同濟大學
TONGJI UNIVERSITY

计算机系统结构课程实验
总结报告

实验题目：简单的流水线 CPU 设计与性能分析

学号：2152809

姓名：曾崇然

指导教师：陆有军老师

日期：2023-11-26

一、实验环境部署与硬件配置说明

1. 实验环境部署：windows11，使用 vivado 作为开发工具，vivado 自带的仿真工具进行仿真

2. 硬件配置说明：Xilinx FPGA 器件：Nexys DDR 开发板

二、实验的总体结构

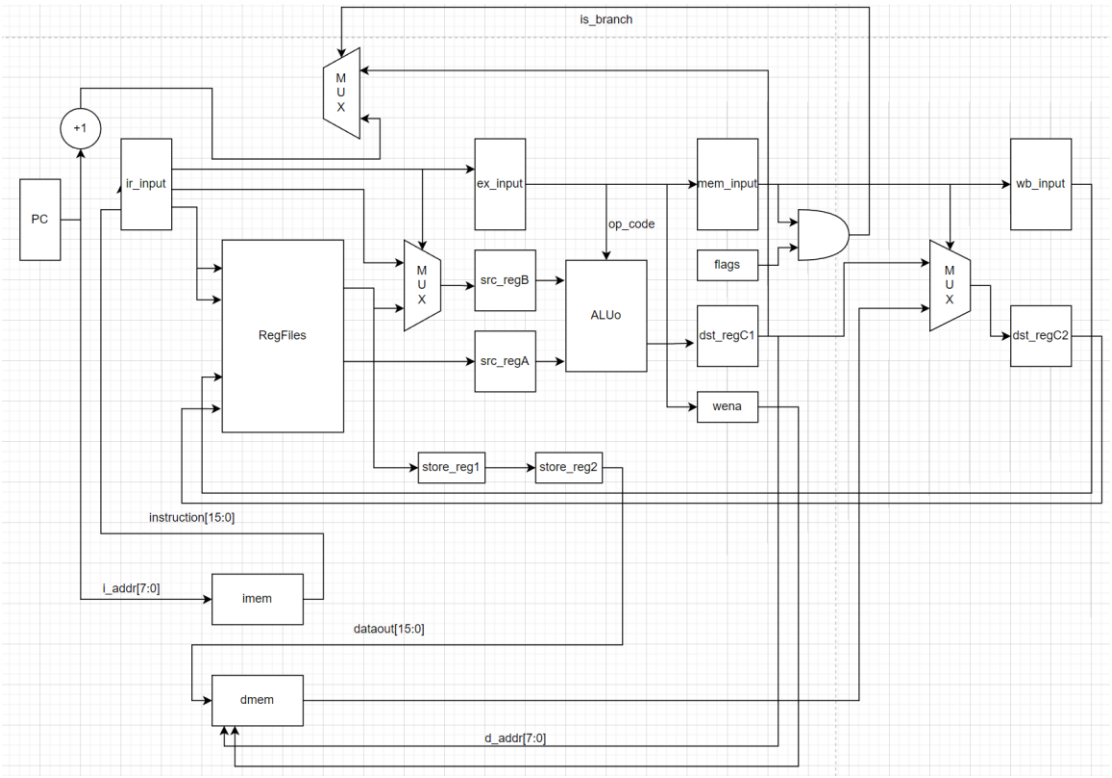
1、 指令选取：共选取常用指令 15 条，以完成指定的功能，依次为：NOP、HALT、ADD、ADDI、SUB、SUBI、SRL、CMP、JUMP、BN、BNN、BZ、BNZ、LOAD、STORE

2、 指令码设计，因为指令为 15 条，所以可用 4 位表示操作码，每 4 位表示一个寄存器或者立即数，后 8 位可拼接起来表示一个立即数。具体设计如下，该 CPU 可执行 15 条指令，拥有 16 个通用寄存器

指令码格式:

种类	操作码	op1	op2	op3	operation
NOP	0000	0000	0000	0000	no operation
HALT	0001	0000	0000	0000	halt
ADD	0010	r1	r2	r3	r1 <- r2+r3
ADDI	0011	r1	val1	val2	r1 <- r1+{val1,val2}
SUB	0100	r1	r2	r3	r1 <- r2-r3
SUBI	0101	r1	val1	val2	r1 <- r1-{val1,val2}
SRL	0110	r1	r2	val	r1 <- r2>>val
CMP	0111	0000	r2	r3	r2-r3 set cf,zf,nf
JUMP	1000	0000	val1	val2	jump to {val1,val2}
BN	1001	r1	val1	val2	jump to r1+{val1,val2} if nf==1
BNN	1010	r1	val1	val2	jump to r1+{val1,val2} if nf==0
BZ	1011	r1	val1	val2	jump to r1+{val1,val2} if zf==1
BNZ	1100	r1	val1	val2	jump to r1+{val1,val2} if zf==0
LOAD	1101	r1	r2	val	r1 <- dmem[r2+val]
STORE	1110	r1	r2	val	r1 -> dmem[r2+val]

3、 静态流水线的总体结构



4、结构解释

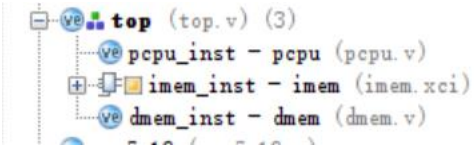
①总体采用哈佛结构，指令和数据分开存储，通过数据总线和指令总线和 CPU 进行数据交换

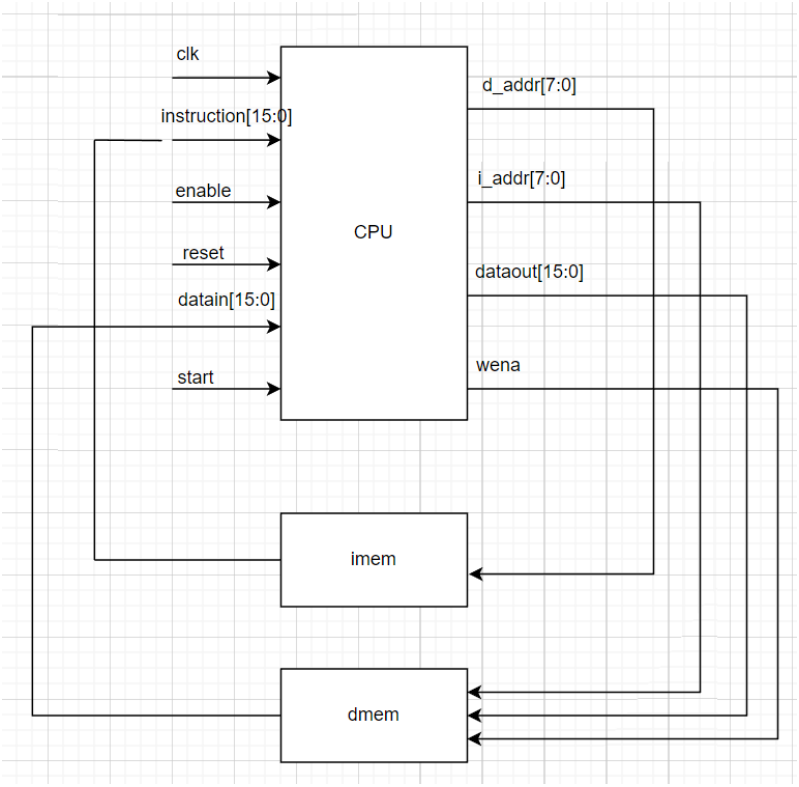
②一条指令的处理划分为 IF、ID、EX、MEM、WB 五个阶段，分别位取址、取值、计算、访存、写回，根据指令类型的不同在不同的阶段完成不同的操作。

③通过设置多个寄存器来实现指令的流水执行

三、总体架构部件的解释说明

1、 总体构架：主体分为三部分，CPU，imem（指令寄存器），dmem（数据寄存器），imem 和 dmem 通过指令总线和数据总线与 CPU 进行数据的交换，使用 top.v 文件将这三个部件进行连接





2、 静态流水线具体部件的解释说明

①PC 寄存器：存放要读取的指令的地址，地址为 7 位，即指令存储器的深度为 256

②id_input，ex_input，mem_input，wb_input：存放对应阶段流入的指令，每个阶段设置一个以保证指令流水执行

③RegFiles：通用寄存器堆，16 个

④src_regA，src_regB：两个源操作数寄存器，在 ID 阶段根据指令类型的不同将这两个寄存器进行赋值

⑤store_reg1，store_reg2：用于 STORE 指令的两个

寄存器，使用两个以保证两个 STORE 指令可以连续执行（流水）

⑥ALU：运算部件，在 EX 阶段进行计算，并置标志位，能进行加法，减法，移位等操作

⑦dst_regC1，dst_regC2：两个结果寄存器，分别在 MEM 和 WB 阶段将结果传出，设置两个是为了保证指令的流水执行

⑧wena：写信号寄存器

⑨flag：各种标志位寄存器，如 nf，zf，cf

⑩imem，dmem：指令存储器和数据存储器，指令存储器位 ROM，数据存储器为 RAM，在使能信号和写入信号有效的上升沿写入，指令和数据的宽度均为 16 位，深度位 256。

四、实验仿真过程

1、 编写用于仿真测试的 C 程序

2、 将 C 程序转化为对应的指令序列，保证指令在实现的指令范围内，同时使用 NOP 指令和调整指令顺序的方法解决冲突

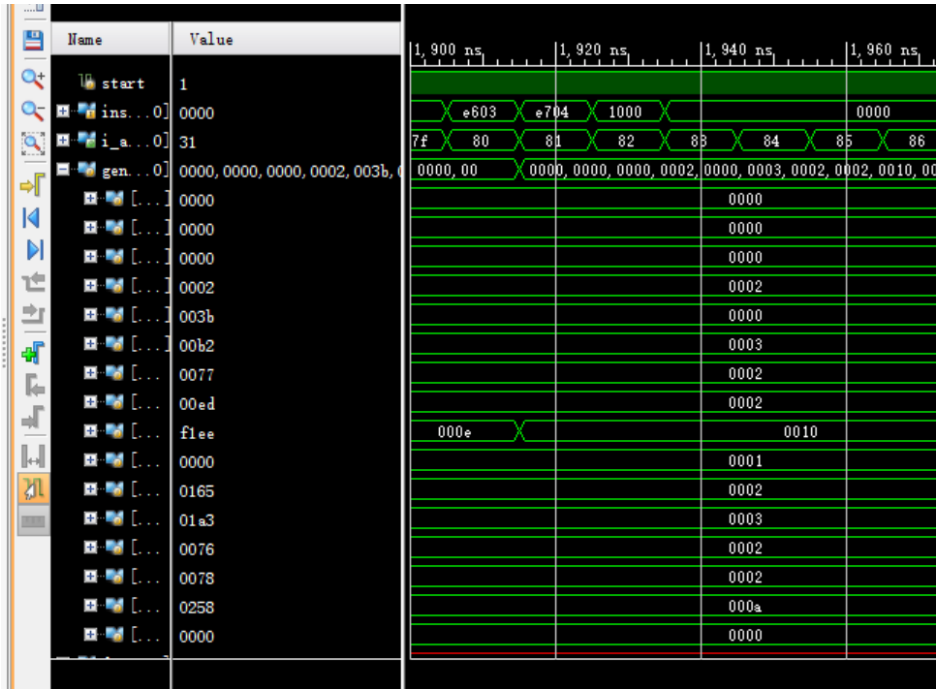
3、 将指令序列转化为机器码，形成 coe 文件

4、 使用该 coe 文件初始化 imem

5、 编写 test_bench 文件进行测试仿真

五、实验仿真的波形图及某时刻寄存器值的物理意义

1、 静态流水线的波形图（未展示全部）：以下波形包含 start（开始信号），instruction（指令），i_addr（指令地址），general_reg（通用寄存器堆）



2、 寄存器值的物理意义

①cpu_state、next_cpu_state：CPU 当前和下一个状态寄存器

②id_input、ex_input、mem_input、wb_input，存储正在某个阶段执行的对应的指令

③src_regA，src_regB 两个源操作数寄存器

④dst_regC1，dst_regC2 两个目标寄存器，两个保证流水线不断

⑤store_reg1，store_reg2 用于 store 指令的两个寄存器，两个保证流水线不断

⑥general_reg[15:0]通用寄存器堆

⑦zf，nf，cf 运算的标志位

⑧ALU_result 运算器的运算结果

⑨cf_buf 存储溢出位

六、流水线 CPU 实验性能验证模型

实验性能验证模型：比萨塔摔鸡蛋游戏。两个同学在可变换层数的比萨塔上摔鸡蛋，一个同学秘密设定同一批鸡蛋耐摔值；另一个同学在指定层高的比萨塔拿着鸡蛋往下摔，用最少的摔次数和摔破的鸡蛋数求出鸡蛋的耐摔值。假定在耐摔值的楼层及其下面楼层，鸡蛋摔不破，可以重复使用，否则鸡蛋摔破。要求模型的算法输出包括：摔的总次数、摔的总鸡蛋数、最后摔的鸡蛋是否摔破。请使用 C 语言设计该验证模型的算法，并把 C 语言汇编为 RISC-V 指令汇编程序，同时利用编译器生成 RISC-V 指令集可执行目标程序。

1、 编写 C 语言程序

采用二分的方法进行耐摔值的查找，评估函数为爬的楼层数+5*摔碎鸡蛋数

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    //初始参数
    int totalFloors = 10,  fallResistance = 2;
    //要求的
    int result = 0, count = 0, totalEggs = 0, isBroken = 0, value = 0;

    int high = totalFloors, low = 1;
    int stumbFloor = 0;
    while (1) {
        count++;
        stumbFloor = (high + low) / 2;
        value = value + (high - low) / 2;
        if (stumbFloor <= fallResistance) {
            isBroken = 0;
            low = stumbFloor;
        }
        else {
```

```
            isBroken = 1;
            totalEggs++;
            high = stumbFloor - 1;
        }
        if ((high - low) < 2) {
            break;
        }
    }

    if (high == low) {
        result = high;
    }
    else {
        count++;
        if (high <= fallResistance) {
            isBroken = 0;
            result = high;
        }
        else {
            isBroken = 1;
            totalEggs++;
            result = low;
        }
    }

    value = value + totalEggs * 5;

    printf("result:%d count:%d totalEggs:%d isBroken:%d value:%d", result, count, totalEggs, isBroken, value);

    return 0;
}

```

2、 编写对应的指令序列（汇编语言）

因为指令码的格式是自己确定的且只有 15 条，所以只能手动编译，手动调整指令顺序和添加 NOP 指令以解决冲突（部分截图如下）：

```
0       ADDI r1 0 a//totalFloors
1       ADDI r2 0 2 //fallResisitance

2       ADDI r3 0 0//result
3       ADDI r4 0 0//count
4       ADDI r5 0 0//totalEggs
5       ADDI r6 0 0//isBroken
6       ADDI r7 0 0//value

7       ADD r8 r0 r1//high
8       ADDI r9 0 1//low
9       ADDI r10 0 0//stumbFloor
10      NOP

11      ADDI r4 0 1//count++
12      ADD r10 r8 r9
13      NOP
14      NOP
15      NOP
16      SRL r10 r10 1//stumbFloor = (high + low) / 2;
17      SUB r11 r8 r9
18      NOP
19      NOP
20      NOP
21      SRL r11 r11 1
22      NOP
23      NOP
24      NOP
25      ADD r7 r7 r11//value = value + (high - low) / 2;

26      CMP r2 r10
27      BN r0 2 9
28      NOP
29      NOP
30      NOP//if (stumbFloor <= fallResistance)
```

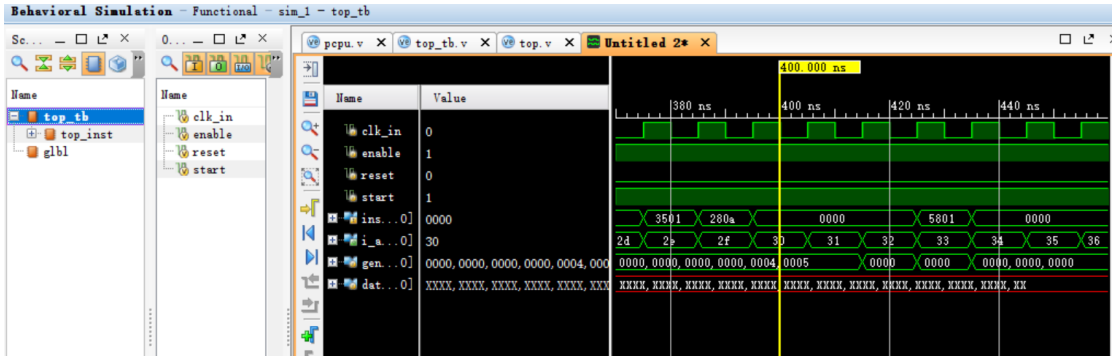
3、 将指令序列转化为机器码

编写了一个 C++程序将指令转为机器码，部分截图如下：

```
memory_initialization_radix = 16;
memory_initialization_vector =
310a
3202
3300
3400
3500
3600
3700
2801
3901
3a00
0000
3401
2a89
0000
0000
```

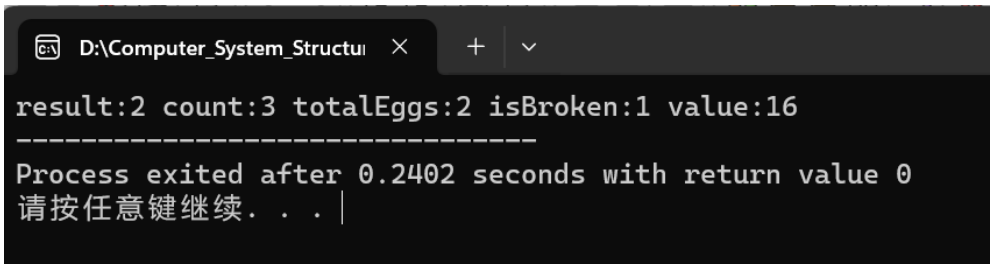
4、 将 coe 文件导入，并进行仿真

dmem 初始值未知，所以为 xxxxxxxx

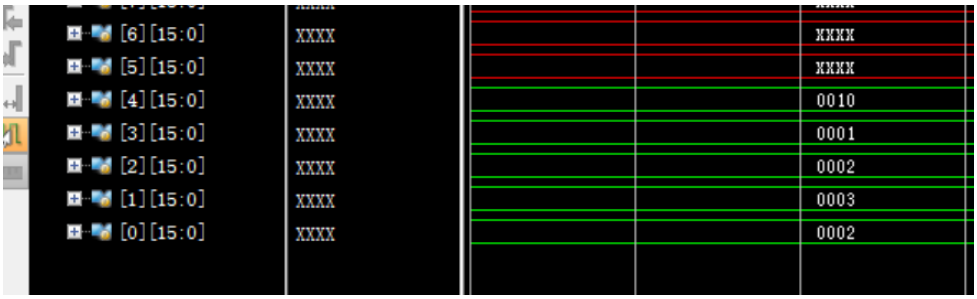


5、结果比较

C 语言程序结果：



流水线 CPU 计算结果：



可以看出数据存储器中存储了对应值的计算结果，测试出的耐摔值为 2，摔的次数为 3，总共摔碎的鸡蛋为 2，最后一个鸡蛋摔碎了，评估函数的值为 10（16 进制），与 C 语言程序计算的结果一致，CPU 和汇编语言正确。

七、实验验算程序下板测试过程与实现

1、重新配置顶层文件，加入分频和七段数码管以方便观察实验现象，如下：

```
module top(
    input clk_in,
    input enable,
    input reset,
    input start,
    output [7:0] o_seg,
    output [7:0] o_sel
    );
    wire [15:0] instruction;
    wire [15:0] datain;
    wire [7:0] i_addr;
    wire [7:0] d_addr;
    wire wena;
    wire [15:0] dataout;

    reg [24 : 0] cnt;
    always @ (posedge clk_in, negedge reset)
    if (reset)
        cnt <= 0;
    else
        cnt <= cnt + 1'b1;
    wire clk_cpu = cnt[24];
    seg7x16(clk_in, reset, 1, {8'b0000_0000, instruction}, o_seg, o_sel);
    pcpu pcpu_inst(clk_cpu, enable, reset, start, instruction, datain, i_addr, d_addr, wena, dataout);
    imem imem_inst(i_addr, instruction);
    dmem dmem_inst(clk_cpu, enable, wena, d_addr, dataout, datain);
endmodule

```

2、配置 xdc 文件，部分截图如下：

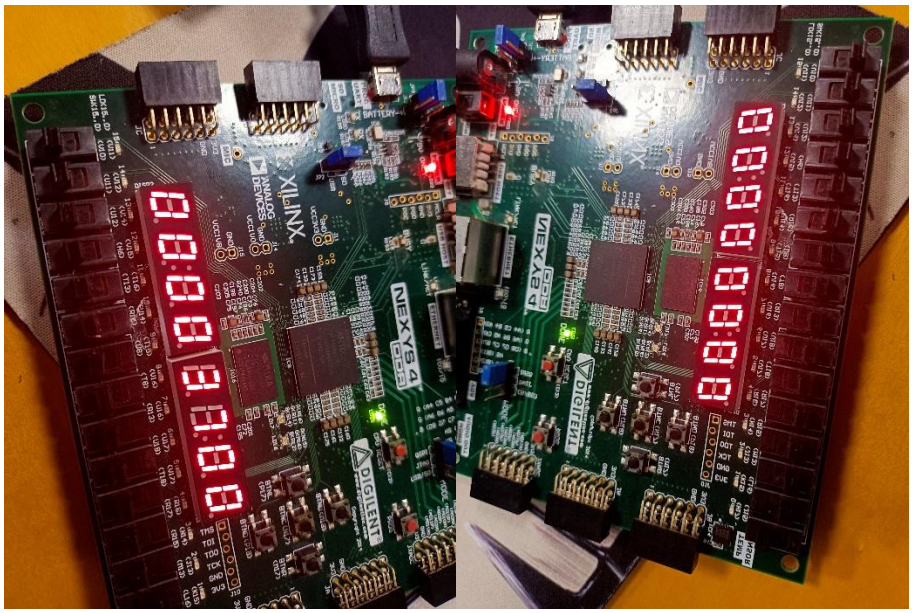
```
set_property IOSTANDARD LVCMOS33 [get_ports {o_seg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {o_seg[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports enable]
set_property IOSTANDARD LVCMOS33 [get_ports start]
set_property IOSTANDARD LVCMOS33 [get_ports clk_in]

create_clock -period 100.000 -name clk_pin -waveform {0.000 50.000} [get_ports clk_in]
set_input_delay -clock [get_clocks *] 1.000 [get_ports reset]
set_output_delay -clock [get_clocks *] 0.000 [get_ports -filter { NAME =~  "*" && DIRECTION == "OUT" }]
```

3、综合、布线、下板：

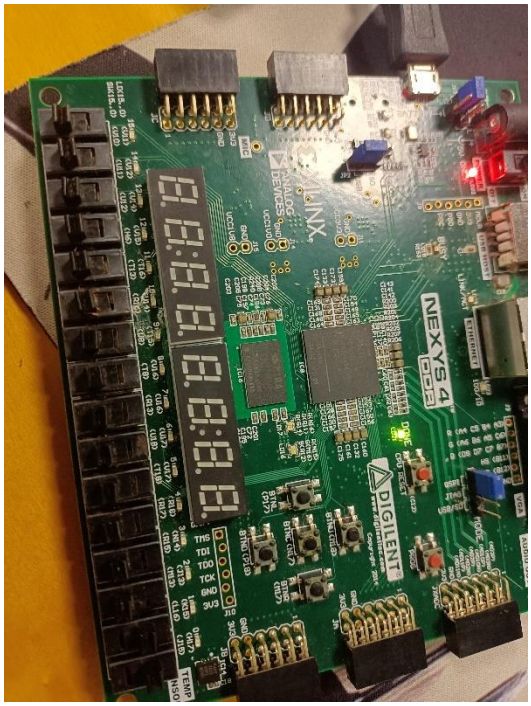
实验结果如下图：

成功显示第一条指令：

开启 enable 和 start 能够正常流动：



关闭 enable 能停止流动，reset 能复位



八、流水线的性能指标定性分析（包括：吞吐率、加速比、效率及相关与冲突分析、CPU 的运行时间及存储器空间的使用）

通过阅读时序报告可知延迟大概在 90ns 左右，因此时钟周期最短可设置在 90ns 附近。

1、 吞吐率：最大：1/t=11111111，实际：10893246

2、 加速比：n*m/(n+m-1) ，n 趋于无穷大时为 5，在测试程序中约为 4.901

2、 效率：n*m/(m*(m+n-1))，n 趋于无穷大时为 1，在测试程序中约为 0.9804

3、 冲突分析

在遇到先写后读冲突时，可以使用 NOP 指令进

行缓冲或者调整指令执行顺序来解决冲突；在执行跳转指令时，由于跳转地址和跳转标志位的产生延迟的缘故会引发冲突，不能通过调整顺序解决，只能使用 NOP 进行缓冲。因此在程序执行的过程中需要大量的 NOP 指令，CPU 空转，很大程度上影响了 CPU 的工作效率。

4、 CPU 运行时间：测试程序大约执行了几百条指令，运行时间在 20000ns 左右

5、 存储器空间的使用：测试程序只在最后 STORE 了 5 个值进数据存储器，其余计算均在寄存器，因此数据存储器空间消耗为 5，指令存储器消耗空间为指令的数量为 130。

九、总结与体会

在本次实验中，我选择并设计了 15 条指令，并实现了能够执行这 15 条指令的五段流水线 CPU。然后编写了对应的摔鸡蛋的验证程序，将其转化为汇编和机器码置于 CPU 上进行运算，验证其正确性以及性能。

在这个过程中，我理解和掌握了指令设计的基本方法和原理；掌握了简单的流水线 CPU 的设计和实现方法，明白其每个阶段的作用和意义；在编写汇编语言并转为机器码的过程中，我了解了通过 NOP 和调整指令顺序来解决流水线当中的冲突的方法。

这次实验使我对指令的设计，流水线 CPU 的设计和性能评估，以及指令冲突的解决都有了更加深刻的认识。

十、附件（所有程序）

cpu 代码：cpu 的代码置于代码文件夹中的 cpu 源码文件夹中，其中包含.v 文件，ip 核，xdc 配置文件，tb 测试文件（使用 tb 需要注释掉分频和七段数码管相关内容，并改变对应的时钟），以及时序报告。

验证文件：验证相关文件置于代码文件夹中的验证文件夹中，stumbEggs.c 是验证的 C 程序，stumbEggsComplication.txt 是对应的汇编语言，stumbEggsComplicationClear.txt 是去掉注释和空行的汇编语言，stumbEggsMachine.coe 是根据汇编语言转化的机器码，stumbEggsMachineLook.coe 是每行加了序号的机器码，stumbEggsMachine.coe 用于初始化 imem。TranslateToMachine.cpp 是一个能够将 stumbEggsComplicationClear.txt 汇编代码自动转化为 stumbEggsMachine.coe 机器语言代码初始化文件和 stumbEggsMachineLook.coe 文件的程序。