به نام او

علی امینی

401170529

سوال 7

در این سوال می خواهیم cpu ای را پیاده سازی کنیم که دارای 3 بخش اصلی reg , memory , alu هست که این 3 بخش توسط بخش اصلی پردازنده به کار گرفته می شوند.

بخشreg دارای 4 آرایه 512 بیتی به نام های a , b , s0 , s1 هست.

بخش alu بخش انجام دهنده جمع و ضرب ما هست که محتوای جمع را روی s0 و محتوای کم و پر ارزش ضرب را به ترتیب بر روی s0 و s1 می ریزد.

بخش mem بخشی هست که ما در آن داده های خود را ذخیره کرده و از آن می خوانیم.

بخش cpu بخشی هست که این 3 بخش را به هم وصل کرده و به آن ها سیگنال های کنترلی ارسال می کند.

## ALU

این بخش دو سیگنال se1 , se0 را به عنوان کنترل می گیرد.

سیگنال se1 مشخص می کند که دستور ما از نوع ضرب یا جمع هست یا نه و سیگنال se0 هم بین ضرب و جمع انتخاب می کند.

همچنین a , b به عنوان ورودی به این بخش داده می شوند و s0 , s1 به عنوان خروجی از این بخش خارج می شوند.

```
module ALU (
    input se1,
    input se0,
    input signed [511:0] a,
    input signed [511:0] b,
    output reg [511:0] s0,
    output reg [511:0] s1
);

reg signed [63:0] res;
integer i = 0;

always @(*) begin
        if (se1) begin
            for (i = 0; i < 16; i = i + 1) begin
                if (!se0) begin
                    res = $signed(a[i*32 +: 32]) + $signed(b[i*32 +: 32]);
                    s0[i*32 +: 32] = res[31:0];
                    s1[i*32 +: 32] = 32'h00000000;
                end
                else if(se0) begin
                    res = $signed(a[i*32 +: 32]) * $signed(b[i*32 +: 32]);
                    s0[i*32 +: 32] = res[31:0];
                    s1[i*32 +: 32] = res[63:32];
                end

            end
        end
    end

endmodule
```

ابتدا یک رجیستر ساینڈ برای ضرب و جمع در ALU تعریف می شود.

سپس در بلوک always اگر se1 دارای مقدار 1 باشد یعنی باید عملیات انجام بدهیم و با 0 بودن se0 این عملیات از نوع جمع هست. مقدار جمع a + b در رجیستر res ریخته می شود و 32 بیت کم ارزش آن درون s0 ذخیره می شود. همچنین مقدار 32’h00000000 درون خروجی s1 ریخته می شود.

در صورت 1 بودنse0 عملیات ما ضرب هست. ابتدا مقدار ضرب با علامت a * b در رجیستر res محاسبه شده و سپس 32 بیت کم ارزش و پر ارزش آن به ترتیب در s0 و s1 ریخته شده و به عنوان خروجی پس فرستاده می شود.

# بخش MEMORY

این بخش وظیفه نوشتن و خواندن به صورت خالص از مموری را بر عهده دارد.

این بخش آدرس حافظه را ورودی گرفته و کار خود را بر روی آن خانه انجام می دهد.

همچنین دو سیگنال کنترلیre, we را ورودی گرفته تا مشخص شود باید روی خانه ی مذکور بنویسد یا از آن خانه بخواند.

یک din هم به عنوان ورودی دریافت می کند که دیتا ی ورودی برای نوشتن هست.

و در آخر یک dout را خروجی می دهد که دیتای خوانده شده از مموری هست.

```
reg [8:0] add;
reg [31:0] ram [0:511];

integer i;

always @(*) begin
    add = madd;

    if (re) begin
        add = add - (add%512);
        for (i = 0; i < 16; i = i + 1) begin
            dout[i*32 +: 32] = ram[add];
            add = add + 1;
        end
    end

    else if (we) begin
        add = add - (add%512);
        for (i = 0; i < 16; i = i + 1) begin
            ram[add + i] = din[i*32 +: 32];
            add = add + 1;
        end
    end
end
```

بخش کلی این ماژول به این شکل هست.

در بلوک always چک می کنیم که re یا we 1 هستند و طبق آن می خوانیم یا می نویسیم.

در ابتدا هر کدام آدرس را به اولین نقطه هر بلوک حافظه می رسانیم یعنی اگر برای مثال آدرس 6 به ما داده شده باشد می دانیم که می خواهیم بر روی بلوک 0 حافظه کار انجام دهیم پس آدرس را برابر با

اولین خانه این بلوک قرار می دهیم. سپس اگز بخواهیم بخوانیم دیتا ی رم را روی dout ریخته و اگر بخواهیم بنویسیم دیتا ی din را روی رم قرار می دهیم.

در آخر آدرس +1 می شود و به سراغ خانه بعدی آن بلوک حافظه می رویم.

در آخر این ماژول یک بخش initial قرار داده ایم که در فایل TB برای محاسبه edge case ها و تست های رندوم از آن ها استفاده می کنیم.

## بخش REG

این بخش وظیفه متصل کردن دو بخش قبل را دارد. طراحی این CPU به شکلی است که ALU به صورت مستقیم دسترسی به MEMORY ندارد و ارتباط بین این دو بخش توسط بخش REG برقرار می شود.

خروجی های این بخش a,b,s0,s1 وread data هستند که به بخش های دیگر داده می شوند.

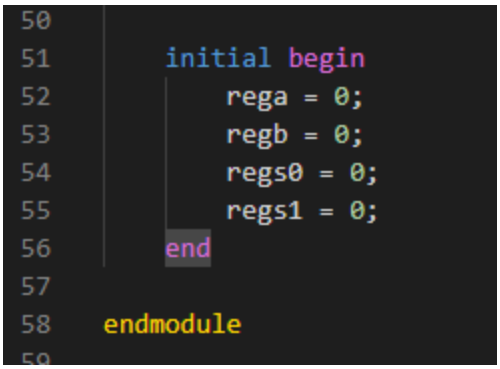
ورودی های این بخش شامل low data , high data , write data هستند و برای این بخش 3 سیگنال controller که مشخص کند دستور چیست و we , re که اینیبل برای خواندن و نوشتن هستند در نظر گرفته شده است. همچنین دیگر ورودی این بخش address هست.

```
module REG (
    input [511:0] ldata,
    input [511:0] hdata,
    input [511:0] wdata,
    input we,
    input re,
    input select,
    input clk,
    input [1:0] add,
    output reg [511:0] read_data,
    output reg [511:0] a,
    output reg [511:0] b,
    output reg [511:0] s0,
    output reg [511:0] s1
);

    reg signed [511:0] rega, regb, regs0, regs

    always @(*) begin
        if (re) begin
            if (add == 2'b00)
                read_data = rega;
            else if (add == 2'b01)
                read_data = regb;
            else if (add == 2'b10)
                read_data = regs0;
            else if (add == 2'b11)
                read_data = regs1;
        end
        else if (we) begin
            if (add == 2'b00)
                rega <= wdata;
            else if (add == 2'b01)
                regb <= wdata;
            else if (add == 2'b10)
                regs0 <= wdata;
            else if (add == 2'b11)
                regs1 <= wdata;
        end
        else if (select) begin
            regs0 = ldata;
            regs1 = hdata;
        end

        a = rega;
        b = regb;
        s0 = regs0;
        s1 = regs1;
    end
```



```
50
51      initial begin
52          rega = 0;
53          regb = 0;
54          regs0 = 0;
55          regs1 = 0;
56      end
57
58  endmodule
```

در ابتدا 4 رجیستر داخلی با علامت درست می کنیم.

سپس در بلوک always اگر re فعال باشد محتوای یکی از رجیستر های درونی را بر روی خروجی read data می ریزیم و اگر we فعال باشد ورودی write data را بر روی یکی از رجیستر های درونی می ریزیم.

دقت کنید که انتخاب رجیستر های درونی بر حسب سیگنال کنترلی controller هست به این شکل که دو بیت کنترلر از اعداد 0 تا 3 به ترتیب s1 , s0 , b, a را نشان می دهند که با توجه به آن یکی از این رجیستر ها انتخاب شده و عملیات مورد نظر ما روی آن ها انجام می شود.

در بلوک initial محتوای ابتدایی این 4 رجیستر داخلی را برابر با 0 قرار می دهیم.

## بخش CPU

ورودی های این بخش از نوع آدرس و سیگنال کنترلی هستند و خروجی های آن s1 , s0 , b ,aو مموری و رجیستر هستند تا بتوانیم آن ها را روی TB مشاهده کنیم.

در ابتدا ی این بخش یکسیری سیم و رجیستر تعریف می کنیم و 3 بخش قبلی را تعریف کرده و به هم وصل می کنیم که چیز جدیدی ندارد.

```
always @(*) begin
    if (select == 2'b00) begin
        mem_re = 1;
        mem_we = 0;
        reg_we = 1;
        reg_re = 0;
        controller = 0;
        se0 = 0;
        se1 = 0;
    end
    else if (select == 2'b01) begin
        reg_we = 0;
        reg_re = 1;
        mem_re = 0;
        mem_we = 1;
        controller = 0;
        se1 = 1;
        se0 = 0;
    end
    else if (select == 2'b10) begin
        se1 = 1;
        se0 = 0;
        mem_re = 0;
        mem_we = 0;
        reg_we = 0;
        reg_re = 0;
        controller = 1;
    end
    else if (select == 2'b11) begin
        se1 = 1;
        se0 = 1;
        mem_re = 0;
        mem_we = 0;
        reg_we = 0;
        reg_re = 0;
        controller = 1;
    end
    a = rega;
    b = regb;
    s0 = regs0;
    s1 = regs1;
    mem_out = dout;
    reg_out = din;
end
```

در بلوک alway این بخش با توجه به سیگنال کنترلی select سیگنال های کنترلی دیگر بخش ها را تنظیم کرده و برای آن ها می فرستیم.

00: خواندن دیتا از روی رم و بیختن آن ها بر روی یکی از رجیستر ها

01: نوشتن دیتا ی یکی از رجیستر ها روی رم

10: با توجه به 1 بودن se1 این دستور مربوط به ALU هست و با 0 بودن se0 این عملیات از نوع جمع هست.

11: مانند دستور بالا اما با این تفاوت که با 1 بودن se0 این دستور مربوط به ضرب می شود.

## فایل TB

در این فایل ابتدا یک CPU تعریف کرده و سپس آن را امتحان می کنیم.

```
always
    #1 clk = ~clk;

initial begin
    #2 select = 2'b00;
        regnumber = 2'b00;
        madd = 278;
    #2 select = 2'b00;
        regnumber = 2'b01;
        madd = 345;
    #2 select = 2'b10;
    #2 select = 2'b11;
    #2 select = 2'b00;
        regnumber = 2'b00;
        madd = 1;
    #2 select = 2'b00;
        regnumber = 2'b01;
        madd = 1;
    #2 select = 2'b11;
    #2 select = 2'b00;
        regnumber = 2'b01;
        madd = 17;
    #2 select = 2'b11;
    #2 select = 2'b10;
    #2 select = 2'b00;
        regnumber = 2'b00;
        madd = 33;
    #2 select = 2'b10;
    #2 select = 2'b11;
    #50 $stop;
end
```

1) محتوای بلوکی که خانه 278 در آن قرار دارد را خوانده و بر روی رجیستر a می ریزد.
2) محتوای بلوکی که خانه 345 در آن قرار دارد را خوانده و بر روی رجیستر b می ریزد.
3) عملیات جمع a + b را انجام داده حاصل آن را در s0 و عدد 0 را در s1 می ریزد این عمل برای تست جمع دو عدد رندوم انجام شده. اعداد این بلوک ها را در بخش initial ماژول MEMORY ست کرده ایم که شکل آن را در زیر می بینید.

```
36      for (i = 0; i < 512; i = i + 1) begin
37      if (i % 16 == 0) begin
38          ram[i] = 32'hFFFFFFF0;
39      end else if (i % 16 == 1) begin
40          ram[i] = 32'hFFFFFFF1;
41      end else if (i % 16 == 2) begin
42          ram[i] = 32'hFFFFFFF2;
43      end else if (i % 16 == 3) begin
44          ram[i] = 32'hFFFFFFF3;
45      end else if (i % 16 == 4) begin
46          ram[i] = 32'hFFFFFFF4;
47      end else if (i % 16 == 5) begin
48          ram[i] = 32'hFFFFFFF5;
49      end else if (i % 16 == 6) begin
50          ram[i] = 32'hFFFFFFF6;
51      end else if (i % 16 == 7) begin
52          ram[i] = 32'hFFFFFFF7;
53      end else if (i % 16 == 8) begin
54          ram[i] = 32'hFFFFFFF8;
55      end else if (i % 16 == 9) begin
56          ram[i] = 32'hFFFFFFF9;
57      end else if (i % 16 == 10) begin
58          ram[i] = 32'hFFFFFFFA;
59      end else if (i % 16 == 11) begin
60          ram[i] = 32'hFFFFFFFB;
61      end else if (i % 16 == 12) begin
62          ram[i] = 32'hFFFFFFFC;
63      end else if (i % 16 == 13) begin
64          ram[i] = 32'hFFFFFFFD;
65      end else if (i % 16 == 14) begin
66          ram[i] = 32'hFFFFFFFE;
67      end else if (i % 16 == 15) begin
68          ram[i] = 32'hFFFFFFFF;
69      end
```

4) دستور 11 به عمل ضرب مربوط می شود. حاصل ضرب a * b را به شکل توضیح داده شده در بخش ALU در رجیستر های s0 و s1 می ریزد. این دستور برای تست ضرب دو عدد رندوم انجام شده است که نحوه انتخاب این دو عدد در دستور قبل توضیح داده شده است.

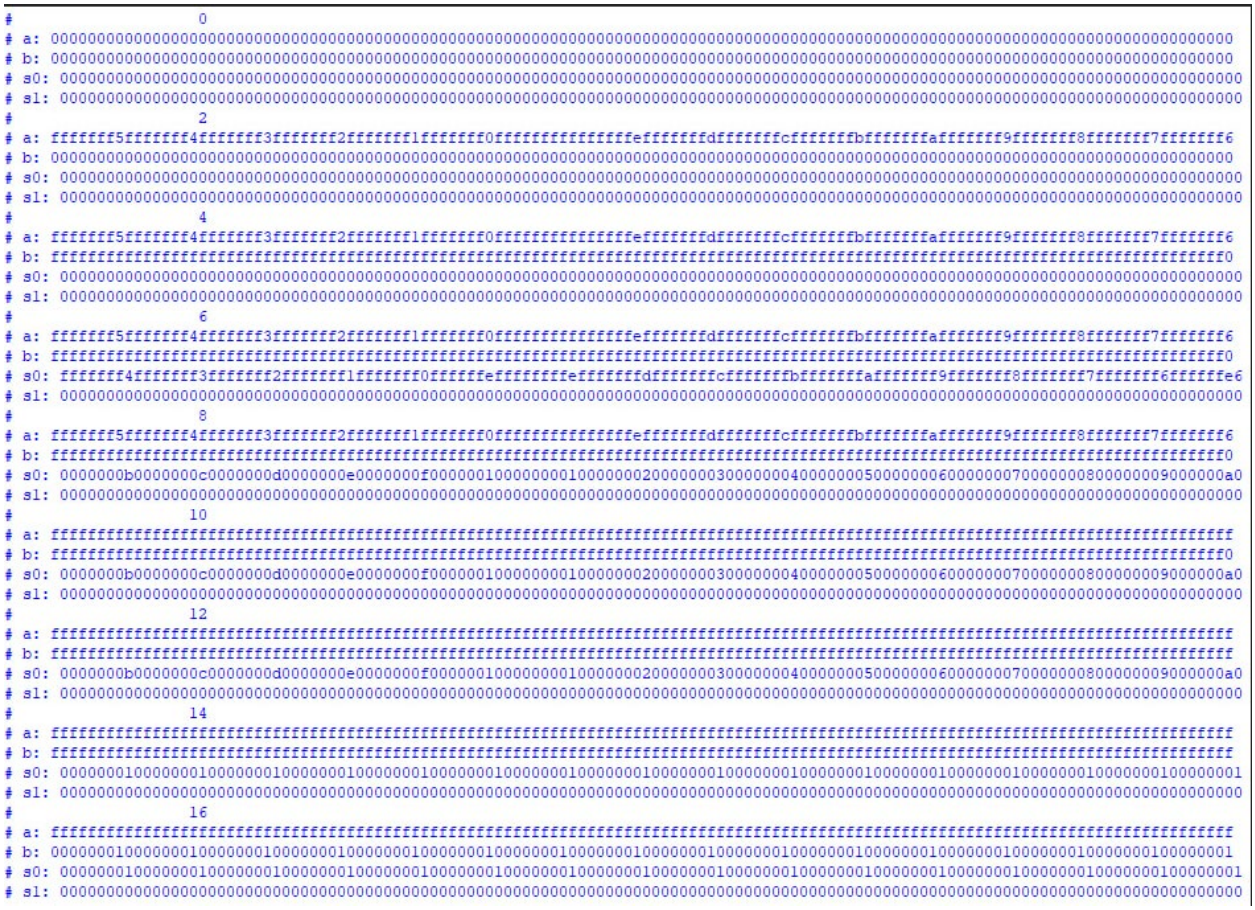
از این جا به بعد به تست کردن edge test ها می پردازیم.
اعداد مهم ما عدد 1 و -1 و بزرگترین عدد مثبت ممکن هستند.
این اعداد از پیش در 3 بلوک اولیه MEMORY ذخیره شده اند.
بدین شکل:

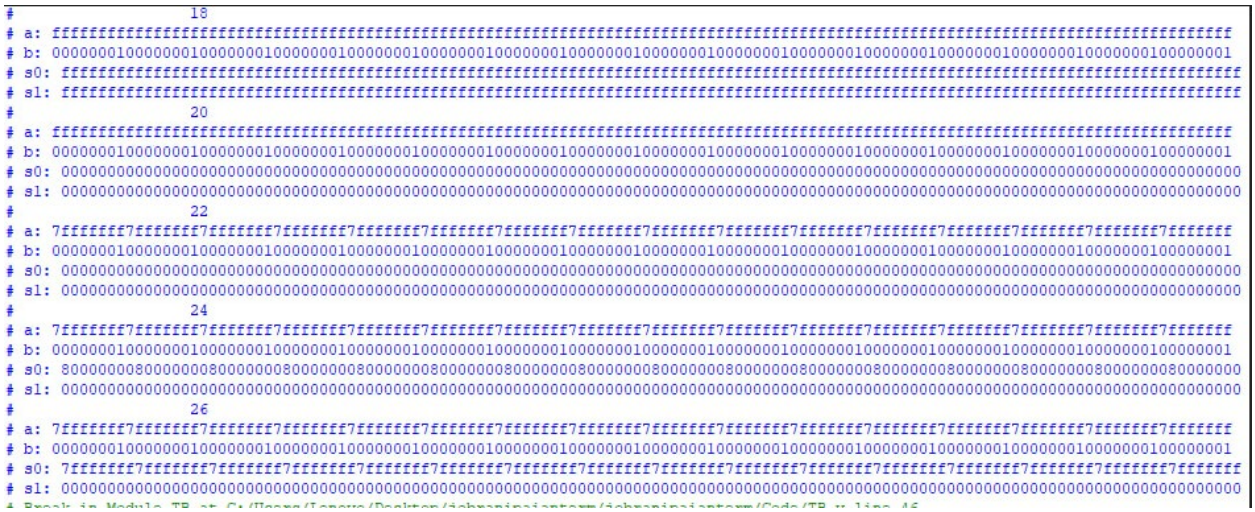
```
71      for (i = 1; i < 17; i = i + 1) begin
72          ram[i] = 32'hFFFFFFFF;
73      end
74      for (i = 17; i < 33; i = i + 1) begin
75          ram[i] = 32'h00000001;
76      end
77
78      for (i = 33; i < 49; i = i + 1) begin
79          ram[i] = 32'h7FFFFFFF;
80      end
81  end
```

5) عدد -1 را در a می ریزیم.
6) عدد -1 را در b می ریزیم.
7) با ضرب کردن a * b داریم ضرب -1 در -1 را انجام می دهیم که حاصل آن برابر با 1 می شود و در عکس خروجی ها که در پایان قرار می دهم این حاصل قابل مشاهده هست.
8) عدد 1 را در b لود می کنیم.
9) ضرب -1 در 1 را انجام می دهیم که حاصل آن برابر با -1 می شود.
10) جمع 1 با -1 را انجام می دهیم که حاصل آن برابر با 0 می شود.
11) بزرگترین عدد مثبت را در a لود می کنیم.
12) آن را با 1 جمع می کنیم که جواب باید 8FFFFFF شود.
13) آن را در 1 ضرب می کنیم که جواب باید با خود آن (7FFFFFFF) برابر شود.

با تمام شدن edge caseها و random case ها تست های ما stop می شوند.

عکس خروجی ها در تمامی مراحل:




هر کدام از مرحله های بالا 2 ثانیه طول می کشند که برای مشاهده نتیجه آن ها می توان از عکس بالا کمک گرفت.