《计算机组成原理》实验报告

年级、专业、班级	2022级计算机科学与技术03班/01班/06班	姓名	叶旭航，解吴雪，李佳玲
实验题目	实验一简单流水线与运算器实验		
实验时间	2024 年 3 月 19 日	实验地点	DS1410
实验成绩	优秀/良好/中等	实验性质	□验证性 ☑设计性 □综合性

教师评价：

☑算法/实验过程正确； ☑源程序/实验内容提交； ☑程序结构/实验步骤合理；

☑实验结果正确； ☑语法、语义正确； ☑报告规范；

其他:

评价教师: 冯永

实验目的

（1）理解流水线 (Pipeline) 设计原理；

（2）了解算术逻辑单元 ALU 的原理；

（3）熟悉并运用 Verilog 语言设计 ALU；

（4）熟悉并运用 Verilog 语言设计流水线全加器；

报告完成时间: 2024年 4月 6日

1 实验内容

1.1 ALU设计实验

实验要求实现以下算术运算功能，其对应的控制码及功能如下：

$F_{2:0}$	功能	$F_{2:0}$	功能
000	A + B(Unsigned)	100	$\overline{A}$
001	A - B	***101***	***SLT***
010	A AND B	110	未使用
011	A OR B	111	未使用

表 1: 算数运算控制码及功能

实验要求：

1. 根据ALU原理图，使用Verilog语言定义ALU模块，其中输入输出端口参考实验原理，运算指令码长度为[2:0]。
2. 仿真时B端口输入为32h'01, A端口输入参照 4.1 中表格
3. 实现SLT功能。
4. 验证表 1中所有功能。
5. 给出RTL源程序（.v文件）

1.2 流水线实验

本次实验为仿真实验，设计完成后仅需进行**行为仿真**。

实验要求：

1. 实现4级流水线8bit全加器，需带有流水线暂停和刷新；
2. 模拟流水线暂停，仿真时控制10周期后暂停流水线2周期（第2级），流水线恢复流动；
3. 模拟流水线刷新，仿真时控制15周期时流水线刷新（第3级）。

2 实验设计

2.1 ALU

2.1.1 功能描述

实现一个计算逻辑单元，对数据进行基本的操作并在数码管上直观的显示

2.1.2 接口定义

表 2: 接口定义模版

信号名	方向	位宽	功能描述
ans	output	32-bit	ALU的结果输出。
num1	input	8-bit	计算操作数。
num2	input	32-bit	第二个计算操作数。
op	input	3-bit	多位选择器的选择数，不同数字对应着对Input进行不同的操作。
clk	input	1-bit	时钟信号。
reset	input	1-bit	复位信号。
s	input	32-bit	ALU计算结果。
seg	output	7-bit	Seg7输出，七段数码管显示的数字。
ans	output	8-bit	控制哪个管子亮。
sw	input	3-bit	op信号。
ins	input	8-bit	Num1信号。

2.1.3 逻辑控制

主要分为四个文件，ALU为核心部分，输入数据num1和num2并根据op对数据进行计算，ALU_top为顶层文件，将所有文件功能串联起来从而实现功能，display利用时间clk并对之进行计数，从而在不同时间里让不同的管子输出alu_result的不同位数的结果，seg7为七段数码管的控制文件根据display的din实现不同的dout

2.2 有阻塞4级8bit全加器

2.2.1 功能描述

将数据位数比较长的数分为多个阶段进行求解，从而提升整体运算速度

2.2.2 接口定义

2.2.3 逻辑控制

3 实验过程记录

表 3: 接口定义模版

信号名	方向	位宽	功能描述
cin_a	input	8-bit	要进行计算的8位数据。
cin_b	input	8-bit	要进行计算的另一组8位数据。
rst	input	1-bit	复位信号。
clk	input	1-bit	时钟信号。
stop	input	1-bit	实现对流水线的阻塞。
c_in	input	1-bit	模拟最开始的进位。
c_out	output	1-bit	进位输出。
sum	output	8-bit	最终结果。

3.1 问题1: 实现各指令码所对应的功能

问题描述：在实现过程中需要根据指令码执行相应的功能，以确保输入的指令码能够准确执行相应的算术运算。

解决方案：使用 Verilog 中的 case 语句来根据指令码执行相应的操作。

3.2 问题2: num1无符号扩展

问题描述：num1由外部输入后, 需经过无符号扩展至 32 位后输入到运算器的端口B

解决方案：通过在 num1 的高位添加 24 个零来实现的。具体来说，24'b0, num1 就是将 num1 的低 8 位放在结果的低位，然后在高位添加 24 个零。这样就实现了对 num1 进行无符号扩展，确保了它与 num2 在进行运算时具有相同的位数。

3.3 问题3: SLT功能的实现

问题描述：实现num1与 num2 的比较，并根据比较结果设置相应的输出。

解决方案：首先对 num1 进行无符号扩展，即在 num1 的高位添加 24 个零。将无符号扩展后的 num1 与 num2 进行比较。如果 num1 小于 num2，则将结果设置为 32 位全 1 的值；否则，将结果设置为 32 位全 0 的值。

3.4 问题4: ALU约束文件完善

问题描述：给出的约束文件中存在部分端口缺失的情况，上板验证时出现错误。

解决方案：补充缺失的端口到约束文件中，确保所有端口都得到正确约束，重新上板验证后

无误

3.5 问题5: 验证ALU功能

问题描述：编写仿真文件验证 ALU 的各项功能是否符合预期。

解决方案：在仿真文件中设置不同的操作码 op_tb 和操作数 num1_tb，然后观察输出结果 ans_tb 来验证 ALU 的各种功能是否正确。对于每种操作，将适当的操作码赋给 op_tb 变量，并设置适当的输入操作数。每次更改操作后，等待一段时间以确保 ALU 有足够的时间执行操作。通过观察输出结果，可以验证 ALU 模块中对应操作的功能是否正确。

3.6 问题6: 实现4级流水线全加器

问题描述：每个级别进行 8 位加法运算，并实现流水线结构以提高计算效率。

解决方案：使用四个always块进行流水线的实现，每个阶段可以在不同的时钟周期内独立计算部分8bit的结果，从而实现了流水线加法器的功能。在每个时钟周期内，每个阶段的计算结果被传递到下一个阶段，并随着时钟信号的推动完成整个加法操作。

3.7 问题7: 实现对流水线的阻塞

问题描述：实现带有流水线暂停和刷新 4 级流水线全加器。

解决方案：通过在每个阶段中加入对 stop 信号的判断逻辑，并在满足阻塞条件时暂停计算，可以实现有阻塞的流水线设计。

3.8 问题8: 对4级流水线全加器进行行为仿真

问题描述：编写仿真文件实现对流水线功能的验证和模拟流水线的暂停和刷新。

解决方案：按照时钟周期，逐步输入测试数据，观察每个阶段的计算结果是否符合预期。并在文件中要求的时刻设置暂停信号，验证流水线的暂停功能。然后恢复暂停，继续观察流水线的计算结果。

4 实验结果及分析

4.1 ALU验证实验结果

4.2 流水线阻塞（暂停）仿真图

操作	Num1	Result
A + B(Unsigned)	8'b00000010	
A - B	8'b11111111	
A AND B	8'b11111110	
A OR B	8'b10101010	
$\overline{A}$	8'b11110000	
SLT	8'b10000001	

表 4: ALU结果表

图 1: 占位图

4.3 流水线刷新(清空)仿真图

A ALU代码

```
module ALU(
    input [7:0] num1,
    input [31:0] num2,
    input [2:0] op,
    output reg [31:0] ans
    );

always @ (*)
begin
    case(op)
        3'b000: ans = {24'b0, num1} + num2;
        3'b001: ans = {24'b0, num1} - num2;
        3'b010: ans = {24'b0, num1} & num2;
        3'b011: ans = {24'b0, num1} | num2;
        3'b100: ans = ~{24'b0, num1};
        3'b101: ans = ({24'b0, num1} < num2) ? 32'b1 : 32'b0;
        3'b110: ans = 32'b0;
        3'b111: ans = 32'b0;
        default : ans = 32'hX;
    endcase
end

endmodule
```

B 32bit流水线全加器代码

```
module stallable_pipeline_adder(
    input [7:0] cin_a, cin_b,
```

```
    input rst, clk, stop, c_in,
    output reg c_out,
    output reg [7:0] sum
);

reg cout1, cout2, cout3, cout4;

reg [1:0] sum1;
reg [3:0] sum2;
reg [5:0] sum3;
reg [7:0] sum4;

reg [5:0] surA1, surB1;
reg [3:0] surA2, surB2;
reg [1:0] surA3, surB3;

// Stage 1
always @(posedge clk) begin
    if (rst)
    begin
        cout1 <= 0;
        sum1 <= 0;
        surA1 <= 0;
        surB1 <= 0;
    end
    else if (stop)
    begin
        cout1 <= cout1;
        sum1 <= sum1;
        surA1 <= surA1;
        surB1 <= surB1;
    end
    else
    begin
        {cout1, sum1} = cin_a[1:0] + cin_b[1:0] + c_in;
        surA1 <= cin_a[7:2];
        surB1 <= cin_b[7:2];
    end
end

// Stage 2
always @(posedge clk) begin
    if (rst)
    begin
        cout2 <= 0;
        sum2 <= 0;
        surA2 <= 0;
        surB2 <= 0;
    end
```

```
    else if (stop)
    begin
        cout2 <= cout2;
        sum2 <= sum2;
        surA2 <= surA2;
        surB2 <= surB2;
    end
    else
    begin
        {cout2, sum2[3:2]} = surA1[1:0] + surB1[1:0] + cout1;
        sum2[1:0] <= sum1;
        surA2 <= surA1[5:2];
        surB2 <= surB1[5:2];
    end
end

// Stage 3
always @(posedge clk) begin
    if (rst)
    begin
        cout3 <= 0;
        sum3 <= 0;
        surA3 <= 0;
        surB3 <= 0;
    end
    else if (stop)
    begin
        c_out <= c_out;
        sum3 <= sum3;
        surA3 <= surA3;
        surB3 <= surB3;
    end
    else
    begin
        {cout3, sum3[5:4]} = surA2[1:0] + surB2[1:0] + cout2;
        sum3[3:0] <= sum2;
        surA3 <= surA2[3:2];
        surB3 <= surB2[3:2];
    end
end

// Stage 4
always @(posedge clk) begin
    if (rst)
    begin
        cout4 <= 0;
        sum4 <= 0;
    end
    else if (stop)
```

```
    begin
        cout4 <= cout4;
        sum4 <= sum4;
    end
    else
    begin
        {cout4, sum4[7:6]} = surA3[1:0] + surB3[1:0] + cout3;
        sum4[5:0] <= sum3;
    end
end

// Assign the output
always @(*) begin
    sum = sum4;
    c_out = cout4;
end

endmodule
```