میان ترم درس طراحی سیستم های دیجیتال

نام و نام خانوادگی: صادق محمدیان

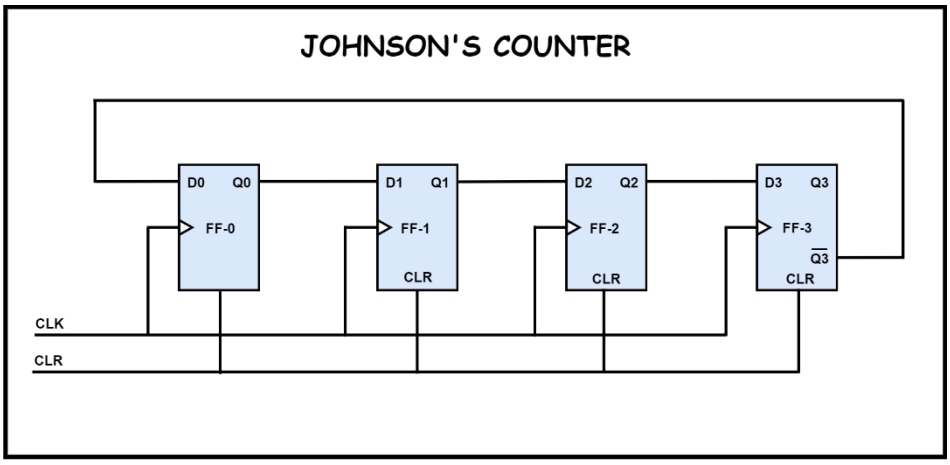
شماره دانشجویی:۴۰۱۱۰۹۴۷۷

نیم سال دوم تحصیلی ۱۴۰۲-۱۴۰۳

من از بین سوالات داده شده سوال ۶ و ۷ را انتخاب کرده ام.

سوال ششم:

در این سوال از ما خواسته شده تا یک شمارنده جانسون N بیتی طراحی کنیم و در طراحی آن از D-flip flop استفاده کنیم. شمای کلی این شمارنده برای N=4 به شکل زیر می باشد:

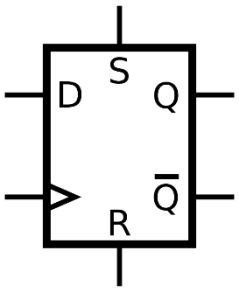


دنباله ای ک ه این شمارنده تولید می کند به صورت زیر می باشد:

$$0000 \rightarrow 0001 \rightarrow 0011 \rightarrow 0111 \rightarrow 1111$$

$$\rightarrow 1110 \rightarrow 1100 \rightarrow 1000 \rightarrow 0000 \rightarrow$$

از آنجایی که در این مدار قصد داریم به ازای Nهای مختلف این شمارنده را طراحی کنیم باید از ویژگی پارامتر در کد وریلاگ خود استفاده کنیم. ابتدا اقدام به طراحی یک D-flip flop می کنیم. جدول درستی یک D-flip flop به شکل زیر می باشد:



$\overline{SET}$	$\overline{RESET}$	D	CK	Q	$\overline{Q}$
0	1	-	-	1	0
1	0	-	-	0	1
0	0	-	-	1	1
1	1	1	↓	1	0
1	1	0	↓	0	1

در این فلیپ فلاپ ورودی های reset و set به صورت آسنکرون هستند.

کد وریلاگ زیر طراحی یک D-flip flopمی باشد.

```
module DFF(input clk , set , reset , d ,
           output reg q);
always @(posedge clk or posedge reset or posedge set)
begin
    if (reset)
    begin
        q <= 1'b0;
    end
    else if(set)begin
        q <= 1'b1;
    end
    else begin
        q <= d;
    end
end

endmodule
```

در این طراحی سیگنال های set و reset را active highدر نظر میگیریم یعنی در صورتی که یک باشند عمل مربوطه انجام خواهد شد و آسنکرون هم هستند. حال به سراغ طراحی شمارنده خود می رویم. آن را در یک ماژول دیگر طراحی می کنیم . در طراحی این شمارنده توجه داریم که ورودی D فلاپ فلاپ اول مانند دیگر فلیپ فلاپ ها نیست و ورودی آن $\tilde{Q}$ می باشد.

کد وریلاگ زیر طراحی این شمارنده می باشد.

```
module jCounter #(parameter N = 4)(input clk , set , reset ,
                                   output [N-1 : 0] out);

        DFF dff (clk , set , reset , ~out[N-1] , out[0]);

        genvar j;
        generate

            for(j = 1; j < N; j=j+1)
            begin

                DFF dffs (clk , set , reset , out[j-1] , out[j] );
            end

        endgenerate

endmodule
```

در توضیح این ماژول می تولن گفت که در بلاک generate از فلیپ فلاپ ها اینستس گرفته ایم و خروجی های q هرکدام را به ورودی های فلیپ فلاپ های بعدی وصل کرده ایم البته خروجی q` فلیپ فلاپ آخری را به ورودی فلیپ فلاپ اول وصل کرده ایم .

حال باید برای حالت های مختلف N تست بنچ یا ماژول تحریک بنویسیم.

برای شبیه سازی کلاک از تکه کد زیر استفاده میکنیم که کلاک با دوره تناوب ۱۰ واحد زمانی ایجاد می کند .

برای حالت N=4:

کد وریلاگ زیر برای تست در حالت N=4 می باشد.

```
module N4TB;

    parameter N=4;
    reg clk , set , reset;
    wire [N-1 : 0] out;

    initial
        clk = 0;
    always #5 clk = ~clk;

    jCounter #(.N(N)) johnsonCounter(clk , set , reset , out);

    initial
    begin
        set = 0;
        reset = 1;
        #5
        reset = 0;
        #100
        set = 1;
        #5
        set = 0;
        #100

        $stop();

    end

    initial
    begin
        $display("\t\tTime\tq");
        $monitor($time, "\t%b\t", out);
    end

  initial
  begin
    $dumpfile("TB4_VCD.vcd");
    $dumpvars;
  end
endmodule
```

برای ذخیره نتایج در فایل VCD تکه زیر را به آن ماژول تست اضافه شده است:

```
initial
begin
  $dumpfile("TB4_VCD.vcd");
  $dumpvars;
end
```

نتایج زیر بعد از شبیه سازی قابل مشاهده می باشد که درستی طراحی را تایید می کند.

Time	q
0	0000
5	0001
15	0011
25	0111
35	1111
45	1110
55	1100
65	1000
75	0000
85	0001
95	0011
105	1111
115	1110
125	1100
135	1000
145	0000
155	0001
165	0011
175	0111
185	1111
195	1110
205	1100

برای حالت N=8 :

کد وریلاگ زیر برای تست در حالت N=8 می باشد:

```
module N8TB;

    parameter N=8;
    reg clk , set , reset;
    wire [N-1 : 0] out;

    initial
        clk = 0;
    always #5 clk = ~clk;

    jCounter #(.N(N)) johnsonCounter(clk , set , reset , out);

    initial
    begin
        set = 0;
        reset = 1;
        #5
        reset = 0;
        #200
        set = 1;
        #5
        set = 0;
        #200

        $stop();

    end

    initial
    begin
        $display("\t\tTime\tq");
        $monitor($time, "\t%b\t", out);
    end
    initial
    begin
    $dumpfile("TB8_VCD.vcd");
    $dumpvars;
    end
endmodule
```

برای ذخیره نتایج در فایل VCD تکه زیر را به آن ماژول تست اضافه شده است:

```
initial
begin
  $dumpfile("TB8_VCD.vcd");
  $dumpvars;
end
```

نتایج زیر بعد از شبیه سازی قابل مشاهده می باشد که درستی طراحی را تایید می کند.

```
Time     q
     0 00000000
     5 00000001
    15 00000011
    25 00000111
    35 00001111
    45 00011111
    55 00111111
    65 01111111
    75 11111111
    85 11111110
    95 11111100
   105 11111000
   115 11110000
   125 11100000
   135 11000000
   145 10000000
   155 00000000
   165 00000001
   175 00000011
   185 00000111
   195 00001111
   205 11111111
   215 11111110
   225 11111100
   235 11111000
   245 11110000
   255 11100000
   265 11000000
   275 10000000
   165 00000001
   175 00000011
   185 00000111
   195 00001111
   205 11111111
   215 11111110
   225 11111100
   235 11111000
   245 11110000
   255 11100000
   265 11000000
   275 10000000
   285 00000000
   295 00000001
   305 00000011
   315 00000111
   325 00001111
   335 00011111
   345 00111111
   355 01111111
   365 11111111
   375 11111110
   385 11111100
   395 11111000
   405 11110000
```

برای حالت N=16 :

کد وریلاگ زیر برای تست در حالت N=16 می باشد:

```
module N16TB;

    parameter N=16;
    reg clk , set , reset;
    wire [N-1 : 0] out;

    initial
        clk = 0;
    always #5 clk = ~clk;

    jCounter #(.N(N)) johnsonCounter(clk , set , reset , out);

    initial
    begin
        set = 0;
        reset = 1;
        #5
        reset = 0;
        #400
        set = 1;
        #5
        set = 0;
        #400

        $stop();

    end

    initial
    begin
        $display("\t\tTime\tq");
        $monitor($time, "\t%b\t", out);
    end
    initial
    begin
    $dumpfile("TB16_VCD.vcd");
    $dumpvars;
    end
endmodule
```

برای ذخیره نتایج در فایل VCD تکه زیر را به آن ماژول تست اضافه شده است:

```
initial
begin
  $dumpfile("TB16_VCD.vcd");
  $dumpvars;
end
```

نتایج زیر بعد از شبیه سازی قابل مشاهده می باشد که درستی طراحی را تایید می کند.

```
Time     q
   0 0000000000000000
   5 0000000000000001
  15 0000000000000011
  25 0000000000000111
  35 0000000000001111
  45 0000000000011111
  55 0000000000111111
  65 0000000001111111
  75 0000000011111111
  85 0000000111111111
  95 0000001111111111
 105 0000011111111111
 115 0000111111111111
 125 0001111111111111
 135 0011111111111111
 145 0111111111111111
 155 1111111111111111
 165 1111111111111110
 175 1111111111111100
 185 1111111111111000
 195 1111111111110000
 205 1111111111100000
 215 1111111111000000
 225 1111111110000000
 235 1111111100000000
 245 1111111000000000
 255 1111110000000000
 265 1111100000000000
 275 1111000000000000
 285 1110000000000000
 295 1100000000000000
 305 1000000000000000
 315 0000000000000000
 325 0000000000000001
 335 0000000000000011
 345 0000000000000111
 355 0000000000001111
 365 0000000000011111
 375 0000000000111111
 385 0000000001111111
 395 0000000011111111
 405 1111111111111111
 415 1111111111111110
 425 1111111111111100
 435 1111111111111000
 445 1111111111110000
 455 1111111111100000
 465 1111111111000000
 475 1111111110000000
 485 1111111100000000
 495 1111111000000000
 505 1111110000000000
 515 1111100000000000
 525 1111000000000000
 535 1110000000000000
 545 1100000000000000
 555 1000000000000000
 565 0000000000000000
 575 0000000000000001
 585 0000000000000011
 595 0000000000000111
 605 0000000000001111
 615 0000000000011111
 625 0000000000111111
 635 0000000001111111
 645 0000000011111111
 655 0000000111111111
 665 0000001111111111
 675 0000011111111111
 685 0000111111111111
 695 0001111111111111
 705 0011111111111111
 715 0111111111111111
 725 1111111111111111
 735 1111111111111110
```

برای حالت N=32 :

کد وریلاگ زیر برای تست در حالت N=32 می باشد:

```
module N32TB;

    parameter N=32;
    reg clk , set , reset;
    wire [N-1 : 0] out;

    initial
        clk = 0;
    always #5 clk = ~clk;

    jCounter #(.N(N)) johnsonCounter(clk , set , reset , out);

    initial
    begin
        set = 0;
        reset = 1;
        #5
        reset = 0;
        #800
        set = 1;
        #5
        set = 0;
        #800

        $stop();

    end

    initial
    begin
        $display("\t\tTime\tq");
        $monitor($time, "\t%b\t", out);
    end
    initial
    begin
    $dumpfile("TB32_VCD.vcd");
    $dumpvars;
    end
endmodule
```

برای ذخیره نتایج در فایل VCD تکه زیر را به آن ماژول تست اضافه شده است:

```
initial
begin
  $dumpfile("TB32_VCD.vcd");
  $dumpvars;
end
```

نتایج زیر بعد از شبیه س ازی قابل مشاهده می باشد که درستی طراحی را تایید می کند.

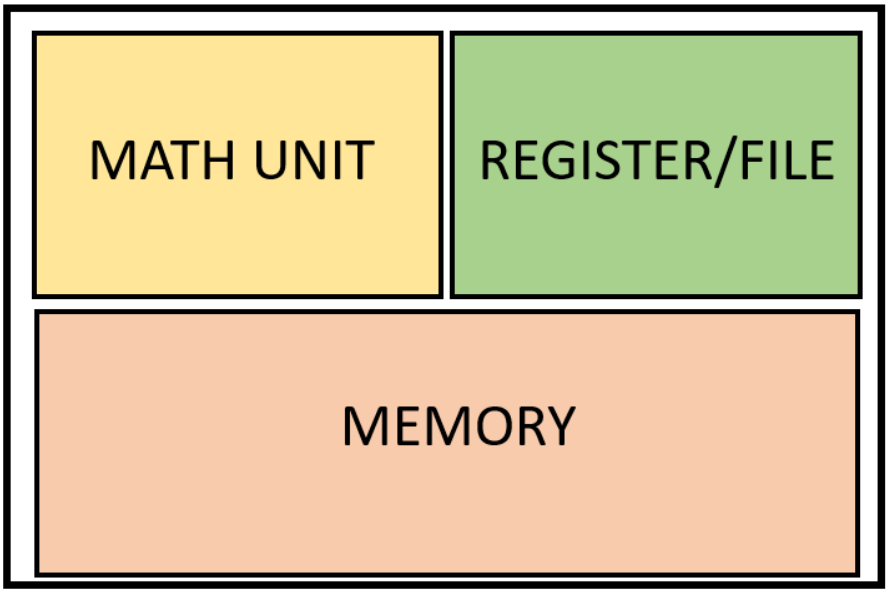
```
Time     q
      0 00000000000000000000000000000000
      5 00000000000000000000000000000001
     15 00000000000000000000000000000011
     25 00000000000000000000000000000111
     35 00000000000000000000000000001111
     45 00000000000000000000000000011111
     55 00000000000000000000000000111111
     65 00000000000000000000000001111111
     75 00000000000000000000000011111111
     85 00000000000000000000000111111111
     95 00000000000000000000001111111111
    105 00000000000000000000011111111111
    115 00000000000000000000111111111111
    125 00000000000000000001111111111111
    135 00000000000000000011111111111111
    145 00000000000000000111111111111111
    155 00000000000000001111111111111111
    165 00000000000000011111111111111111
    175 00000000000000111111111111111111
    185 00000000000001111111111111111111
    195 00000000000011111111111111111111
    205 00000000000111111111111111111111
    215 00000000001111111111111111111111
    225 00000000011111111111111111111111
    235 00000000111111111111111111111111
    245 00000001111111111111111111111111
    255 00000011111111111111111111111111
    265 00000111111111111111111111111111
    275 00001111111111111111111111111111
    285 00011111111111111111111111111111
    295 00111111111111111111111111111111
    305 01111111111111111111111111111111
    315 11111111111111111111111111111111
    325 11111111111111111111111111111110
    335 11111111111111111111111111111100
    345 11111111111111111111111111111000
    355 11111111111111111111111111110000
    365 11111111111111111111111111100000
    375 11111111111111111111111111000000
    385 11111111111111111111111110000000
    395 11111111111111111111111100000000
    405 11111111111111111111111000000000
    415 11111111111111111111110000000000
    425 11111111111111111111100000000000
    435 11111111111111111111000000000000
    445 11111111111111111110000000000000
    455 11111111111111111100000000000000
    465 11111111111111111000000000000000
    475 11111111111111110000000000000000
    485 11111111111111100000000000000000
    495 11111111111111000000000000000000
    505 11111111111110000000000000000000
    515 11111111111100000000000000000000
    525 11111111111000000000000000000000
    535 11111111110000000000000000000000
    545 11111111100000000000000000000000
    555 11111111000000000000000000000000
    565 11111110000000000000000000000000
    575 11111100000000000000000000000000
    585 11111000000000000000000000000000
    595 11110000000000000000000000000000
    605 11100000000000000000000000000000
    615 11000000000000000000000000000000
    625 10000000000000000000000000000000
    635 00000000000000000000000000000000
    645 00000000000000000000000000000001
    655 00000000000000000000000000000011
    665 00000000000000000000000000000111
    675 00000000000000000000000000001111
    685 00000000000000000000000000011111
    695 00000000000000000000000000111111
    705 00000000000000000000000001111111
    715 00000000000000000000000011111111
    725 00000000000000000000000111111111
    735 00000000000000000000001111111111
    745 00000000000000000000011111111111
    755 00000000000000000000111111111111
    765 00000000000000000001111111111111
```

سوال هفتم:

در این سوال قصد داریم که پردازنده طراحی کنیم که شامل ۳ بخش زیر می باشد:

۱-رجیستر فایل ۲-واحد محاسبات ۳-حافظه

در ابتدا هر یک از این سه بخش را به طور جداگانه طراحی می کنیم و سپس در یک ماژول نهایی از هر سه اینستنس میگیریم و کارکرده نهایی پردازنده را پیاده سازی می کنیم.



واحد محاسبات:

این واحد برای انجام محاسبات و کارهای منطقی است این واحد باید بتواند دو عملیات ضرب و جمع را انجام دهد .لذا باید نوع عملیاتی که باید انجام شود و دو عدد را به آن به عنوان ورودی بدهیم. و یک خروجی برای آن در نظر بگیریم .البته باید توجه داشته باشیم چون عملیات ضرب این قسمت انجام می دهد و حاصل دو عدد nبیتی یک عدد 2n بیتی خواهد بود و خروجی باید اندازه دوبرابر ورودی ها داشته باشد.

کد وریلاگ زیر طراحی این بخش از پردازنده می باشد. اگر instr که ورودی می دهیم برابر با صفر باشد عملیات جمع و اگر یک باشد عملیات ضرب را انجام میدهد.

اگر instr برابر با صفر باشد در هر مرحله از حلقه، عمل جمع دو عدد ۳۲ بیتی از ورودی (input1 و input2) انجام می‌شود و نتیجه در regout ذخیره می‌شود. عمل جمع اینجا با استفاده از signed$ انجام می‌شود که جمع عددهای با علامت را انجام می‌دهد. همچنین j * 64 +: 64 به این معنا است که نتایج جمع در بازه‌های ۶۴ بیتی مختلف (j=0 تا j=15) در regout ذخیره شوند.

اگر instr برابر با یک باشد در هر مرحله از حلقه، عمل ضرب دو عدد ۳۲ بیتی از ورودی (input1 و input2) انجام می‌شود و نتیجه در regout ذخیره می‌شود. عمل ضرب اینجا با استفاده از signed$ انجام می‌شود که عمل ضرب عددهای با علامت را انجام می‌دهد. همچنین i * 64 +: 64 به این معنا است که نتایج ضرب در بازه‌های ۶۴ بیتی مختلف (i=0 تا i=15) در regout ذخیره می‌شوند.

```
module mathUnit(input [511 : 0] input1 , [511 : 0] input2 , instr ,
                output signed [1023 : 0] out
                );

    reg [1023 : 0] regout;
    integer i , j;

    always @(*)
    begin
        //sum instruction
        if(instr == 1'b0)
        begin
            for (j = 0; j < 16; j = j + 1)
            begin
            regout[j * 64 +: 64] <= $signed(input1[j * 32 +: 32]) +
$signed(input2[j * 32 +: 32]);
            end
        end
        //multiply instruction
        else if (instr == 1'b1)
        begin
            for (i = 0; i < 16; i = i + 1)
```

```
            begin
            regout[i * 64 +: 64] <= $signed(input1[i * 32 +: 32]) *
$signed(input2[i * 32 +: 32]);
            end

        end

    end
    assign out = regout;

endmodule
```

واحد register file:

این واحد باید توانایی ذخیره ۴ آرایه ی ۵۱۲ بیتی با نام های A1تاA4 را داشته باشد. برای آدرس دهی هریک از آنها به دو بیت برای آدرس دهی نیاز داریم و آنها را طبق جدول زیر آدرس دهی میکنیم:

A1	00
A2	01
A3	10
A4	11

این بخش باید توانایی خواندن و نوشتن را داشته باشد پس به آن سیگنال های نوشتن را به عنوان ورودی برای آن ایجاد می کنیم.

همچنین سیگنال ریست را نیز برای آن در نظر می گیریم که در صورت فعال شدن تمامی ثبات ها صفر می شوند و سیگنال ست را نیز برای آن در نظر می گیریم که در صورت فعال شدن تمامی ثبات ها یک می شوند.

طراحی را به گونه ای انجام میدهیم که رجیستر فایل ها همواره در دسترس دیگر بخش ها باشند.

دو درگاه نوشتن داده نیز برای این بخش در نظر میگیریم که نیاز به دو سیگنال فعال سازی می باشند همانطور که قابل مشاهده است با توجه به انتظارات که از طراحی خود داریم تعداد ورودی ها و خروجی های این ماژول زیاد می باشد.

کد وریلاگ زیر طراحی این بخش از پردازنده می باشد:

```
module register (
        input clk , reset , set ,
        input [511 : 0] input1 , [511 : 0] input2,
        input [1 : 0] wAdd1 , [1 : 0] wAdd2,
        input wEnable1 , wEnable2 ,[1 : 0] rAdd,
        output signed [511 : 0] out,
        output signed [511 : 0] A1 , signed [511 : 0] A2 , signed [511 : 0] A3 ,
signed [511 : 0] A4
    );


        // 512*4 registers!
    reg signed [511 : 0] registers [0 : 3];

    integer i,j;
    always @(posedge clk or negedge reset or negedge set)
    begin
        //reset signal is enabled!
        if(!reset)
        begin
            for (j = 0; j < 16; j = j + 1)
            begin
            registers[j] = 512'b0;
            end
        end
        //set signal is enabled!
        else if(!set)
        begin
            for (i = 0; i < 16; i = i + 1)
            begin
            registers[j] = 512'b1;
            end
        end
        //write signal is enabled!
        else
        begin
            if (wEnable1)
            begin
            registers[wAdd1] <= $signed(input1);
            end
            if (wEnable2)
            begin
            registers[wAdd2] <= $signed(input2);
```

```
            end
        end
    end

    assign A1 = registers[0];
    assign A2 = registers[1];
    assign A3 = registers[2];
    assign A4 = registers[3];

    assign out = registers[rAdd];

endmodule
```

واحد memory:

چون حافظه دارای ۵۱۲ خانه می باشد پس برای ادرس دهی به ۹ بیت نیاز داریم.

$$lg\ 512 = 9$$

برای حافظه دو سیگنال set و reset را قرار می دهیم که در صورتی که فعال شوند(active low) تمام خانه های حافظه را به ترتیب ۱و۰ می کنند . همچنین برای نوشتن در حافظه یک سیگنال فعالسازی قرار می دهیم و با فعال کردن آن در حافظه می نویسیم.

طراحی که انجام داده ایم به اینگونه می باشد که نوشتن و خواندن در حافظه با گرفتن یک ادرس پایه و دادن آن به عنوان ورودی به memory داده می شود و خروجی memoryیک بردار شامل ۱۶ عدد ۳۲ بیتی علامت دار است که ۱۶ خانه ی متوالی از حافظه با شروع از آدرس پایه می باشند. در طراحی خود در اول کار خانه های حافظه را مقدار دهی اولیه میکنیم که در فایل Initialldata.txtمی باشند و با دستور زیر مقدار دهی اولیه می شود.

```
    initial
    begin
        $readmemh("Initialldata.txt",data_memory)
    end
```

کد وریلاگ زیر طراحی قسمت memory unit می باشد:

```
module memory (
        input clk , reset , set ,
        input signed [511 : 0] input1 , [8 : 0] address , wEnable,
        output signed [511 : 0] out
    );

    reg signed [31 : 0] mmemory [0 : 511] , signed [511 : 0] MemOut;
    integer i , j , k , l;
```

```
    initial
    begin
        $readmemh("Initialldata.txt",mmemory)
    end

    always @(posedge clk or negedge reset or negedge set)
    begin
        if(!reset)
        begin
            for (k = 0 ; k < 32 ; k = k + 1)
            begin
                mmemory[k] <= 32'h0;
            end
        end

        else if(!set)
        begin
            for (l = 0 ; l < 32 ; l = l + 1)
            begin
                mmemory[l] <= 32'h1;
            end
        end

        else
        begin
            if (wEnable)
            begin
                for (i = 0; i < 16; i = i + 1)
                begin
                    mmemory[(i + maddress) % 512] <= $signed(input1[32 * i +:
32]);
                end
            end
        end
    end

    always @(*) begin
        for (j = 0; j < 16; j = j + 1)
        begin
            MemOut[32 * j +: 32] = $signed(mmemory[(j + address) % 512]);
        end
    end

    assign out = MemOut;
endmodule
```

حال ما تمامی بخش های پردازنده موردنظر را طراحی کردیم و باید با اتصال آنها به یکدیگر در ماژول اصلی خود یعنی processor پردازنده را طراحی کنیم.در ایم ماژول از هر ۳ ماژول قبل instanceمیگیریم.

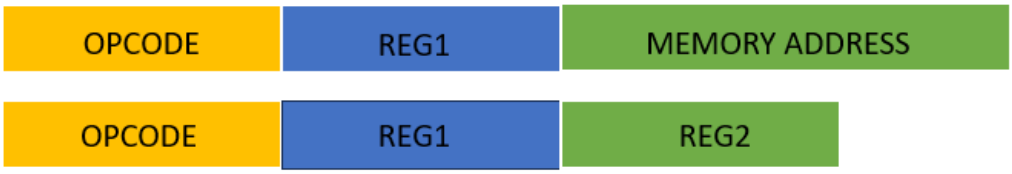
این پردازنده یک سری دستور را به فرم مشخصی مانند آنچه در ماشین هایی که در درس ساختار و زبان کامپیوتر خواندیم گرفته و آنها را اجرا می کند پس برای آنها یک فرم کلی ارایه میدهیم

باتوجه به اینکه این پردازنده باید چهار عمل گفته شده در صورت سوال را انجام دهد پس برای مقادیر opcode چهار مقدار مختلف داریم و برای مشخص کردن آن دو بیت در نظر می گیریم.

با توجه به اینکه چهار رجیستر داریم برای مشخص کردن رجیستر مورد نیز نیاز به دو بیت داریم.

اندازه حافظه ۵۱۲ خانه می باشد پس با $lg\ 512 = 9$ بیت قابل آدرس دهی می باشد.

پس فرم کلی دستورات را میتوان به شکل زیر در نظر گرفت:



برای دستور loadکردن ابتدا سیگنال نوشتن رجیستر و شماره ۲حافظه را صفر می کنیم و سیگنال ۱حافظه را فعال میشود سپس بیت ۹و ۱۰دستور در ادرس نوشتن اول قرار میگیرد و بیت های ۰تا۸ دستور در آدرس مموری قرار می گیرد.

برای دستور store کردن سیگنال فعال سازی نوشتن ۱و۲ غیرفعال می شوند سپس سیگنال نوشتن حافظه فعال می شود سپس بیت های ۹و ۱۰ دستور در آدرس خواندن رجیستر قرار می گیرد و در آدرسی که از ۰تا ۸ مشخص می شود ریخته می شود.

برای انجام عمل جمع سیگنال نوشتن مموری صفر می شود و سیگنال های نوشتن رجیستر یک می شوند و خروجی عمل نیز درون ثبات ۳و۴ ریخته می شود همچنین instr مربوط به جمع به sumunitداده می شود تا عملیات جمع را انجام دهد.

برای انجام عمل ضرب سیگنال نوشتن مموری صفر می شود و سیگنال های نوشتن رجیستر یک می شوند و خروجی عمل نیز درون ثبات ۳و۴ ریخته می شود همچنین instr مربوط به ضرب به sumunitداده می شود تا عملیات جمع را انجام دهد.

خروجی های پردازنده هم بصورت آسنکرون آپدیت می شوند.

کد دستور های مختلف در جدول زیر قابل مشاهده می باشد:

Load	00
Store	01
Sum	10
multiply	11

طراحی ماژول اصلی به زبان وریلاگ به شکل زیر می باشد:

```
module processor(input clk , reset , set , [12 : 0] instruction ,
                 output signed [511 : 0] A1 , signed [511 : 0] A2 , signed [511 :
0] A3 , signed [511 : 0] A4);


    //ports for mathUnit
    reg [511 : 0] mathUnitInput1;
    reg [511 : 0] mathUnitInput2;
    reg instr;
    wire signed [1023 : 0] mathUnitOutput;
    //ports register
    reg [511 : 0] registerInput1;
    reg [511 : 0] registerInput2;
    reg [1 : 0] registerWriteAdd1;
    reg [1 : 0] registerWriteAdd2;
    reg registerWriteEnable1;
    reg registerWriteEnable2;
    reg [1 : 0] registerReadAdd;
    wire signed [511 : 0] registerOut;
    wire signed [511 : 0] registerA1;
    wire signed [511 : 0] registerA2;
    wire signed [511 : 0] registerA3;
    wire signed [511 : 0] registerA4;
    //ports for memory
```

```
    reg signed [511 : 0] memoryInput;
    reg [8 : 0] memoryAddress;
    reg memoryWriteEnable;
    wire signed [511 : 0] memoryOut;


    //instance from mathUnit
    mathUnit mathUnit (mathUnitInput1 , mathUnitInput2 , instr , mathUnitOutput);
    //instance from register
    register registerr (clk , reset , set , registerInput1 , registerInput2 ,
registerWriteAdd1 ,
                                    registerWriteAdd2 , registerWriteEnable1 ,
registerWriteEnable2 ,
                                    registerReadAdd , registerOut , registerA1 ,
registerA2 , registerA3 , registerA4);
    //instance from memory
    memory memoryy (clk , reset , set , memoryInput, memoryAddress,
memoryWriteEnable, memoryOut);

    integer j;

     always @(negedge clk)
     begin
        #5
        if(instruction[12 : 11] == 2'b00)
            begin
                memoryWriteEnable <= 0;
                memoryAddress <= instruction[8 : 0];
                registerWriteEnable1 <= 1;
                registerWriteEnable2 <= 0;
                registerWriteAdd1 <= instruction[10 : 9];
                #5
                registerInput1 <= memoryOut;
            end

        else if(instruction[12 : 11] == 2'b01)
            begin
                memoryWriteEnable <= 1;
                memoryAddress <= instruction[8 : 0];
                registerWriteEnable1 <= 0;
                registerWriteEnable2 <= 0;
                registerWriteAdd1 <= instruction[10 : 9];
                #5
                registerInput1 <= memoryOut;
            end
```

```
        else if(instruction[12 : 11] == 2'b10)
            begin
                memoryWriteEnable <= 0;
                registerWriteEnable1 <= 1;
                registerWriteEnable2 <= 1;
                registerWriteAdd1 <= 2'b10;
                registerWriteAdd2 <= 2'b11;
                instr = 1'b0;
                mathUnitInput1 <= registerA1;
                mathUnitInput2 <= registerA2;
                #5
                for(j = 0 ; j < 16 ; j = j + 1)
                begin
                    registerInput1[32 * j +: 32] <= mathUnitOutput[64 * j +: 32];
                    registerInput2[32 * j +: 32] <= mathUnitOutput[64 * j + 32 +:
32];
                end
            end

        else if(instruction[12 : 11] == 2'b11)
            begin
                memoryWriteEnable <= 0;
                registerWriteEnable1<= 1;
                registerWriteEnable1 <= 1;
                registerWriteAdd1 <= 2'b10;
                registerWriteAdd1  <= 2'b11;
                instr = 1'b1;
                mathUnitInput1 <= registerA1;
                mathUnitInput2 <= registerA2;
                #5
                for(j = 0; j < 16; j = j + 1)
                begin
                    registerInput1[32 * j +: 32] <= mathUnitOutput[64 * j +: 32];
                    registerInput2[32 * j +: 32] <= mathUnitOutput[64 * j + 32 +:
32];
                end
            end
    end



    assign A1 = registerA1;
    assign A2 = registerA2;
    assign A3 = registerA3;
```

```
    assign A4 = registerA4;


endmodule
```

حال پس از طراحی پردازنده خود باید آن را در حالت های مرزی تست کنیم:

حالت های مرزی می توانند اعداد 1,0,-1بزگترین عدد مثبت و کوچکترین عدد منفی باشند.

تمامی این حالات در مقدار دهی اولیه حافظه در حافظه وجود دارند.

کلاک را برابر ۸۰ واحد زمانی می گیریم تا مطمعن شویم از بیشینه تاخیر های موجود در مدار بیشتر است و این مورد در عملکرد مدار مشکلی ایجاد نمی کند.

ابتدا بدون دادن تست مدار را اجرا می کنیم که خروجی زیر حاصل می شود:

```
module test1;
    reg clk , reset , set ;
    reg [12 : 0] instruction;
    wire [511 : 0] A1 ;
    wire [511 : 0] A2 ;
    wire [511 : 0] A3 ;
    wire [511 : 0] A4;

    initial
        clk = 0;
    always
        #40 clk = ~clk;


    processor Processor (clk, reset , set , instruction , A1 , A2 , A3 , A4);
    initial
        $monitor($time , ":\nA1 = %h\nA2 = %h\nA3 = %h\nA4 = %h\n " , A1 , A2 ,
A3 , A4 );
endmodule
```

```
                     0:
A1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

حال تست زیر را انجام میدهیم که در صورت تطبیق با داده های اولیه مموری و دستور عمل ها متوجه می شویم که پردازنده به درستی کار می کند.

```
module test1;
    reg clk , reset , set ;
    reg [12 : 0] instruction;
    wire [511 : 0] A1 ;
    wire [511 : 0] A2 ;
    wire [511 : 0] A3 ;
    wire [511 : 0] A4;

    initial
        clk = 0;
    always
        #40 clk = ~clk;
    processor Processor (clk, reset , set , instruction , A1 , A2 , A3 , A4);

    initial
    begin
        instruction <= {2'b10, 11'b0};
        #80
        instruction <= {2'b11, 11'b0};
        #80
        instruction <= {4'b0000, 9'b0};
        #80
        instruction <= {4'b0001, 9'b10000};
        #80
        instruction <= {2'b10, 11'b0};
        #80
        instruction <= {2'b11, 11'b0};
        #80
        $stop();
    end
    initial
        $monitor($time , ":\nA1 = %h\nA2 = %h\nA3 = %h\nA4 = %h\n " , A1 , A2 ,
A3 , A4 );
endmodule
```

```
                   0:
A1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                 200:
A1 = 0000000f0000000e0000000d0000000c0000000b0000000a00000779000000078000006070006009800000000050200004400000000300000112000000001000000000
A2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                 280:
A1 = 0000000f0000000e0000000d0000000c0000000b0000000a00000779000000078000006070006009800000000050200004400000000300000112000000001000000000
A2 = 000000000000000100000002000000030000000400000005000000060000000700000008000000090000000a0000000b0000000c0000000d0000000e0000000f
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
module test1;
    reg clk , reset , set ;
    reg [12 : 0] instruction;
    wire [511 : 0] A1 ;
    wire [511 : 0] A2 ;
    wire [511 : 0] A3 ;
    wire [511 : 0] A4;

    initial
        clk = 0;
    always
        #40 clk = ~clk;


    processor Processor (clk, reset , set , instruction , A1 , A2 , A3 , A4);

    initial
    begin
        instruction <= {2'b10, 11'b0};
        #80
        instruction <= {2'b11, 11'b0};
        #80
        instruction <= {4'b0000, 9'b0};
        #80
        instruction <= {4'b0001, 9'b10000};
        #80
        instruction <= {2'b10, 11'b0};
        #80
        instruction <= {2'b11, 11'b0};
        #80
        instruction <= {4'b0110, 9'b111111};
        #80
        instruction <= {4'b0111, 9'b1111111};
        #80
        instruction <= {4'b0000, 9'b111111};
        #80
        instruction <= {4'b0001, 9'b1111111};
        #80
        instruction <= {2'b10, 11'b0};
        #80
        instruction <= {2'b11, 11'b0};
        #80
        $stop();
    end
    initial
```

```
        $monitor($time , ":\nA1 = %h\nA2 = %h\nA3 = %h\nA4 = %h\n " , A1 , A2 ,
A3 , A4 );
endmodule
```

```
                0:
A1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                120:
A1 = 0000000f0000000e0000000d0000000c0000000b0000000a00007790000000780000060700060098000000005020000440000000300000112000000010000000
A2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                360:
A1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

این تست سوم می باشد که نشان می دهد مدار ما به درستی کار می کند.

حال داده های مموری را که در فایل اولیه هستند تغییر می دهیم و دوباره اقدام به تست می کنیم. فایل داده های اولیه قبلی در پوشه سوال ۷ به نام initialldataold هنوز قرار دارد و فایل جدید initialldata جدید را با داده های جدید پر می کنیم و با شبیه سازی دوباره ماژول تست نتایج زیر را می بینیم

```
                0:
A1 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
A2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

                200:
A1 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
A2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                360:
A1 = 000000150000ffffffff00007fffffff000ff0000f0000f00f0000f080000000000000001000000010000000ff0000000f0000000ffffffff0000000000000000
A2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                440:
A1 = 000000150000ffffffff00007fffffff000ff0000f0000f00f0000f080000000000000001000000010000000ff0000000f0000000ffffffff0000000000000000
A2 = 000000020000ffff0000ffff7fffffff000ff000000ff0000f0000f0800000008000000000000000100000000f0000000ff000000ffffffffffffffff00000000
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                840:
A1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A2 = 000000020000ffff0000ffff7fffffff000ff000000ff0000f0000f0800000008000000000000000100000000f0000000ff000000ffffffffffffffff00000000
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                920:
A1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```