begin
                if (asn)
                    result   <= op2 + op1;
                else
                    result   <= op2 - op1;
                state <= `PUSH;
            end
            `PUSH:
            begin
                s_flag      <= (result < 0);
                z_flag      <= (result == 0);
                stk_push    <= 1;
                stk_data_in <= result;
                state       <= `PUSH_W;
            end
            `PUSH_W:
            begin
                stk_push    <= 0;
                state       <= `INIT;
                fin_sig     <= 1;
            end
        endcase
    end
end

//debugging
always @(*)
    $display($time, "\t [ALU::%d] rstn = %b, control_bus = %b, opt1 = %d, opt2 = %d,
result = %d, stk_data_in = %d, stk_push = %b, stk_pop = %b, stk_data_out = %d z_flag = %b,
```

```
s_flag = %b,", state, rstn, control_bus, op1, op2, result, stk_data_in, stk_push, stk_pop,
stk_data_out, z_flag, s_flag);

endmodule
```

این ماژول برای دستورهای ADD/SUB بکار میرود. که برای این دستورات، نیاز داریم به POP/PUSH روی استک داریم.

یک استیت INIT داریم که در آن سیگنالهای اولیه را ست میکنیم و کار شروع میشود.

برای عملیات جمع و تفریق نیاز است دو تا عدد از استک POP کنیم.

کل عملیات POP کردن 3 حالت مختلف دارد. ابتدا سیگنال درخواست POP کردن فعال میشود. و بعد باید یک کلاک صبر کنیم و بعد عدد را از اول استک بیرون بیاوریم. و سپس در کلاک بعدی این عدد را در operand مان بریزیم. پس کلا 3 کلاک طول میکشد. پس برای POP کردن دو عدد 6 مرحله داریم که هرکدام یک کلاک طول میکشد. پس از اینکه هر دو عدد دریافت شد، در استیت ALU عملیات را انجام میدهیم. حال خروجی نهایی را نیز در استک باید ذخیره کنیم. که این عملیات PUSH کردن هم نیازمند 3 کلاک است. ابتدا سیگنالها فعال میشوند. سپس یک کلاک صبر میکنیم تا این سیگنال اعمال شود و PUSH انجام شود. یک مرحله سوم هم نیاز داریم که کارمان تمام شود و مثلا POP یا PUSH بعدی را شروع کنیم. که این مرحله می تواند همان استیت INIT باشد. بنابراین پس از PUSH_W مستقیم به INIT میرویم.

ماژول PC:

```
`define INIT    3'b000
`define POP     3'b001
`define NXTL    3'b010
`define BR      3'b011
`define EXIT    3'b100
`define POP_W   3'b101

`define OP_JP   2'b01
`define OP_JS   2'b10
`define OP_JZ   2'b11

module PC (control_bus,
```

```
        clk,
        rstn,
        en,
        pc,
        z_flag,
        s_flag,
        stk_pop,
        stk_push,
        stk_data_out,
        fin_sig);

parameter INST_CAP = 20;
parameter DATA_LEN = 8;

input clk, en, rstn, z_flag, s_flag;
input [3:0] control_bus;
input [DATA_LEN-1:0] stk_data_out;
output reg stk_pop,stk_push, fin_sig;
output reg [$clog2(INST_CAP):0] pc;

wire[1:0] opc = {control_bus[2], control_bus[0]};
wire branch   = (control_bus > 2 && control_bus < 6);
wire exit     = (control_bus == 4'b1111);
wire stall    = !(control_bus < 4'b1000 || control_bus == 4'b1111);

reg [2:0]state;

always @(posedge clk, negedge rstn) begin
    if (!rstn && !en) begin
        state    <= `INIT;
        stk_pop  <= 1'bz;
        stk_push <= 1'bz;
        pc       <= 0;
        fin_sig  <= 0;
    end
    else begin
        case(state)
            `INIT:
            begin
                if (en) begin
                    if (branch)
                        state <= `POP;
                    else if (exit)
                        state <= `EXIT;
                    else
                        state <= `NXTL;
                end
                fin_sig  <= 0;
                stk_pop  <= 1'bz;
                stk_push <= 1'bz;
            end
            `POP:
```

```
                begin
                    stk_push <= 0;
                    if ((opc == `OP_JP) || (opc == `OP_JZ && z_flag) ||  (opc == `OP_JS &&
s_flag)) begin
                        stk_pop <= 1;
                        state   <= `POP_W;
                    end
                    else begin
                        state <= `NXTL;
                    end
                end
                `POP_W:
                begin
                    stk_pop <= 1;
                    state   <= `BR;
                end
                `NXTL:
                begin
                    if (pc < INST_CAP - 1) begin
                        pc <= pc + 1;
                    end
                    state <= `INIT;
                    if (stall) begin
                        fin_sig <= 1;
                    end
                end
                `BR:
                begin
                    pc      <= stk_data_out;
                    state   <= `INIT;
                    fin_sig <= 1;
                end
                `EXIT:
                begin
                    $writememb("report/result.mem", cpu0.memory0.mem);
                    $finish;
                end
            endcase
        end

    end

    //debugging
    always @(*)
        $display($time, "\t [PC::%d] rstn = %b, en = %b, control_bus = %b, pc = %d, z_flag =
%b, s_flag = %b, stk_pop = %b, stk_data_out = %d", state, rstn, en, control_bus, pc, z_flag,
s_flag, stk_pop, stk_data_out);

endmodule
```

کلا ماژول PC سه حالت دارد. یا می خواهد برنامه را تمام کند و شروع به نوشتن اطلاعات جدید در حافظه کند، یا می خواهد دستور Jump ای را اجرا کند. و یا صرفا میخواهد Program Counter را یک واحد افزایش دهد و برود خط بعدی را بخواند و الی آخر.

برای حالت اول که صرفا کافیست $writememb را فراخوانی کنیم.

برای حالت دوم، لازم است ابتدا یک بار از استک POP کنیم و بدانیم که به کدام خانه می خواهیم برویم. و سپس عملیات Jump را انجام دهیم و PC را برابر مقدار مربوطه ست کنیم. که این POP کردن و انجام عملیات، مشابه بخش ALU، در 3 مرحله مختلف انجام میشود. POP, POP_W, BR

برای حالتی که صرفا میخواهیم PC را یک واحد افزایش دهیم، کارمان راحت است. در حالت NXTL چک میکنیم که آیا به انتهای ظرفیت دستورات نرسیده باشیم، و سپس pc را یک واحد افزایش میدهیم. همچنین، اگر دستور قبلی stall بوده باشد، سیگنال fin_sig که نشان دهنده پایان عملیات اجرای دستور است را = 1 می کنیم.

ماژول PP:

```
`define INIT    3'b000

`define LOAD    3'b011
`define PUSH    3'b001
`define PUSH_W  3'b010

`define POP     3'b101
`define POP_W   3'b110
`define STORE   3'b111

`define OP_PUSHC    2'b00
`define OP_PUSHM    2'b01
`define OP_POPM     2'b10

module PP (control_bus,
           clk,
           en,
           rstn,
```

```
        addr_const,
        stk_data_in,
        stk_push,
        stk_pop,
        stk_data_out,
        mem_data_in,
        mem_addr,
        mem_r_en,
        mem_w_en,
        mem_data_out,
        fin_sig);

parameter ADDR_LEN = 8;
parameter DATA_LEN = 8;

input clk, rstn, en;
input [3:0] control_bus;
input [DATA_LEN-1:0] stk_data_out, mem_data_out, addr_const;
output reg [DATA_LEN-1:0] stk_data_in, mem_data_in;
output reg [ADDR_LEN-1:0] mem_addr;
output reg stk_push, stk_pop, mem_r_en, mem_w_en, fin_sig;

wire [1:0] opc = control_bus[1:0];
wire pp_en     = (en && control_bus < 3);

reg [2:0]state;

always @(posedge clk, negedge rstn) begin
    if (!rstn && !pp_en) begin
        state       <= `INIT;
        stk_data_in <= {DATA_LEN{1'bz}};
        stk_push    <= 1'bz;
        stk_pop     <= 1'bz;
        mem_data_in <= {DATA_LEN{1'bz}};
        mem_addr    <= {ADDR_LEN{1'bz}};
        mem_r_en    <= 1'bz;
        mem_w_en    <= 1'bz;
        fin_sig     <= 0;
    end
    else begin
        case(state)
            `INIT:
            begin
                if (pp_en) begin
                    state <= {opc,1'b1};
                end
                fin_sig     <= 0;
                stk_data_in <= {DATA_LEN{1'bz}};
                stk_push    <= 1'bz;
                stk_pop     <= 1'bz;
                mem_data_in <= {DATA_LEN{1'bz}};
                mem_addr    <= {ADDR_LEN{1'bz}};
```

```
                mem_r_en    <= 1'bz;
                mem_w_en    <= 1'bz;
            end
            `PUSH:
            begin
                mem_r_en    <= 0;
                mem_w_en    <= 0;
                stk_pop     <= 0;
                stk_data_in <= (opc == `OP_PUSHC) ? addr_const : mem_data_out;
                state       <= `PUSH_W;
            end
            `PUSH_W:
            begin
                stk_push <= 1;
                state    <= `INIT;
                fin_sig  <= 1;
            end
            `STORE:
            begin
                mem_w_en    <= 1;
                mem_r_en    <= 0;
                mem_data_in <= stk_data_out;
                mem_addr    <= addr_const;
                state       <= `INIT;
                fin_sig     <= 1;
            end
            `POP:
            begin
                stk_push <= 0;
                stk_pop  <= 1;
                state    <= `POP_W;
            end
            `POP_W:
            begin
                state <= `STORE;
            end
            `LOAD:
            begin
                mem_r_en <= 1;
                mem_w_en <= 0;
                mem_addr <= addr_const;
                state    <= `PUSH;
            end
            default:
            state <= `INIT;
        endcase
    end

end

//debugging
always @(*)
```

```
        $display($time, "\t [WBPB::%d] rstn = %b, en = %b, fin_sig = %b, control_bus = %b,
addr_const = %d, stk_data_in = %d, stk_push = %b, stk_pop = %b, stk_data_out = %d,
mem_data_in = %d, mem_addr = %d, mem_r_en = %b, mem_w_en = %b, mem_data_out = %d", state,
rstn, en, fin_sig, control_bus, addr_const, stk_data_in, stk_push, stk_pop, stk_data_out,
mem_data_in, mem_addr, mem_r_en, mem_w_en, mem_data_out);

endmodule
```

در این ماژول، وظایف PUSH, POP, LOAD, STORE را داریم.

کلا سه نوع دستور داریم که به PP محول میشود:

PUSHC: push c
PUSHM: load m, push m
POPM: pop m, store m

در ابتدا در حالت INIT هستیم. اگر enable = 1 شود، باید وارد state ای بشویم که وابسته به نوع دستورمان است. اگر از نوع PUSHC باشیم، همان ابتدا مستقیما وارد فاز push میشویم و در 2 کلاک و در حالت های PUSH, PUSH_W عملیات را انجام میدهیم. اگر دستورمان از نوع PUSH_M باشد، ابتدا باید LOAD کنیم. و از حافظه بخوانیم. سپس، مشابه نوع قبلی وارد فاز PUSH و سپس PUSH_W میشویم. اما اگر دستورمان POPM بود، لازم است ابتدا یک بار از استک POP کنیم، و سپس داده مورد نظر را در حافظه ذخیره کنیم. که این کار را در فاز STORE انجام میدهیم.

پس در نهایت، استیت های پایانی ما، وابسته به اینکه نوع دستورمان چه باشد، یکی از دو استیت STORE و PUSH_W است. پس در این دو حالت پس از اینکه کارها انجام شد، باید به استیت اولیه INIT برگردیم.

بقیه ماژول های مورد استفاده:

Stack:

```
module Stack (rstn,
              data_in,
              push,
              pop,
              clk,
              data_out,
```

```
            full,
            empty);

    parameter STACK_DEPTH = 8;
    parameter WORD_LEN    = 8;

    input wire rstn, push, pop, clk;
    input wire[WORD_LEN-1:0] data_in;
    output full, empty;
    output reg [WORD_LEN-1:0] data_out;

    reg[$clog2(STACK_DEPTH):0] stack_ptr;
    reg[WORD_LEN-1:0] memory [0:STACK_DEPTH-1];

    assign empty = (stack_ptr == 0) ? 1'b1 : 1'b0;
    assign full  = (stack_ptr == STACK_DEPTH) ? 1'b1 : 1'b0;

    wire [WORD_LEN-1:0]top = memory[stack_ptr-1];

    // stack reset
    task reset_memory;
        integer i;
        begin
            for (i = 0; i < STACK_DEPTH;i++) begin
                memory[i] <= {WORD_LEN{1'b0}};
            end
            stack_ptr <= 0;
        end
    endtask

    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            reset_memory;
        end
        else begin
            // pushing into stack
            if (push && !pop && !full) begin
                memory[stack_ptr] <= data_in;
                stack_ptr         <= stack_ptr + 1;
            end
            // pop from stack
            if (pop && !push && !empty) begin
                data_out  <= memory[stack_ptr - 1];
                stack_ptr <= stack_ptr - 1;
            end
        end

    end

    //debugging
    always @(*)
        $display($time, "\t [STACK] rstn = %b, stack_ptr = %d, top = %d, data_in = %d, push
```

```
= %b, pop = %b, data_out = %d, full = %b, empty = %b", rstn, stack_ptr, top, data_in, push,
pop, data_out, full, empty);

endmodule
```

CPU

```
`define IFIDC   2'b00
`define IFIDC_W 2'b01
`define EXEC    2'b10
`define EXEC_W  2'b11

module CPU (rstn,
            clk);

    parameter DATA_LEN    = 8;
    parameter ADDR_LEN    = 8;
    parameter INST_CAP    = 20;
    parameter INST_LEN    = 12;
    parameter WORD_LEN    = 8;
    parameter MEM_SIZE    = 256;
    parameter STACK_DEPTH = 8;

    input rstn, clk;

    wire stk_full, stk_empty, stk_push, stk_pop, mem_r_en, mem_w_en, exec_fin_sig,IS_ready;
    wire [3:0] ifidc_control_bus;
    wire [DATA_LEN-1:0] stk_data_in, stk_data_out, mem_data_in, mem_data_out,
ifidc_addr_const;
    wire [ADDR_LEN-1:0] mem_addr;
    wire [$clog2(INST_CAP):0] exec_pc;

    reg ifidc_fetch, exec_en;
    reg [1:0] state;

    always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
            ifidc_fetch <= 0;
            exec_en     <= 0;
            state       <= `IFIDC;
        end
        else begin
            case(state)
                `IFIDC:
                begin
                    ifidc_fetch <= 1;
                    state       <= `IFIDC_W;
                end
```

```
                `IFIDC_W:
                begin
                    ifidc_fetch <= 0;
                    if (IS_ready) begin
                        state <= `EXEC;
                    end
                end
                `EXEC:
                begin
                    exec_en <= 1;
                    state   <= `EXEC_W;
                end
                `EXEC_W:
                begin
                    exec_en <= 0;
                    if (exec_fin_sig) begin
                        state <= `IFIDC;
                    end
                end
            endcase
        end
    end

    IFIDC #(
    .INST_CAP(INST_CAP),
    .INST_LEN(INST_LEN),
    .DATA_LEN(DATA_LEN)
    ) ifidc0 (
    .clk(clk),
    .en(ifidc_fetch),
    .rstn(rstn),
    .pc(exec_pc),
    .IS_ready(IS_ready),
    .control_bus(ifidc_control_bus),
    .data(ifidc_addr_const)
    );

    EXEC #(
    .DATA_LEN(DATA_LEN),
    .ADDR_LEN(ADDR_LEN),
    .INST_CAP(INST_CAP)
    ) exec0 (
    .clk(clk),
    .en(exec_en),
    .rstn(rstn),
    .pc(exec_pc),
    .control_bus(ifidc_control_bus),
    .addr_const(ifidc_addr_const),
    .stk_data_in(stk_data_in),
    .stk_push(stk_push),
    .stk_pop(stk_pop),
    .stk_data_out(stk_data_out),
```

```
    .mem_data_in(mem_data_in),
    .mem_addr(mem_addr),
    .mem_r_en(mem_r_en),
    .mem_w_en(mem_w_en),
    .mem_data_out(mem_data_out),
    .fin_sig(exec_fin_sig)
    );

    MEMORY #(
    .ADDR_LEN(ADDR_LEN),
    .WORD_LEN(WORD_LEN),
    .MEM_SIZE(MEM_SIZE)
    ) memory0 (
    .clk(clk),
    .addr(mem_addr),
    .r_en(mem_r_en),
    .w_en(mem_w_en),
    .data_out(mem_data_out),
    .data_in(mem_data_in)
    );

    Stack #(
    .WORD_LEN(WORD_LEN),
    .STACK_DEPTH(STACK_DEPTH)
    ) stack0 (
    .clk(clk),
    .rstn(rstn),
    .data_in(stk_data_in),
    .push(stk_push),
    .pop(stk_pop),
    .data_out(stk_data_out),
    .full(stk_full),
    .empty(stk_empty)
    );

    //debugging
    always @(*)
        $display($time, "\t [CPU::%d] ifidc_fetch = %b, exec_en = %b, exec_finish = %b",
state, ifidc_fetch, exec_en, exec_fin_sig);


endmodule
```

Testbench

```
module testbench();

    parameter CLK_C       = 10;
    parameter DATA_LEN    = 8;
```

```
    parameter ADDR_LEN    = 8;
    parameter INST_CAP    = 20;
    parameter INST_LEN    = 12;
    parameter WORD_LEN    = 8;
    parameter MEM_SIZE    = 256;
    parameter STACK_DEPTH = 8;

    reg clk, rstn;

    CPU #(
    .WORD_LEN(WORD_LEN),
    .DATA_LEN(DATA_LEN),
    .ADDR_LEN(ADDR_LEN),
    .MEM_SIZE(MEM_SIZE),
    .STACK_DEPTH(STACK_DEPTH),
    .INST_LEN(INST_LEN),
    .INST_CAP(INST_CAP)
    ) cpu0 (
    .clk(clk),
    .rstn(rstn)
    );

    initial begin
        $dumpfile("report/waveform.vcd");
        $dumpvars(0,cpu0);
    end

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

    initial begin
        $readmemb("report/memory.mem", cpu0.memory0.mem, 0, MEM_SIZE-1);
        $readmemb("report/is.mem", cpu0.ifidc0.inst_mem, 0, INST_CAP-1);
    end

    initial begin
        rstn = 0;
        #CLK_C
        rstn = 1;
        #(100000 * CLK_C);
        $finish;
    end

endmodule
```

ls_math.mem

```
// Example: Y=((X+23)+(X+23))-12
0001_0000_0001              // PUSHM X
0000_0001_0111              // PUSHC 23
0110_xxxx_xxxx              // ADD
0010_0000_0010              // POPM Y
0001_0000_0010              // PUSHM Y
0001_0000_0010              // PUSHM Y
0110_xxxx_xxxx              // ADD
0000_0000_1100              // PUSHC 12
0111_xxxx_xxxx              // SUB
0010_0000_0010              // POPM RES
1111_xxxx_xxxx              // EXIT
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
```

ls.mem

```
// Example: Y=((X+23)+(X+23))-12
0001_0000_0001              // PUSHM X
0000_0001_0111              // PUSHC 23
0110_xxxx_xxxx              // ADD
0010_0000_0010              // POPM Y
0001_0000_0010              // PUSHM Y
0001_0000_0010              // PUSHM Y
0110_xxxx_xxxx              // ADD
0000_0000_1100              // PUSHC 12
0111_xxxx_xxxx              // SUB
0010_0000_0010              // POPM RES
1111_xxxx_xxxx              // EXIT
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
```

Memory.mem

```
// main memory: 256 * word, word = 8 bits
0000_0000
0000_0101
0000_1010
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
0000_0000
```

Result.mem

```
// 0x00000000
00000000
00000101
00101100
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

Is_jump.mem

```
// Branch instruction tests
```

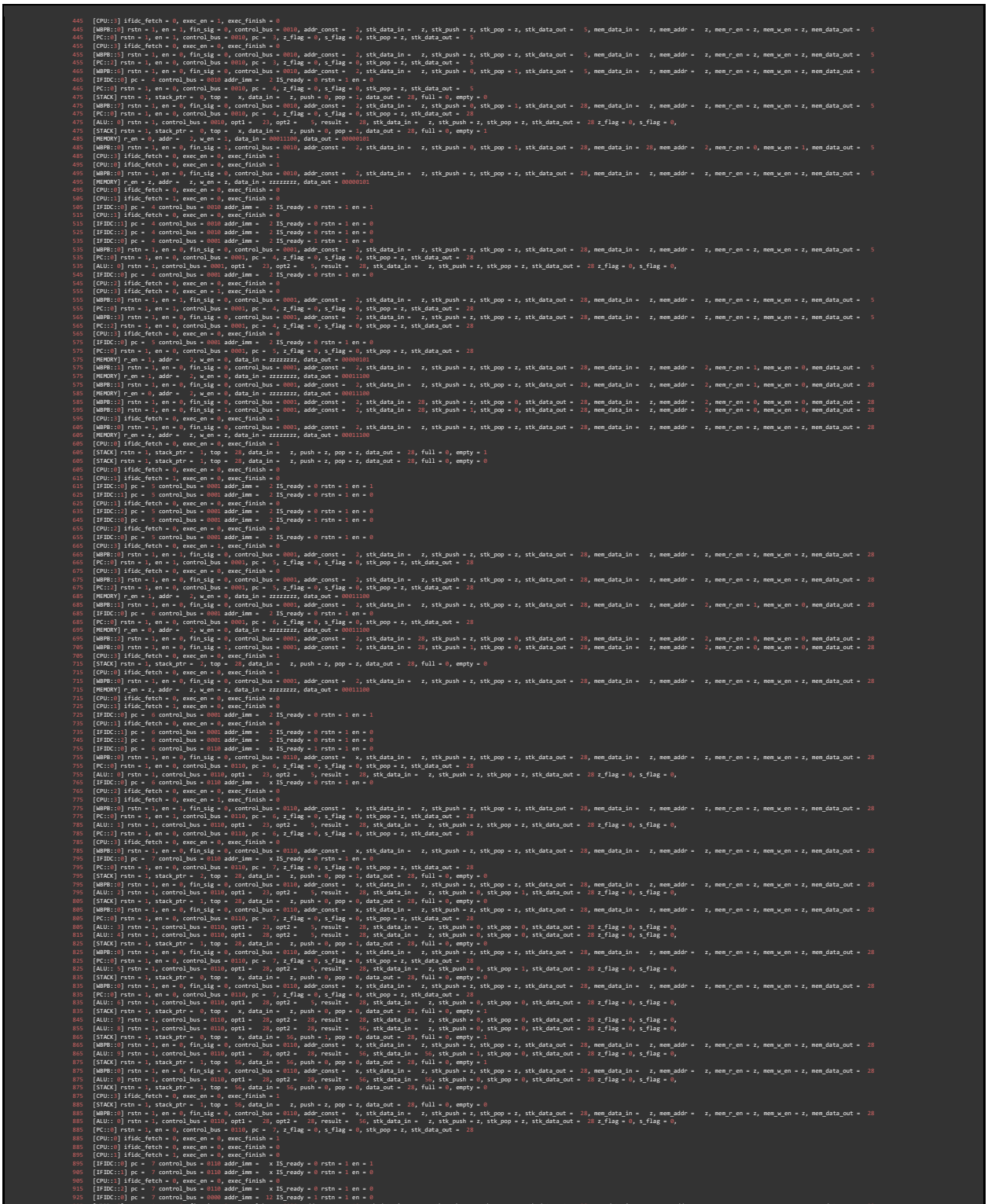
```
0000_0000_0011              // 0.PUSHC 3
0000_0000_0011              // 1.PUSHC 3
0111_xxxx_xxxx              // 2.SUB
0000_0000_0110              // 3.PUSHC 6
0100_xxxx_xxxx              // 4.JZ
1111_xxxx_xxxx              // 5.EXIT
0000_0000_1000              // 6.PUSHC 8
0000_0001_0000              // 7.PUSHC 16
0111_xxxx_xxxx              // 8.SUB
0000_0000_1100              // 9.PUSHC 12
0101_xxxx_xxxx              // 10.JS
1111_xxxx_xxxx              // 11.EXIT
0000_0000_0101              // 12.PUSHC 5
0011_xxxx_xxxx              // 13.JUMP
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
```

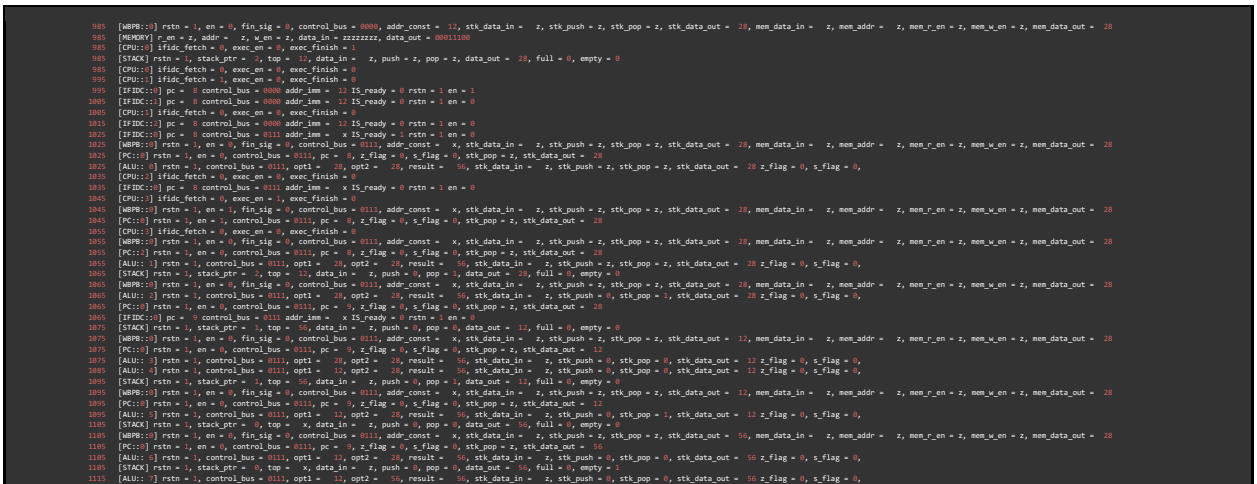
ls_swap.mem

```
// SWAP
0001_0000_0001              // PUSHM ARG1
0001_0000_0010              // PUSHM ARG2
0010_0000_0001              // POPM ARG2
0010_0000_0010              // POPM ARG1
1111_xxxx_xxxx              // EXIT
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
0000_0000_0000
```

Results.txt

```
VCD info: dumpfile report/waveform.vcd opened for output.
                0  [STACK] rstn = 0, stack_ptr =  x, top =   x, data_in =   x, push = x, pop = x, data_out =   x, full = x, empty = x
                0  [IFIDC::x] pc =  x control_bus = xxxx addr_imm =   x IS_ready = x rstn = 1 en = x
                0  [WBPB::x] rstn = 0, en = x, fin_sig = x, control_bus = xxxx, addr_const =   x, stk_data_in =   x, stk_push = x, stk_pop = x, stk_data_out =   x, mem_data_in =   x, mem_addr =   x, mem_r_en = x, mem_w_en = x, mem_data_out =   x
                0  [PC::x] rstn = 0, en = x, control_bus = xxxx, pc =  x, z_flag = x, s_flag = x, stk_pop = x, stk_data_out =   x
                0  [ALU:: x] rstn = 0, control_bus = xxxx, opt1 =   x, opt2 =   x, result =   x, stk_data_in =   x, stk_push = x, stk_pop = x, stk_data_out =   x z_flag = x, s_flag = x,
                0  [CPU::0] ifidc_fetch = 0, exec_en = 0, exec_finish = x
                0  [IFIDC::x] pc =  x control_bus = xxxx addr_imm =   x IS_ready = x rstn = 0 en = 0
                0  [WBPB::0] rstn = 0, en = 0, fin_sig = x, control_bus = xxxx, addr_const =   x, stk_data_in =   x, stk_push = x, stk_pop = x, stk_data_out =   x, mem_data_in =   x, mem_addr =   x, mem_r_en = x, mem_w_en = x, mem_data_out =   x
                0  [PC::x] rstn = 0, en = 0, control_bus = xxxx, pc =  x, z_flag = x, s_flag = x, stk_pop = x, stk_data_out =   x
                0  [STACK] rstn = 0, stack_ptr =  0, top =   x, data_in =   x, push = x, pop = x, data_out =   x, full = 0, empty = 1
                5  [IFIDC::0] pc =  0 control_bus = 1000 addr_imm =   x IS_ready = 0 rstn = 0 en = 0
                5  [WBPB::0] rstn = 0, en = 0, fin_sig = 0, control_bus = 1000, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   x
                5  [PC::0] rstn = 0, en = 0, control_bus = 1000, pc =  0, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
                5  [ALU:: 0] rstn = 0, control_bus = 1000, opt1 =   x, opt2 =   x, result =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x z_flag = 0, s_flag = 0,
                5  [STACK] rstn = 0, stack_ptr =  0, top =   x, data_in =   z, push = z, pop = z, data_out =   x, full = 0, empty = 1
                5  [MEMORY] r_en = z, addr =   z, w_en = z, data_in = zzzzzzzz, data_out = xxxxxxxx
                5  [CPU::0] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
               10  [STACK] rstn = 1, stack_ptr =  0, top =   x, data_in =   z, push = z, pop = z, data_out =   x, full = 0, empty = 1
               10  [IFIDC::0] pc =  0 control_bus = 1000 addr_imm =   x IS_ready = 0 rstn = 1 en = 0
               10  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 1000, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   x
               10  [PC::0] rstn = 1, en = 0, control_bus = 1000, pc =  0, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
               10  [ALU:: 0] rstn = 1, control_bus = 1000, opt1 =   x, opt2 =   x, result =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x z_flag = 0, s_flag = 0,
               15  [CPU::1] ifidc_fetch = 1, exec_en = 0, exec_finish = 0
               15  [IFIDC::0] pc =  0 control_bus = 1000 addr_imm =   x IS_ready = 0 rstn = 1 en = 1
               25  [IFIDC::1] pc =  0 control_bus = 1000 addr_imm =   x IS_ready = 0 rstn = 1 en = 0
               25  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
               35  [IFIDC::2] pc =  0 control_bus = 1000 addr_imm =   x IS_ready = 0 rstn = 1 en = 0
               45  [IFIDC::3] pc =  0 control_bus = 0001 addr_imm =   1 IS_ready = 1 rstn = 1 en = 0
               45  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0001, addr_const =   1, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   x
               45  [PC::0] rstn = 1, en = 0, control_bus = 0001, pc =  0, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
               45  [ALU:: 0] rstn = 1, control_bus = 0001, opt1 =   x, opt2 =   x, result =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x z_flag = 0, s_flag = 0,
               55  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
               55  [IFIDC::0] pc =  0 control_bus = 0001 addr_imm =   1 IS_ready = 0 rstn = 1 en = 0
               65  [CPU::1] ifidc_fetch = 0, exec_en = 1, exec_finish = 0
               65  [WBPB::0] rstn = 1, en = 1, fin_sig = 0, control_bus = 0001, addr_const =   1, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   x
               65  [PC::0] rstn = 1, en = 1, control_bus = 0001, pc =  0, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
               75  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
               75  [WBPB::1] rstn = 1, en = 0, fin_sig = 0, control_bus = 0001, addr_const =   1, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   x
               75  [PC::1] rstn = 1, en = 0, control_bus = 0001, pc =  0, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
               85  [MEMORY] r_en = 1, addr =   1, w_en = 0, data_in = zzzzzzzz, data_out = xxxxxxxx
               85  [WBPB::1] rstn = 1, en = 0, fin_sig = 0, control_bus = 0001, addr_const =   1, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   1, mem_r_en = 1, mem_w_en = 0, mem_data_out =   x
               85  [IFIDC::0] pc =  1 control_bus = 0001 addr_imm =   1 IS_ready = 0 rstn = 1 en = 0
               85  [PC::0] rstn = 1, en = 0, control_bus = 0001, pc =  1, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
               85  [MEMORY] r_en = 1, addr =   1, w_en = 0, data_in = zzzzzzzz, data_out = 00000101
               85  [WBPB::1] rstn = 1, en = 0, fin_sig = 0, control_bus = 0001, addr_const =   1, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   1, mem_r_en = 1, mem_w_en = 0, mem_data_out =   5
               95  [MEMORY] r_en = 0, addr =   1, w_en = 0, data_in = zzzzzzzz, data_out = 00000101
               95  [WBPB::2] rstn = 1, en = 0, fin_sig = 0, control_bus = 0001, addr_const =   1, stk_data_in =   5, stk_push = z, stk_pop = 0, stk_data_out =   x, mem_data_in =   z, mem_addr =   1, mem_r_en = 0, mem_w_en = 0, mem_data_out =   5
              105  [WBPB::0] rstn = 1, en = 0, fin_sig = 1, control_bus = 0001, addr_const =   1, stk_data_in =   5, stk_push = 1, stk_pop = 0, stk_data_out =   x, mem_data_in =   z, mem_addr =   1, mem_r_en = 0, mem_w_en = 0, mem_data_out =   5
              105  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 1
              115  [STACK] rstn = 1, stack_ptr =  1, top =   5, data_in =   z, push = z, pop = z, data_out =   x, full = 0, empty = 1
              115  [CPU::0] ifidc_fetch = 0, exec_en = 0, exec_finish = 1
              115  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0001, addr_const =   1, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              115  [MEMORY] r_en = z, addr =   z, w_en = z, data_in = zzzzzzzz, data_out = 00000101
              115  [STACK] rstn = 1, stack_ptr =  1, top =   5, data_in =   z, push = z, pop = z, data_out =   x, full = 0, empty = 0
              115  [CPU::0] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
              125  [CPU::1] ifidc_fetch = 1, exec_en = 0, exec_finish = 0
              125  [IFIDC::0] pc =  1 control_bus = 0001 addr_imm =   1 IS_ready = 0 rstn = 1 en = 1
              135  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
              135  [IFIDC::1] pc =  1 control_bus = 0001 addr_imm =   1 IS_ready = 0 rstn = 1 en = 0
              145  [IFIDC::2] pc =  1 control_bus = 0001 addr_imm =   1 IS_ready = 0 rstn = 1 en = 0
              155  [IFIDC::0] pc =  1 control_bus = 0000 addr_imm =  23 IS_ready = 1 rstn = 1 en = 0
              155  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0000, addr_const =  23, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              155  [PC::0] rstn = 1, en = 0, control_bus = 0000, pc =  1, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
              155  [ALU:: 0] rstn = 1, control_bus = 0000, opt1 =   x, opt2 =   x, result =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x z_flag = 0, s_flag = 0,
              165  [IFIDC::0] pc =  1 control_bus = 0000 addr_imm =  23 IS_ready = 0 rstn = 1 en = 0
              165  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
              165  [CPU::1] ifidc_fetch = 0, exec_en = 1, exec_finish = 0
              175  [WBPB::0] rstn = 1, en = 1, fin_sig = 0, control_bus = 0000, addr_const =  23, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              175  [PC::0] rstn = 1, en = 1, control_bus = 0000, pc =  1, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
              185  [WBPB::1] rstn = 1, en = 0, fin_sig = 0, control_bus = 0000, addr_const =  23, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              185  [PC::1] rstn = 1, en = 0, control_bus = 0000, pc =  1, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
              185  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
              195  [IFIDC::0] pc =  2 control_bus = 0000 addr_imm =  23 IS_ready = 0 rstn = 1 en = 0
              195  [PC::0] rstn = 1, en = 0, control_bus = 0000, pc =  2, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
              195  [MEMORY] r_en = 0, addr =   2, w_en = 0, data_in = zzzzzzzz, data_out = 00000101
              195  [WBPB::2] rstn = 1, en = 0, fin_sig = 0, control_bus = 0000, addr_const =  23, stk_data_in =  23, stk_push = z, stk_pop = 0, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = 0, mem_w_en = 0, mem_data_out =   5
              205  [WBPB::0] rstn = 1, en = 0, fin_sig = 1, control_bus = 0000, addr_const =  23, stk_data_in =  23, stk_push = 1, stk_pop = 0, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = 0, mem_w_en = 0, mem_data_out =   5
              205  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 1
              215  [STACK] rstn = 1, stack_ptr =  2, top =  23, data_in =   z, push = z, pop = z, data_out =   x, full = 0, empty = 0
              215  [CPU::0] ifidc_fetch = 0, exec_en = 0, exec_finish = 1
              215  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0000, addr_const =  23, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              215  [MEMORY] r_en = z, addr =   z, w_en = z, data_in = zzzzzzzz, data_out = 00000101
              215  [CPU::0] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
              225  [CPU::1] ifidc_fetch = 1, exec_en = 0, exec_finish = 0
              225  [IFIDC::0] pc =  2 control_bus = 0000 addr_imm =  23 IS_ready = 0 rstn = 1 en = 1
              235  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
              235  [IFIDC::1] pc =  2 control_bus = 0000 addr_imm =  23 IS_ready = 0 rstn = 1 en = 0
              245  [IFIDC::2] pc =  2 control_bus = 0000 addr_imm =  23 IS_ready = 0 rstn = 1 en = 0
              255  [IFIDC::0] pc =  2 control_bus = 0110 addr_imm =   x IS_ready = 1 rstn = 1 en = 0
              255  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0110, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              255  [PC::0] rstn = 1, en = 0, control_bus = 0110, pc =  2, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
              255  [ALU:: 0] rstn = 1, control_bus = 0110, opt1 =   x, opt2 =   x, result =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x z_flag = 0, s_flag = 0,
              265  [IFIDC::0] pc =  2 control_bus = 0110 addr_imm =   x IS_ready = 0 rstn = 1 en = 0
              265  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
              265  [CPU::1] ifidc_fetch = 0, exec_en = 1, exec_finish = 0
              275  [WBPB::0] rstn = 1, en = 1, fin_sig = 0, control_bus = 0110, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              275  [PC::0] rstn = 1, en = 1, control_bus = 0110, pc =  2, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
              285  [ALU:: 1] rstn = 1, control_bus = 0110, opt1 =   x, opt2 =   x, result =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x z_flag = 0, s_flag = 0,
              285  [PC::2] rstn = 1, en = 0, control_bus = 0110, pc =  2, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
              285  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
              285  [WBPB::0] rstn = 1, en = 1, fin_sig = 0, control_bus = 0110, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              295  [IFIDC::0] pc =  3 control_bus = 0110 addr_imm =   x IS_ready = 0 rstn = 1 en = 0
              295  [PC::0] rstn = 1, en = 0, control_bus = 0110, pc =  3, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   x
              295  [STACK] rstn = 1, stack_ptr =  2, top =  23, data_in =   z, push = 0, pop = 1, data_out =   x, full = 0, empty = 0
              295  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0110, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   x, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              305  [ALU:: 2] rstn = 1, control_bus = 0110, opt1 =   x, opt2 =   x, result =   x, stk_data_in =   z, stk_push = 0, stk_pop = 1, stk_data_out =   x z_flag = 0, s_flag = 0,
              305  [STACK] rstn = 1, stack_ptr =  1, top =   5, data_in =   z, push = 0, pop = 0, data_out =  23, full = 0, empty = 0
              305  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0110, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  23, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              305  [PC::0] rstn = 1, en = 0, control_bus = 0110, pc =  3, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  23
              305  [ALU:: 3] rstn = 1, control_bus = 0110, opt1 =   x, opt2 =   x, result =   x, stk_data_in =   z, stk_push = 0, stk_pop = 0, stk_data_out =  23 z_flag = 0, s_flag = 0,
              315  [ALU:: 4] rstn = 1, control_bus = 0110, opt1 =  23, opt2 =   x, result =   x, stk_data_in =   z, stk_push = 0, stk_pop = 0, stk_data_out =  23 z_flag = 0, s_flag = 0,
              325  [STACK] rstn = 1, stack_ptr =  1, top =   5, data_in =   z, push = 0, pop = 1, data_out =  23, full = 0, empty = 0
              325  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0110, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  23, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              325  [PC::0] rstn = 1, en = 0, control_bus = 0110, pc =  3, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  23
              325  [ALU:: 5] rstn = 1, control_bus = 0110, opt1 =   5, opt2 =  23, result =   x, stk_data_in =   z, stk_push = 0, stk_pop = 1, stk_data_out =  23 z_flag = 0, s_flag = 0,
              335  [STACK] rstn = 1, stack_ptr =  0, top =   x, data_in =   z, push = 0, pop = 0, data_out =   5, full = 0, empty = 0
              335  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0110, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   5, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              335  [PC::0] rstn = 1, en = 0, control_bus = 0110, pc =  3, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   5
              335  [ALU:: 6] rstn = 1, control_bus = 0110, opt1 =  23, opt2 =   x, result =   x, stk_data_in =   z, stk_push = 0, stk_pop = 0, stk_data_out =   5 z_flag = 0, s_flag = 0,
              335  [STACK] rstn = 1, stack_ptr =  0, top =   x, data_in =   z, push = 0, pop = 0, data_out =   5, full = 0, empty = 1
              345  [ALU:: 7] rstn = 1, control_bus = 0110, opt1 =  23, opt2 =   5, result =   x, stk_data_in =   z, stk_push = 0, stk_pop = 0, stk_data_out =   5 z_flag = 0, s_flag = 0,
              355  [ALU:: 8] rstn = 1, control_bus = 0110, opt1 =  23, opt2 =   x, result =  28, stk_data_in =   z, stk_push = 0, stk_pop = 0, stk_data_out =   5 z_flag = 0, s_flag = 0,
              355  [STACK] rstn = 1, stack_ptr =  0, top =   x, data_in =  28, push = 1, pop = 0, data_out =   5, full = 0, empty = 1
              365  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0110, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   5, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              365  [ALU:: 9] rstn = 1, control_bus = 0110, opt1 =  23, opt2 =   5, result =  28, stk_data_in =  28, stk_push = 1, stk_pop = 0, stk_data_out =   5 z_flag = 0, s_flag = 0,
              375  [STACK] rstn = 1, stack_ptr =  1, top =  28, data_in =  28, push = 0, pop = 0, data_out =   5, full = 0, empty = 1
              375  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0110, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   5, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              375  [ALU:: 0] rstn = 1, control_bus = 0110, opt1 =  23, opt2 =   5, result =  28, stk_data_in =  28, stk_push = 0, stk_pop = 0, stk_data_out =   5 z_flag = 0, s_flag = 0,
              375  [STACK] rstn = 1, stack_ptr =  1, top =  28, data_in =  28, push = 0, pop = 0, data_out =   5, full = 0, empty = 0
              375  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 1
              385  [STACK] rstn = 1, stack_ptr =  1, top =  28, data_in =   z, push = z, pop = z, data_out =   5, full = 0, empty = 0
              385  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0110, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   5, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              385  [ALU:: 0] rstn = 1, control_bus = 0110, opt1 =  23, opt2 =   5, result =  28, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   5 z_flag = 0, s_flag = 0,
              385  [PC::0] rstn = 1, en = 0, control_bus = 0110, pc =  3, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   5
              385  [CPU::0] ifidc_fetch = 0, exec_en = 0, exec_finish = 1
              385  [CPU::1] ifidc_fetch = 1, exec_en = 0, exec_finish = 0
              395  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
              395  [IFIDC::0] pc =  3 control_bus = 0110 addr_imm =   x IS_ready = 0 rstn = 1 en = 1
              405  [IFIDC::1] pc =  3 control_bus = 0110 addr_imm =   x IS_ready = 0 rstn = 1 en = 0
              405  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
              415  [IFIDC::2] pc =  3 control_bus = 0110 addr_imm =   x IS_ready = 0 rstn = 1 en = 0
              425  [IFIDC::0] pc =  3 control_bus = 0010 addr_imm =   2 IS_ready = 1 rstn = 1 en = 0
              425  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0010, addr_const =   2, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   5, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =   5
              425  [PC::0] rstn = 1, en = 0, control_bus = 0010, pc =  3, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =   5
              435  [ALU:: 0] rstn = 1, control_bus = 0010, opt1 =  23, opt2 =   5, result =  28, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =   5 z_flag = 0, s_flag = 0,
              435  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
              435  [IFIDC::0] pc =  3 control_bus = 0010 addr_imm =   2 IS_ready = 0 rstn = 1 en = 0
```

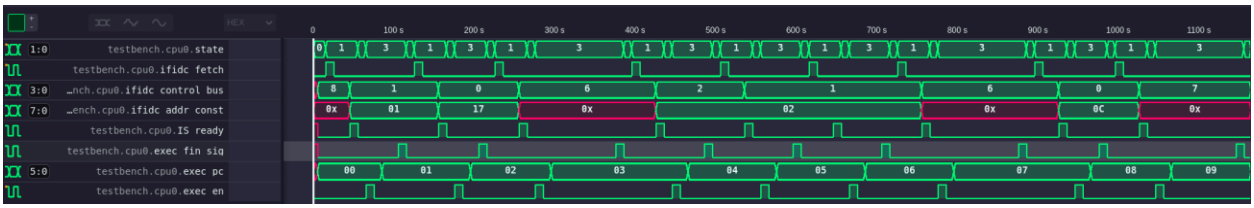




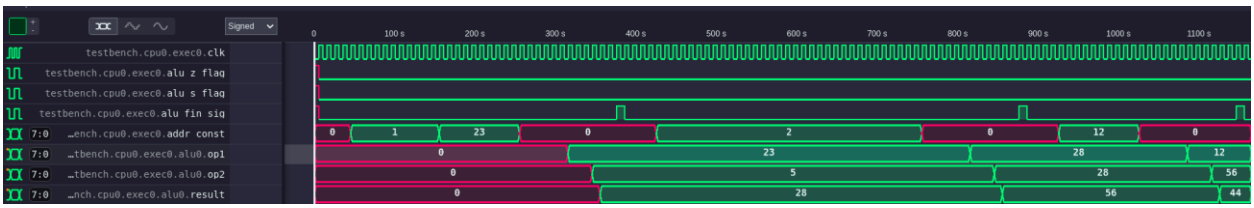
```
  985  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0000, addr_const =  12, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  28, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
  985  [MEMORY] r_en = z, addr =   z, w_en = z, data_in = zzzzzzzz, data_out = 00011100
  985  [CPU::0] ifidc_fetch = 0, exec_en = 0, exec_finish = 1
  985  [STACK] rstn = 1, stack_ptr =  2, top =  12, data_in =   z, push = z, pop = z, data_out =  28, full = 0, empty = 0
  985  [CPU::0] ifidc_fetch = 0, exec_en = 0, exiec_finish = 0
  995  [CPU::1] ifidc_fetch = 1, exec_en = 0, exec_finish = 0
  995  [IFIDC::0] pc =  8 control_bus = 0000 addr_imm =  12 IS_ready = 0 rstn = 1 en = 1
 1005  [IFIDC::1] pc =  8 control_bus = 0000 addr_imm =  12 IS_ready = 0 rstn = 1 en = 0
 1005  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
 1015  [IFIDC::1] pc =  8 control_bus = 0000 addr_imm =  12 IS_ready = 0 rstn = 1 en = 0
 1025  [IFIDC::0] pc =  8 control_bus = 0111 addr_imm =   x IS_ready = 1 rstn = 1 en = 0
 1025  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  28, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1025  [PC::0] rstn = 1, en = 0, control_bus = 0111, pc =  8, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  28
 1025  [ALU:: 0] rstn = 1, control_bus = 0111, opt1 =  28, opt2 =  28, result =  56, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  28 z_flag = 0, s_flag = 0,
 1035  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
 1035  [IFIDC::0] pc =  8 control_bus = 0111 addr_imm =   x IS_ready = 0 rstn = 1 en = 0
 1045  [CPU::1] ifidc_fetch = 0, exec_en = 1, exec_finish = 0
 1045  [WBPB::0] rstn = 1, en = 1, fin_sig = 0, control_bus = 0111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  28, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1045  [PC::0] rstn = 1, en = 1, control_bus = 0111, pc =  8, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  28
 1055  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
 1055  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  28, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1055  [PC::1] rstn = 1, en = 0, control_bus = 0111, pc =  8, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  28
 1055  [ALU:: 1] rstn = 1, control_bus = 0111, opt1 =  28, opt2 =  28, result =  56, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  28 z_flag = 0, s_flag = 0,
 1065  [STACK] rstn = 1, stack_ptr =  2, top =  12, data_in =   z, push = 0, pop = 1, data_out =  28, full = 0, empty = 0
 1065  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  28, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1065  [ALU:: 2] rstn = 1, control_bus = 0111, opt1 =  28, opt2 =  28, result =  56, stk_data_in =   z, stk_push = 0, stk_pop = 1, stk_data_out =  28 z_flag = 0, s_flag = 0,
 1065  [PC::0] rstn = 1, en = 0, control_bus = 0111, pc =  9, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  28
 1065  [IFIDC::0] pc =  9 control_bus = 0111 addr_imm =   x IS_ready = 0 rstn = 1 en = 0
 1075  [STACK] rstn = 1, stack_ptr =  1, top =  56, data_in =   z, push = 0, pop = 1, data_out =  12, full = 0, empty = 0
 1075  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  12, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1075  [PC::0] rstn = 1, en = 0, control_bus = 0111, pc =  9, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  12
 1075  [ALU:: 3] rstn = 1, control_bus = 0111, opt1 =  28, opt2 =  28, result =  56, stk_data_in =   z, stk_push = 0, stk_pop = 0, stk_data_out =  12 z_flag = 0, s_flag = 0,
 1085  [ALU:: 4] rstn = 1, control_bus = 0111, opt1 =  12, opt2 =  28, result =  56, stk_data_in =   z, stk_push = 0, stk_pop = 0, stk_data_out =  12 z_flag = 0, s_flag = 0,
 1095  [STACK] rstn = 1, stack_ptr =  1, top =  56, data_in =   z, push = 0, pop = 1, data_out =  12, full = 0, empty = 0
 1095  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  12, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1095  [PC::0] rstn = 1, en = 0, control_bus = 0111, pc =  9, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  12
 1095  [ALU:: 5] rstn = 1, control_bus = 0111, opt1 =  12, opt2 =  28, result =  56, stk_data_in =   z, stk_push = 0, stk_pop = 1, stk_data_out =  12 z_flag = 0, s_flag = 0,
 1105  [STACK] rstn = 1, stack_ptr =  0, top =  x, data_in =   z, push = 0, pop = 0, data_out =  56, full = 0, empty = 0
 1105  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  56, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1105  [PC::0] rstn = 1, en = 0, control_bus = 0111, pc =  9, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  56
 1105  [ALU:: 6] rstn = 1, control_bus = 0111, opt1 =  12, opt2 =  28, result =  56, stk_data_in =   z, stk_push = 0, stk_pop = 0, stk_data_out =  56 z_flag = 0, s_flag = 0,
 1115  [STACK] rstn = 1, stack_ptr =  0, top =  x, data_in =   z, push = 0, pop = 0, data_out =  56, full = 0, empty = 1
 1115  [ALU:: 7] rstn = 1, control_bus = 0111, opt1 =  12, opt2 =  56, result =  56, stk_data_in =   z, stk_push = 0, stk_pop = 0, stk_data_out =  56 z_flag = 0, s_flag = 0,
 1125  [ALU:: 8] rstn = 1, control_bus = 0111, opt1 =  12, opt2 =  56, result =  44, stk_data_in =   z, stk_push = 0, stk_pop = 0, stk_data_out =  56 z_flag = 0, s_flag = 0,
 1135  [STACK] rstn = 1, stack_ptr =  0, top =  x, data_in =  44, push = 1, pop = 0, data_out =  56, full = 0, empty = 1
 1135  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  56, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1135  [ALU:: 9] rstn = 1, control_bus = 0111, opt1 =  12, opt2 =  56, result =  44, stk_data_in =  44, stk_push = 1, stk_pop = 0, stk_data_out =  56 z_flag = 0, s_flag = 0,
 1145  [STACK] rstn = 1, stack_ptr =  1, top =  44, data_in =  44, push = 0, pop = 0, data_out =  56, full = 0, empty = 1
 1145  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  56, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1145  [ALU:: 0] rstn = 1, control_bus = 0111, opt1 =  12, opt2 =  56, result =  44, stk_data_in =  44, stk_push = 0, stk_pop = 0, stk_data_out =  56 z_flag = 0, s_flag = 0,
 1145  [STACK] rstn = 1, stack_ptr =  1, top =  44, data_in =  44, push = 0, pop = 0, data_out =  56, full = 0, empty = 0
 1145  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 1
 1155  [CPU::0] ifidc_fetch = 0, exec_en = 0, exec_finish = 1
 1155  [STACK] rstn = 1, stack_ptr =  2, top =  44, data_in =   z, push = z, pop = z, data_out =  56, full = 0, empty = 0
 1155  [WBPB::0] rstn = 1, en = 1, fin_sig = 0, control_bus = 0111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  56, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1155  [ALU:: 0] rstn = 1, control_bus = 0111, opt1 =  12, opt2 =  56, result =  44, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  56 z_flag = 0, s_flag = 0,
 1155  [PC::0] rstn = 1, en = 0, control_bus = 0111, pc =  9, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  56
 1155  [CPU::0] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
 1165  [CPU::1] ifidc_fetch = 1, exec_en = 0, exec_finish = 0
 1165  [IFIDC::0] pc =  9 control_bus = 0111 addr_imm =   x IS_ready = 0 rstn = 1 en = 1
 1175  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
 1175  [IFIDC::1] pc =  9 control_bus = 0111 addr_imm =   x IS_ready = 0 rstn = 1 en = 0
 1185  [IFIDC::1] pc =  9 control_bus = 0111 addr_imm =   x IS_ready = 0 rstn = 1 en = 0
 1195  [IFIDC::0] pc =  9 control_bus = 0010 addr_imm =   2 IS_ready = 1 rstn = 1 en = 0
 1195  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0010, addr_const =   2, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  56, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1195  [PC::0] rstn = 1, en = 0, control_bus = 0010, pc =  9, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  56
 1195  [ALU:: 0] rstn = 1, control_bus = 0010, opt1 =  12, opt2 =  56, result =  44, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  56 z_flag = 0, s_flag = 0,
 1205  [IFIDC::0] pc =  9 control_bus = 0010 addr_imm =   2 IS_ready = 0 rstn = 1 en = 0
 1205  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
 1215  [CPU::1] ifidc_fetch = 0, exec_en = 1, exec_finish = 0
 1215  [WBPB::0] rstn = 1, en = 1, fin_sig = 0, control_bus = 0010, addr_const =   2, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  56, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1215  [PC::0] rstn = 1, en = 1, control_bus = 0010, pc =  9, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  56
 1225  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0010, addr_const =   2, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  56, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1225  [PC::1] rstn = 1, en = 0, control_bus = 0010, pc =  9, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  56
 1225  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
 1235  [IFIDC::0] pc =  10 control_bus = 0010 addr_imm =   2 IS_ready = 0 rstn = 1 en = 0
 1235  [PC::0] rstn = 1, en = 0, control_bus = 0010, pc =  10, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  56
 1235  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0010, addr_const =   2, stk_data_in =   z, stk_push = 0, stk_pop = 1, stk_data_out =  56, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1245  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0010, addr_const =   2, stk_data_in =   z, stk_push = 0, stk_pop = 1, stk_data_out =  44, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1245  [STACK] rstn = 1, stack_ptr =  0, top =  x, data_in =   z, push = 0, pop = 1, data_out =  44, full = 0, empty = 0
 1245  [PC::0] rstn = 1, en = 0, control_bus = 0010, pc =  10, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  44
 1245  [ALU:: 0] rstn = 1, control_bus = 0010, opt1 =  12, opt2 =  56, result =  44, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  44 z_flag = 0, s_flag = 0,
 1245  [STACK] rstn = 1, stack_ptr =  0, top =  x, data_in =   z, push = 0, pop = 1, data_out =  44, full = 0, empty = 1
 1255  [MEMORY] r_en = 0, addr =   2, w_en = 1, data_in = 00101100, data_out = 00011100
 1255  [WBPB::0] rstn = 1, en = 0, fin_sig = 1, control_bus = 0010, addr_const =   2, stk_data_in =   z, stk_push = 0, stk_pop = 1, stk_data_out =  44, mem_data_in =  44, mem_addr =   2, mem_r_en = 0, mem_w_en = 1, mem_data_out =  28
 1255  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 1
 1265  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 0010, addr_const =   2, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  44, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1265  [MEMORY] r_en = z, addr =   z, w_en = z, data_in = zzzzzzzz, data_out = 00011100
 1265  [CPU::0] ifidc_fetch = 0, exec_en = 0, exec_finish = 1
 1265  [CPU::0] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
 1275  [CPU::1] ifidc_fetch = 1, exec_en = 0, exec_finish = 0
 1275  [IFIDC::0] pc = 10 control_bus = 0010 addr_imm =   2 IS_ready = 0 rstn = 1 en = 1
 1285  [IFIDC::1] pc = 10 control_bus = 0010 addr_imm =   2 IS_ready = 0 rstn = 1 en = 0
 1285  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
 1295  [IFIDC::2] pc = 10 control_bus = 0010 addr_imm =   2 IS_ready = 0 rstn = 1 en = 0
 1305  [IFIDC::0] pc = 10 control_bus = 1111 addr_imm =   x IS_ready = 1 rstn = 1 en = 0
 1305  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 1111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  44, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1305  [PC::0] rstn = 1, en = 0, control_bus = 1111, pc = 10, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  44
 1305  [ALU:: 0] rstn = 1, control_bus = 1111, opt1 =  12, opt2 =  56, result =  44, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  44 z_flag = 0, s_flag = 0,
 1315  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
 1315  [IFIDC::0] pc = 10 control_bus = 1111 addr_imm =   x IS_ready = 0 rstn = 1 en = 0
 1325  [CPU::1] ifidc_fetch = 0, exec_en = 1, exec_finish = 0
 1325  [WBPB::0] rstn = 1, en = 1, fin_sig = 0, control_bus = 1111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  44, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1325  [PC::0] rstn = 1, en = 1, control_bus = 1111, pc = 10, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  44
 1335  [CPU::1] ifidc_fetch = 0, exec_en = 0, exec_finish = 0
 1335  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 1111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  44, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1335  [PC::1] rstn = 1, en = 0, control_bus = 1111, pc = 10, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  44
 1335  [ALU:: 1] rstn = 1, control_bus = 1111, opt1 =  12, opt2 =  56, result =  44, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  44 z_flag = 0, s_flag = 0,
WARNING: PC.v:98: $writememb: Standard inconsistency, following 1364-2005.
 1345  [STACK] rstn = 1, stack_ptr =  0, top =  x, data_in =   z, push = 0, pop = 1, data_out =  44, full = 0, empty = 1
 1345  [WBPB::0] rstn = 1, en = 0, fin_sig = 0, control_bus = 1111, addr_const =   x, stk_data_in =   z, stk_push = z, stk_pop = z, stk_data_out =  44, mem_data_in =   z, mem_addr =   z, mem_r_en = z, mem_w_en = z, mem_data_out =  28
 1345  [ALU:: 2] rstn = 1, control_bus = 1111, opt1 =  12, opt2 =  56, result =  44, stk_data_in =   z, stk_push = 0, stk_pop = 1, stk_data_out =  44 z_flag = 0, s_flag = 0,
 1345  [PC::1] rstn = 1, en = 0, control_bus = 1111, pc = 10, z_flag = 0, s_flag = 0, stk_pop = z, stk_data_out =  44
```

شکل موجها به شرح زیر است:

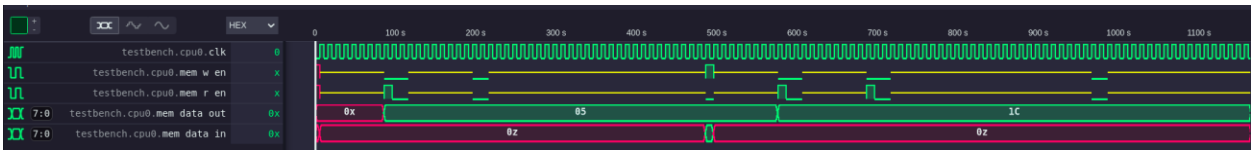
IFIDC_EXEC



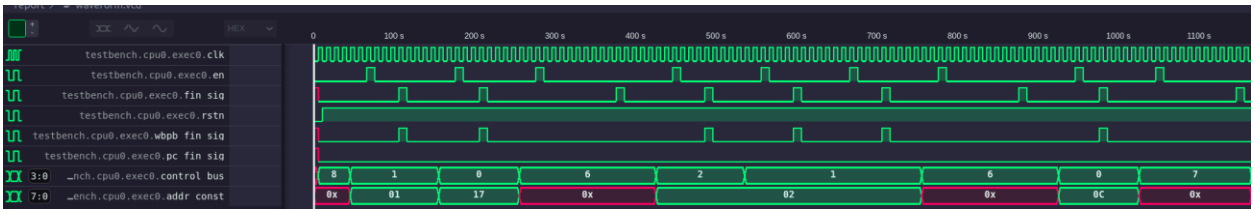
ALU



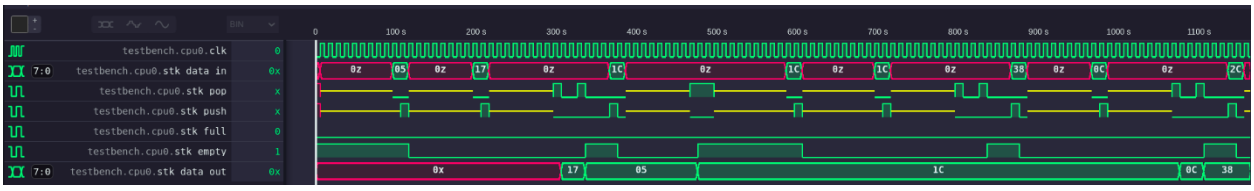
Memory

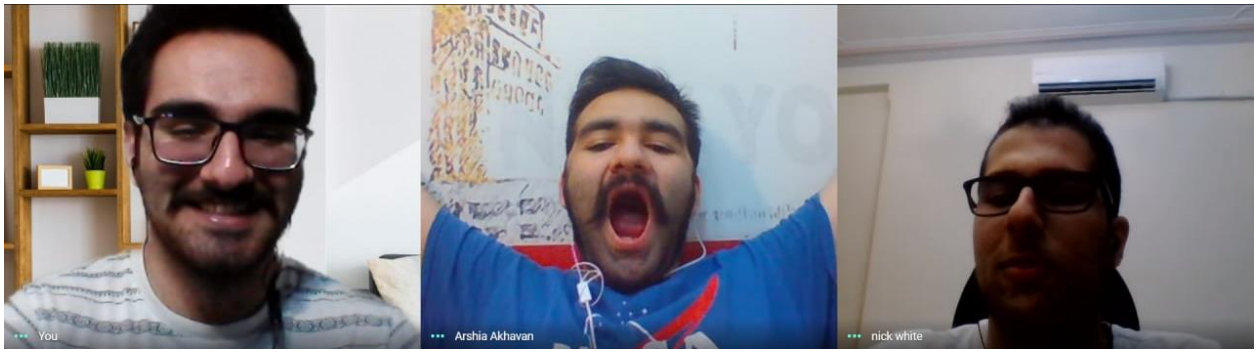


PP_PC



Stack





پایان :)