

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：

姓名：

指导教师：陆有军

日期：11月2日

目录

[一、 实验环境部署与硬件配置说明 3](#_Toc15278)

[二、 实验的总体结构 3](#_Toc20395)

[1、 IF（指令获取）阶段： 3](#_Toc1848)

[2、 ID（指令译码）阶段： 3](#_Toc13038)

[3、 EX（执行）阶段： 3](#_Toc29159)

[4、 MEM（访存）阶段： 3](#_Toc796)

[5、 WB（写回）阶段： 3](#_Toc26571)

[三、 总体架构部件的解释说明 5](#_Toc10439)

[1、 指令获取（IF）阶段： 5](#_Toc12522)

[2、 指令译码（ID）阶段： 6](#_Toc11949)

[3、 执行（EX）阶段： 8](#_Toc19602)

[4、 访存（MEM）阶段： 10](#_Toc24644)

[5、 写回（WB）阶段： 11](#_Toc1087)

[四、 实验仿真过程 12](#_Toc17805)

[1、 指令流水线的仿真过程 12](#_Toc23187)

[2、 构建tb文件 12](#_Toc11941)

[五、 实验仿真的波形图及某时刻寄存器值的物理意义 13](#_Toc24372)

[六、 流水线CPU实验性能验证模型 14](#_Toc21672)

[七、 实验验算程序下板测试过程与实现 17](#_Toc30356)

[1、 下板操作说明： 17](#_Toc741)

[2、 下板结果说明： 17](#_Toc23783)

[八、 流水线性能指标定性分析...................................................................................................](#_Toc30356)20

[1、 吞吐率分析 21](#_Toc4721)

[2、 加速比分析 21](#_Toc9683)

[3、 流水线效率分析 21](#_Toc5021)

[4、 指令相关与冲突分析 21](#_Toc18081)

[九、 总结与体会 21](#_Toc7135)

[十、 附件（见压缩包） 21](#_Toc26359)

1. 实验环境部署与硬件配置说明

系统环境：windows11

开发环境：vivado2016

硬件环境：Xilinix NEXYS4 DDR

1. 实验的总体结构

该流水线总体结构分为五个阶段，分别是 IF（指令获取）、ID（指令译码）、EX（执行）、MEM（访存）、WB （写回）。该流水线的目标是将此前的31条指令单周期cpu进行修改以实现对 31 条 MIPS 指令的五段流水线处理。

以下是实验的总体结构部分：

* 1. IF（指令获取）阶段：

功能：负责从指令存储器中获取指令。

控制逻辑：管理指令的流入，并将其传递到下一个阶段。

关键模块：pipe\_if模块，包含pcreg模块（维护和更新当前指令地址）和mux4\_32模块（4选1的32位多路选择器，用于选择下一条指令的地址）。

* 1. ID（指令译码）阶段：

功能：解析输入的指令并产生对应的控制信号。

控制信号类型：包括源寄存器读使能、目标寄存器读使能、目标寄存器写使能、数据存储器读写使能等。

实现方式：通过逻辑比较和组合逻辑判断当前指令的类型，并生成相应的控制信号。

* 1. EX（执行）阶段：

功能：执行ALU（算术逻辑单元）操作，进行算术和逻辑运算。

ALU操作：包括加法、减法、逻辑与、逻辑或等操作。

分支处理：处理分支和跳转条件，计算分支目标地址。

* 1. MEM（访存）阶段：

功能：处理访存指令，包括加载和存储。

数据存储器访问：访问数据存储器，读取或写入数据。

数据传递：将访存阶段计算的结果传递给写回阶段。

* 1. WB（写回）阶段：

功能：将计算结果写回寄存器文件，更新寄存器的值。

寄存器更新：根据执行阶段的结果，更新目标寄存器的值。

一些其他的重要模块如下：

流水线寄存器模块：

功能：在流水线中存储并传递各阶段的结果。

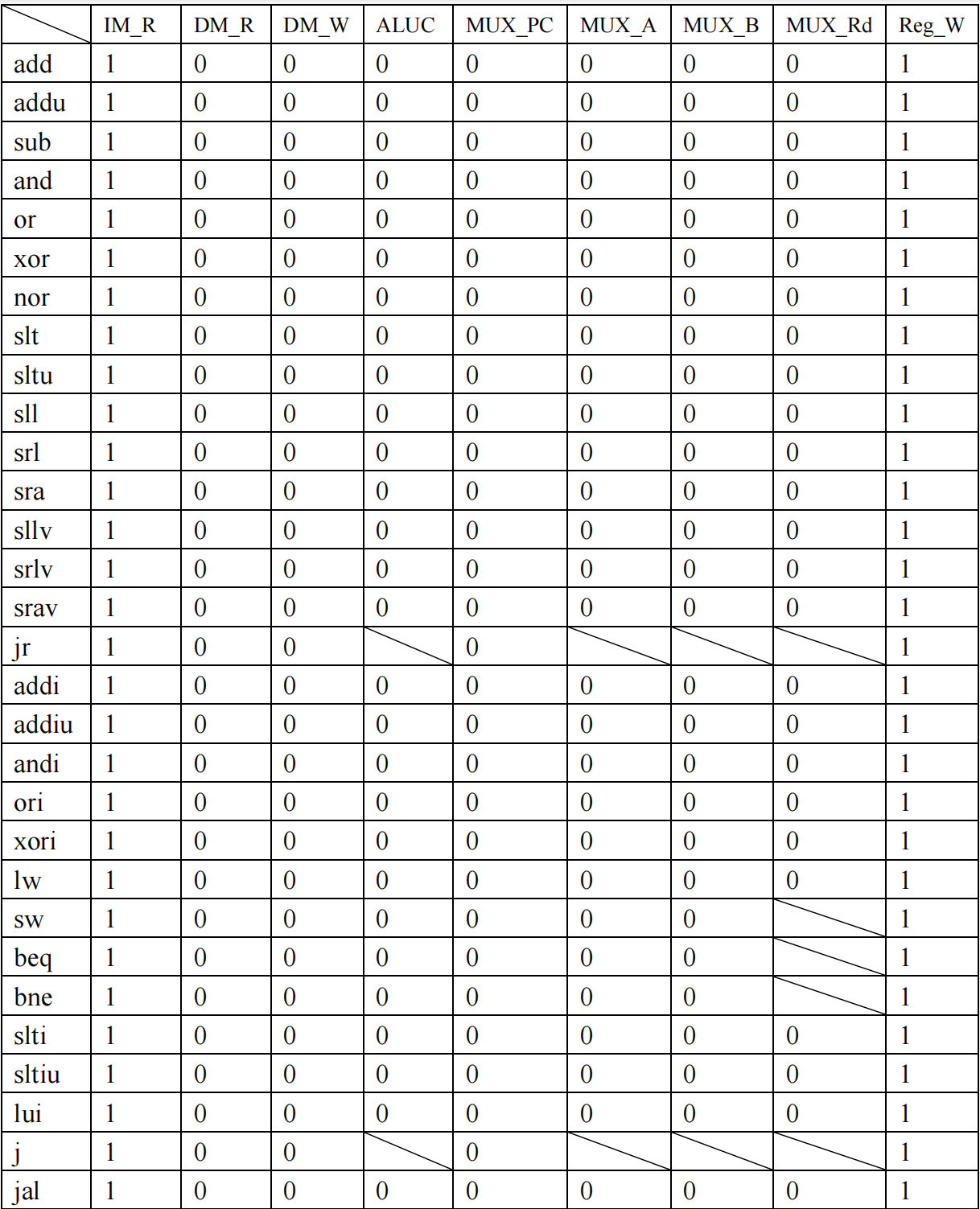
关键模块：pipe\_if\_id模块：存储并传递指令取指阶段的结果pipe\_id\_ex模块：存储并传递译码阶段的结果到执行阶段。

数据冲突与暂停机制模块：

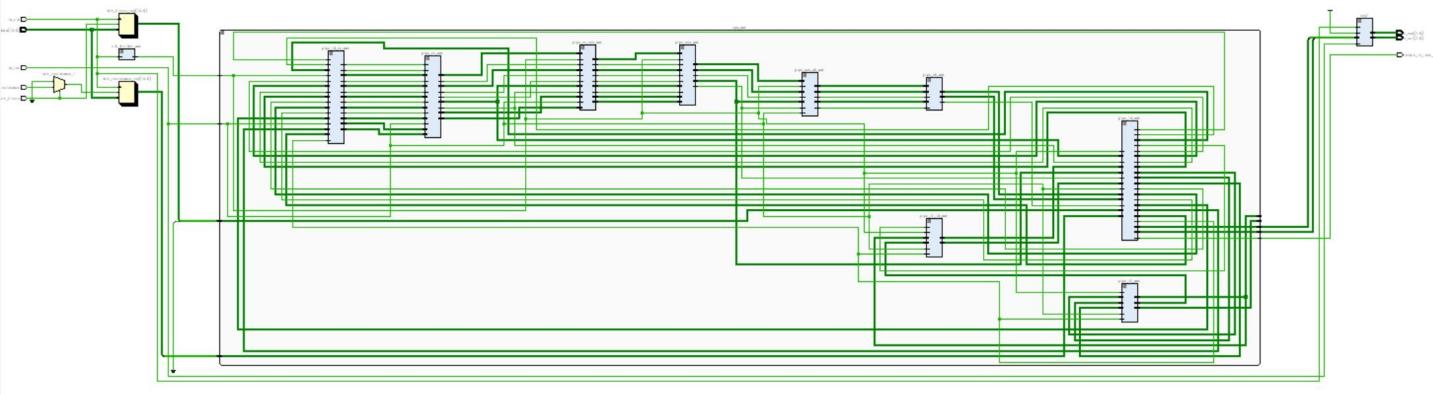
数据冲突类型：ID-EX、ID-MEM。

暂停机制：通过stall信号模块检测并处理数据冲突，实现对应的暂停机制，确保数据的正确传递和处理。

控制信号表如下：



cpu的总体构建模式图如下：



1. 总体架构部件的解释说明
   1. 指令获取（IF）阶段：

指令内存访问模块（pipe\_if）：这是流水线中的首个阶段，负责从指令存储器中读取指令。它包含pcreg模块以更新和维护当前指令地址，以及mux4\_32模块来根据控制信号选择下一条指令的地址。

pipe\_if模块的定义如下：

module pipe\_if(

    input           in\_clk,

    input           in\_rst,

    input           in\_stall,

    input   [31:0]  in\_pc\_jaddr,

    input   [31:0]  in\_pc\_baddr,

    input   [1:0]   in\_pc\_sel,

    output  [31:0]  out\_pc,

    output  [31:0]  out\_npc,

    output  [31:0]  out\_instruction

    );

mux4\_32 模块：该模块是一个 4 选 1 的 32 位多路选择器，根据 in\_pc\_sel 选择不同的地址输入，输出到 next\_pc 中，作为下一条指令的地址。其中，out\_npc 通过简单加法操作得到，表示当前指令的地址加上 4。模块定义如下：

module mux4\_32(

    input [31:0] d0,

    input [31:0] d1,

    input [31:0] d2,

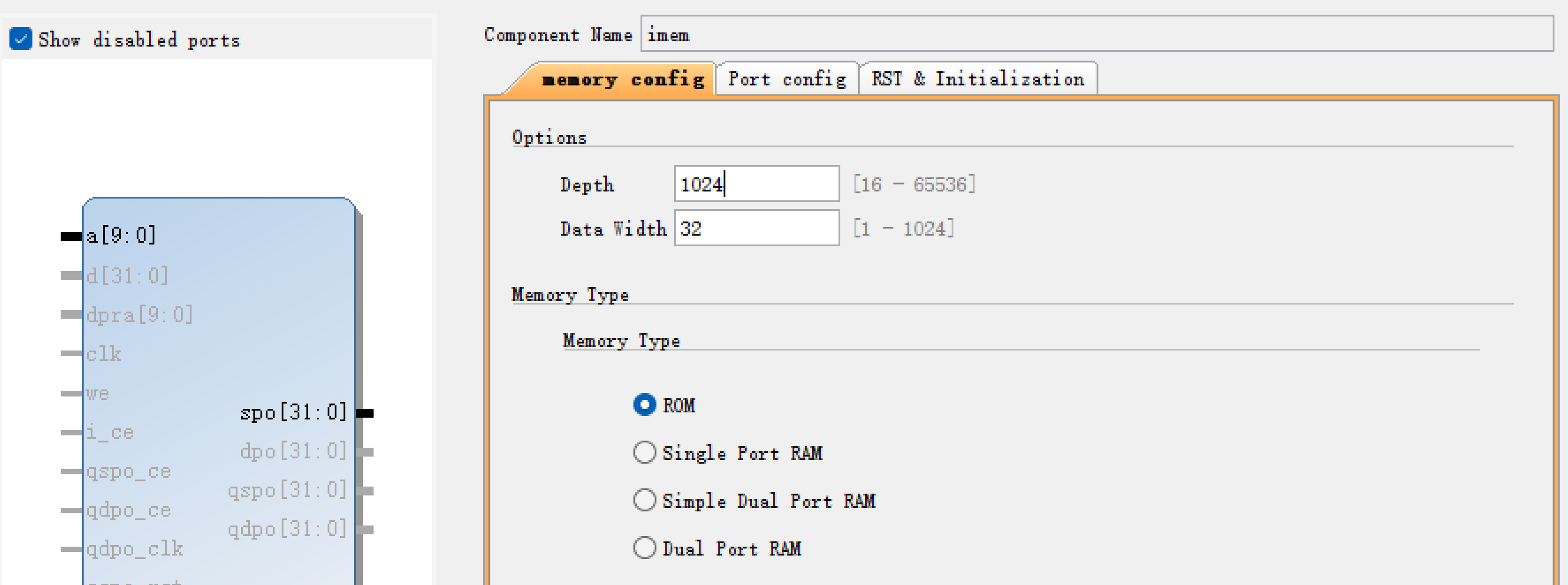
    input [31:0] d3,

    input [1:0]  s,

    output reg [31:0] y

    );

imem 模块：用来读取对cpu的指令，该部分通过使用ip核来实现：



* 1. 指令译码（ID）阶段：

译码模块：负责解析从IF阶段传来的指令，提取操作码、功能码以及源、目标寄存器的地址等关键信息。它进一步通过寄存器堆模块读取源寄存器的数据，并通过控制器模块生成相应的控制信号，这些信号将指导流水线后续阶段的操作。

module pipe\_id(

    input           in\_clk,

    input           in\_rst,

    input   [31:0]  in\_npc,

    input   [31:0]  in\_instruction,

    input   [4:0]   in\_ex\_waddr,

    input   [4:0]   in\_mem\_waddr,

    input           in\_ex\_wena,

    input           in\_mem\_wena,

    input   [4:0]   in\_wb\_reg\_addr,

    input           in\_wb\_reg\_ena,

    input   [31:0]  in\_wb\_reg\_data,

    input   [31:0]  init\_floors,

    input   [31:0]  init\_resistance,

    output  [31:0]      out\_rs\_data,

    output  [31:0]      out\_rt\_data,

    output  [4:0]       out\_rd\_waddr,

    output              out\_rd\_sel,

    output              out\_rd\_wena,

    output  [31:0]      out\_immed,

    output  [31:0]      out\_shamt,

    output              out\_dmem\_ena,

    output              out\_dmem\_wena,

    output  [1:0]       out\_dmem\_type,

    output  [31:0]      out\_pc\_baddr,

    output  [31:0]      out\_pc\_jaddr,

    output  [1:0]       out\_pc\_sel,

    output              out\_alu\_a\_sel,

    output              out\_alu\_b\_sel,

    output [3:0]        out\_alu\_sel,

    output              out\_stall,

    output              out\_branch,

    output  [31:0]      result\_attempt\_count,

    output  [31:0]      result\_broken\_count,

    output              result\_is\_last\_broken,

    output [15:0] out\_up\_floor,

    output [15:0] out\_down\_floor

    );

段间寄存模块（pipe\_if\_id）：此模块作为IF阶段和ID阶段之间的流水线寄存器，用于存储并传递从IF阶段获取的指令及其相关信息，确保数据在流水线中的正确传递。

pipe\_if\_id模块的定义如下：

module pipe\_if\_id(

    input               in\_clk,

    input               in\_rst,

    input               in\_stall,

    input               in\_branch,

    input       [31:0]  in\_npc,

    input       [31:0]  in\_instruction,

    output reg  [31:0]  out\_npc,

    output reg  [31:0]  out\_instruction

    );

控制器模块：核心功能是解析输入的指令并产生对应的控制信号。通过检测指令的操作码和功能码，该模块能够确定指令类型，并生成如寄存器读写使能信号、ALU操作选择信号等，以协调流水线的运行。模块的定义如下：

module controller(

    input           in\_branch,

    input [31:0]    in\_instruction,

    output          out\_rs\_rena,

    output          out\_rt\_rena,

    output          out\_rd\_wena,

    output [4:0]    out\_rd\_addr,

    output          out\_rd\_sel,

    output          out\_dmem\_ena,

    output          out\_dmem\_wena,

    output [1:0]    out\_dmem\_type,

    output          out\_ext\_signed,

    output          out\_alu\_a\_sel,

    output          out\_alu\_b\_sel,

    output [3:0]    out\_alu\_sel,

    output [1:0]    out\_pc\_sel

    );

stall 模块：该模块的机制是为了解决流水线中可能发生的数据冲突导致的暂停情况。在每个时钟周期的下降沿，模块会检查当前是否需要暂停流水线。 模块的定义如下：

module stall(

    input           in\_clk,

    input           in\_rst,

    input   [4:0]   in\_rs\_addr,

    input   [4:0]   in\_rt\_addr,

    input           in\_rs\_rena,

    input           in\_rt\_rena,

    input           in\_ex\_wena,

    input           in\_mem\_wena,

    input   [4:0]   in\_ex\_waddr,

    input   [4:0]   in\_mem\_waddr,

    output  reg     out\_stall

    );

* 1. 执行（EX）阶段：

流水线的ID到EX阶段的寄存器模块：在时钟上升沿或复位信号上升沿时，根据输入的控制信号和数据，更新流水线的输出信号，为下一个阶段的执行提供必要的输入。模块中包含了对输入信号的判断和处理逻辑，以及根据输入更新输出的逻辑。模块的定义如下：

module pipe\_id\_ex(

    input               in\_clk,

    input               in\_rst,

    input               in\_dmem\_ena,

    input               in\_dmem\_wena,

    input       [1:0]   in\_dmem\_type,

    input       [31:0]  in\_rs\_data,

    input       [31:0]  in\_rt\_data,

    input       [4:0]   in\_rd\_waddr,

    input               in\_rd\_sel,

    input               in\_rd\_wena,

    input       [31:0]  in\_immed,

    input       [31:0]  in\_shamt,

    input               in\_alu\_a\_sel,

    input               in\_alu\_b\_sel,

    input       [3:0]   in\_alu\_sel,

    input               in\_stall,

    output reg          out\_dmem\_ena,

    output reg          out\_dmem\_wena,

    output reg  [1:0]   out\_dmem\_type,

    output reg  [31:0]  out\_rs\_data,

    output reg  [31:0]  out\_rt\_data,

    output reg  [4:0]   out\_rd\_waddr,

    output reg          out\_rd\_sel,

    output reg          out\_rd\_wena,

    output reg  [31:0]  out\_immed,

    output reg  [31:0]  out\_shamt,

    output reg          out\_alu\_a\_sel,

    output reg          out\_alu\_b\_sel,

    output reg  [3:0]   out\_alu\_sel

    );

执行模块（pipe\_ex）：此阶段主要负责执行ALU（算术逻辑单元）的运算。它接收来自ID阶段的控制信号和操作数，执行算术和逻辑运算，并输出结果。同时，它还负责处理分支和跳转条件，计算分支目标地址。模块的定义如下：

module pipe\_ex(

    input           in\_rst,

    input           in\_dmem\_ena,

    input           in\_dmem\_wena,

    input   [1:0]   in\_dmem\_type,

    input   [31:0]  in\_rs\_data,

    input   [31:0]  in\_rt\_data,

    input   [4:0]   in\_rd\_waddr,

    input           in\_rd\_sel,

    input           in\_rd\_wena,

    input   [31:0]  in\_immed,

    input   [31:0]  in\_shamt,

    input           in\_alu\_a\_sel,

    input           in\_alu\_b\_sel,

    input   [3:0]   in\_alu\_sel,

    output          out\_dmem\_ena,

    output          out\_dmem\_wena,

    output  [1:0]   out\_dmem\_type,

    output  [31:0]  out\_rs\_data,

    output  [31:0]  out\_rt\_data,

    output  [4:0]   out\_rd\_waddr,

    output          out\_rd\_sel,

    output          out\_rd\_wena,

    output  [31:0]  out\_alu\_result

    );

ALU模块：运算数选择方面，输入的ALU运算数根据选择信号（in\_alu\_a\_sel、in\_alu\_b\_sel）进行选择，可以是寄存器数据或者立即数。模块的定义如下：

module alu(

    input   [31:0]  a,

    input   [31:0]  b,

    output  [31:0]  y,

    input   [3:0]   aluc,

    output          zero,

    output          carry,

    output          negative,

    output          overflow

    );

* 1. 访存（MEM）阶段：

EX到MEM段间寄存模块：在EX和MEM阶段之间传递控制信号和计算结果，确保访存操作能够正确执行。模块的定义如下：

module pipe\_ex\_mem(

    input               in\_clk,

    input               in\_rst,

    input               in\_dmem\_ena,

    input               in\_dmem\_wena,

    input   [1:0]       in\_dmem\_type,

    input   [31:0]      in\_rs\_data,

    input   [31:0]      in\_rt\_data,

    input   [4:0]       in\_rd\_waddr,

    input               in\_rd\_sel,

    input               in\_rd\_wena,

    input   [31:0]      in\_alu\_result,

    output reg          out\_dmem\_ena,

    output reg          out\_dmem\_wena,

    output reg [1:0]    out\_dmem\_type,

    output reg [31:0]   out\_rs\_data,

    output reg [31:0]   out\_rt\_data,

    output reg [4:0]    out\_rd\_waddr,

    output reg          out\_rd\_sel,

    output reg          out\_rd\_wena,

    output reg [31:0]   out\_alu\_result

    );

访存模块：处理访存指令，包括加载和存储操作。它接收来自EX阶段的数据和控制信号，访问数据存储器，读取或写入数据，并将结果传递给写回阶段。模块的定义如下：

module pipe\_mem(

    input           in\_clk,

    input           in\_dmem\_ena,

    input           in\_dmem\_wena,

    input   [1:0]   in\_dmem\_type,

    input   [31:0]  in\_rs\_data,

    input   [31:0]  in\_rt\_data,

    input   [4:0]   in\_rd\_waddr,

    input           in\_rd\_sel,

    input           in\_rd\_wena,

    input   [31:0]  in\_alu\_result,

    output  [4:0]   out\_rd\_waddr,

    output          out\_rd\_sel,

    output          out\_rd\_wena,

    output  [31:0]  out\_alu\_result,

    output  [31:0]  out\_dmem\_data

    );

* 1. 写回（WB）阶段：

MEM到WB段间寄存模块：在MEM和WB阶段之间传递最终的计算结果和控制信号，确保数据能够正确写回寄存器。模块的定义如下：

module pipe\_mem\_wb(

    input               in\_clk,

    input               in\_rst,

    input       [4:0]   in\_rd\_waddr,

    input               in\_rd\_sel,

    input               in\_rd\_wena,

    input       [31:0]  in\_alu\_result,

    input       [31:0]  in\_dmem\_data,

    output reg  [4:0]   out\_rd\_waddr,

    output reg          out\_rd\_wena,

    output reg          out\_rd\_sel,

    output reg  [31:0]  out\_alu\_result,

    output reg  [31:0]  out\_dmem\_data

    );

写回模块：将计算结果写回寄存器文件，更新寄存器的值。它接收来自MEM阶段的数据和控制信号，并更新寄存器堆中的相应寄存器。模块的定义如下：

module pipe\_wb(

    input   [4:0]   in\_rd\_waddr,

    input           in\_rd\_wena,

    input           in\_rd\_sel,

    input   [31:0]  in\_alu\_result,

    input   [31:0]  in\_dmem\_data,

    output  [4:0]   out\_rd\_waddr,

    output          out\_rd\_wena,

    output  [31:0]  out\_rd\_wdata

    );

1. 实验仿真过程
   1. 指令流水线的仿真过程

选用了比萨塔摔鸡蛋程序作为测试案例，在将汇编程序完成后，使用mars4将其转为coe文件后初始化imem的IP核，作为输入的指令从而不需要在tb文件中占用大量的段落来构建要输出的指令。

* 1. 构建tb文件

为了在一次下板过程中进行多组数据的调试，编写了testbench程序。此程序用来实现了以下功能：

1.初始化的层高与耐摔值从外层模块传入。

2.在reset时直接初始化对应的regfile寄存器。

3.可以将每个时钟周期（clk）时的CPU值输出到log文件中，便于后续分析和调试。

tb文件的部分核心代码如下：

module testbench(

    );

    reg         clk;

    reg         rst;

    integer     regfile\_output;

    reg [15:0]  init\_data;

    reg         is\_init\_floors;

    reg         is\_init\_resistance;

    wire        last\_broken;

    initial

    begin

        regfile\_output = $fopen("regfile\_output.txt");

        clk = 0;

        rst = 1;

        #20 init\_data = 16'd00128;

        #20 is\_init\_floors = 1'b1;

        #20 is\_init\_floors = 1'b0;

        #20 init\_data = 16'd0020;

        #20 is\_init\_resistance = 1'b1;

        #20 is\_init\_resistance = 1'b0;

        #20 rst = 0;

    end

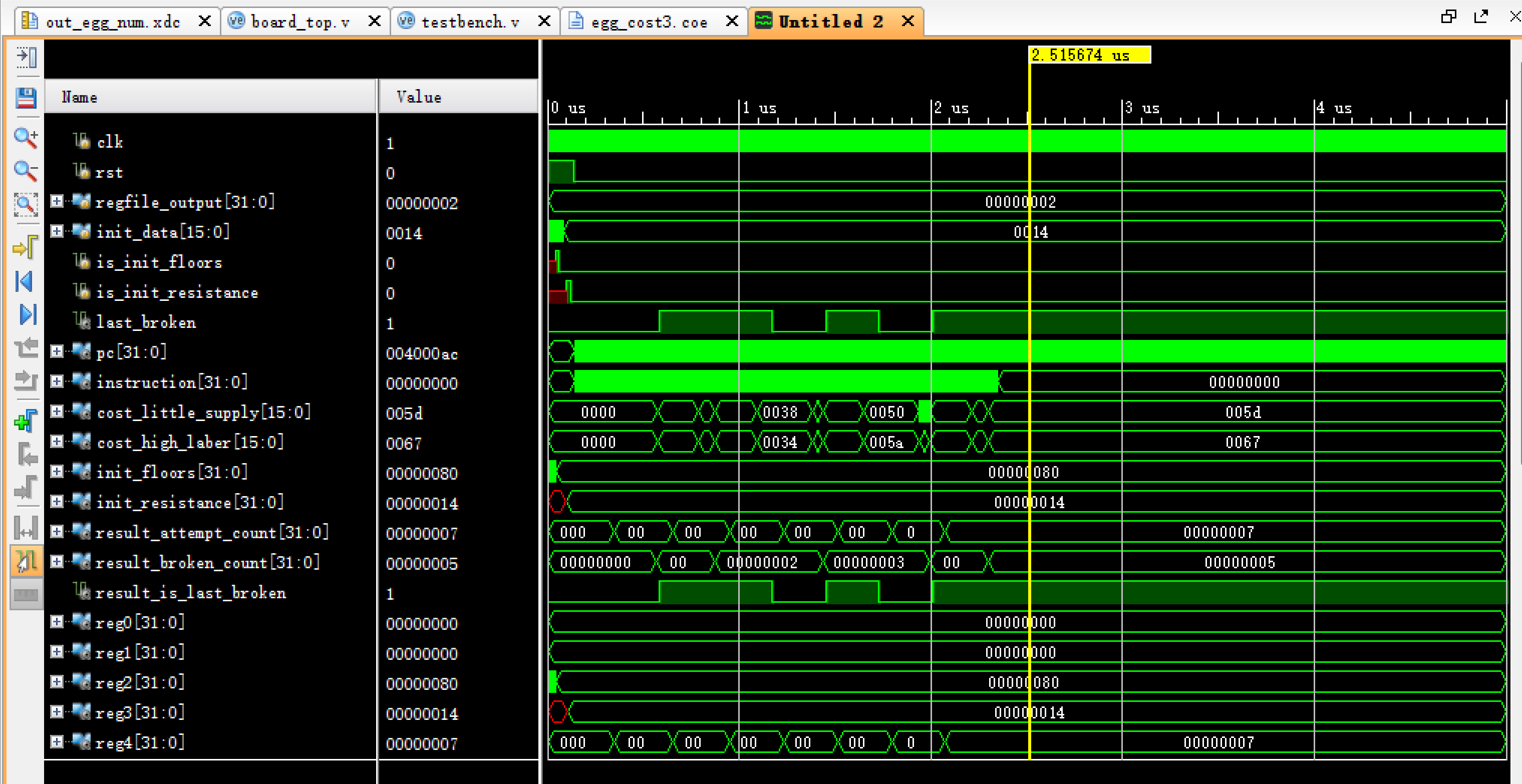
    always

    begin

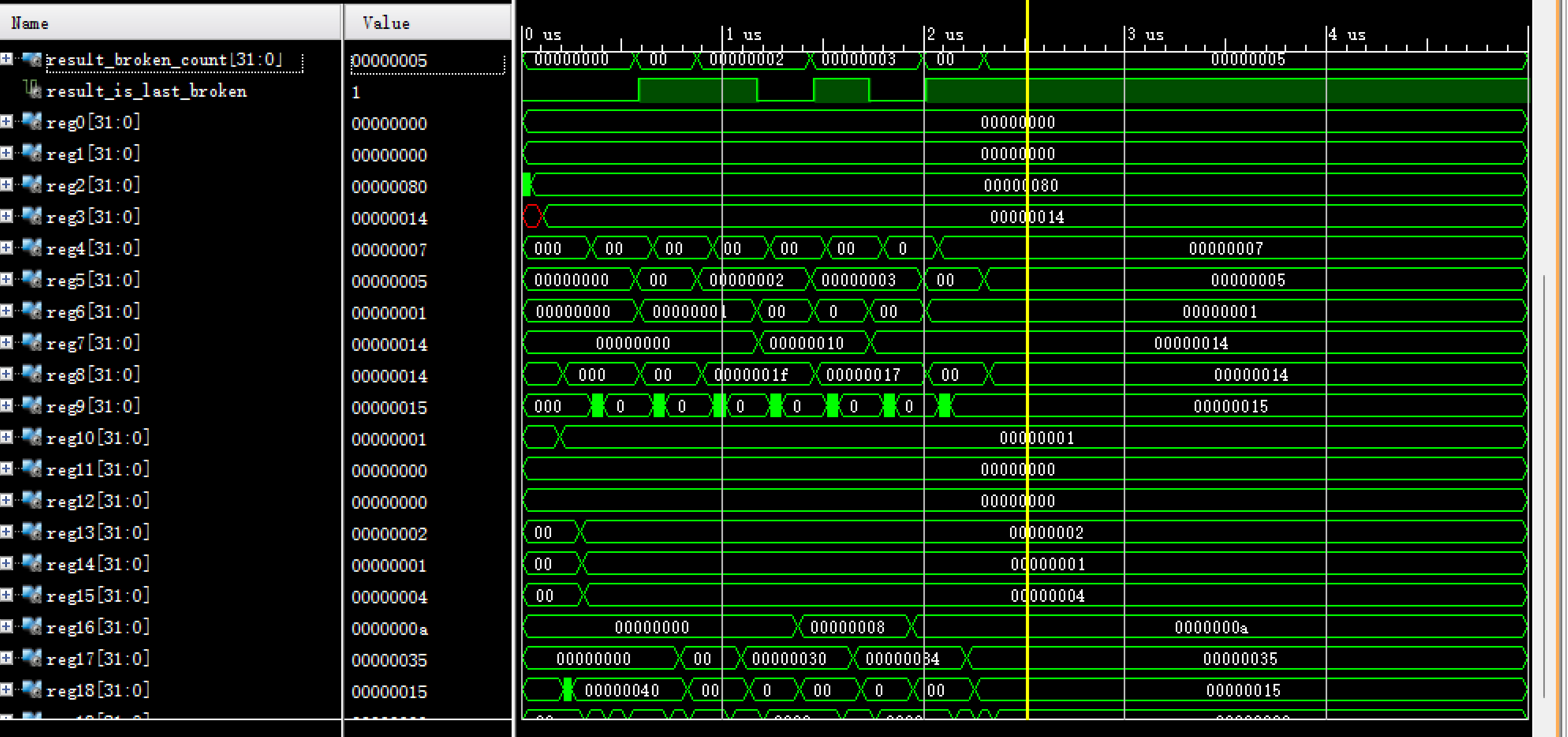
        #1 clk = ~clk;

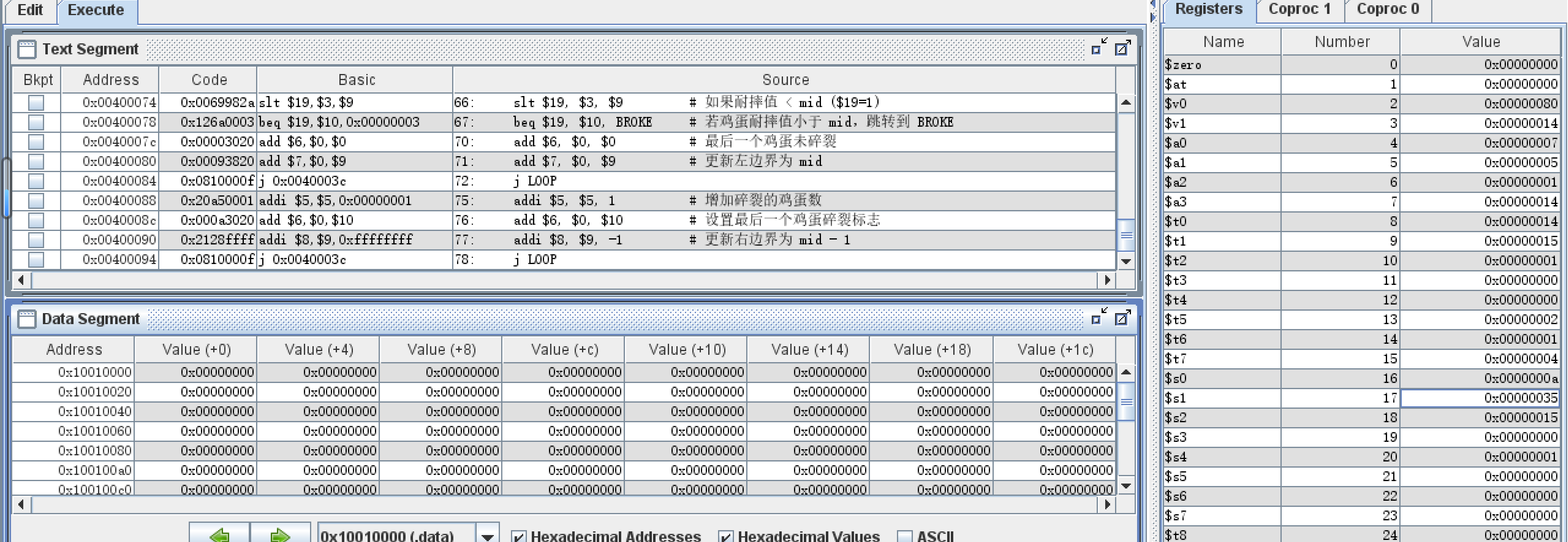
    end

1. 实验仿真的波形图及某时刻寄存器值的物理意义



仿真结果如上图：输入的总的楼层数init\_floors为128，耐摔值init\_resistance为20，得到的结果是实验的尝试次数result\_attempt\_count为7次，鸡蛋摔碎的数量result\_broken\_count为5个，两个不同时期的消耗cost\_little\_supply与cost\_high\_laber分别为93和103





各寄存器的值与使用mars4汇编器得到的结果相同：

* 输入的楼层数和耐摔值存储在2号和3号寄存器中，分别为128和20；
* 实验的尝试次数和鸡蛋摔碎的数量存储在4号和5号寄存器中，分别为7次和5个；
* 最后的一次是否摔碎为6号寄存器，此时为1，说明鸡蛋摔碎；
* 二分法的上下边界和中值分别是7号、8号和9号寄存器；
* 11号和12号寄存器本想作为两个时期的成本，单因为要使用乘法计算，本次实验中没有考虑到mul指令，需要不断作加法循环来计算，导致vivado卡死，于是就只能在顶层模块中形式化得由上下楼的层数和鸡蛋摔碎数来计算；
* 13、14和15号寄存器用来记录不同时期的上下楼成本和鸡蛋成本
* 总共上楼层数与下楼层数存储在16号和17号寄存器中，分别为10层和53层。
* 18号寄存器用来记录上一次实验的楼层数，作为累加总的上下楼层数的中间变量，初始为 (n + 1) / 2，n为输入的总楼层数

1. 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。用你的模型评价该游戏在两个不同历史时期花费的总成本f=mp1+np2+hp3，m为上的楼层总数，n为下的楼层总数，h为摔破的鸡蛋总数，p1为每上1层的成本，p2为每下1层的成本，p3为每个鸡蛋的成本；在物质匮乏时期，p1=2，p2=1，p3=4；在人力成本增长时期，p1=4，p2=1，p3=2。请使用C语言设计该验证模型的算法，并把C语言汇编为MIPS或RISC-V指令汇编程序，同时利用编译器生成MIPS或RISC-V指令集可执行目标程序。  
  
根据要求可以写出cpp：

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

int main()

{

    int init\_floor;

    int init\_resistance;

    printf("请输入楼层数：");

    scanf("%d", &init\_floor);

    printf("请输入耐摔值：");

    scanf("%d", &init\_resistance);

    int result\_attempt\_cnt = 0;

    int result\_broken\_cnt = 0;

    bool result\_is\_last\_broken = 0;

    int up\_floor = 0, down\_floor=0,last\_try\_floor=(init\_floor+1)/2;

    int l = 0, r = init\_floor;

    int cost\_little\_supply = 0, cost\_high\_laber = 0;

    while (l < r) {

        result\_attempt\_cnt++;

        int mid = (l + r + 1) / 2;

        if (mid > last\_try\_floor)

            up\_floor += (mid - last\_try\_floor);

        else

            down\_floor += (last\_try\_floor - mid);

        if (mid > init\_resistance) {

            result\_is\_last\_broken = 1;

            r = mid - 1;

            result\_broken\_cnt++;

        }

        else {

            result\_is\_last\_broken = 0;

            l = mid;

        }

        last\_try\_floor = mid;

    }

    cost\_little\_supply = 2 \* up\_floor + 1 \* down\_floor + 4 \* result\_broken\_cnt;

    cost\_high\_laber= 4 \* up\_floor + 1 \* down\_floor + 2 \* result\_broken\_cnt;

    printf("共尝试%d次，摔碎鸡蛋%d个，最后一次鸡蛋", result\_attempt\_cnt, result\_broken\_cnt);

    printf((result\_is\_last\_broken ? "摔碎\n" : "没碎\n"));

    printf("共上楼%d层，下楼%d层\n", up\_floor, down\_floor);

    printf("物质匮乏期的总成本为：%x，", cost\_little\_supply);

    printf("人力成本增长期的总成本为：%x", cost\_high\_laber);

    return 0;

}

算法分析：在这里使用的算法思想是二分法搜索，逐次将楼层细分直至找到可以将鸡蛋摔破的最小楼层。与传统的二分法不同这里需要注意在最后一次分治时，鸡蛋直到摔破才能判断其耐摔值所以还要将求得的楼层数减一，并且每次二分时将下界要保持不变，否则可能会导致最后的结果比真实值小。时间复杂度：O(log)

得到的汇编文件为：

# $2 (楼层上限): 建筑物的最大楼层数 (1024)

# $3 (鸡蛋耐摔值): 鸡蛋的最大可忍受摔落楼层 (65)

# $4 (总摔鸡蛋次数): 记录总共的摔鸡蛋次数

# $5 (碎裂的鸡蛋数): 记录摔碎的鸡蛋数

# $6 (最后一个鸡蛋是否碎裂): 标志位，1 表示鸡蛋碎裂，0 表示未碎裂

# $7 (左边搜索边界): 二分查找时的左边搜索边界

# $8 (右边搜索边界): 二分查找时的右边搜索边界

# $9 (中间值): 二分查找时的中间值

# $10 (常数1): 常数值1

# $11 (物质匮乏时期总成本): 记录物质匮乏时期的总成本

# $12 (人力成本增长时期总成本): 记录人力成本增长时期的总成本

# $13 (p1): 上楼成本

# $14 (p2): 下楼成本

# $15 (p3): 鸡蛋成本

# $16 (上楼层次数 m): 累加上楼的总层数

# $17 (下楼层次数 n): 累加下楼的总层数

# $18 (上次尝试的楼层): 上一次尝试楼层的值，初始为 (n + 1) / 2

.text

    addi $2, $0, 128      # 设置楼层上限 n

    addi $3, $0, 20       # 设置鸡蛋耐摔值 m

    addi $10, $0, 1         # 常数 1

    add $7, $0, $0          # 左边界 l = 0

    add $8, $0, $2          # 右边界 r = n

    addi $18, $2, 1         # 上次尝试楼层初始值为 (n + 1) / 2

    sra $18, $18, 1         # $18 = (n + 1) / 2

    add $11, $0, $0         # 初始化物质匮乏时期总成本为0

    add $12, $0, $0         # 初始化人力成本增长时期总成本为0

    add $16, $0, $0         # 初始化上楼层次数为0

    add $17, $0, $0         # 初始化下楼层次数为0

    add $5, $0, $0          # 初始化摔碎的鸡蛋次数为0

    # 设置物质匮乏时期成本参数 (p1=2, p2=1, p3=4)

    addi $13, $0, 2         # p1 = 2

    addi $14, $0, 1         # p2 = 1

    addi $15, $0, 4         # p3 = 4

LOOP:

    slt $19, $7, $8         # 检查左边界是否小于右边界 ($19=1 表示 l < r)

    beq $19, $0, END  # 如果 l >= r，跳转到 CALC\_COST

    addi $4, $4, 1          # 增加总摔鸡蛋次数

    add $9, $7, $8          # 计算中间值 (l + r + 1) / 2

    addi $9, $9, 1

    sra $9, $9, 1           # mid = (l + r + 1) / 2

    # 计算上楼或下楼的层数

    slt $19, $18, $9        # 如果 $18 < $9，表示上楼

    bne $19, $0, GOING\_UP   # 若上楼，跳转到 GOING\_UP

    sub $20, $18, $9        # 下楼层数 = last\_try\_floor - mid

    add $17, $17, $20       # 累加下楼层次数

    j UPDATE\_LAST\_FLOOR

GOING\_UP:

    sub $20, $9, $18        # 上楼层数 = mid - last\_try\_floor

    add $16, $16, $20       # 累加上楼层次数

UPDATE\_LAST\_FLOOR:

    add $18, $0, $9         # 更新上次尝试的楼层为 mid

    # 判断鸡蛋是否碎裂

    slt $19, $3, $9         # 如果耐摔值 < mid ($19=1)

    beq $19, $10, BROKE     # 若鸡蛋耐摔值小于 mid，跳转到 BROKE

RESIST:

    add $6, $0, $0          # 最后一个鸡蛋未碎裂

    add $7, $0, $9          # 更新左边界为 mid

j LOOP

BROKE:

    addi $5, $5, 1          # 增加碎裂的鸡蛋数

    add $6, $0, $10         # 设置最后一个鸡蛋碎裂标志

    addi $8, $9, -1         # 更新右边界为 mid - 1

    j LOOP

END:

    #j END                   # 无限循环

.text  ends

1. 实验验算程序下板测试过程与实现
   1. 下板操作说明：

按p17，高电平时将输入的总楼层数传入2号寄存器；

按m17，高电平时将输入的耐摔值传入3号寄存器；

按n17，高电平时确认输入完成，开始运算；

j15-v10：用于输入总楼层数和耐摔值；

H17：显示最后一次鸡蛋是否摔碎；

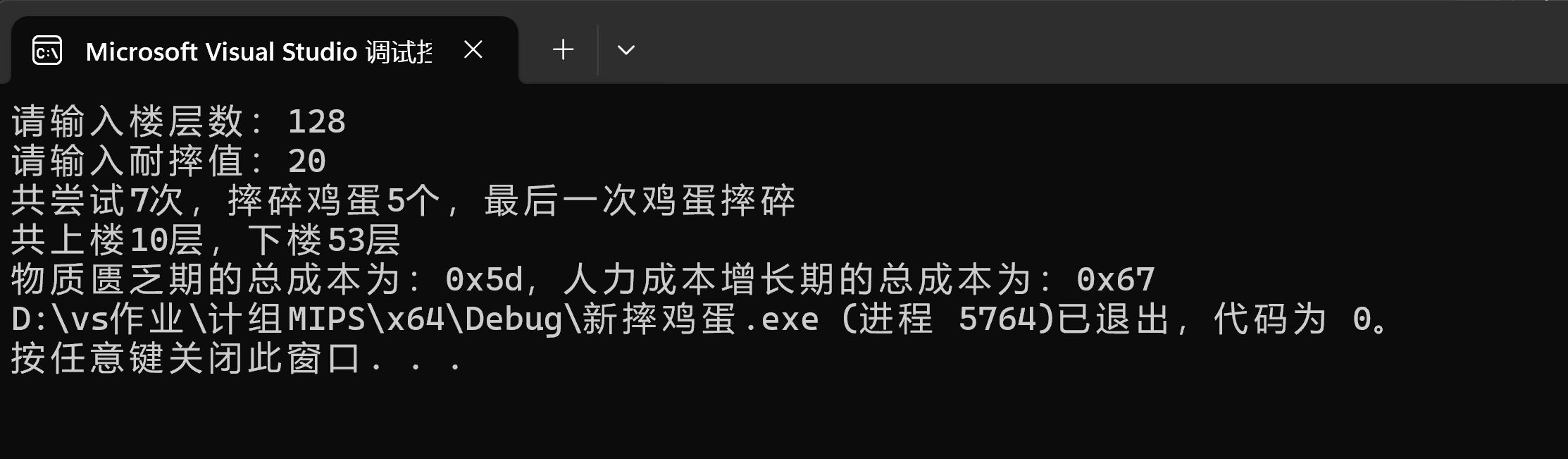
八位数码管：前4位是十六进制的物质匮乏时期总成本，后4位为人力成本增长时期总成本。

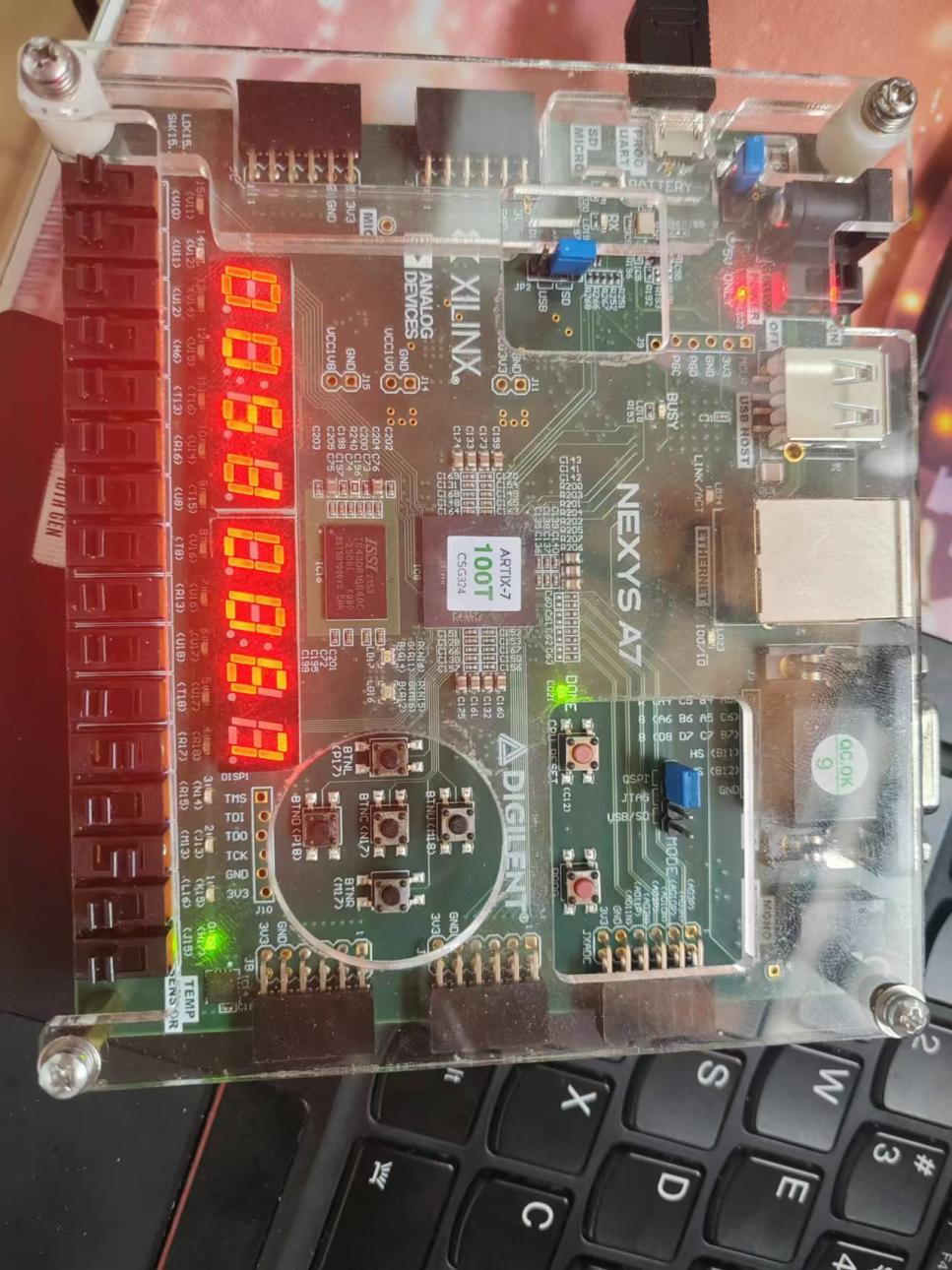
* 1. 下板结果说明：

总楼层数128，耐摔值 20:

cpp运行的结果和下板结果如下图：

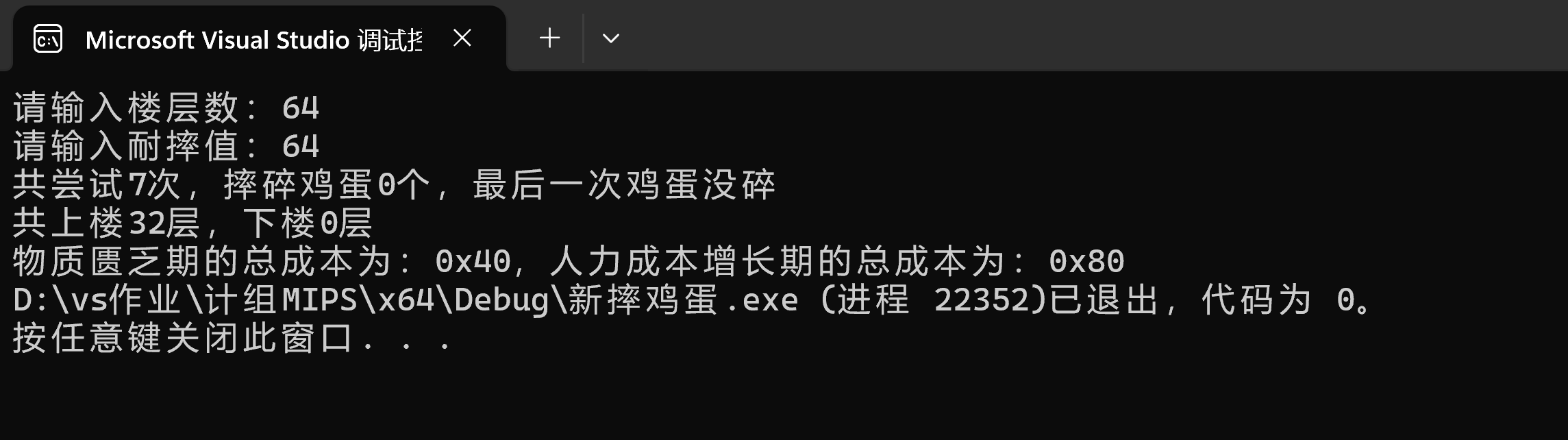
可以得到物质匮乏时期总成本为0x5d，人力成本增长时期总成本为0x67，最后一次鸡蛋摔碎，下板结果与cpp得到的结果相同。

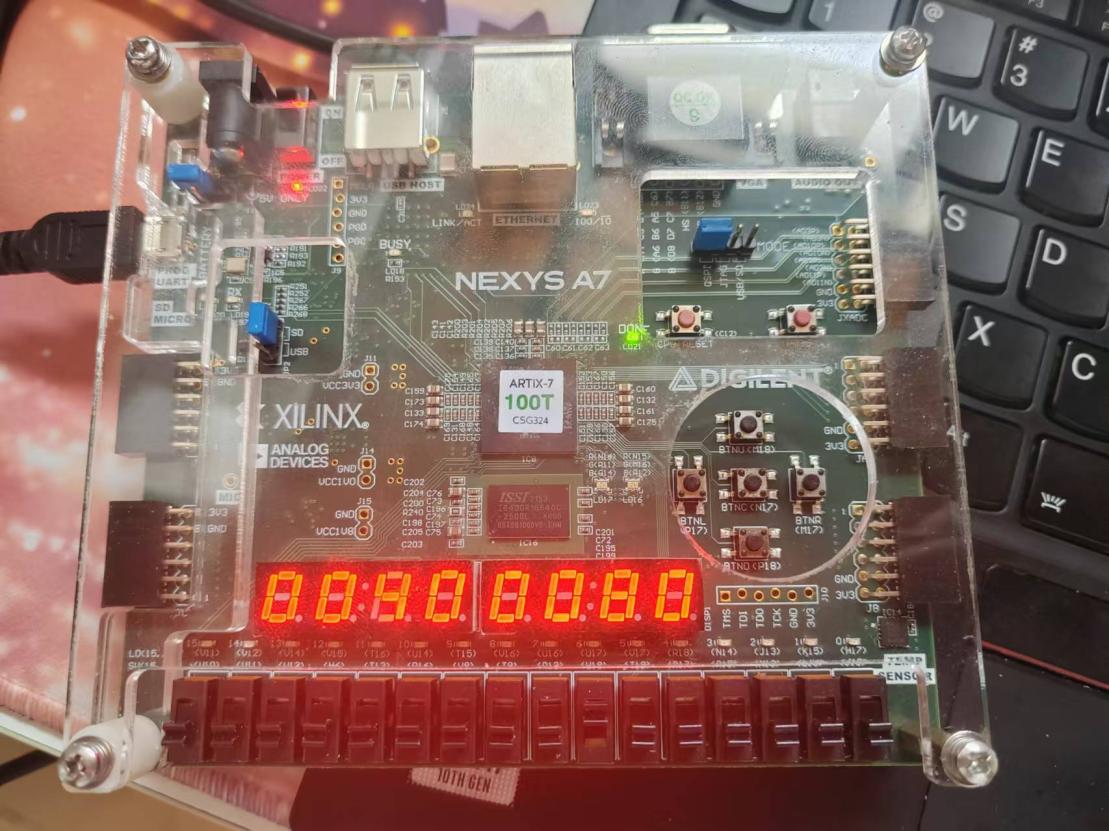




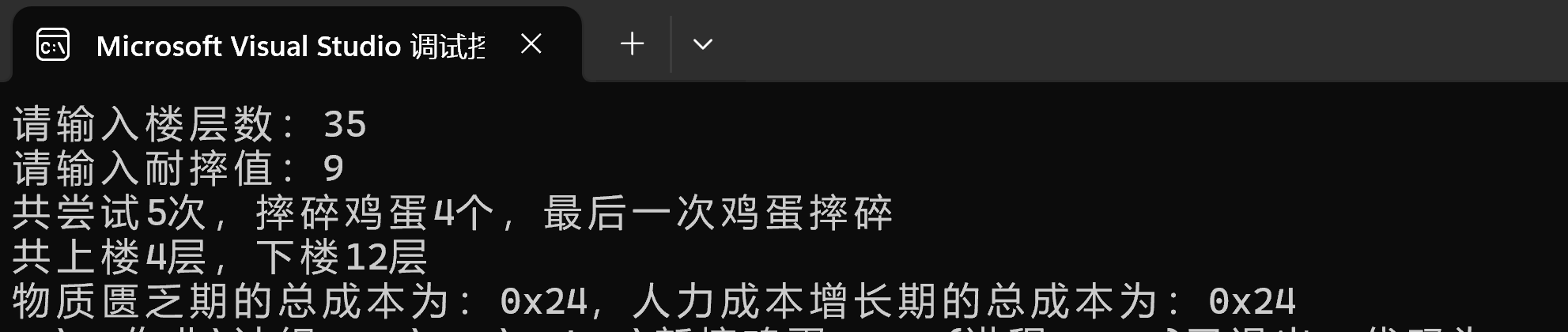
总楼层数64，耐摔值 64:

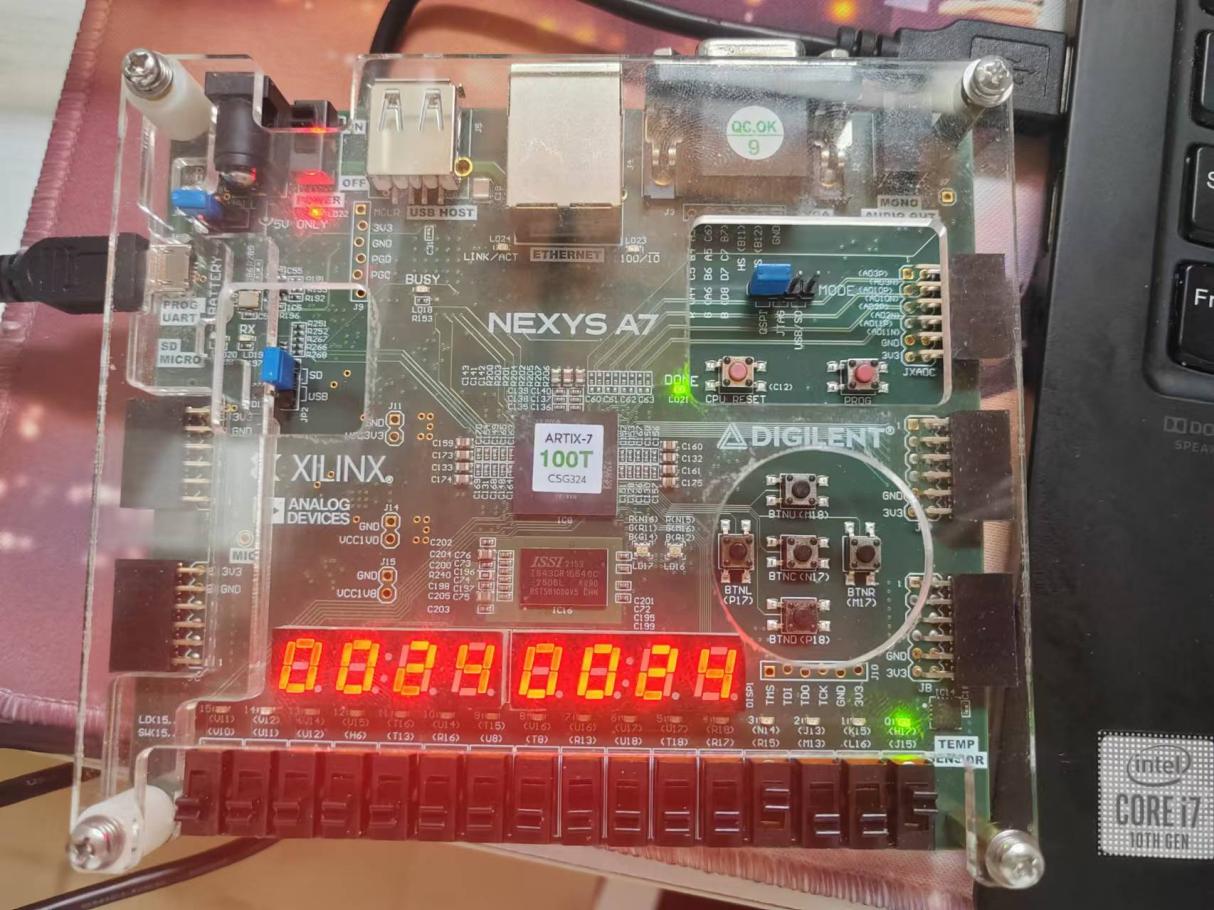
cpp运行的结果和下板结果如下图：

可以得到物质匮乏时期总成本为0x40，人力成本增长时期总成本为0x80，最后一次鸡蛋没摔碎，下板结果与cpp得到的结果相同。  


  
总楼层数35，耐摔值 9:

cpp运行的结果和下板结果如下图：

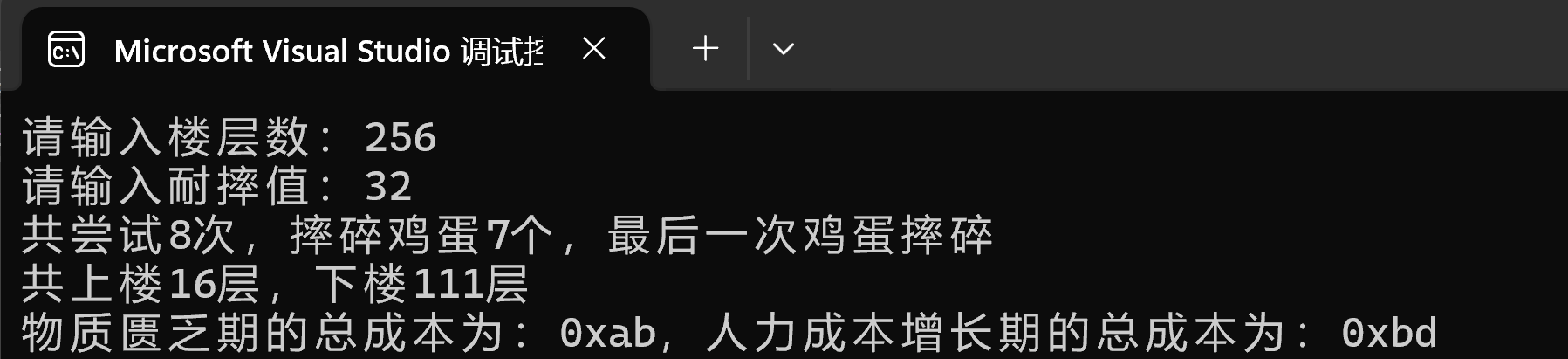
可以得到物质匮乏时期总成本为0x24，人力成本增长时期总成本为0x24，最后一次鸡蛋摔碎，下板结果与cpp得到的结果相同。  


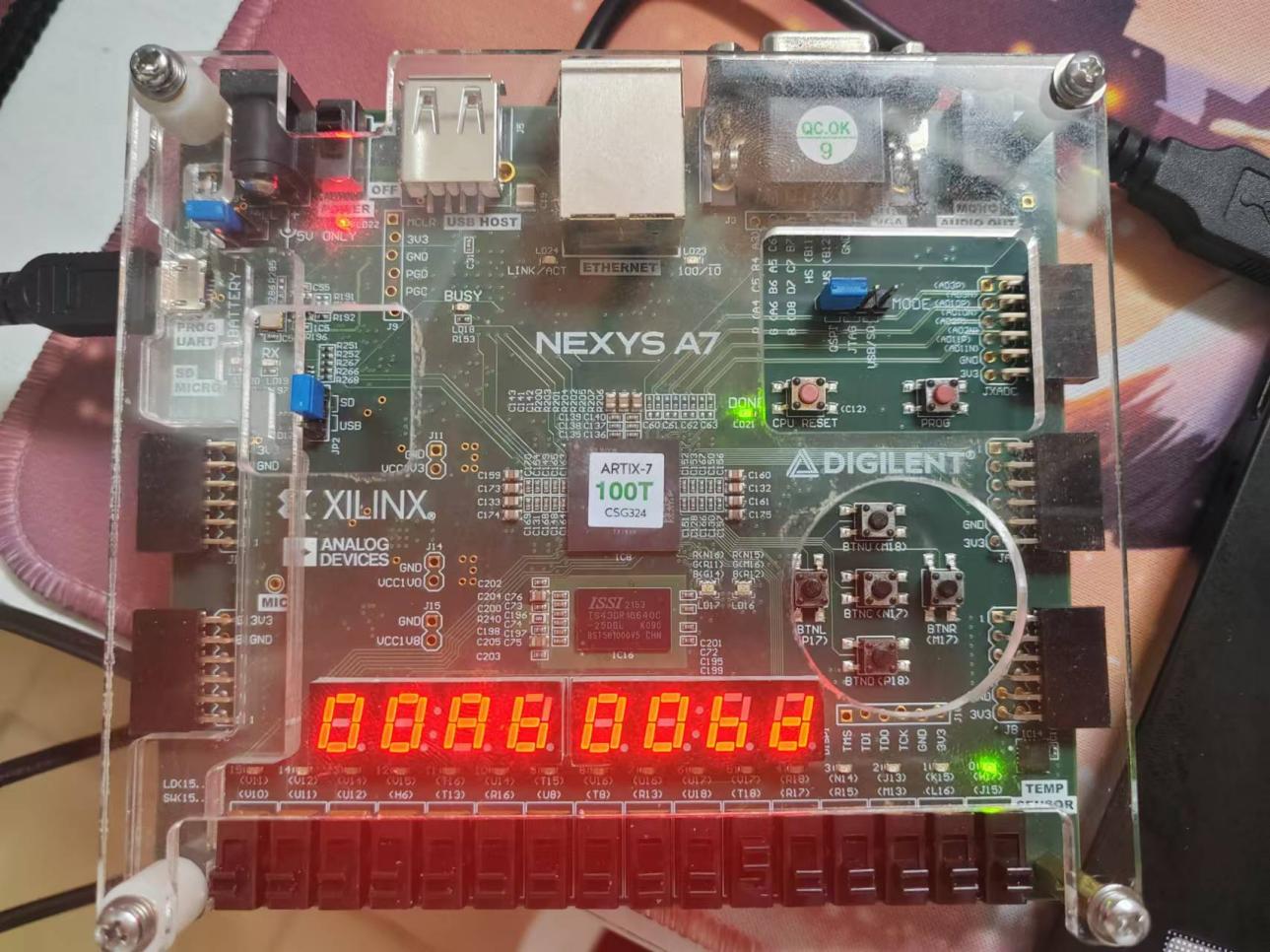


总楼层数256，耐摔值 32:

cpp运行的结果和下板结果如下图：

可以得到物质匮乏时期总成本为0xab，人力成本增长时期总成本为0xbd，最后一次鸡蛋摔碎，下板结果与cpp得到的结果相同。



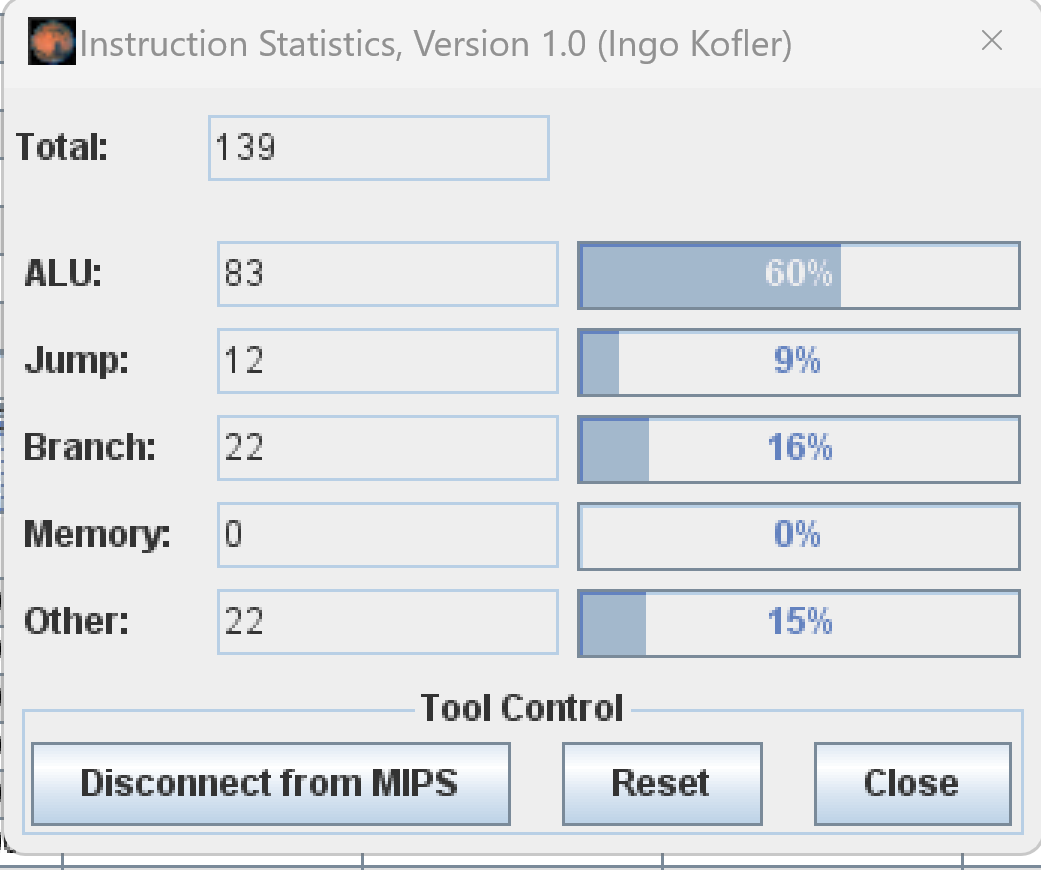


1. 流水线的性能指标定性分析（包括：吞吐率、加速比、效率及相关与冲突分析、CPU的运行时间及存储器空间的使用）

以输入的总楼层数为128，耐摔值为20分析：

通过mars4的工具可以得到该程序的总的指令执行数和各类指令的数量以及占比：

其中总的指令数为139条



* 1. 吞吐率分析

由波形图可得总的时钟周期数为221，则吞吐率为：

* 1. 加速比分析

指令类型汇总：

共有12条跳转指令、22 条分支指令，其余 83+22=105 条为运算指令，则顺序执行所有指令所需的时钟周期为

则加速比为：

* 1. 流水线效率分析

由顺序执行所有指令所需的时钟周期和总的时钟周期可得流水线的效率为：

* 1. 指令相关与冲突分析

数据相关冲突通过暂停流水线的方式解决，特别是针对先写后读RAW的数据相关情况，根据指令在流水线中的具体位置决定stall几个周期以保证数据正确传递。控制相关冲突则通过冻结或排空流水线的方式处理，当检测到分支指令时，暂停其后指令的执行，并计算新的PC值，再重新启动流水线，以减少分支延迟。

1. 总结与体会

本次实验，我深入理解了五段静态流水线的架构与工作原理，通过实际设计和实现流水线CPU，我熟悉了流水线cpu的开发流程。在实验过程中，我掌握了流水线仿真、性能验证模型的构建方法，并能使用C语言和MIPS汇编语言进行程序设计。此外，我还学习到了如何借助mars4工具分析流水线的性能指标，包括吞吐率、加速比和效率，以及如何结合课堂所学处理流水线中的相关与冲突问题。通过此次实验，我对计算机系统结构有了更全面的认识，为之后的学习和研究打下了坚实基础。

1. 附件（见压缩包）