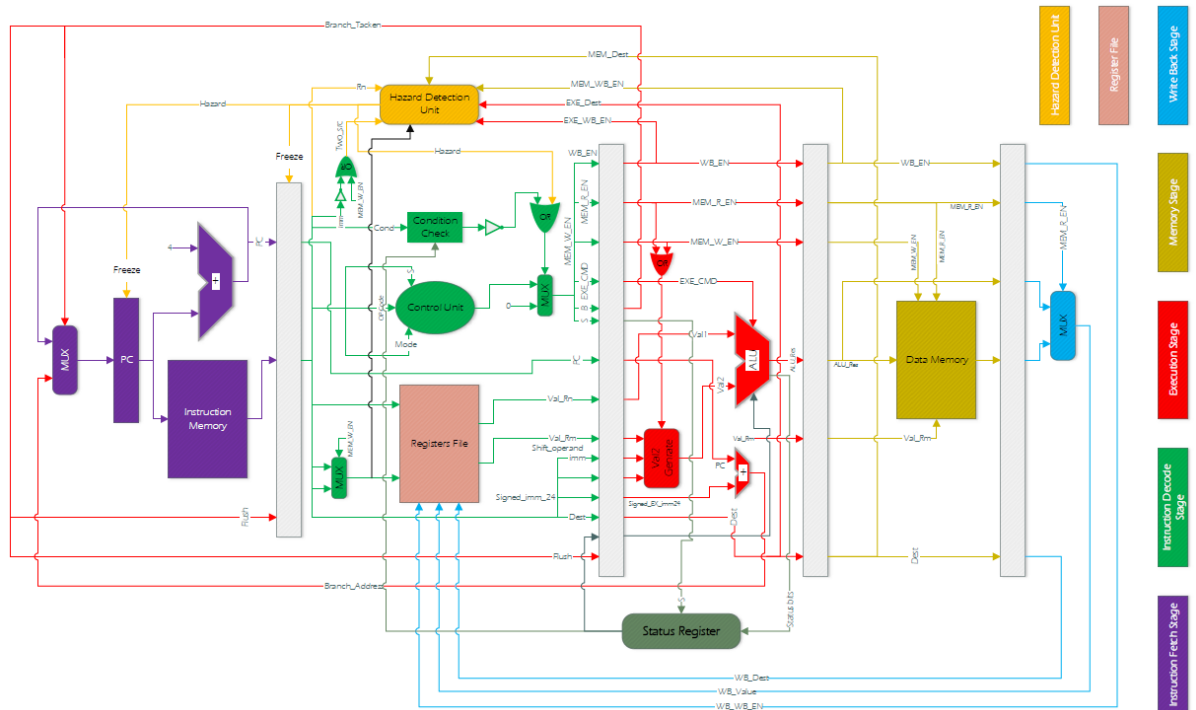


توضیحات آزمایش

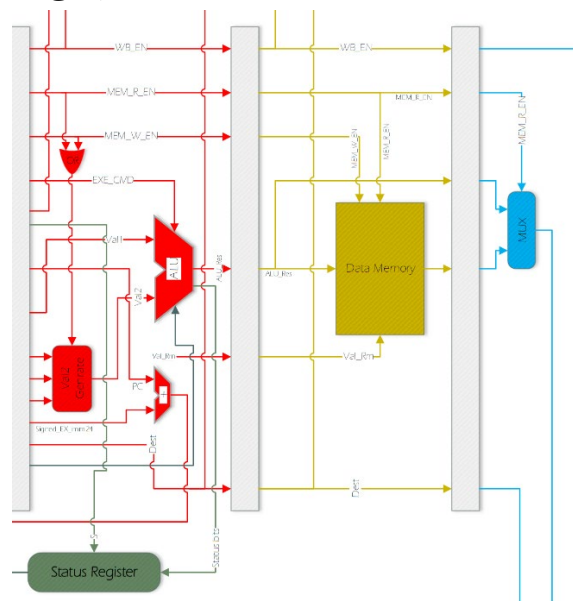
در این آزمایش پردازنده ARM به صورت پایپ‌لاین پیاده‌سازی می‌شود. دیاگرام این پردازنده به صورت زیر است:



این پردازنده دارای 13 دستور اصلی است. پیاده‌سازی باید در زبان ورپلاگ باشد و در نهایت پس از شبیه‌سازی در نرم‌افزار ModelSim، با استفاده از نرم‌افزار Quartus سنتز می‌شود و روی FPGA قرار می‌گیرد. سپس، با استفاده از یک تست‌بنچ، پردازنده پیاده‌سازی شده تست می‌شود. از اهداف این آزمایش می‌توان به یادگیری نحوه عیب‌یابی و تست مدارهای سخت‌افزاری طراحی شده اشاره کرد.

جلسه دوم

در این جلسه سه ماژول اجرای عملیات (EXE)، مموری (MEM) و ذخیره پاسخ (WB) پیاده‌سازی می‌شوند:



ماژول EXE

این ماژول از 4 بخش اصلی تشکیل شده است:

1. Val2Generator

این بخش که وظیفه محاسبه اپرند دوم ALU را دارد، دارای 4 ورودی زیر است:

MemInst:

این ورودی که از OR شدن نتیجه MEM_R_EN و MEM_W_EN بدست می‌آید، نشان می‌دهد که آیا دستور فعلی مربوط به خواندن یا نوشتن در مموری است یا خیر.

Imm:

این بیت، بیت شماره 25 دستور است که نشان می‌دهد آیا اپرند دوم از نوع immediate است یا شماره رجیستر را نشان می‌دهد.

ShifterOperand:

این ورودی شامل 12 بیت سمت راست دستور فعلی است.

ValRm:

یک ورودی 32 بیتی است که از خروجی دوم رجیستر فایل تامین می‌شود.

همچنین، این ماژول شامل 1 خروجی 32 بیتی است که ورودی دوم ALU را مشخص می‌کند.

تعدادی حالت مختلف برای این ماژول وجود دارد:

1. اگر دستور مربوط به مموری باشد، خروجی مقدار Sign-Extend شده 12 بیت Shifter Operand است.

2. اگر مقدار Imm برابر با 1 باشد، 8 بیت سمت راست Shifter Operand به مقدار دو برابر 4 بیت سمت چپ Shifter Operand به صورت دایره‌ای و از سمت راست شیفت می‌خورد.

3. در حالت بعدی 4 بیت سمت راست Shifter Operand نشان‌دهنده شماره رجیستر است که از قبل مقدار این رجیستر در قالب ورودی ValRm وارد این ماژول شده است. در این حالت، بیت 5 و 6 ورودی Shifter Operand نوع شیفت (LSL و LSR و ASR و ROR) را مشخص می‌کند و سپس مقدار ValRm به اندازه 5 بیت سمت چپ Shifter Operand شیفت می‌خورد.

4. حالت دیگری نیز وجود دارد که بنابر صورت سوال، نیازی به پیاده‌سازی آن نیست.

کد این ماژول در ادامه آورده شده است:

```
module Val2Generator(  
    input memInst, imm,  
    input [31:0] valRm,  
    input [11:0] shifterOperand,  
    output reg [31:0] val2  
);  
    integer i;  
  
    always @(memInst or imm or valRm or shifterOperand) begin  
        val2 = 32'd0;  
        if (memInst) begin // LDR, STR  
            val2 = {{20{shifterOperand[11]}}, shifterOperand};  
        end  
        else begin  
            if (imm) begin // immediate  
                val2 = {24'd0, shifterOperand[7:0]};  
                for (i = 0; i < 2 * shifterOperand[11:8]; i = i + 1) begin  
                    val2 = {val2[0], val2[31:1]};  
                end  
            end  
            else begin // shift Rm  
                case (shifterOperand[6:5])  
                    2'b00: val2 = valRm << shifterOperand[11:7]; // LSL  
                    2'b01: val2 = valRm >> shifterOperand[11:7]; // LSR  
                    2'b10: val2 = valRm >>> shifterOperand[11:7]; // ASR  
                    2'b11: begin // ROR  
                        val2 = valRm;  
                        for (i = 0; i < shifterOperand[11:7]; i = i + 1) begin  
                            val2 = {val2[0], val2[31:1]};  
                        end  
                    end  
                    default: val2 = 32'd0;  
                endcase  
            end  
        end  
    end  
endmodule
```

2. ALU

این ماژول دو اپرند 32 بیتی، یک دستور 4 بیتی و یک بیت carryIn را به عنوان ورودی می‌گیرد و یک خروجی 32 بیتی به همراه یک خروجی 4 بیتی برای Status Register تولید می‌کند. مقادیر رجیستر Status به همراه توضیحاتشان در بخش قبل آزمایش ذکر شده‌اند. واحد عملیات (ALU) می‌تواند عملیات‌های زیر را انجام دهد:

- MOV .1
- MVN .2
- ADD .3
- ADC .4
- SUB .5
- SBC .6
- AND .7
- ORR .8
- EOR .9

نوع عملیات با توجه به ورودی 4 بیتی exeCmd مشخص می‌شود.
کد این ماژول در ادامه آورده شده است:

```
module ALU #(
    parameter N = 32
) (
    input [N-1:0] a, b,
    input carryIn,
    input [3:0] exeCmd,
    output reg [N-1:0] out,
    output [3:0] status
);

    reg c, v;
    wire z, n;
    assign status = {n, z, c, v};
    assign z = ~|out;
    assign n = out[N-1];

    wire [N-1:0] carryExt, nCarryExt;
    assign carryExt = {(N-1){1'b0}}, carryIn;
    assign nCarryExt = {(N-1){1'b0}}, ~carryIn;

    always @(exeCmd or a or b or carryIn) begin
        out = {N{1'b0}};
        c = 1'b0;

        case (exeCmd)
            4'b0001: out = b; // MOV
            4'b1001: out = ~b; // MVN
            4'b0010: {c, out} = a + b; // ADD
            4'b0011: {c, out} = a + b + carryExt; // ADC
            4'b0100: {c, out} = a - b; // SUB
            4'b0101: {c, out} = a - b - nCarryExt; // SBC
            4'b0110: out = a & b; // AND
            4'b0111: out = a | b; // ORR
            4'b1000: out = a ^ b; // EOR
            default: out = {N{1'b0}};
        endcase

        v = 1'b0;
        if (exeCmd[3:1] == 3'b001) begin // ADD, ADC
            v = (a[N-1] == b[N-1]) && (a[N-1] != out[N-1]);
        end
        else if (exeCmd[3:1] == 3'b010) begin // SUB, SBC
            v = (a[N-1] != b[N-1]) && (a[N-1] != out[N-1]);
        end
    end
endmodule
```

3. Adder

این ماژول که یک جمع‌کننده 32 بیتی است برای محاسبه آدرس Branch مورد استفاده قرار می‌گیرد. ورودی‌های این ماژول شامل آدرس PC+4 و مقدار Sign Extend شده 24 بیت سمت راست دستور Branch است که جمع این 2 به مرحله اول پایپ‌لاین (IF) بازمی‌گردد.

MEM ماژول

این ماژول تنها شامل Data Memory است که یک ورودی 32 بیتی آدرس، یک ورودی 32 بیتی داده و بیت‌های memRead و memWrite را می‌گیرد و در صورت لزوم همگام با کلاک داده‌ای در مموری نوشته می‌شود و یا به صورت async از آن خوانده می‌شود.
کد این ماژول در ادامه آورده شده است:

```
module DataMemory(  
    input clk, rst,  
    input [31:0] memAdr, writeData,  
    input memRead, memWrite,  
    output reg [31:0] readData  
);  
    localparam WordCount = 4096;  
  
    reg [7:0] dataMem [0:WordCount-1]; // 4KB memory  
  
    wire [31:0] adr;  
    // Align address to the word boundary  
    assign adr = {memAdr[31:2], 2'b00};  
  
    integer i;  
    always @(posedge clk or posedge rst) begin  
        if (rst)  
            for (i = 0; i < WordCount; i = i + 1) begin  
                dataMem[i] <= 8'd0;  
            end  
        else if (memWrite)  
            {dataMem[adr + 3], dataMem[adr + 2],  
             dataMem[adr + 1], dataMem[adr]} <= writeData;  
        end  
  
        always @(memRead or adr) begin  
            if (memRead)  
                readData = {dataMem[adr + 3], dataMem[adr + 2],  
                             dataMem[adr + 1], dataMem[adr]};  
        end  
    end  
endmodule
```

لازم به ذکر است که شبیه‌سازی یک مموری 4 گیگ در ModelSim کار بسیار زمان‌بری خواهد بود و به همین دلیل از یک مموری 4 کیلوبایتی استفاده شده است.

2. `MOV R1, #4096` $\rightarrow R1 = 4096$

/TopLevelTB/t/stId/rf/dk	St1				
/TopLevelTB/t/stId/rf/rst	St0				
/TopLevelTB/t/stId/rf/regFile	20 8192 -10737...	20	1 2 3 4 5 6 7 8 ...	20 4096 2 3 4 5 6 7 8 9 10	
[0]	20	20			
[1]	8192	1		4096	
[2]	-1073741824	2			
[3]	4	3			
[4]	40	4			
[5]	-12	5			
[6]	9	6			
[7]	805306373	7			
[8]	4	8			
[9]	-10	9			
[10]	-36	10			

مقدار R1 پس از اجرای این دستور برابر با 4096 شده است.

3. `MOV R2, #0xC0000000` $\rightarrow R2 = -1073741824$

/TopLevelTB/t/stId/rf/dk	St1				
/TopLevelTB/t/stId/rf/rst	St0				
/TopLevelTB/t/stId/rf/regFile	20 8192 -10737...	20	4096 2 3 ...	20 4096 -1073741824 3 4	
[0]	20	20			
[1]	8192	4096			
[2]	-1073741824	2		-1073741824	
[3]	4	3			
[4]	40	4			
[5]	-12	5			
[6]	9	6			
[7]	805306373	7			

مقدار R2 پس از اجرای این دستور برابر با -1073741824 شده است.

4. `ADDS R3, R2, R2` $\rightarrow R3 = -2147483648$

/TopLevelTB/t/stId/rf/rst	St0				
/TopLevelTB/t/stId/rf/regFile	20 8192 -10737...	20	4096 -1073741...	20 4096 -1073741824 4 4 5..	
[0]	20	20			
[1]	8192	4096			
[2]	-1073741824	-1073741824			
[3]	4	3		4	
[4]	40	4			
[5]	-12	5			
[6]	9	6			
[7]	805306373	7			
[8]	4	8			
[9]	-10	9			
[10]	-36	10			
[11]	11	11			
[12]	12	12			
[13]	13	13			
[14]	14	14			
/TopLevelTB/t/stEx/alu/c	0				
/TopLevelTB/t/stEx/alu/v	0				
/TopLevelTB/t/stEx/alu/z	0				
/TopLevelTB/t/stEx/alu/n	0				

در این دستور به دلیل عدم وجود Forwarding Unit و عدم پیاده‌سازی Hazard Unit، مقدار قبلی R2 (2) در محاسبات دخیل می‌شود و به همین دلیل مقدار 4 در R3 قرار می‌گیرد.

5. \square ADC R4, R0, R0 \rightarrow R4 = 41

/TopLevelTB/t/stId/rf/dk	St1		
/TopLevelTB/t/stId/rf/rst	St0		
/TopLevelTB/t/stId/rf/regFile	20 8192 -10737...	20 4096 -10737...	20 4096 -10737418
[0]	20	20	
[1]	8192	4096	
[2]	-1073741824	-1073741824	
[3]	4	4	
[4]	40	4	40
[5]	-12	5	
[6]	9	6	
[7]	805306373	7	
[8]	4	8	
[9]	-10	9	
[10]	-36	10	
[11]	11	11	

به دلیل وجود اشتباه در محاسبه قبلی، carry برابر با 1 نبوده و خروجی ALU به جای 41 برابر با 40 شده است.

6. \square SUB R5, R4, R4, LSL #2 \rightarrow R5 = -123

/TopLevelTB/t/stId/rf/dk	St1		
/TopLevelTB/t/stId/rf/rst	St0		
/TopLevelTB/t/stId/rf/regFile	20 8192 -10737...	20 4096 -10737...	20 4096 -10737418
[0]	20	20	
[1]	8192	4096	
[2]	-1073741824	-1073741824	
[3]	4	4	
[4]	40	40	
[5]	-12	5	-12
[6]	9	6	
[7]	805306373	7	
[8]	4	8	
[9]	-10	9	

در این دستور نیز به دلیل ذکر شده، خروجی به جای -123 برابر با -12 شده است.

7. \square SBC R6, R0, R0, LSR #1 \rightarrow R6 = 10

/TopLevelTB/t/stId/rf/regFile	20 8192 -10737...	20 4096 -10737...	20 4096 -10737418
[0]	20	20	
[1]	8192	4096	
[2]	-1073741824	-1073741824	
[3]	4	4	
[4]	40	40	
[5]	-12	-12	
[6]	9	6	9
[7]	805306373	7	
[8]	4	8	
[9]	-10	9	
[10]	-36	10	
[11]	11	11	

در این دستور مقدار R6 برابر با 9 شده است. دلیل این تفاوت این است که در دستور 4 باید مقدار Status Register آپدیت می‌شد و بیت C آن برابر با 1 می‌شد. اما چون این اتفاق نیفتاده و carry برابر 0 است، مقدار تفاضل یکی کمتر شده و در این بخش مقدار 9 را خواهیم داشت.

8. \square ORR R7, R5, R2 \rightarrow R7 = -123

با توجه به نادرست بودن مقدار R5، مقدار R7 نیز صحیح نخواهد بود.

9. \square AND R8, R7, R3 \rightarrow R8 = -2147483648

این مقدار نیز به دلیل مشابه متفاوت خواهد بود.

10. MVN R9, R6 → R9 = -11

/TopLevelTB/tl/std/rf/regFile		20 8192 -10737...	20 4096 -1073...	20 4096 -1073...
[0]	+	20	20	
[1]	+	8192	4096	
[2]	+	-1073741824	-1073741824	
[3]	+	4	4	
[4]	+	40	40	
[5]	+	-12	-12	
[6]	+	9	9	
[7]	+	805306373	805306373	
[8]	+	4	4	
[9]	+	-10	9	-10
[10]	+	-36	10	
[11]	+	11	11	
[12]	+	12	12	

در این دستور به دلیل تفاوت یک واحدی در R6 با مقدار واقعی، R9 نیز به جای 11- برابر با 10- شده است.

11. EOR R10, R4, R5 → R10 = -84

این مورد نیز نادرست خواهد بود.

12. CMP R8, R6

با توجه به نادرست اپرندهای این دستور، از توضیح این مورد نیز صرف نظر می‌شود.

13. ADDNE R1, R1, R1 → R1 = 8192

/TopLevelTB/tl/std/rf/rst		St0	20 8192 -10737...	20 4096 -1073741824 4 40 -12...	20 8192 -1073741824 4 40 -12 9 80
/TopLevelTB/tl/std/rf/regFile			20	20	
[0]	+	8192	4096	8192	
[1]	+	-1073741824	-1073741824		
[2]	+	4	4		
[3]	+	40	40		
[4]	+	-12	-12		
[5]	+	9	9		
[6]	+	805306373	805306373		

همانطور که مشاهده می‌شود، پس از این دستور مقدار R1 برابر با 8192 شده است.

14. TST R9, R8

در این بخش R8 و R9 برابر نیستند که نتیجه در دستور بعدی خودش را نشان می‌دهد.

15. ADDEQ R2, R2, R2 → R2 = -1073741824

/TopLevelTB/tl/std/rf/rst		St0	1024 8192 -107...	20 8192 -1073741824 4 40 -12 9 805306373 4 -
/TopLevelTB/tl/std/rf/regFile			1024	20
[0]	+	8192	8192	
[1]	+	-1073741824	-1073741824	
[2]	+	4	4	
[3]	+	40	40	
[4]	+	-12	-12	
[5]	+	9	9	

در اینجا چون شرط EQ صحیح نمی‌باشد، هیچ جمعی صورت نمی‌پذیرد.

16. MOV R0, #1024 → R0 = 1024

/TopLevelTB/tl/std/rf/rst		St0	1024 8192 -107...	20 8192 -1073741824 4 40 -12 9 ...	1024 8192 -10737
/TopLevelTB/tl/std/rf/regFile			1024	20	1024
[0]	+	8192	8192		
[1]	+	-1073741824	-1073741824		
[2]	+	4	4		
[3]	+	40	40		
[4]	+	-12	-12		
[5]	+	9	9		

در اینجا نیز مقدار R0 برابر با 1024 شده است.

17. STR R1, [R0], #0 -> MEM[1024] = 8192

/TopLevelTB/t/stdId/rf/dk	St0			
/TopLevelTB/t/stdId/rf/rst	St0			
/TopLevelTB/t/stdId/rf/regFile	1024 8192 -107...	1024 8192 -1073741824 4 40 -12 9 805306373		
[0]	1024	1024		
[1]	8192	8192		
[2]	-1073741824	-1073741824		
[3]	4	4		
[4]	40	40		
[5]	-12	-12		
[6]	9	9		
[7]	805306373	805306373		
[8]	4	4		
[9]	-10	-10		
[10]	-36	-36		
[11]	0	11		
[12]	12	12		
[13]	13	13		
[14]	14	14		
/TopLevelTB/t/stMem/mem/mem20	8192	0 8192		
/TopLevelTB/t/stMem/mem/memAdr	1024	20		
/TopLevelTB/t/stMem/mem/writeData	1024	8192 20		
/TopLevelTB/t/stEx/alu/c	0			
/TopLevelTB/t/stMem/mem/memWrite	St0			
/TopLevelTB/t/stMem/mem/memRead	St1			
/TopLevelTB/t/stMem/mem/readData	0	8192		
/TopLevelTB/t/stEx/alu/out	1024	20 1024		
/TopLevelTB/t/stEx/alu/v	0			

در اینجا به دلیل عدم وجود Forwarding Unit و عدم پیاده‌سازی Hazard Unit، مقدار 8192 در خانه 20-ام مموری نوشته می‌شود زیرا مقدار 1024 تازه به استیج EXE رسیده است و مقدار 20 است که در استیج MEM حضور دارد.

18. LDR R11, [R0], #0 -> R11 = 8192

/TopLevelTB/t/dk	St1			
/TopLevelTB/t/rst	St0			
/TopLevelTB/t/stdId/rf/regFile	0 1 2 3 4 5 6 7 8 ...	1024 8192 -10... 1024 8192 ... 1024 8192 -10...		
[0]	0	1024		
[1]	1	8192		
[2]	2	-1073741824		
[3]	3	4		
[4]	4	40		
[5]	5	-12		
[6]	6	9		
[7]	7	805306373		
[8]	8	4		
[9]	9	-10		
[10]	10	-36		
[11]	11	11 8192 0		
[12]	12	12		
[13]	13	13		
[14]	14	14		
/TopLevelTB/t/stMem/mem/mem20	0	8192		
/TopLevelTB/t/stMem/mem/memAdr	4096	20 1024		
/TopLevelTB/t/stMem/mem/memRead	St0			
/TopLevelTB/t/stMem/mem/readData	x	8192 0		

همانطور که مشاهده می‌شود، پس از این دستور مقدار R11 برابر با 8192 شده است.

نتیجه سنتز در کوارتس

Flow Summary	
Flow Status	Successful - Fri Apr 07 00:49:30 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	ARM
Top-level Entity Name	ARM
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	0 / 33,216 (0 %)
Total combinational functions	0 / 33,216 (0 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)