"بسمه تعالی"

دانشکده فنی

برگ امتحانی شماره: .....

مهر امتحانات

نام خانوادگی ..... نام ..... رشته تحصیلی ..... شماره دانشجویی .....

نام و تعداد واحد درس ..... نیمسال اول / دوم سال تحصیلی ..... نام استاد .....

تاریخ امتحان ..... ساعت .....

شماره سوال	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	نمره نهایی	امضاء استاد
نمره												

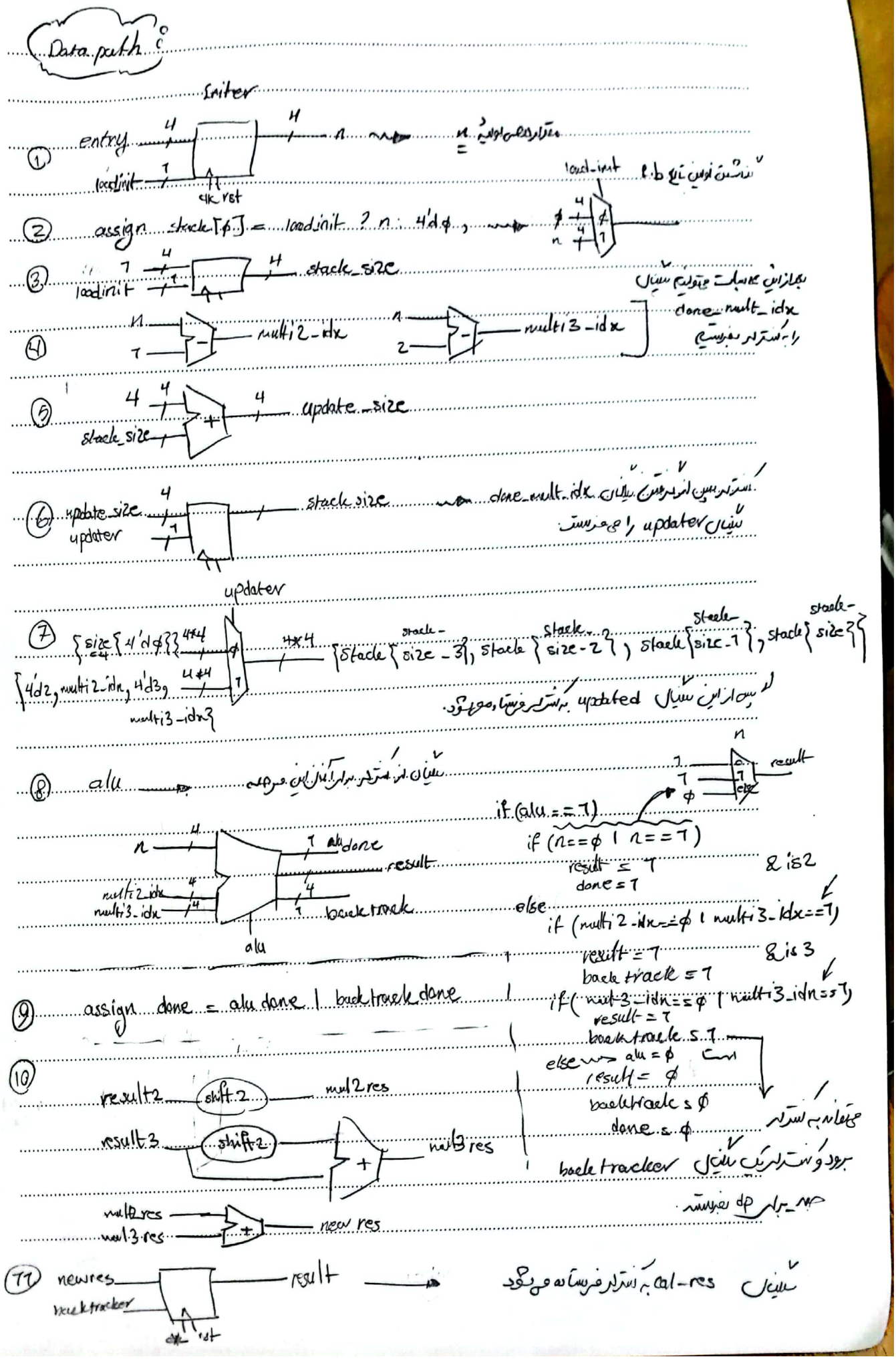
این دفترچه شامل ۴ برگ امتحانی و ۲ برگ پیش نویسی رنگی می باشد.

8101984494

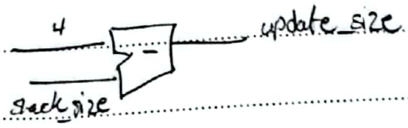
ستاره‌ای دهستانی

```
int fib(int n)
if n<1
return 1;
else
return 2*fib(n-1) + 3*fib(n-2);
```

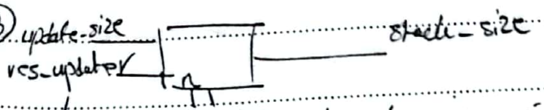
Data path


Initer
entry
loadinit
clk rst
load_init
stack_size
multi2_idx
multi3_idx
done_mult_idx
update_size
updater
alu
result
if (alu == 1)
if (n == 0 | n == 1)
result = 1
done = 1
else
if (multi2_idx == 0 | multi3_idx == 1)
result = 1
back track = 1
if (multi3_idx == 0 | multi3_idx == 1)
result = 1
back track = 1
else alu = 0
result = 0
back track = 0
done = 0
shift 2
mul2res
result2
result3
mul3res
new res
newres
result

(12)



(13)



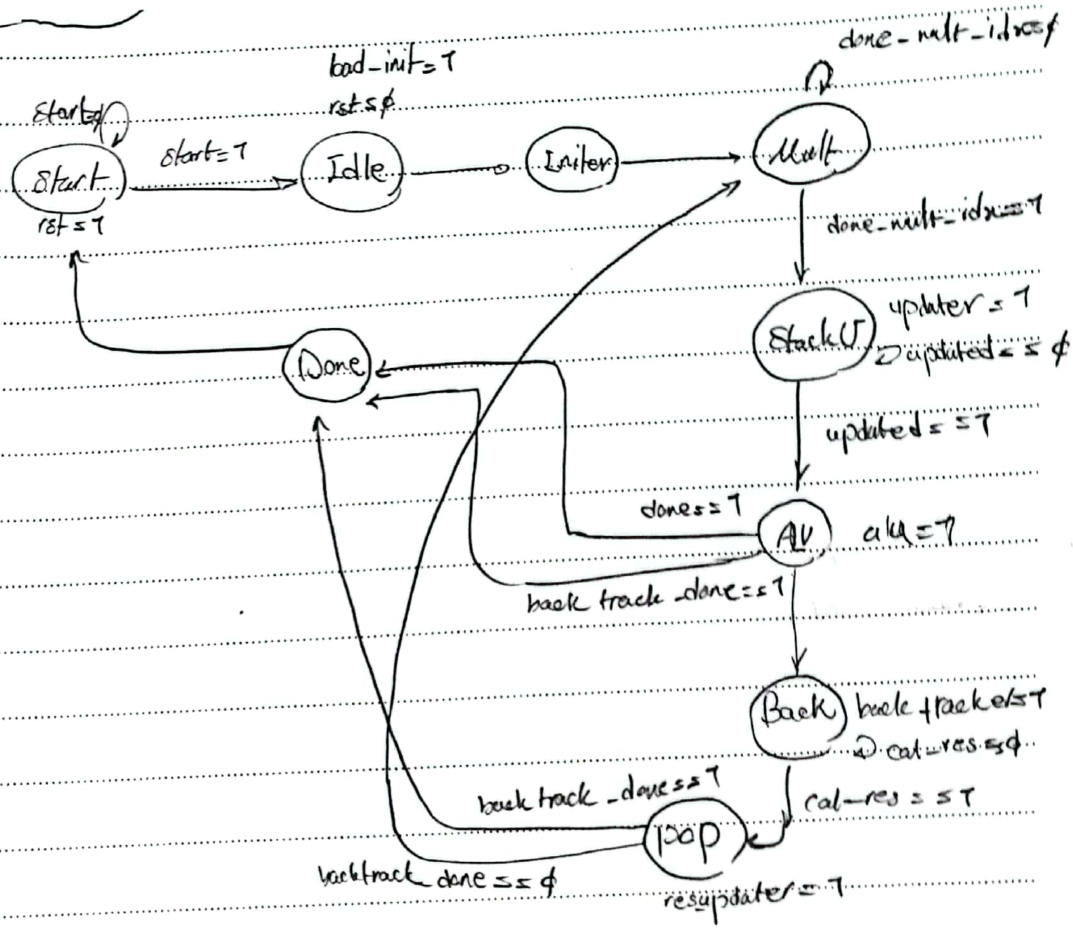
(14) assign {stack{stack_size+1}, stack{stack_size+2} ...} = (res_updater) ? {4'd0, ...} : { }

+39*4

خروجی

assign backtrack_done = stack_size == 1;

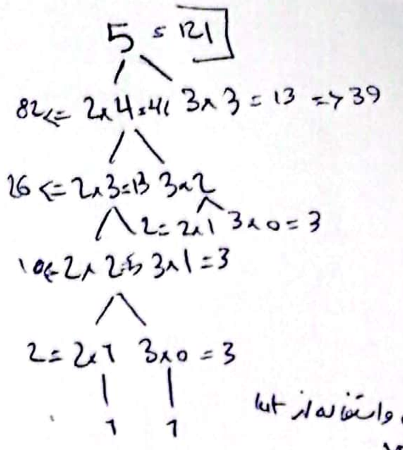
Controller :

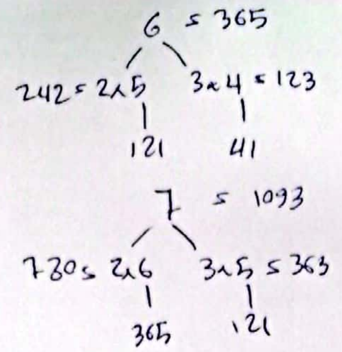


به نام خدا

رامین احمدپور - 810198494

الگوریتم مثال از نحوه انجام سئوال:

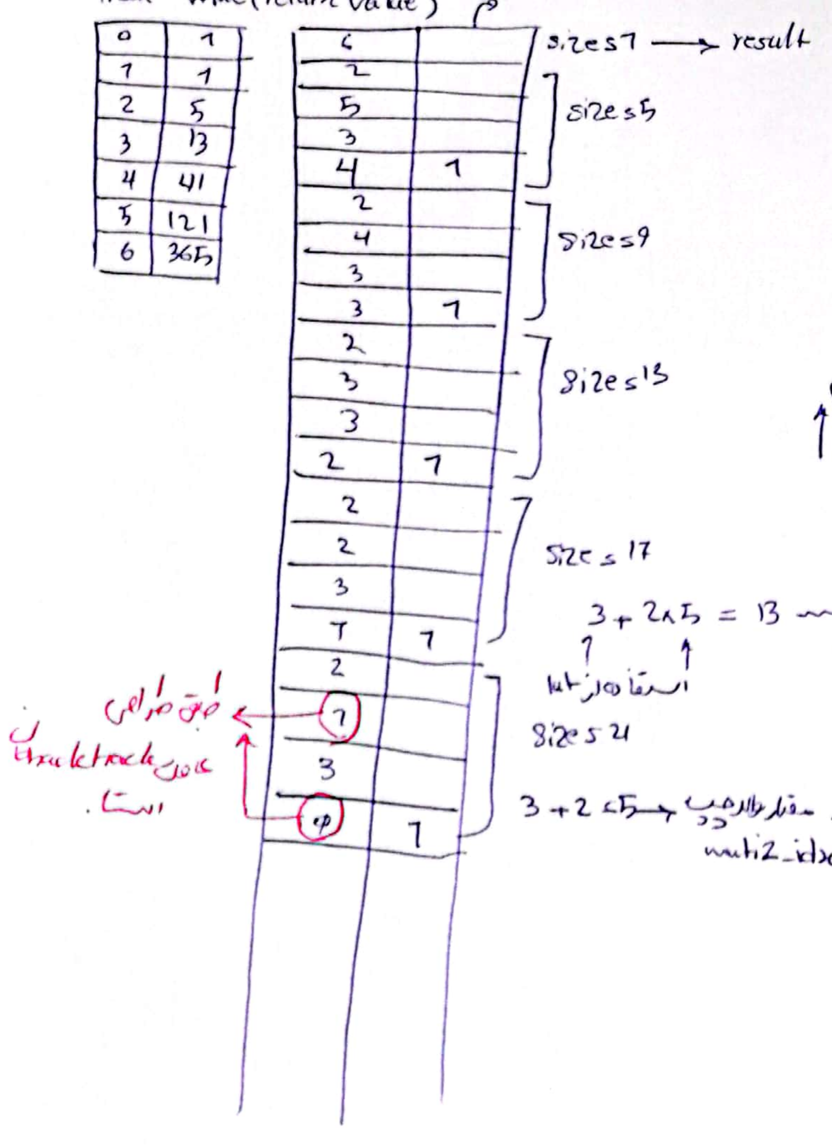




عدم نیاز به پیمایش و استفاده از lut برای value

به طور مشابه در ادامه آرایه value داریم که مقادیر محاسبه شده را نگه می‌دارد.

index	value (return value)
0	1
1	1
2	5
3	13
4	41
5	121
6	365



به طور مشابه و با همان فرم پیش می‌رویم و backtrack می‌کنیم و همه چیز را آپدیت می‌کنیم.

$3 + 2 \times 5 = 13$ ← مقدار محاسبه شده (در واقع مقدار والد فرزند است) ← مقدار 3 = 13 → ذخیره در lut

استفاده از lut

$3 + 2 = 5$ ← مقدار والد multi2_idx+1 → مقدار 2 = 5 → ذخیره در lut

کد دیتاپث:

```
`timescale 1ns/1ns
module Datapath (
    clk,
    rst,

    entry,
    load_init,
    updater,
    alu,
    cal_res,
    res_updater,
    poping,
    dont_check,
    push,

    updated,
    done,
    backtrack,
    cal_update,
    result
);

    parameter size = 4;
    parameter stack_length = 128;
    parameter stack_digit = 7;
    parameter max_length = 15; // max entry number

    input clk,rst;
    input [size-1:0] entry;
    input load_init;
    input updater;
    input alu;
    input cal_res;
    input res_updater;
    input poping;
    input dont_check;
    input push;

    output updated;
    output done;
    output backtrack;
    output cal_update;
    output [5*size:0] result;
```

```
    wire [size-1:0] n;
    wire [size-1:0] n_input;
    wire [size-1:0] multi2_idx;
    wire [size-1:0] multi3_idx;
    wire [size-1:0] multi2_idx_;
    wire [size-1:0] multi3_idx_;
    wire [size-1:0] multi2_in;
    wire [size-1:0] multi3_in;
    wire [size-1:0] rm2;
    wire [size-1:0] rm3;
    reg [size-1:0] stack [stack_length:0];
    reg [stack_length-1:0] stack_right;
    reg [max_length-1:0] visited;
    reg [5*size-1:0] value [max_length:0];
    wire [5*size-1:0] multres;
    wire [stack_digit-1:0] stack_size;
    wire [stack_digit-1:0] update_size;
    wire [stack_digit-1:0] updater_size;
    wire [stack_digit-1:0] size_input;
    wire d2, d3;
    wire bt2, bt3;
    wire [size-1:0] ass2, ass3,assM2, assM3;
    wire assW;

    reg [size-1:0] two = {{(size-2){1'b0}}, 2'b10};
    reg [size-1:0] three = {{(size-2){1'b0}}, 2'b11};

    assign value[0] = { {(2*size-1){1'b0}}, 1'b1};
    assign value[1] = { {(2*size-1){1'b0}}, 1'b1};

    AddSub nMinus1(.left(n), .right(4'd1), .addsub(2'd0), .res(multi2_idx_));
    AddSub nMinus2(.left(n), .right(4'd2), .addsub(2'd0), .res(multi3_idx_));

    AddSub #(7) addStackLen(.left(stack_size), .right(7'd4), .addsub(2'd1),
.res(update_size));

    Register #(4) assign2(.clk(clk), .rst(rst), .en(updater), .pi(two),
.po(ass2));
    Register #(4) assign3(.clk(clk), .rst(rst), .en(updater), .pi(three),
.po(ass3));
    Register #(4) assignM2(.clk(clk), .rst(rst), .en(updater), .pi(multi2_idx),
.po(assM2));
```

```
    Register #(4) assignM3(.clk(clk), .rst(rst), .en(updater), .pi(multi3_idx),
.po(assM3));
    Register #(1) assignWait(.clk(clk), .rst(rst), .en(updater), .pi(1'b1),
.po(assW));

    always @(*) begin
        if (load_init) begin
            stack[1] = entry;
            visited[entry] = 1'b1;
        end
        else begin
            if (updater) begin
                visited[multi2_idx] <= 1'b1;
            end
            if (push) begin
                if (stack_size-7'd1 == 0) begin
                end
                else begin
                    stack[stack_size-7'd3] <= ass2 ;
                    stack[stack_size-7'd1] <= ass3 ;
                    stack[stack_size-7'd2] <= assM2 ;
                    stack[stack_size] <= assM3 ;
                    stack_right[stack_size] <= assW ;
                end
            end
            if (bt2) begin
                value[multi2_idx] <= {4'd0,rm2};
            end
            if (bt3) begin
                value[multi3_idx] <= {4'd0,rm3};
            end
            if (cal_res) begin
                value[n] <= multres;
            end
        end
    end
    Register #(1) ackUpdate(.clk(clk), .rst(rst), .en(updater), .pi(1'b1),
.po(updated));

    ALU #(4) alu2(.alu(alu), .twothree(1'b1), .n(n), .m2(multi2_idx),
.m3(multi3_idx),
        .result(rm2), .done(d2), .backtrack(bt2));
    ALU #(4) alu3(.alu(alu), .twothree(1'b0), .n(n), .m2(multi2_idx),
.m3(multi3_idx),
        .result(rm3), .done(d3), .backtrack(bt3));
```

```
    assign done = (dont_check == 1'b0) & (stack_size==1'b1) & (load_init==1'b0);
    // assign backtrack = bt3 & bt2;

    Register #(1) backtrackerReg(.clk(clk), .rst(rst), .en(bt3 & bt2), .pi(1'b1),
.po(backtrack));

    assign multres = two*value[multi2_idx] + three*value[multi3_idx];

    AddSub #(7) subStackLen(.left(stack_size), .right(7'd4), .addsub(2'd0),
.res(updater_size));

    assign size_input = load_init ? 7'd1 : (cal_res ? updater_size : (updater ?
update_size : 7'd0));

    Register #(7) sizeRegUpdate(.clk(clk), .rst(rst),
.en(cal_res|load_init|updater), .pi(size_input), .po(stack_size));
    Register #(1) cal_Updater(.clk(clk), .rst(rst), .en(cal_res), .pi(1'b1),
.po(cal_update));

    assign multi3_in = poping ? stack[stack_size] : multi3_idx_ ;
    assign multi2_in = poping ? stack[stack_size-2] : multi2_idx_ ;

    Register assignNewMult3(.clk(clk), .rst(rst), .en(1'b1), .pi(multi3_in),
.po(multi3_idx));
    Register assignNewMult2(.clk(clk), .rst(rst), .en(1'b1), .pi(multi2_in),
.po(multi2_idx));

    assign n_input = res_updater ? multi2_idx : ( poping ? stack[stack_size-
2]+1'b1 : (cal_res ? multi2_idx+1'b1 : (load_init ? entry : n_input)));
    Register res_update_n(.clk(clk), .rst(rst),
.en(res_updater|poping|cal_res|load_init), .pi(n_input), .po(n));

    assign result = value[entry];

endmodule
```

کد کنترلر:

```
`timescale 1ns/1ns
module Controller (
    clk,
    rst,
    start,
```

```
    updated,
    done,
    backtrack,
    cal_update,

    load_init,
    updater,
    alu,
    cal_res,
    res_updater,
    poping,
    dont_check,
    push
);

    input clk, rst;
    input start;
    input updated;
    input done;
    input backtrack;
    input cal_update;

    output reg load_init;
    output reg updater;
    output reg alu;
    output reg cal_res;
    output reg res_updater;
    output reg poping;
    output reg dont_check;
    output reg push;


    parameter [3:0]
        Start = 4'd0,
        Idle = 4'd1,
        Initer = 4'd2,
        Mult = 4'd3,
        Stack = 4'd4,
        ALU = 4'd5,
        Back = 4'd6,
        Pop = 4'd7,
        Poper = 4'd10,
        Update = 4'd8,
        Done = 4'd9;
```

```
    reg [3:0] ps, ns;


    always @(posedge clk, posedge rst) begin
        if(rst)begin
            ps <= Start;
        end
        else
            ps <= ns;
    end

    always @(ps, start, updated, done, backtrack, cal_update) begin
        case (ps)
            Start:      ns = start ? Idle : Start;
            Idle:       ns = Initer;
            Initer:     ns = Mult;
            Mult:       ns = Stack;
            Stack:      ns = ALU;
            ALU:        ns = Back;
            Back:       ns = backtrack ? Pop : Update;
            Pop:        ns = done ? Done : Poper;
            Poper:      ns = Pop;
            Update:     ns = Mult;
            Done:       ns = Start;

            default: ns = Start;
        endcase
    end

    always @(ps) begin
        {load_init, updater, alu, cal_res, res_updater, poping, dont_check, push}
= 0;
        case (ps)
            Start: begin
                dont_check = 1'b1;

            end
            Idle: begin
                load_init = 1'b1;
                dont_check = 1'b1;
            end
            Initer: begin
                dont_check = 1'b1;

            end
```

```
            Mult: begin
                dont_check = 1'b1;
            end

            Stack: begin
                dont_check = 1'b1;
                updater = 1'b1;
                push = 1'b1;
            end
            ALU: begin
                alu = 1'b1;
            end
            Back: begin

            end
            Pop: begin
                cal_res = 1'b1;
            end
            Poper: begin
                poping = 1'b1;
            end
            Update: begin
                res_updater = 1'b1;
            end
            Done: begin

            end
        endcase
    end


endmodule
```

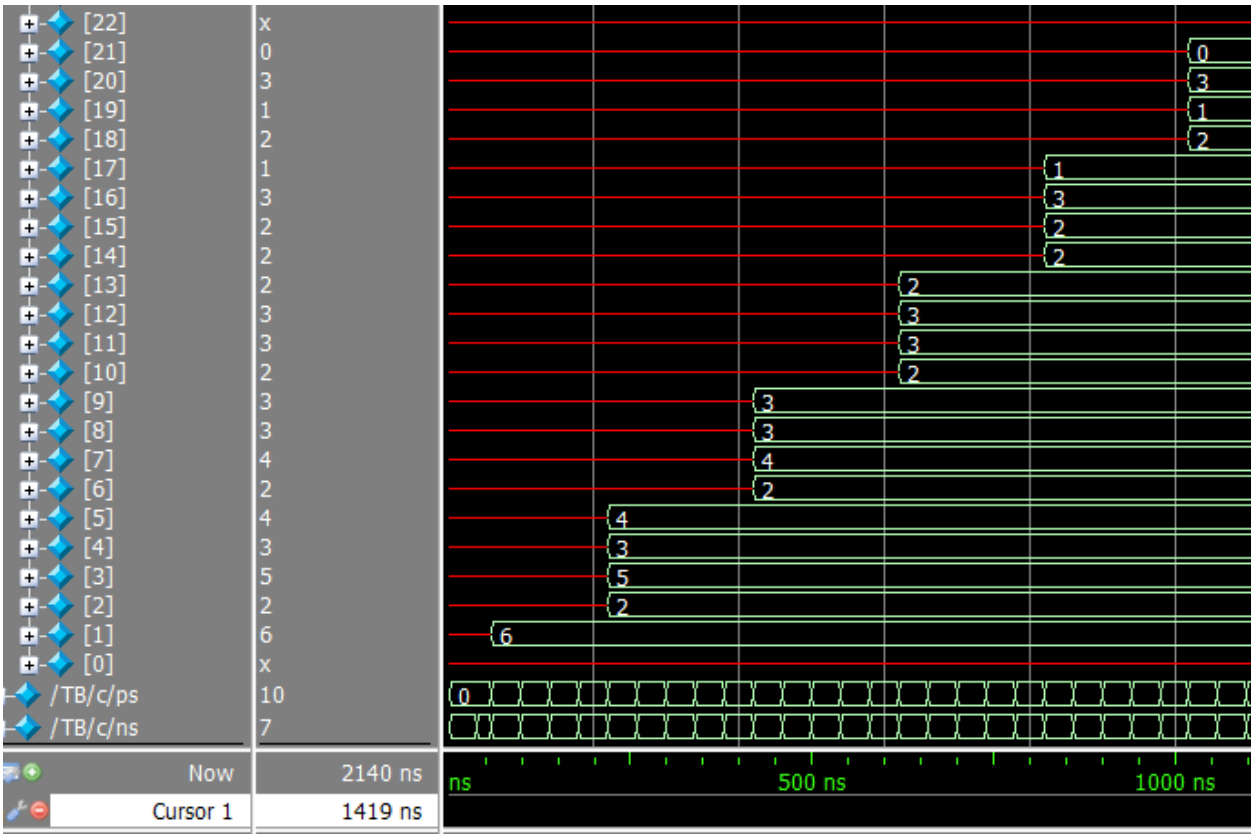
کد تست بنچ:

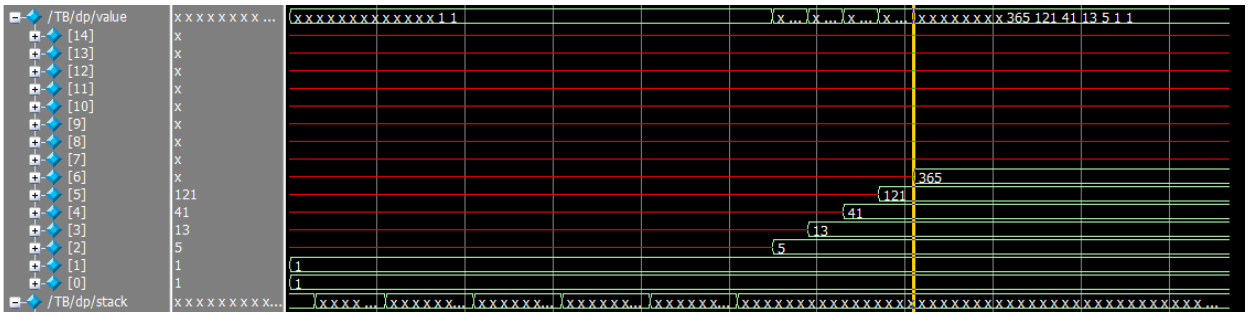
```
`timescale 1ns/1ns

module TB ();

    reg clk=1'b0, rst=1'b1, start=1'b0;
    reg [3:0] entry = 4'd15;
    wire load_init;
    wire updater;
    wire alu;
    wire cal_res;
```

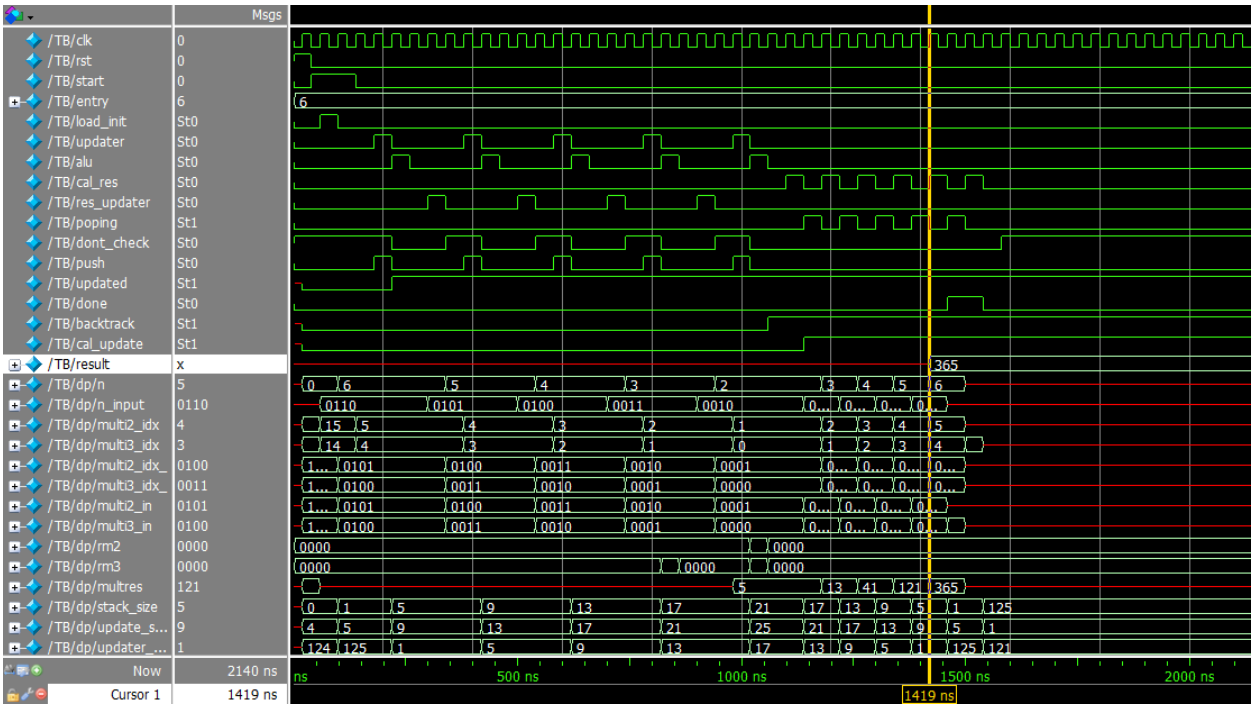
```
    wire res_updater;
    wire poping;
    wire dont_check;
    wire push;
    wire updated;
    wire done;
    wire backtrack;
    wire cal_update;
    wire [20:0] result;

    Controller c(
    clk,
    rst,
    start,

    updated,
    done,
    backtrack,
    cal_update,

    load_init,
    updater,
    alu,
    cal_res,
    res_updater,
    poping,
    dont_check,
    push
    );

    Datapath dp(
    clk,
    rst,

    entry,
    load_init,
    updater,
    alu,
    cal_res,
    res_updater,
    poping,
    dont_check,
    push,

    updated,
```

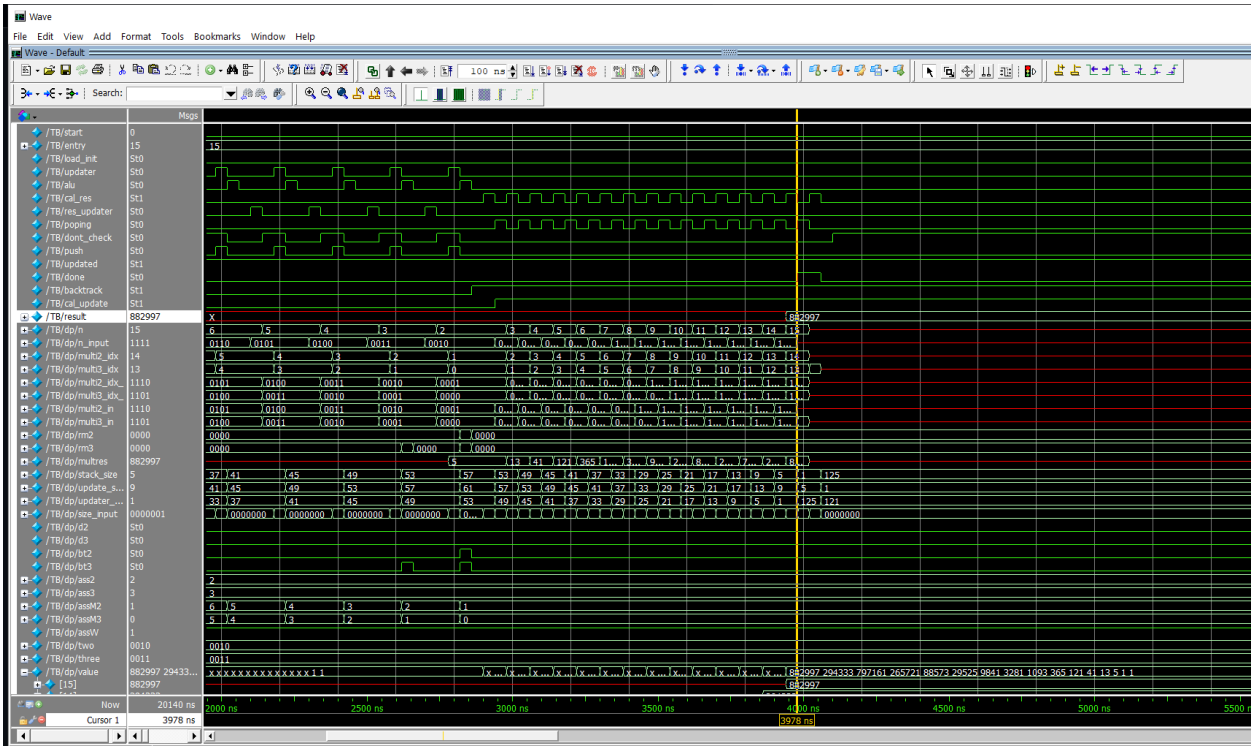
```
    done,
    backtrack,
    cal_update,
    result
    );

    always #20 clk = ~clk;

    initial begin
        #40 rst = 1'b0; start = 1'b1;
        #100 start = 1'b0;
        #20000 $stop;
    end

endmodule
```

خروجی:




/TB/dp/value
[14]
[13]
[12]
[11]
[10]
[9]
[8]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
/TB/dp/stack
365
121
41
13
5
1
1


Msgs
/TB/clk
/TB/rst
/TB/start
/TB/entry
/TB/load_init
/TB/updater
/TB/alu
/TB/cal_res
/TB/res_updater
/TB/poping
/TB/dont_check
/TB/push
/TB/updated
/TB/done
/TB/backtrack
/TB/cal_update
/TB/result
/TB/dp/n
/TB/dp/n_input
/TB/dp/multi2_idx
/TB/dp/multi3_idx
/TB/dp/multi2_idx_
/TB/dp/multi3_idx_
/TB/dp/multi2_in
/TB/dp/multi3_in
/TB/dp/rm2
/TB/dp/rm3
/TB/dp/multres
/TB/dp/stack_size
/TB/dp/update_s...
/TB/dp/updater_...
Now
2140 ns
Cursor 1
1419 ns
500 ns
1000 ns
1500 ns
2000 ns


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/TB/start
/TB/entry
/TB/load_init
/TB/updater
/TB/alu
/TB/cal_res
/TB/res_updater
/TB/popping
/TB/dont_check
/TB/push
/TB/updated
/TB/done
/TB/backtrack
/TB/cal_update
/TB/result
/TB/dp/n
/TB/dp/n_input
/TB/dp/multi2_idx
/TB/dp/multi3_idx
/TB/dp/multi2_idx_
/TB/dp/multi3_idx_
/TB/dp/multi2_in
/TB/dp/multi3_in
/TB/dp/rm2
/TB/dp/rm3
/TB/dp/multres
/TB/dp/stack_size
/TB/dp/update_s...
/TB/dp/updater_...
/TB/dp/size_input
/TB/dp/d2
/TB/dp/d3
/TB/dp/bt2
/TB/dp/bt3
/TB/dp/ass2
/TB/dp/ass3
/TB/dp/assM2
/TB/dp/assM3
/TB/dp/assW
/TB/dp/two
/TB/dp/three
/TB/dp/value
[15]
Now
Cursor 1
20140 ns
3978 ns
2000 ns
2500 ns
3000 ns
3500 ns
4000 ns
4500 ns
5000 ns
882997
882997 294333 797161 265721 88573 29525 9841 3281 1093 365 121 41 13 5 1 1