به نام خدا

فاطمه حمدی (۴۰۱۱۰۵۸۴۸)

برای طراحی این مدار، آن را به اجزای کوچکتری تقسیم می کنیم، از صحت عملکرد هر قسمت اطمینان حاصل می کنیم و در آخر همه جزییات را در کنار هم میچینیم تا به مدار نهایی برسیم.

ALU :

برای واحد محاسبات ماژول جداگانه طراحی می کنیم:

```
module alu (
    input reg [1:0]operation,
    input reg signed [511:0] A,
    input reg signed [511:0] B,
    output reg signed [511:0] C,
    output reg signed [511:0] D
);
reg signed [1023:0] E;
always @(A or B or operation) begin

    if (operation==10) begin
        E = A*B;
    end
    if (operation == 11) begin
        E = A + B;
    end
    C = E[511:0];
    D = E[1023:511];
end
endmodule
```

این ماژول 2 ورودی و 2 خروجی دارد که ورودی ها همان رجیستر های 1 و 2 هستند و خروجی ها رجیستر های 3 و 4. با توجه به opcode عملیات انجام شده و پاسخ در E نوشته می شود. بیت های پرارزش E در رجیستر 4 و بیت های کم ارزش آن در رجیستر 3 ذخیره می شوند. برای تست ، از آنجایی که اعداد 511 بیتی بسیار بزرگ هستند، با اعداد 4 بیتی آن را چک می کنیم. (کد تست در انتهای فایل کامنت شده است)

```
#  -8+  7=  -1
#  -8*  7= -56
#   6+ -5=   1
#   6* -5= -30
#  -8+ -8= -16
#  -8* -8=  64
#   3+  0=   3
#   3*  0=   0
```

همانطور که مشاهده می شود، عملیات اعداد با علامت های منفی و مثبت و همچنین اورفلو به درستی هندل شده اند.

حال که نحوه انجام عملیات ضرب و جمع توضیح داده شد، به شرح عملیات ذخیره و بارگزاری می پردازیم.

بارگزاری:

```
tmp = 512'b0;
for (int i = 15; i >= 0; i = i - 1) begin
    if (i+entry1<=511) begin
        tmp = {tmp[479:0], mem[i+entry1]};
    end
end
registers[entry2] = tmp;
```

tmp یک متغیر 512 بیتی است که برای ذخیره و بارگزاری از آن کمک می گیریم. در ابتدا این متغیر را برابر 0 قرار می دهیم و پس در 32 بیت کم ارزش یکی از خانه های حافظه را قرار می دهیم. سپس در حلقه for هر بار آن را 32 بیت به سمت چپ شیفت می دهیم و یک خانه جدید در tmp بارگزاری می کنیم. بعد از 16 دور، 16 عدد 32 بیتی از حافظه درون tmp بارگزاری شده است. در نهایت با انجام concatenate ، محتویات tmp به این شکل خواهد بود:

tmp = {mem[entry1+15], mem [entry1+14],..., mem[entry1+1], mem[entry1] }

entry در واقع شماره ایندکسی از حافظه است که به عنوان ورودی می دهیم. مسلما اگر entry1 عددی باشد که تا 15 تا بعد آن را در حافظه نداشته باشیم، (ایندکس بزرگتر از 511 می شود) باید شرطی داشته باشیم که دیگر بارگزاری از حافظه انجام نشود. در این حالت بیت های دست نخورده tmp صفر باقی می مانند. پس از مقدار دهی کامل tmp آن را در رجیستر با ایندکس entry2 ذخیره می کنیم.

ذخیره سازی:

```
tmp = registers[entry2];
for (int i = 0; i < 16; i = i + 1) begin
```

```
                    if (i+entry1<=511) begin
                        mem[i + entry1] = tmp[i * 32 +: 32];
                    end
                end
```

برای ذخیره سازی، باید خلاف عملیات بارگزاری عمل کنیم. در ابتدا مقدار رجیستر مورد نظر که ایندکس آن در entry2 است را در tmp ذخیره می کنیم تا این متغیر را 32 بیت 32 بیت جدا کنیم و در خانه های حافظه ذخیره سازی کنیم. از ایندکس entry1 حافظه شروع کرده و تا ایندکس entry1+15 ادامه می دهیم. مسلما باید تا جایی باید ادامه بدهیم که ایندکس حافظه که میخواهیم بخشی از tmp را در آن ذخیره کنیم، از 511 بیشتر نشود. در پایان مقداردهی حافظه به این حالت خواهد بود:

mem[entry1+15] = temp[511:479]

...

mem[entry1+1] = temp[63:32]

mem[entry1] = temp[31:0]

مدار نهایی:

```
module processor (
input reg clk,
    input reg [1:0] opcode,
    input reg [9:0] entry1,
    input reg [1:0] entry2
    );
    reg we;
    reg signed [511:0] tmp;
    reg signed [31:0] mem [511:0];
    reg signed [511:0] registers [3:0];
    alu alu(opcode, registers[0], registers[1], registers[2], registers[3]);

    localparam [1:0]
            load = 2'b00, store = 2'b01, mul = 2'b10, sum = 2'b11;
```

ورودی و خروجی های مدار را مشاهده می کنیم. با entry1 ایندکس حافظه و entry2 ایندکس رجیستر ها را ورودی می گیریم، همچنین با ورودی opcode تعیین می کنیم چه عملیاتی انجام شود. رجیستر ها ، حافظه، متغیر tmp و حالت ها (برای اینکه بدانیم چه عملیاتی انجام می گیرد با توجه به ورودیopcode) را نیز تعریف می کنیم. از آنجایی

که میخواهیم مدار را در شرایط مختلف تست کنیم، با توجه به هر تست رجیستر ها و حافظه را مقدار دهی اولیه می کنیم تا عملکرد مدار را با توجه به آن مقادیر اولیه بسنجیم.

```
if (entry1<0 || entry2 < 0 || entry2 >3 || entry1 > 511) begin
            $display("invalid index");
        end
```

این کد عملکرد مدار را در حالتی که ایندکس ها نامعتبر هستند، متوقف می کند.

تست مدار:

عملیات جمع و ضرب:

عملیات ضرب: opcode =2

عملیات جمع: opcode =3

Test bench

```
always begin
    #1 clk = !clk;
end
initial begin
    clk=1;
    opcode = 3;
    # 10
    opcode = 2;
```

مقدار دهی اولیه رجیستر ها (رندم)

```
initial begin
        registers[0] = -10;
        registers[1] = 2345;
        registers[2] = -511'd20938447598233;
        registers[3] = 512'hFFFFFFFFFFFFFFFF;
    end
```

+ به معنای این است که مقادیر رجیستر ها را بعد از عملیات جمع نشان می دهد و * به این معناست که مقادیر را بعد از عملیات ضرب نشان می دهد.

```
+ register[0] = -10
+ register[1] = 2345
+ register[2] = 2335
+ register[3] = 0
```

```
* register[0] = -10
* register[1] = 2345
* register[2] = -23450
* register[3] = -1
```

همانطور که مشاهده می شود، برای هر دو عملیات، جواب های درست داریم.

مثالی دیگر

```
initial begin
        registers[0] = -1024;
        registers[1] = -2048;
        registers[2] = -511'hABCD;
        registers[3] = 512'd29384758923;
    end
```

```
+ register[0] = -1024
+ register[1] = -2048
+ register[2] = -3072
+ register[3] = -1

* register[0] = -1024
* register[1] = -2048
* register[2] = 2097152
* register[3] = 0
```

جواب ها کاملا درست است. بالا تر، از درستی عملکرد ALU مطمئن شدیم، حال مشاهده کردیم عملکرد مدار برای عملیات ضرب و جمع روی رجیستر ها و مقدار دهی مناسب آن ها، درست است.

عملیات بارگزاری:

برای تست بارگزاری و ذخیره سازی، اعداد را به صورت هگزادسیمال نشان می دهیم تا با تعداد بیت بالا، خواناتر باشند. حافظه را با توان هایی از 16 مقدار دهی می کنیم (تا خواندن و دیدن درستی مقدار رجیستر راحت تر باشد) :

```
        mem[5] = 0;
        mem[6] = 0;
        mem[7] = 23'd16;
        mem[8] = 23'd16;
        for ( int i = 0 ; i<12 ;i=i+2 ) begin
            mem[9+i] =  mem[8+i] * 16;
            mem[9+i+1] =  mem[8+i] * 16;
            $display("%d",i);
        end
```

حال با test bench زیر، آن را تست می کنیم.

```
initial begin
    clk=1;
    opcode = 0;
    entry1 = 5;
    entry2 = 2;
end
```

مقادیر حافظه از entry1 تا entry1+15 و رجیستر با ایندکسentry2 را پرینت می کنیم:

```
# register[2] = 10000000100000000100000001000000001000000010000000010000000100000001000000010000000100000001000000010000000100000001000000010000000000000000
# mem[5] = 0
# mem[6] = 0
# mem[7] = 10
# mem[8] = 10
# mem[9] = 100
# mem[10] = 100
# mem[11] = 1000
# mem[12] = 1000
# mem[13] = 10000
# mem[14] = 10000
# mem[15] = 100000
# mem[16] = 100000
# mem[17] = 1000000
# mem[18] = 1000000
# mem[19] = 10000000
# mem[20] = 10000000
```

یک تست دیگر برای حالت خاص انجام می دهیم:

```
for ( int i = 0 ; i<512 ;i++ ) begin
            mem[i] =  1;
        end
```

به همه خانه های حافظه مقدار 1 می دهیم.

```
opcode = 0;
    entry1 = 500;
    entry2 = 1;
```

ایندکسی که به حافظه می دهیم طوری است که اگر آن را به علاوه 15 کنیم، (515) از تعداد خانه های حافظه بیشتر است، طراحی مدار طوری است که انتظار داریم تعدادی از بیت های سمت چپ رجیستری که در آن load می کنیم، صفر باشد.

```
# register[1] = 00000000000000000000000000000000000000010000000100000001000000010000000100000001000000010000000100000001000000010000000100000001000000010000000100000001
# mem[500] = 1
# mem[501] = 1
# mem[502] = 1
# mem[503] = 1
# mem[504] = 1
# mem[505] = 1
# mem[506] = 1
# mem[507] = 1
# mem[508] = 1
# mem[509] = 1
# mem[510] = 1
# mem[511] = 1
```

همانطور که مشاهده کردیم بارگزاری در حالت کلی و برای نقاط مرزی درست کار می کند.

عملیات ذخیره سازی:

حالت برعکس تست اول بارگزاری را برای این حالت با تست و مقداردهی اولیه زیر انجام می دهیم:

```
    opcode = 1;
    entry1 = 400;
    entry2 = 3;
```

```
        registers[entry2] =
512'h1000000010000000010000000100000000100000001000000001000000010000000010000000
1000000001000000010000000010000000100000000000000000;
```

```
mem[400] = 0
mem[401] = 0
mem[402] = 10
mem[403] = 10
mem[404] = 100
mem[405] = 100
mem[406] = 1000
mem[407] = 1000
mem[408] = 10000
mem[409] = 10000
mem[410] = 100000
mem[411] = 100000
mem[412] = 1000000
mem[413] = 1000000
mem[414] = 10000000
mem[415] = 10000000
register[3] = 10000000100000000100000001000000001000000010000000010000000100000000100000001000000001000000010000000010000000100000000000000000
```

اکنون یک تست برای حالت مرزی انجام می دهیم.

```
    opcode = 1;
    entry1 = 511;
    entry2 = 0;
```

مقدار دهی رجیستر 1

```
registers[entry2] = -1;
```

انتظار داریم که مقدار 1- ذخیره شده در رجیستر1، فقط در ایندکس 511 حافظه ذخیره شود.

```
# mem[511] = ffffffff
# register[0] = ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
```

جمع بندی:

با توجه به تست ها و توضیحات دیدیم که مدار همه انتظارات ما را برآورده کرد، پس برای هر مقدار دیگری نیز درست است. در فایل ماژول و test bench آن، مقدار دهی ها به ازای تست های مختلف، مشخص و کامنت شده اند.