به نام خدا

جبرانی پایان‌ترم

درس: طراحی سیستم های دیجیتال

نویسنده: امیرکسری احمدی

شماره دانشجویی: ۴۰۱۱۷۰۵۰۷

استاد: مهندس فصحتی

سوال ۷ میان ترم در جبران پایان ترم

در این سوال می‌خواهیم یک پردازنده‌ی آرایه‌ای ۵۱۲ بیتی طراحی کنیم که دارای ۳ بخش زیر است:

۱) یک رجیسترفایل با قابلیت ذخیره سازی ۴ آرایه ۵۱۲ بیتی با نام های A1 تا A4
۲) یک واحد ریاضی که قابلیت ضرب و جمع را دارا باشد. ورودی های این واحد A1 و A2 و خروجی کم‌ارزش آن در A3 و پرارزش آن در A4 است.
۳) یک حافظه با عمق ۵۱۲ و عرض ۳۲ بیت. این پردازنده امکان بارگزاری/ ذخیره‌سازی ۱۶ خانه پشت‌سر هم از حافظه را دارا است.

ما برای پیاده سازی این پردازنده، نیاز به ۴ ماژول register_file، ALU، Memory و خود پردازنده داریم.

به ترتیب نحوه‌ی پیاده سازی هر یک را شرح می‌دهیم:

۱) Register File:

```
module RF (
    input clk,
    input reset,
    input [511:0] input_data_1,
    input [511:0] input_data_2,
    input [1:0] write_address_1,
    input [1:0] write_address_2,
    input write_enable_1,
    input write_enable_2,
    input [1:0] read_address,
    output signed [511:0] output_data,
    output signed [511:0] A1,
    output signed [511:0] A2,
    output signed [511:0] A3,
    output signed [511:0] A4
);

    // Register file array to hold 4 registers of 512 bits each
    reg signed [511:0] reg_file [0:3];

    // Assignments to output individual registers
    assign A1 = reg_file[0];
    assign A2 = reg_file[1];
    assign A3 = reg_file[2];
    assign A4 = reg_file[3];

    integer i;
    // Sequential block to handle reset and write operations
    always @(negedge clk or posedge reset) begin
        if (reset) begin
            // Reset all registers to 0
            for (i = 0; i < 4; i = i + 1) begin
                reg_file[i] <= 512'b0;
            end
        end else begin
            // Write data to registers if write enable signals are active
            if (write_enable_1) begin
                reg_file[write_address_1] <= input_data_1;
            end
            if (write_enable_2) begin
                reg_file[write_address_2] <= input_data_2;
            end
        end
    end

    // Output data from the selected register
    assign output_data = reg_file[read_address];

endmodule
```

در این ماژول، ما ۴ رجیستر را پیاده سازی می‌کنیم که می‌تواند به صورت همزمان توانایی خواندن این ۴ رجیستر را به ما بدهد و به صورت موازی روی دو رجیستر انتخابی بنویسد به این دلیل که پردازنده ما نیاز است پس از محاسبه جمع و تفریق ۵۱۲ بیتی، ۵۱۲ بیت اول را در ثبات سوم و ۵۱۲ بیت نهایی را در ثبات چهارم ریخت.

۲) ALU:

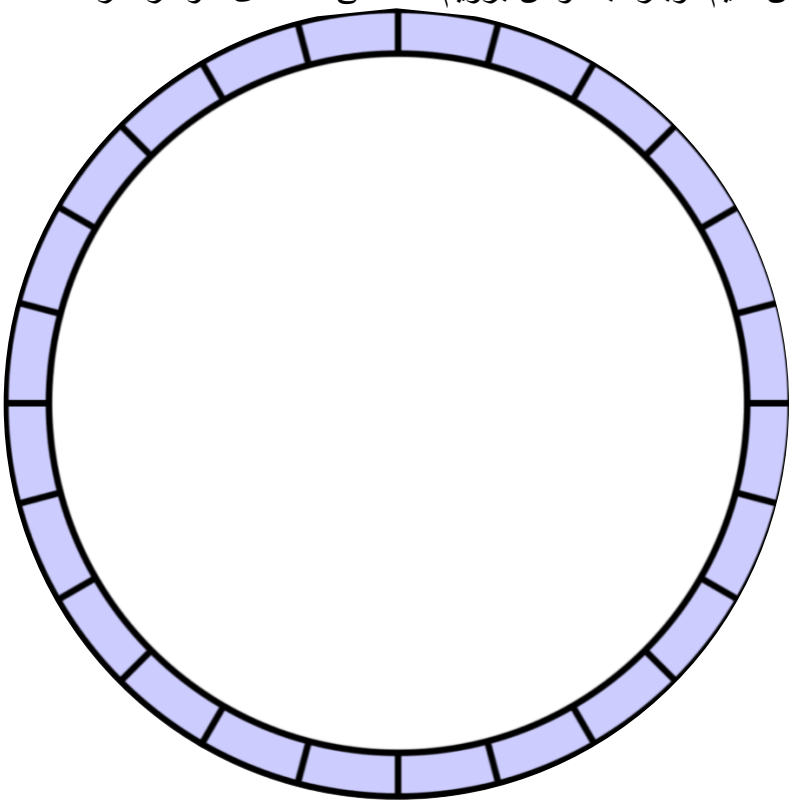
```
module ALU (
    input [511:0] input_data_1,
    input [511:0] input_data_2,
    input ALUOp,
    output signed [1023:0] output_data
);

    reg signed [1023:0] ALUOut;
    integer i;
    integer start_index;
    integer end_index;

    always @(*) begin
        ALUOut = 0; // Initialize ALUOut to avoid latches
        if (ALUOp == 1'b0) begin
            for (i = 0; i < 16; i = i + 1) begin
                start_index = i << 6;
                end_index = i << 5;
                ALUOut[start_index +: 64] = $signed(input_data_1[end_index +: 32]) + $signed(input_data_2[end_index +: 32]);
            end
        end else if (ALUOp == 1'b1) begin
            for (i = 0; i < 16; i = i + 1) begin
                start_index = i << 6;
                end_index = i << 5;
                ALUOut[start_index +: 64] = $signed(input_data_1[end_index +: 32]) * $signed(input_data_2[end_index +: 32]);
            end
        end
    end

    assign output_data = ALUOut;

endmodule
```

در این ماژول تنها دو عملکرد ضرب و جمع ۳۲ بیتی پیاده سازی شده است به شکلی که دو عدد ۳۲ بیتی ورودی می‌گیرد و سپس با توجه به ALUop ورودی گرفته شده است تصمیم می‌گیرد که روی این دو عدد ضرب یا جمع انجام دهد و یک عدد ۶۴ بیتی تولید کند که بتوان نتیجه آن را در دو ثبات R3, R4 نوشت.

: Memory (۳

```
module MEMORY(
        input clk,
        input reset,
        input signed [511 : 0] input_data,
        input [8 : 0] data_address,
        input write_enable,
        output signed [511 : 0] output_data
    );

    reg signed [31 : 0] data_memory [0 : 511];
    reg signed [511 : 0] MemOut;
    integer i, j;
    integer i1, i2;
    integer j1, j2;


    initial
        reset_mem();

    always @(negedge clk or posedge reset) begin
        if(reset) begin
            reset_mem();
        end else begin
            if (write_enable) begin
                for (i = 0; i < 16; i = i + 1) begin
                    i1 = (data_address + i) % 512;
                    i2 = i << 5;
                    data_memory[i1] <= $signed(input_data[i2 +: 32]);
                end
            end
        end
    end

    always @(*) begin
        for (j = 0; j < 16; j = j + 1) begin
            j1 = j << 5;
            j2 = (j + data_address) % 512;
            MemOut[j1 +: 32] = $signed(data_memory[j2]);
        end
    end


    assign output_data = MemOut;

    task reset_mem;
        begin
            $readmemh("hex_file.txt", data_memory);
        end
    endtask

endmodule
```

در این ماژول به طراحی یک حافظه پرداختیم که حافظه دارای ۵۱۲ خانه ۳۲ بیتی است و ۹ بیت برای آدرس‌دهی نیاز دارد. برای مقدار دهی اولیه و ریست کردن این واحد حافظه از یک فایل که مقادیر رندوم درون آن است استفاده می‌کنیم (البته ۱۰۲۴ بیت نهایی edge case هستن تا بتوانیم توانایی پردازنده را به شکل کامل بررسی کنیم.). مود گیری از ۵۱۲ نیز به این دلیل است که اگر از وسط یک خانه شروع به نوشتن کنیم دوباره به اولش برویم تا تمامی داده‌های خود را در حافظه بنویسیم :



۴) پردازنده :
ورودی‌ها، انتساب‌ها و سیم‌های ورودی و خروجی‌ها به همراه یک instance از قطعه‌هایی که ساخته‌ایم:

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

    integer i;

    reg         [511 : 0]   ALUin1;
    reg         [511 : 0]   ALUin2;
    reg                     ALUOp;
    reg         [511 : 0]   RF_in_1;
    reg         [511 : 0]   RF_in_2;
    reg         [1 : 0]     write_address_1;
    reg         [1 : 0]     write_address_2;
    reg                     write_enable_1;
    reg                     write_enable_2;
    reg         [1 : 0]     read_address;
    reg         [8 : 0]     DM_address;
    reg                     DM_write_enable;
    reg  signed [511 : 0]   DM_in;
    wire signed [1023 : 0]  ALUout;
    wire signed [511 : 0]   RF_out;
    wire signed [511 : 0]   RF_A1;
    wire signed [511 : 0]   RF_A2;
    wire signed [511 : 0]   RF_A3;
    wire signed [511 : 0]   RF_A4;
    wire signed [511 : 0]   DM_out;

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

    ALU alu (ALUin1, ALUin2, ALUOp, ALUout);

    RF register_file (clk, reset, RF_in_1, RF_in_2, write_address_1,write_address_2, write_enable_1, write_enable_2,read_address, RF_out, RF_A1, RF_A2, RF_A3, RF_A4);

    MEMORY data_memory (clk, reset, DM_in, DM_address, DM_write_enable, DM_out);

    assign A1 = RF_A1;
    assign A2 = RF_A2;
    assign A3 = RF_A3;
    assign A4 = RF_A4;
```

یک localparam نیز تعریف می‌کنیم که روی opcodeهای دستورهای ورودی حالت بندی می‌کند تا بفهمیم که باید سیگنال‌های کنترلی را چگونه تغییر دهیم.
برای تغییر سیگنال‌های کنترلی تسک‌های زیر تعریف شده‌اند :

```
task control_load;
    begin
        DM_write_enable <= 0;
        DM_address <= instruction[8 : 0];
        write_enable_1 <= 1;
        write_enable_2 <= 0;
        write_address_1 <= instruction[10 : 9];
        #5
        RF_in_1 <= DM_out;
    end
endtask

task control_store;
    begin
        DM_write_enable <= 1;
        DM_address <= instruction[8 : 0];
        write_enable_1 <= 0;
        write_enable_2 <= 0;
        read_address <= instruction[10 : 9];
        #5
        DM_in <= RF_out;
    end
endtask

task control_add_mull;
    begin
        DM_write_enable <= 0;
        write_enable_1 <= 1;
        write_enable_2 <= 1;
        write_address_1 <= 2'b10;
        write_address_2 <= 2'b11;
        ALUin1 <= RF_A1;
        ALUin2 <= RF_A2;
    end
endtask
```

که کنترل یونیت مربوط به add و mul شبیه هم هستند تنها ALUop های متفاوتی دارند که جدا تعیین می‌شوند که در تصویر صفحه بعد می‌توان ادامه ماژول این پردازنده را دید.

```
    integer start_index, finish_index1, finish_index2;

    always @(posedge clk) begin
        #5
        case (instruction[12 : 11])
            load:
                control_load();
            store:
                control_store();
            add: begin
                control_add_mull();
                ALUOp = 1'b0;
                #5
                for(i = 0; i < 16; i = i + 1) begin
                    start_index = i << 5;
                    finish_index1 = i << 6;
                    finish_index2 = finish_index1 + 32;
                    RF_in_1[start_index +: 32] = ALUout[finish_index1 +: 32];
                    RF_in_2[start_index +: 32] = ALUout[finish_index2 +: 32];
                end
            end
            mul: begin
                control_add_mull();
                ALUOp = 1'b1;
                #5
                for(i = 0; i < 16; i = i + 1) begin
                    start_index = i << 5;
                    finish_index1 = i << 6;
                    finish_index2 = finish_index1 + 32;
                    RF_in_1[start_index +: 32] = ALUout[finish_index1 +: 32];
                    RF_in_2[start_index +: 32] = ALUout[finish_index2 +: 32];
                end
            end
        endcase
    end

endmodule
```

هر زمان که کلاک به سمت بالا حرکت کند روی opcodeهای ورودی case زده و با مقادیر لوکال پارام تعریف شده مقایسه می‌کنیم تا متوجه شویم کدوم عملیات‌ها باید صورت بگیرد پس از انجام ضرب یا جمع ۳۲ بیت ابتدایی ALU را به ورودی اول رجیستر فایل و ۳۲ بیت نهایی را به ورودی دوم رجیستر فایل داده تا روی آن‌ها ذخیره شود.

تست بنچ)

در این تست بنچ ابتدا روی ثبات‌های خود لود را انجام داده تا از صحت آن با خبر شویم سپس یک داده را در خانه ۰ حافظه ذخیره کرده و سپس دوباره در یک ثبات دیگر لود می‌کنیم که با توجه به اینکه نتیجه یکسان می‌گیریم می‌توان فهمید که دستور ذخیره سازی نیز به درستی کار می‌کند. سپس به ترتیب دستورهای جمع و ضرب را تست کرده که آن‌ها نیز نتیجه درست گرفته‌اند. سپس تست‌های مرزی که در خانه‌های آخر حافظه ریخته شده بود را در ثبات‌های اول و دوم لود کرده و عملیات ضرب و جمع را انجام داده تا از صحت آن‌ها با خبر شویم.

```
module TB;

    reg clk;
    reg reset;
    reg [12 : 0] instruction;
    wire [511 : 0] A1;
    wire [511 : 0] A2;
    wire [511 : 0] A3;
    wire [511 : 0] A4;

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

    initial
        begin
            clk = 0;
            forever begin
                #20
                clk = ~clk;
            end
        end


    initial begin
        // LOAD FROM DIFFERENT ADDRESSES
        instruction[12:11] = 2'b00;
        instruction[10:9] = 2'b00;
        instruction[8:0] = 9'b01;
        #40
        instruction[12:11] = 2'b00;
        instruction[10:9] = 2'b01;
        instruction[8:0] = 9'b0;
        #40
        instruction[12:11] = 2'b00;
        instruction[10:9] = 2'b10;
        instruction[8:0] = 9'h111;
        #40
        instruction[12:11] = 2'b00;
        instruction[10:9] = 2'b11;
        instruction[8:0] = 9'h21;

        // STORE DATA AND THEN CHECK IF WE STORE IT CORRECT OR NOT
        #40
        instruction[12:11] = 2'b01;
        instruction[10:9] = 2'b11;
        instruction[8:0] = 9'b01;
        #40
        instruction[12:11] = 2'b00;
        instruction[10:9] = 2'b00;
        instruction[8:0] = 9'b01;
```

در صفحه بعد می‌توانید نتایج اجرای این دستورات را ببینید.

```
        // ADD TEST
        #40
        instruction[12:11] = 2'b10;
        instruction[10:0] = 11'b0;

        // MULL TEST
        #40
        instruction[12:11] = 2'b11;
        instruction[10:0] = 11'b0;

        // LOAD EDGE TEST CASES
        #40
        instruction[12:11] = 2'b00;
        instruction[10:9] = 2'b00;
        instruction[8:0] = 9'd480;
        #40
        instruction[12:11] = 2'b00;
        instruction[10:9] = 2'b01;
        instruction[8:0] = 9'd496;

        // ADD TEST EDGE CASES
        #40
        instruction[12:11] = 2'b10;
        instruction[10:0] = 11'b0;

        // MULL TEST EDGE CASES
        #40
        instruction[12:11] = 2'b11;
        instruction[10:0] = 11'b0;
        #40

        #10 $stop;
    end

    initial
        begin
            $monitor($time," A1 = %h\n A2 = %h\n A3 = %h\n A4 = %h\n", A1, A2, A3, A4);

        end

endmodule
```

نتايج :

```
kasrahmedamirkasras-MacBook-Pro-2 Q7_verilog % vvp my_design
                   0 A1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                  40 A1 = cabe2ee946d472e223e8b3d0211651eeba4ac3105e26c659bc78624a5c30b29a743b1ea7bd99009f9268bff77828f0d31c9fd9a5d4ebfa596b367a8aa38c9ce1
A2 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                  80 A1 = cabe2ee946d472e223e8b3d0211651eeba4ac3105e26c659bc78624a5c30b29a743b1ea7bd99009f9268bff77828f0d31c9fd9a5d4ebfa596b367a8aa38c9ce1
A2 = 46d472e223e8b3d0211651eeba4ac3105e26c659bc78624a5c30b29a743b1ea7bd99009f9268bff77828f0d31c9fd9a5d4ebfa596b367a8aa38c9ce1ce6e2560
A3 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                 120 A1 = cabe2ee946d472e223e8b3d0211651eeba4ac3105e26c659bc78624a5c30b29a743b1ea7bd99009f9268bff77828f0d31c9fd9a5d4ebfa596b367a8aa38c9ce1
A2 = 46d472e223e8b3d0211651eeba4ac3105e26c659bc78624a5c30b29a743b1ea7bd99009f9268bff77828f0d31c9fd9a5d4ebfa596b367a8aa38c9ce1ce6e2560
A3 = cc27a02034c0926b8841ec917063496b06dda29c51def56226560b47d261849c953c8e24a732a936f6e1d8802ea47a39002aef4513ca7a04437e0af6e1bfbacf
A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

                 160 A1 = cabe2ee946d472e223e8b3d0211651eeba4ac3105e26c659bc78624a5c30b29a743b1ea7bd99009f9268bff77828f0d31c9fd9a5d4ebfa596b367a8aa38c9ce1
A2 = 46d472e223e8b3d0211651eeba4ac3105e26c659bc78624a5c30b29a743b1ea7bd99009f9268bff77828f0d31c9fd9a5d4ebfa596b367a8aa38c9ce1ce6e2560
A3 = cc27a02034c0926b8841ec917063496b06dda29c51def56226560b47d261849c953c8e24a732a936f6e1d8802ea47a39002aef4513ca7a04437e0af6e1bfbacf
A4 = 5f96612f888cbd80aea257a6b844732257ea76d1309729a04aa0ed9e6a5a804e70ef1ded5a937010a73c010c07b4299cc03ec50c1e1c8489f844e227e2806f55

                 240 A1 = 5f96612f888cbd80aea257a6b844732257ea76d1309729a04aa0ed9e6a5a804e70ef1ded5a937010a73c010c07b4299cc03ec50c1e1c8489f844e227e2806f55
A2 = 46d472e223e8b3d0211651eeba4ac3105e26c659bc78624a5c30b29a743b1ea7bd99009f9268bff77828f0d31c9fd9a5d4ebfa596b367a8aa38c9ce1ce6e2560
A3 = cc27a02034c0926b8841ec917063496b06dda29c51def56226560b47d261849c953c8e24a732a936f6e1d8802ea47a39002aef4513ca7a04437e0af6e1bfbacf
A4 = 5f96612f888cbd80aea257a6b844732257ea76d1309729a04aa0ed9e6a5a804e70ef1ded5a937010a73c010c07b4299cc03ec50c1e1c8489f844e227e2806f55

                 280 A1 = 5f96612f888cbd80aea257a6b844732257ea76d1309729a04aa0ed9e6a5a804e70ef1ded5a937010a73c010c07b4299cc03ec50c1e1c8489f844e227e2806f55
A2 = 46d472e223e8b3d0211651eeba4ac3105e26c659bc78624a5c30b29a743b1ea7bd99009f9268bff77828f0d31c9fd9a5d4ebfa596b367a8aa38c9ce1ce6e2560
A3 = a66ad411ac757150cfb8a994728f3632b6113d2aed0f8beaa6d1a038de959ef52e881e8cecfc30071f64f1df24540341952abf658952ff139bd17f08b0ee94b5
A4 = 00000000ffffffffffffffffffffffff00000000ffffffff000000000000000000000000ffffffff0000000000000000ffffffff00000000ffffffffffffffff

                 320 A1 = 5f96612f888cbd80aea257a6b844732257ea76d1309729a04aa0ed9e6a5a804e70ef1ded5a937010a73c010c07b4299cc03ec50c1e1c8489f844e227e2806f55
A2 = 46d472e223e8b3d0211651eeba4ac3105e26c659bc78624a5c30b29a743b1ea7bd99009f9268bff77828f0d31c9fd9a5d4ebfa596b367a8aa38c9ce1ce6e2560
A3 = aef4b97e96da7800a6ed0254a3ce1820496ef4a9eca148401fa6cd0ce60cd6e2002896330b5d0f70e1501ce496480d8cf444392c0a6ebbdab0ae8847bcc708e0
A4 = 1a727200ef3ea98df57bd535138855a52055688cf32eb2741ae007a2304991bae2b4ea5ad939b5aad655ee5c00dc83ff0aba71670c9c53d302cabac605b63922

                 360 A1 = 000000150000ffffffff00007fffffff000ff0000f0000f00f0000f08000000000000000100000001000000ff0000000f0000000fffffff0000000000000000
A2 = 46d472e223e8b3d0211651eeba4ac3105e26c659bc78624a5c30b29a743b1ea7bd99009f9268bff77828f0d31c9fd9a5d4ebfa596b367a8aa38c9ce1ce6e2560
A3 = aef4b97e96da7800a6ed0254a3ce1820496ef4a9eca148401fa6cd0ce60cd6e2002896330b5d0f70e1501ce496480d8cf444392c0a6ebbdab0ae8847bcc708e0
A4 = 1a727200ef3ea98df57bd535138855a52055688cf32eb2741ae007a2304991bae2b4ea5ad939b5aad655ee5c00dc83ff0aba71670c9c53d302cabac605b63922

                 400 A1 = 000000150000ffffffff00007fffffff000ff0000f0000f00f0000f08000000000000000100000001000000ff0000000f0000000fffffff0000000000000000
A2 = 000000020000ffff0000ffff7fffffff000ff000000ff0000f0000f08000000080000000000000010000000f0000000ff0000000ffffffffffffffff00000000
A3 = aef4b97e96da7800a6ed0254a3ce1820496ef4a9eca148401fa6cd0ce60cd6e2002896330b5d0f70e1501ce496480d8cf444392c0a6ebbdab0ae8847bcc708e0
A4 = 1a727200ef3ea98df57bd535138855a52055688cf32eb2741ae007a2304991bae2b4ea5ad939b5aad655ee5c00dc83ff0aba71670c9c53d302cabac605b63922

                 440 A1 = 000000150000ffffffff00007fffffff000ff0000f0000f00f0000f08000000000000000100000001000000ff0000000f0000000fffffff0000000000000000
A2 = 000000020000ffff0000ffff7fffffff000ff000000ff0000f0000f08000000080000000000000010000000f0000000ff0000000ffffffffffffffff00000000
A3 = 000000170001fffeffffffffffffffffe001fe0000f0ff0f01e0001e00000000080000001000000020000001ef000000fe0000000ffffffeffffffff00000000
A4 = 0000000000000000ffffffff000000000000000000000000000000000ffffffffffffffff0000000000000000ffffffffffffffffffffffffffffffff00000000

                 480 A1 = 000000150000ffffffff00007fffffff000ff0000f0000f00f0000f08000000000000000100000001000000ff0000000f0000000fffffff0000000000000000
A2 = 000000020000ffff0000ffff7fffffff000ff000000ff0000f0000f08000000080000000000000010000000f0000000ff0000000ffffffffffffffff00000000
A3 = 0000002afffe000100010000000000001010000000ef100002000e100000000008000000000000001000000e11000000000000000000000010000000000000000
A4 = 0000000000000000ffffffff3fffffff000000fe0000ef1000e1001c40000000ffffffff0000000000000000ffffffff01000000000000000000000000000000
```