

计算机系统结构课程实验

总结报告

实验题目：动态流水线设计与性能定量分析

学号：

姓名：

指导教师：陆有军

日期：12月1日

目 录

[一、 实验环境部署与硬件配置说明 3](#_Toc489)

[二、 实验的总体结构 3](#_Toc15378)

[三、 总体架构部件的解释说明 4](#_Toc17115)

[1、 动态流水线总体结构部件的解释说明 4](#_Toc17192)

[2、 动态流水线各个结构部件的解释说明 4](#_Toc9974)

[（1） IF 取指模块 4](#_Toc9563)

[（2） IF/ID级间寄存器模块 5](#_Toc25838)

[（3） ID译码模块 5](#_Toc3431)

[（4） EX执行模块 8](#_Toc22546)

[（5） EX/MEM级间寄存器模块 9](#_Toc32442)

[（6） MEM访存模块 11](#_Toc15865)

[（7） MEM/WB级间寄存器模块 12](#_Toc20741)

[（8） WB写回模块 13](#_Toc9784)

[（9） PC寄存器模块 14](#_Toc14730)

[四、 实验仿真过程 15](#_Toc19713)

[1、 流水线仿真过程 15](#_Toc22470)

[2、 构建tb文件 15](#_Toc8105)

[五、 实验仿真的波形图及某时刻寄存器值的物理意义 16](#_Toc25181)

[六、 实验验算数学模型及算法程序 17](#_Toc6260)

[七、 实验验算程序下板测试过程与实现 20](#_Toc26776)

[1、 配置xdc文件： 20](#_Toc1251)

[2、 下板结果说明： 21](#_Toc7694)

[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） 24](#_Toc3038)

[1、 吞吐率分析 24](#_Toc19495)

[2、 加速比分析 25](#_Toc3341)

[3、 流水线效率分析 25](#_Toc10318)

[4、 指令相关与冲突分析 25](#_Toc21281)

[（1） 数据相关 25](#_Toc22930)

[（2） 控制相关： 26](#_Toc31601)

[九、 总结与体会 26](#_Toc22635)

[十、 附件（所有程序见压缩包） 26](#_Toc4902)

# 实验环境部署与硬件配置说明

系统环境：Windows 11

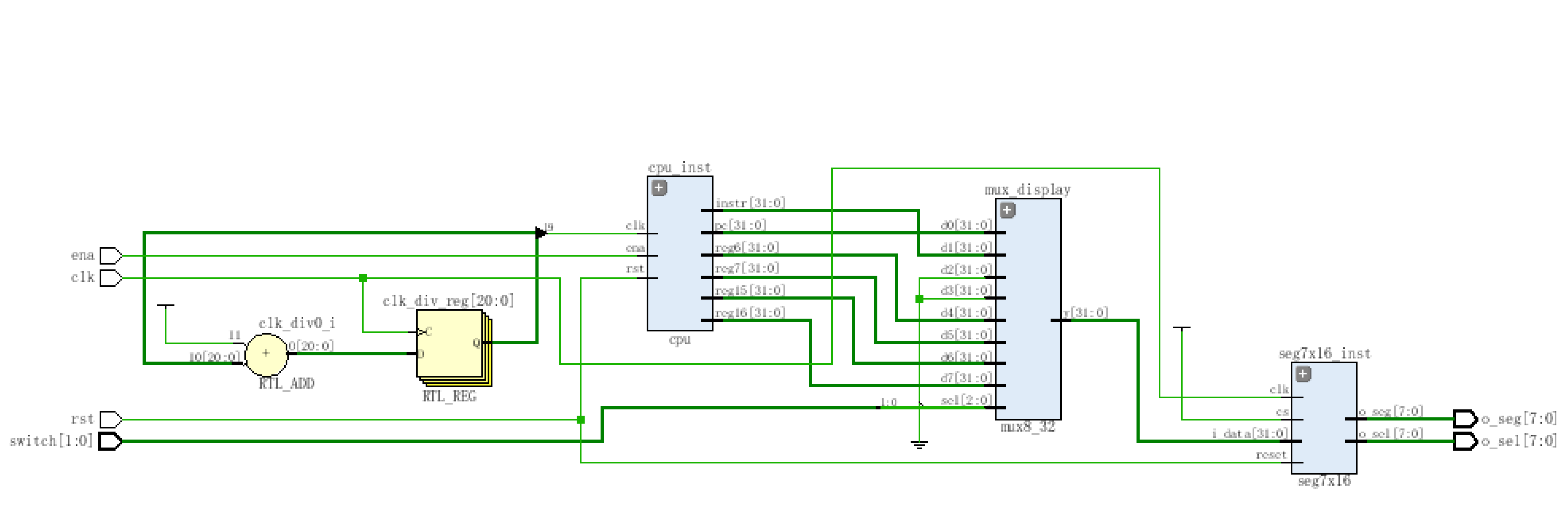
开发环境：Vivado 2016.2

硬件环境：Xilinix NEXYS4 DDR

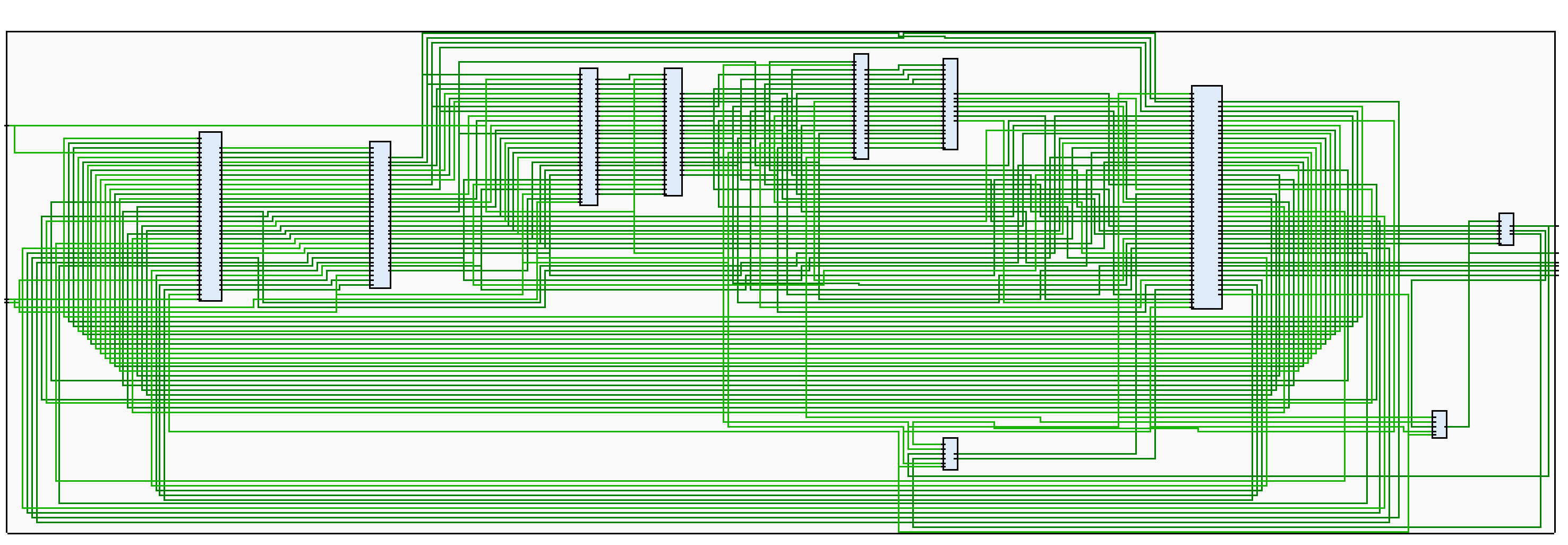
# 实验的总体结构

实验要求完成至少31条MIPS指令的动态流水线CPU设计，并支持中断。在CPU运行验证程序的过程中，由按键或拨动开关产生一个暂停的中断，再次按键或拨动开关结束中断，继续运行后续的运算，并在数码管上动态显示运算值。

首先可以先看看需要的数学建模，有不少的步骤需要用到乘法，而在31条mips中是没有mul的，如果通过循环加法来代替乘法则会非常繁琐，并且导致vivado因为内存不够而卡死。因此在上次实验中设计摔鸡蛋的相关程序时，我最后是求出了上下的楼层数和鸡蛋摔碎数，最后在顶层模块中用行为级描述来计算题目要求的不同时期的成本。在本次实验中如果也是使用行为级描述那就对于整体的实验就没有意义了所以就将上学期期末写的54条指令单周期可中断cpu进行了修改，最终完成了54条MIPS指令的动态流水线可中断CPU设计。

动态流水线的总体结构如下图所示：  


其中cpu的总体架构如下：



# 总体架构部件的解释说明

## 动态流水线总体结构部件的解释说明

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单周期cpu进行修改以实现对54条 MIPS 指令的五段流水线处理。

以下是实验的总体结构部分：

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）。

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。

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。

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。

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。

## 动态流水线各个结构部件的解释说明

### IF 取指模块

**作用：**

负责从指令存储器中取出指令，并根据当前的PC计算下一条指令的地址；

提供当前指令地址的偏移值，PC + 4，供后续阶段使用。  
**实现特点：**

模块内部利用多路选择器处理可能的PC来源即跳转、分支或异常地址。

包含寄存器和加法器协同工作，确保顺利生成连续的指令流。

**模块定义如下：**

module pipe\_if(

    input   [31:0]  in\_pc,

    input   [2:0]   in\_pc\_sel,

    input   [31:0]  in\_pc\_eaddr,

    input   [31:0]  in\_pc\_baddr,

    input   [31:0]  in\_pc\_raddr,

    input   [31:0]  in\_pc\_jaddr,

    output  [31:0]  out\_npc,

    output  [31:0]  out\_pc4,

    output  [31:0]  out\_instr

    );

### IF/ID级间寄存器模块

**功能：**

连接取指阶段和译码阶段，临时存储从IF阶段生成的PC值和指令内容，保证指令与数据流的连贯性。

**特点：**

具有暂停信号（stall）支持流水线阻塞时冻结数据流，避免级间数据混乱；可根据分支指令情况丢弃无效的指令。

**模块定义如下：**

module pipe\_if\_id(

    input               in\_clk,

    input               in\_rst,

    input               in\_stall,

    input               in\_branch,

    input [31:0]        in\_pc4,

    input [31:0]        in\_instr,

    output reg [31:0]   out\_pc4,

    output reg [31:0]   out\_instr

    );

### ID译码模块

**主要任务：**

对指令进行分解和分析，提取操作码、寄存器地址及立即数等信息；

完成数据前推逻辑，减少数据相关导致的冒险；

解析分支条件，提前预测跳转指令是否需要改变PC。

**关键组件：**

包含寄存器堆（register file）、立即数扩展单元、多路选择器等。

比较器（compare）用于实现分支跳转的条件判断。

**模块定义如下：**

module pipe\_id(

    input           in\_clk,

    input           in\_rst,

    input [31:0]    in\_pc4,

    input [31:0]    in\_instr,

    input           in\_hi\_wena,

    input           in\_lo\_wena,

    input           in\_rd\_wena,

    input [4:0]     in\_rd\_waddr,

    input [31:0]    in\_hi\_data,

    input [31:0]    in\_lo\_data,

    input [31:0]    in\_rd\_data,

    input [5:0]     in\_ex\_op,

    input [5:0]     in\_ex\_func,

    input [31:0]    in\_ex\_pc4,

    input [31:0]    in\_ex\_alu\_data,

    input [31:0]    in\_ex\_mul\_hi,

    input [31:0]    in\_ex\_mul\_lo,

    input [31:0]    in\_ex\_div\_r,

    input [31:0]    in\_ex\_div\_q,

    input [31:0]    in\_ex\_clz\_data,

    input [31:0]    in\_ex\_hi\_data,

    input [31:0]    in\_ex\_lo\_data,

    input [31:0]    in\_ex\_rs\_data,

    input           in\_ex\_hi\_wena,

    input           in\_ex\_lo\_wena,

    input           in\_ex\_rd\_wena,

    input [1:0]     in\_ex\_hi\_sel,

    input [1:0]     in\_ex\_lo\_sel,

    input [2:0]     in\_ex\_rd\_sel,

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

    input [31:0]    in\_mem\_pc4,

    input [31:0]    in\_mem\_alu\_data,

    input [31:0]    in\_mem\_mul\_hi,

    input [31:0]    in\_mem\_mul\_lo,

    input [31:0]    in\_mem\_div\_q,

    input [31:0]    in\_mem\_div\_r,

    input [31:0]    in\_mem\_clz\_data,

    input [31:0]    in\_mem\_lo\_data,

    input [31:0]    in\_mem\_hi\_data,

    input [31:0]    in\_mem\_rs\_data,

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

    input           in\_mem\_hi\_wena,

    input           in\_mem\_lo\_wena,

    input           in\_mem\_rd\_wena,

    input [1:0]     in\_mem\_hi\_sel,

    input [1:0]     in\_mem\_lo\_sel,

    input [2:0]     in\_mem\_rd\_sel,

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

    output          out\_stall,

    output          out\_branch,

    output [5:0]    out\_op,

    output [5:0]    out\_func,

    output [2:0]    out\_pc\_sel,

    output [31:0]   our\_pc4,

    output [31:0]   out\_immed,

    output [31:0]   out\_shamt,

    output [31:0]   out\_pc\_eaddr,

    output [31:0]   out\_pc\_baddr,

    output [31:0]   out\_pc\_jaddr,

    output [31:0]   out\_pc\_raddr,

    output [31:0]   out\_rs\_data,

    output [31:0]   out\_rt\_data,

    output [31:0]   out\_hi\_data,

    output [31:0]   out\_lo\_data,

    output [31:0]   out\_cp0\_data,

    output          out\_alu\_a\_sel,

    output [1:0]    out\_alu\_b\_sel,

    output [3:0]    out\_aluc,

    output          out\_mul\_ena,

    output          out\_div\_ena,

    output          out\_clz\_ena,

    output          out\_mul\_sign,

    output          out\_div\_sign,

    output          out\_hi\_wena,

    output          out\_lo\_wena,

    output          out\_rd\_wena,

    output          out\_cutter\_sign,

    output          out\_cutter\_addr\_sel,

    output [2:0]    out\_cutter\_sel,

    output          out\_dmem\_ena,

    output          out\_dmem\_wena,

    output [1:0]    out\_dmem\_wsel,

    output [1:0]    out\_dmem\_rsel,

    output [1:0]    out\_hi\_sel,

    output [1:0]    out\_lo\_sel,

    output [2:0]    out\_rd\_sel,

    output [4:0]    out\_rd\_waddr,

    output [31:0]   out\_reg6,

    output [31:0]   out\_reg7,

    output [31:0]   out\_reg15,

    output [31:0]   out\_reg16

    );

### EX执行模块

**职责：**

根据译码阶段的操作类型，对操作数执行相应运算；

处理算术逻辑运算（ALU）、乘除法计算及位移操作；

生成跳转目标地址（如jal或jr指令）。  
**组件：**

算术逻辑单元（ALU）：提供基本运算能力；

乘法器、除法器：处理复杂运算；

数据选择器：控制操作数的来源（例如立即数扩展或寄存器值）。

module pipe\_ex(

    input           in\_rst,

    input [31:0]    in\_pc4,

    input [31:0]    in\_immed,

    input [31:0]    in\_shamt,

    input [31:0]    in\_rs\_data,

    input [31:0]    in\_rt\_data,

    input [31:0]    in\_hi\_data,

    input [31:0]    in\_lo\_data,

    input [31:0]    in\_cp0\_data,

    input           in\_alu\_a\_sel,

    input [1:0]     in\_alu\_b\_sel,

    input [3:0]     in\_aluc,

    input           in\_mul\_ena,

    input           in\_div\_ena,

    input           in\_clz\_ena,

    input           in\_mul\_sign,

    input           in\_div\_sign,

    input           in\_cutter\_sign,

    input           in\_cutter\_addr\_sel,

    input [2:0]     in\_cutter\_sel,

    input           in\_dmem\_ena,

    input           in\_dmem\_wena,

    input [1:0]     in\_dmem\_wsel,

    input [1:0]     in\_dmem\_rsel,

    input           in\_rd\_wena,

    input           in\_hi\_wena,

    input           in\_lo\_wena,

    input [1:0]     in\_hi\_sel,

    input [1:0]     in\_lo\_sel,

    input [2:0]     in\_rd\_sel,

    input [4:0]     in\_rd\_waddr,

    output [31:0]   out\_pc4,

    output [31:0]   out\_mul\_hi,

    output [31:0]   out\_mul\_lo,

    output [31:0]   out\_div\_r,

    output [31:0]   out\_div\_q,

    output [31:0]   out\_rs\_data,

    output [31:0]   out\_rt\_data,

    output [31:0]   out\_hi\_data,

    output [31:0]   out\_lo\_data,

    output [31:0]   out\_cp0\_data,

    output [31:0]   out\_clz\_data,

    output [31:0]   out\_alu\_data,

    output          out\_cutter\_sign,

    output          out\_cutter\_addr\_sel,

    output [2:0]    out\_cutter\_sel,

    output          out\_dmem\_ena,

    output          out\_dmem\_wena,

    output [1:0]    out\_dmem\_wsel,

    output [1:0]    out\_dmem\_rsel,

    output          out\_hi\_wena,

    output          out\_lo\_wena,

    output          out\_rd\_wena,

    output [1:0]    out\_hi\_sel,

    output [1:0]    out\_lo\_sel,

    output [2:0]    out\_rd\_sel,

    output [4:0]    out\_rd\_waddr

);

### EX/MEM级间寄存器模块

功能：

用于传递EX阶段的计算结果及控制信号到MEM阶段。 特点：

支持高效的数据流动，避免关键数据丢失。

module pipe\_ex\_mem(

    input               in\_clk,

    input               in\_rst,

    input               in\_wena,

    input [31:0]        in\_pc4,

    input [31:0]        in\_rs\_data,

    input [31:0]        in\_rt\_data,

    input [31:0]        in\_hi\_data,

    input [31:0]        in\_lo\_data,

    input [31:0]        in\_cp0\_data,

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

    input [31:0]        in\_mul\_hi,

    input [31:0]        in\_mul\_lo,

    input [31:0]        in\_div\_r,

    input [31:0]        in\_div\_q,

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

    input               in\_cutter\_sign,

    input [2:0]         in\_cutter\_sel,

    input               in\_cutter\_addr\_sel,

    input               in\_dmem\_ena,

    input               in\_dmem\_wena,

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

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

    input               in\_hi\_wena,

    input               in\_lo\_wena,

    input               in\_rd\_wena,

    input [1:0]         in\_hi\_sel,

    input [1:0]         in\_lo\_sel,

    input [2:0]         in\_rd\_sel,

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

    output reg [31:0]   out\_pc4,

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

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

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

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

    output reg [31:0]   out\_cp0\_data,

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

    output reg [31:0]   out\_mul\_hi,

    output reg [31:0]   out\_mul\_lo,

    output reg [31:0]   out\_div\_r,

    output reg [31:0]   out\_div\_q,

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

    output reg          out\_cutter\_sign,

    output reg          out\_cutter\_addr\_sel,

    output reg [2:0]    out\_cutter\_sel,

    output reg          out\_dmem\_ena,

    output reg          out\_dmem\_wena,

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

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

    output reg          out\_rd\_wena,

    output reg          out\_hi\_wena,

    output reg          out\_lo\_wena,

    output reg [1:0]    out\_hi\_sel,

    output reg [1:0]    out\_lo\_sel,

    output reg [2:0]    out\_rd\_sel,

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

);

### MEM访存模块

**主要功能：**

执行加载（load）和存储（store）指令；

管理与数据存储器的交互。  
**特点：**

具备不同字长的数据访问功能（字节、半字和全字）；

使用多路选择器区分访存操作结果或运算结果。

module pipe\_mem(

    input           in\_clk,

    input [31:0]    in\_pc4,

    input [31:0]    in\_rs\_data,

    input [31:0]    in\_rt\_data,

    input [31:0]    in\_hi\_data,

    input [31:0]    in\_lo\_data,

    input [31:0]    in\_cp0\_data,

    input [31:0]    in\_alu\_data,

    input [31:0]    in\_mul\_hi,

    input [31:0]    in\_mul\_lo,

    input [31:0]    in\_div\_r,

    input [31:0]    in\_div\_q,

    input [31:0]    in\_clz\_data,

    input           in\_cutter\_sign,

    input           in\_cutter\_addr\_sel,

    input [2:0]     in\_cutter\_sel,

    input [1:0]     in\_dmem\_wsel,

    input [1:0]     in\_dmem\_rsel,

    input           in\_dmem\_ena,

    input           in\_dmem\_wena,

    input           in\_hi\_wena,

    input           in\_lo\_wena,

    input           in\_rd\_wena,

    input [1:0]     in\_hi\_sel,

    input [1:0]     in\_lo\_sel,

    input [2:0]     in\_rd\_sel,

    input [4:0]     in\_rd\_waddr,

    output [31:0]   our\_pc4,

    output [31:0]   out\_rs\_data,

    output [31:0]   out\_hi\_data,

    output [31:0]   out\_lo\_data,

    output [31:0]   out\_cp0\_data,

    output [31:0]   out\_alu\_data,

    output [31:0]   out\_mul\_hi,

    output [31:0]   out\_mul\_lo,

    output [31:0]   out\_div\_r,

    output [31:0]   out\_div\_q,

    output [31:0]   out\_clz\_data,

    output [31:0]   out\_dmem\_data,

    output          out\_hi\_wena,

    output          out\_lo\_wena,

    output          out\_rd\_wena,

    output [1:0]    out\_hi\_sel,

    output [1:0]    out\_lo\_sel,

    output [2:0]    out\_rd\_sel,

    output [4:0]    out\_rd\_waddr

    );

### MEM/WB级间寄存器模块

**功能：**

保存访存阶段的结果，供写回阶段使用。 特点：

对MEM阶段的结果（如ALU计算值或访存值）进行短期寄存，避免数据丢失。

module pipe\_mem\_wb(

    input               in\_clk,

    input               in\_rst,

    input               in\_wena,

    input [31:0]        in\_pc4,

    input [31:0]        in\_rs\_data,

    input [31:0]        in\_hi\_data,

    input [31:0]        in\_lo\_data,

    input [31:0]        in\_cp0\_data,

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

    input [31:0]        in\_mul\_hi,

    input [31:0]        in\_mul\_lo,

    input [31:0]        in\_div\_r,

    input [31:0]        in\_div\_q,

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

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

    input               in\_hi\_wena,

    input               in\_lo\_wena,

    input               in\_rd\_wena,

    input [1:0]         in\_hi\_sel,

    input [1:0]         in\_lo\_sel,

    input [2:0]         in\_rd\_sel,

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

    output reg [31:0]   out\_pc4,

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

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

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

    output reg [31:0]   out\_cp0\_data,

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

    output reg [31:0]   out\_mul\_hi,

    output reg [31:0]   out\_mul\_lo,

    output reg [31:0]   out\_div\_r,

    output reg [31:0]   out\_div\_q,

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

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

    output reg          out\_hi\_wena,

    output reg          out\_lo\_wena,

    output reg          out\_rd\_wena,

    output reg [1:0]    out\_hi\_sel,

    output reg [1:0]    out\_lo\_sel,

    output reg [2:0]    out\_rd\_sel,

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

    );

### WB写回模块

**作用：**

将之前各阶段生成的数据写回寄存器堆。 特点：

使用多路选择器选择最终写回的值（如ALU结果或存储器读取值），并根据控制信号更新目标寄存器。

module pipe\_wb(

    input [31:0]    in\_pc4,

    input [31:0]    in\_rs\_data,

    input [31:0]    in\_hi\_data,

    input [31:0]    in\_lo\_data,

    input [31:0]    in\_cp0\_data,

    input [31:0]    in\_alu\_data,

    input [31:0]    in\_mul\_hi,

    input [31:0]    in\_mul\_lo,

    input [31:0]    in\_div\_r,

    input [31:0]    in\_div\_q,

    input [31:0]    in\_clz\_data,

    input [31:0]    in\_dmem\_data,

    input           in\_hi\_wena,

    input           in\_lo\_wena,

    input           in\_rd\_wena,

    input [1:0]     in\_hi\_sel,

    input [1:0]     in\_lo\_sel,

    input [2:0]     in\_rd\_sel,

    input [4:0]     in\_rd\_waddr,

    output          out\_hi\_wena,

    output          out\_lo\_wena,

    output          out\_rd\_wena,

    output [4:0]    out\_rd\_waddr,

    output [31:0]   out\_hi\_data,

    output [31:0]   out\_lo\_data,

    output [31:0]   out\_rd\_data

    );

### PC寄存器模块

该模块主要保存程序计数器（PC）的值，用于指向即将执行的指令地址。  
**功能：**

存储当前指令的地址；通过加法器生成下一条指令地址。 控制信号：时钟信号决定寄存器的更新节奏；暂停信号可在流水线中断时冻结PC的更新；复位信号用于在初始化时将PC归零。

module pc(

    input               in\_clk,

    input               in\_rst,

    input               in\_ena,

    input               in\_stall,

    input  [31:0]       in\_pc,

    output reg [31:0]   out\_pc

    );

# 实验仿真过程

## 流水线仿真过程

选用了分段函数程序作为测试案例，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。

## 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.在reset时直接初始化对应的regfile寄存器。

2.可以将每个时钟周期（clk）时的pc值、指令的值和各个寄存器的值输出，便于后续分析和调试。

tb文件的部分核心代码如下：

reg           clk, rst, ena;

    wire [7:0]    o\_seg, o\_sel;

    initial

    begin

        clk = 1'b0;

        rst = 1'b1;

        ena = 1'b1;

        #1

        rst = 1'b0;

    end

    always

    begin

        #1

        clk = ~clk;

    end

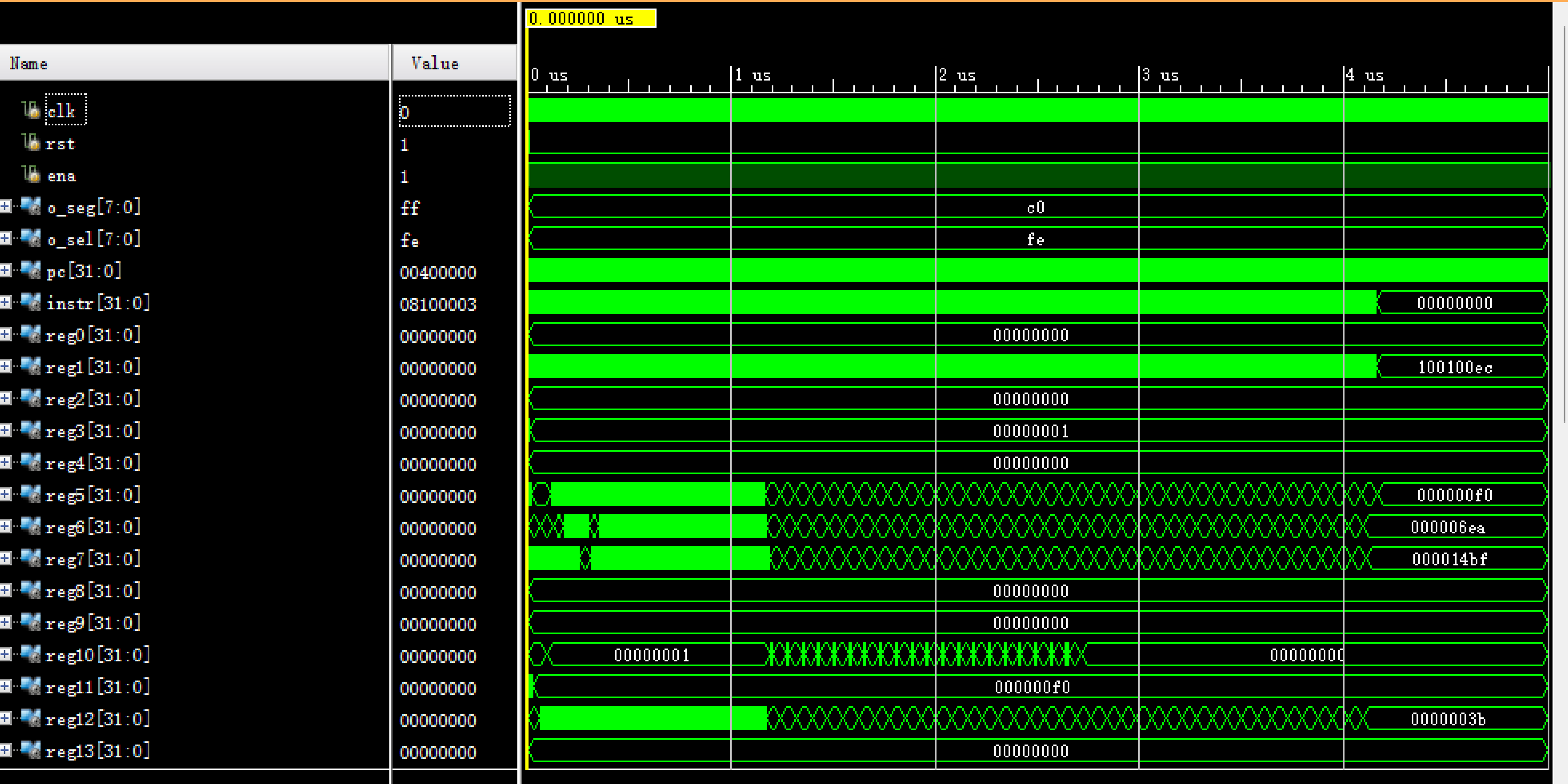
    wire [31:0] pc      = testbench.board\_top\_inst.cpu\_inst.pc;

    wire [31:0] instr   = testbench.board\_top\_inst.cpu\_inst.instr;

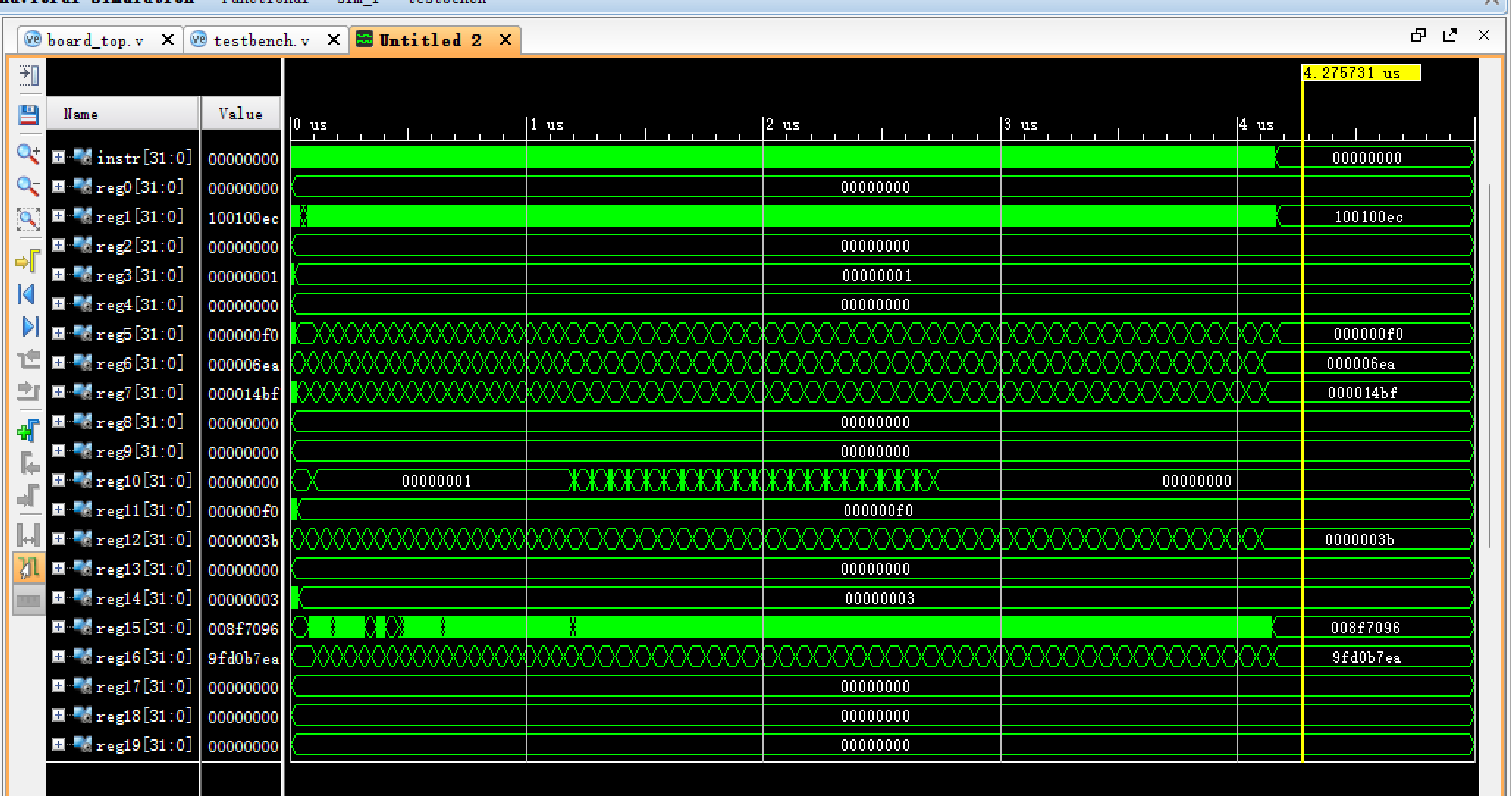
    wire [31:0] reg0    = testbench.board\_top\_inst.cpu\_inst.pipe\_id\_inst.regfile\_inst.array\_reg[0];  
...#其他的寄存器  
board\_top board\_top\_inst(.clk(clk), .rst(rst), .ena(ena), .o\_seg(o\_seg), .o\_sel(o\_sel));

endmodule

对tb文件设为顶层后仿真得到如下波形图：



# 实验仿真的波形图及某时刻寄存器值的物理意义



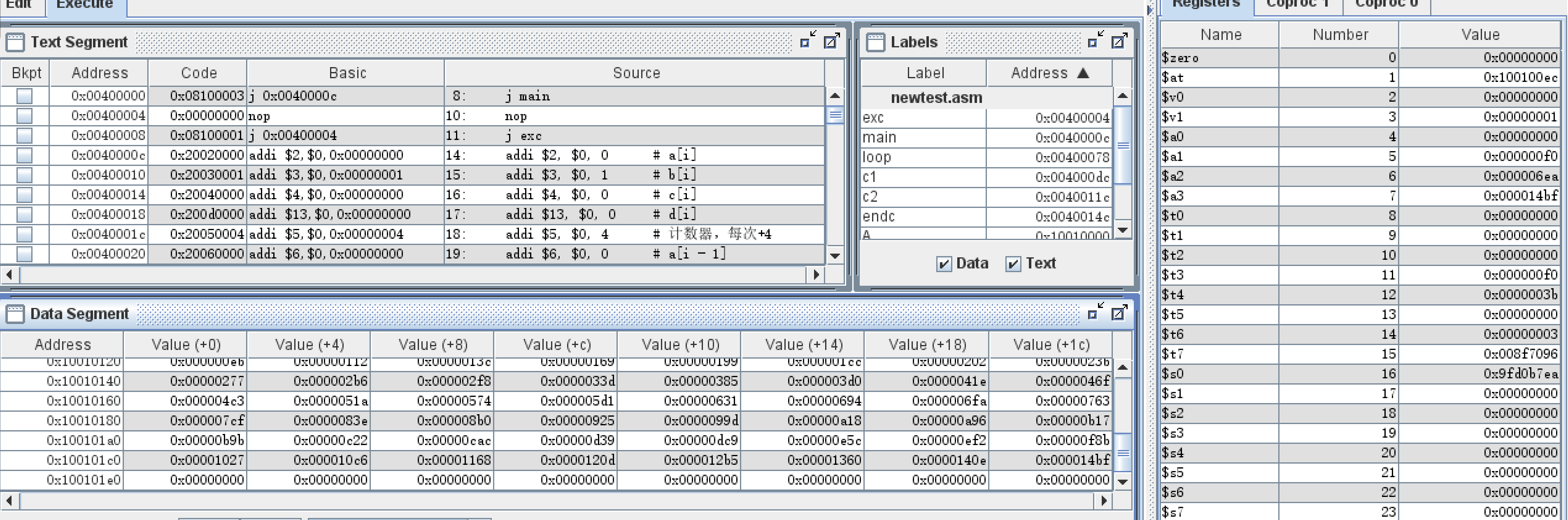
如图所示，波形图中显示了不同的数值，其中instr为输入的十六进制指令，6号和7号寄存器为a[i]与b[i],15号和16号寄存器为c[i]与d[i]。

其他的寄存器的物理意义：

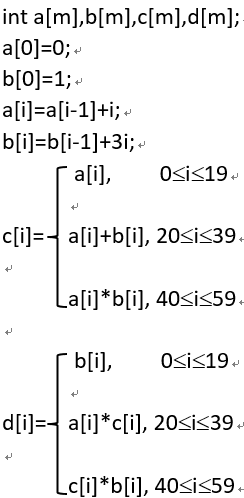
|  |  |  |
| --- | --- | --- |
| 寄存器 | 作用 | 说明 |
| 5 | 计数器 | 每次循环+4 |
| 11 | 数组总长 | 用于存储数组总长度，即 240，作为循环结束条件 |
| 14 | 常数 3 | 用于计算 B 元素值中需要的乘法常数 |
| 27 | 内存地址临时寄存器 | 用于存储数组访问的基地址 |

整个循环的过程可以直接从10号寄存器上体现出来，10号寄存器用于区分分段函数的不同阶段，用来跳转，可以很明显地看到前期值一直为1时是i<20的阶段；中期在20<=i<40频繁变换是因为跳转的逻辑是分两次跳转，第一次i>=20，10号寄存器为0，之后由于小于40值又赋值为1；在i>=40时两次判断的结果都是0就不会来回变值了。

最终得到的几个数值分别为：6ea,14bf,8f7096,9fd0b7ea,仿真的结果与c程序和mars得到的结果相同。



# 实验验算数学模型及算法程序



可以根据这个简单的分段函数写一个c程序方便后续与动态流水线cpu的结果对比：

#include <stdio.h>

using namespace std;

int main()

{

    const int m = 60;

    int a[m], b[m], c[m], d[m];

    a[0] = 0;

    b[0] = 1;

    for (int i = 0; i < m; i++){

        if (i > 0) {

            a[i] = a[i - 1] + i;

            b[i] = b[i - 1] + 3 \* i;

        }

        if (i < 20) {

            c[i] = a[i];

            d[i] = b[i];

        }

        else {

            if (i < 40){

                c[i] = a[i] + b[i];

                d[i] = a[i] \* c[i];

            }

            else {

                c[i] = a[i] \* b[i];

                d[i] = c[i] \* b[i];

            }

        }

    }

    printf("各数组的值为：a[i]        b[i]        c[i]        d[i]\n");

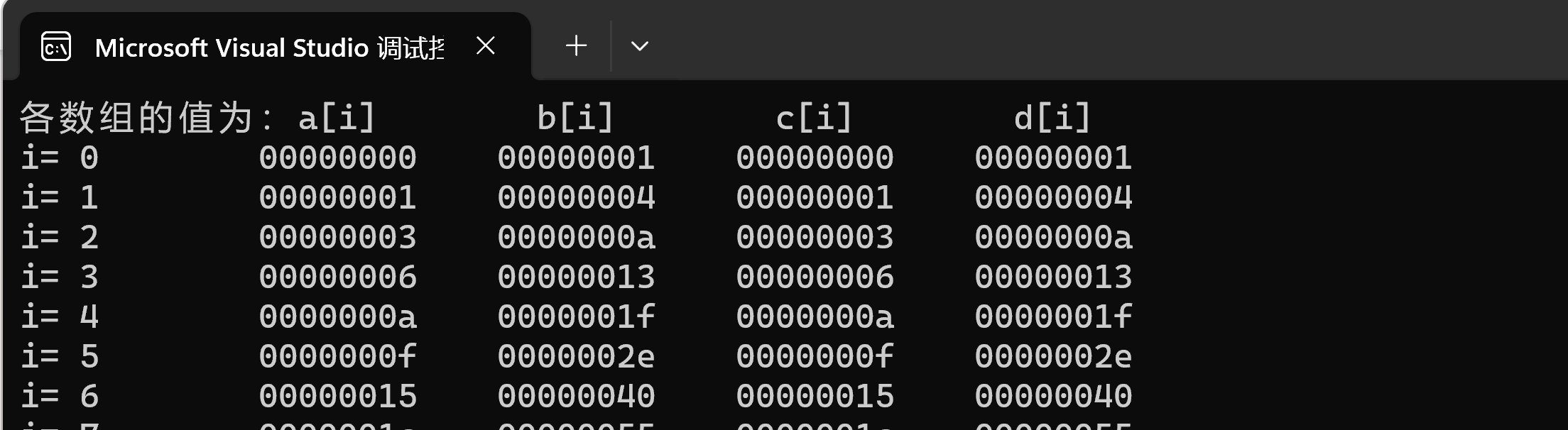
    for (int i = 0; i < m; i++) {

        printf("i=%2d        %08x    %08x    %08x    %08x\n", i, a[i], b[i], c[i], d[i]);

    }

    return 0;

}

其输出形式如下图：  


将其写为mips汇编代码后：

.data

    A:.space 240

    B:.space 240

    C:.space 240

    D:.space 240

.text

    j main

    exc:

    nop

    j exc

main:

    addi $2, $0, 0      # a[i]

    addi $3, $0, 1      # b[i]

    addi $4, $0, 0      # c[i]

    addi $13, $0, 0     # d[i]

    addi $5, $0, 4      # 计数器，每次+4

    addi $6, $0, 0      # a[i - 1]

    addi $7, $0, 1      # b[i - 1]

    addi $10, $0, 0     # 分段区间标识

    addi $11, $0, 240   # 存放数组总长

    addi $14, $0, 3     # 存放常数3

    addi $30, $0, 0

    lui $27, 0x0000

    addu $27, $27, $0

    sw $2, A($27)

    lui $27, 0x0000

    addu $27, $27, $0

    sw $2, B($27)

    addi $31, $31, 1

    lui $27, 0x0000

    addu $27, $27, $0

    sw $3, D($27)

loop:

    # 计算A的值

    srl $12, $5, 2

    add $6, $6, $12

    lui $27, 0x0000

    addu $27, $27, $5

    sw $6, A($27)

    # 计算B的值

    mul $15, $14, $12

    add $7, $7, $15

    lui $27, 0x0000

    addu $27, $27, $5

    sw $7, B($27)

    slti $10, $5, 80

    bne $10, 1, c1

    lui $27, 0x0000

    addu $27, $27, $5

    sw $7, D($27)

    # 本次的值转存记录上一次

    addi $15, $6, 0

    addi $16, $7, 0

    j endc

c1:

    slti $10, $5, 160

    addi $27, $0, 1

    bne $10, $27, c2

    add $15, $6, $7     # c[i] = a[i] + b[i]

    lui $27, 0x0000

    addu $27, $27, $5

    sw $15, C($27)

    mul $16, $15, $6    # d[i] = a[i] \* b[i]

    lui $27, 0x0000

    addu $27, $27, $5

    sw $16, D($27)

    j endc

c2:

    mul $15, $6, $7     # c[i] = a[i] \* b[i]

    lui $27, 0x0000

    addu $27, $27, $5

    sw $15, C($27)

    mul $16, $15, $7    # d[i] = c[i] \* b[i]

    lui $27, 0x0000

    addu $27, $27, $5

    sw $16, D($27)

endc:

    addiu $5, $5, 4

    bne $5, $11, loop

# 实验验算程序下板测试过程与实现

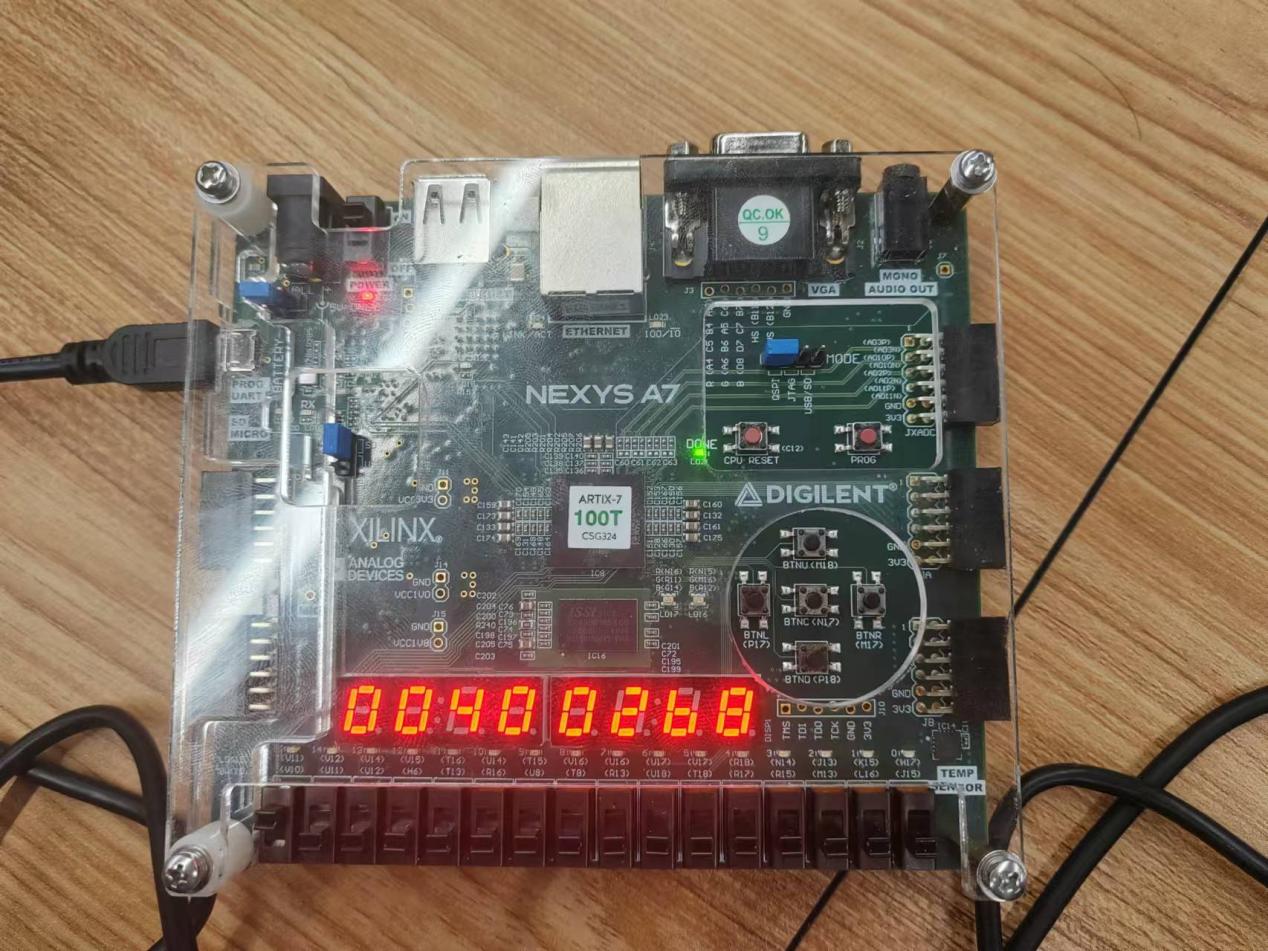
## 配置xdc文件：

一些利用到数码管显示的基本配置如seg和sel的配置已经在之前的数字逻辑和计算机组成原理的课程中详细说明过了，reset键仍然是常见的N17，用于中断的enable设置为v10（即开发板上最左侧的拉杆），右边的三个拉杆则用于切换不同显示的数值。三位可切换8个不同的值，这里使用了6个：  
 switch=0时：pc值

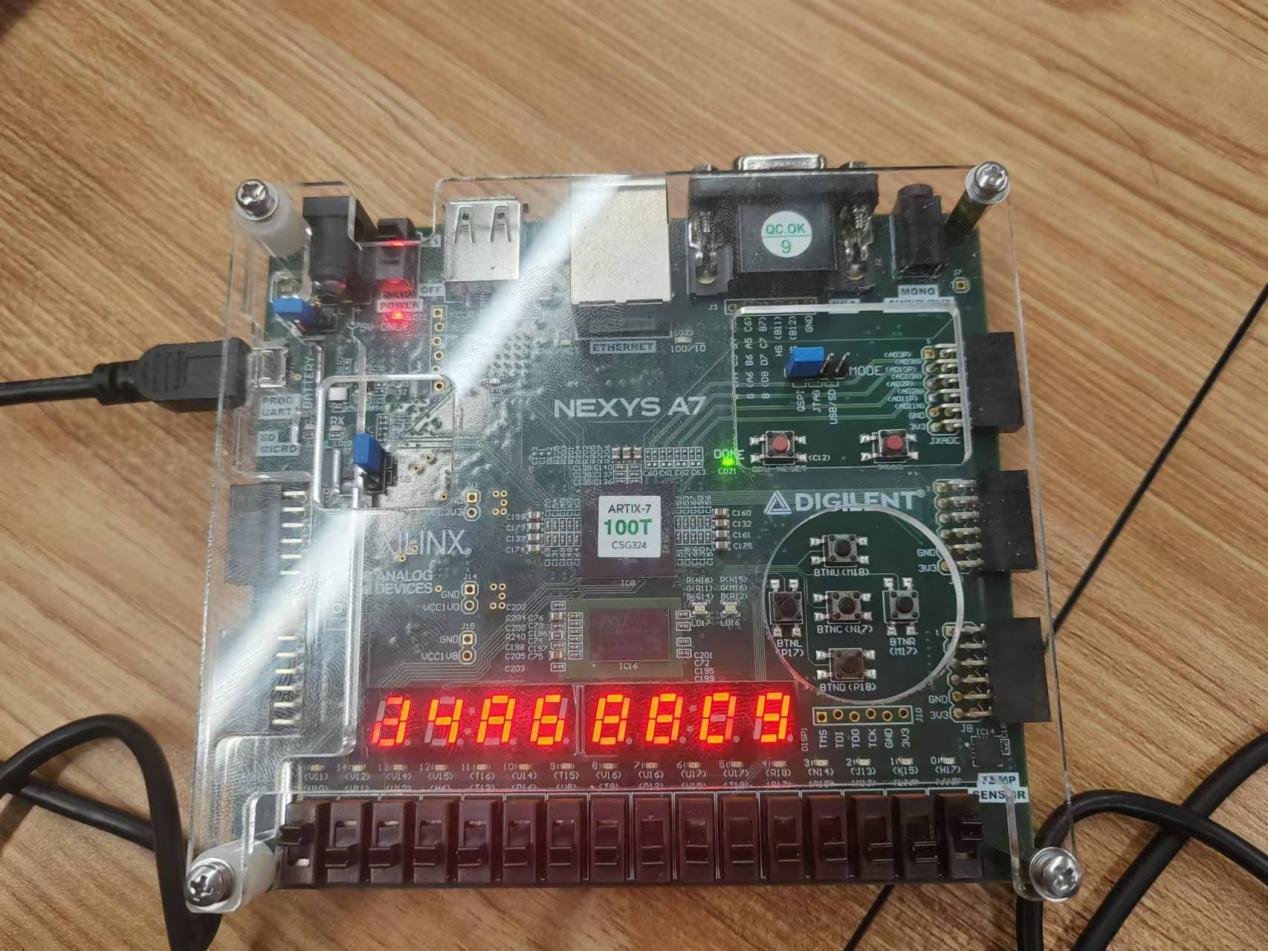
switch=1时：instruction值

switch=4~7时：a[i] , b[i] ,c[i], d[i]分别的值

## 下板结果说明：

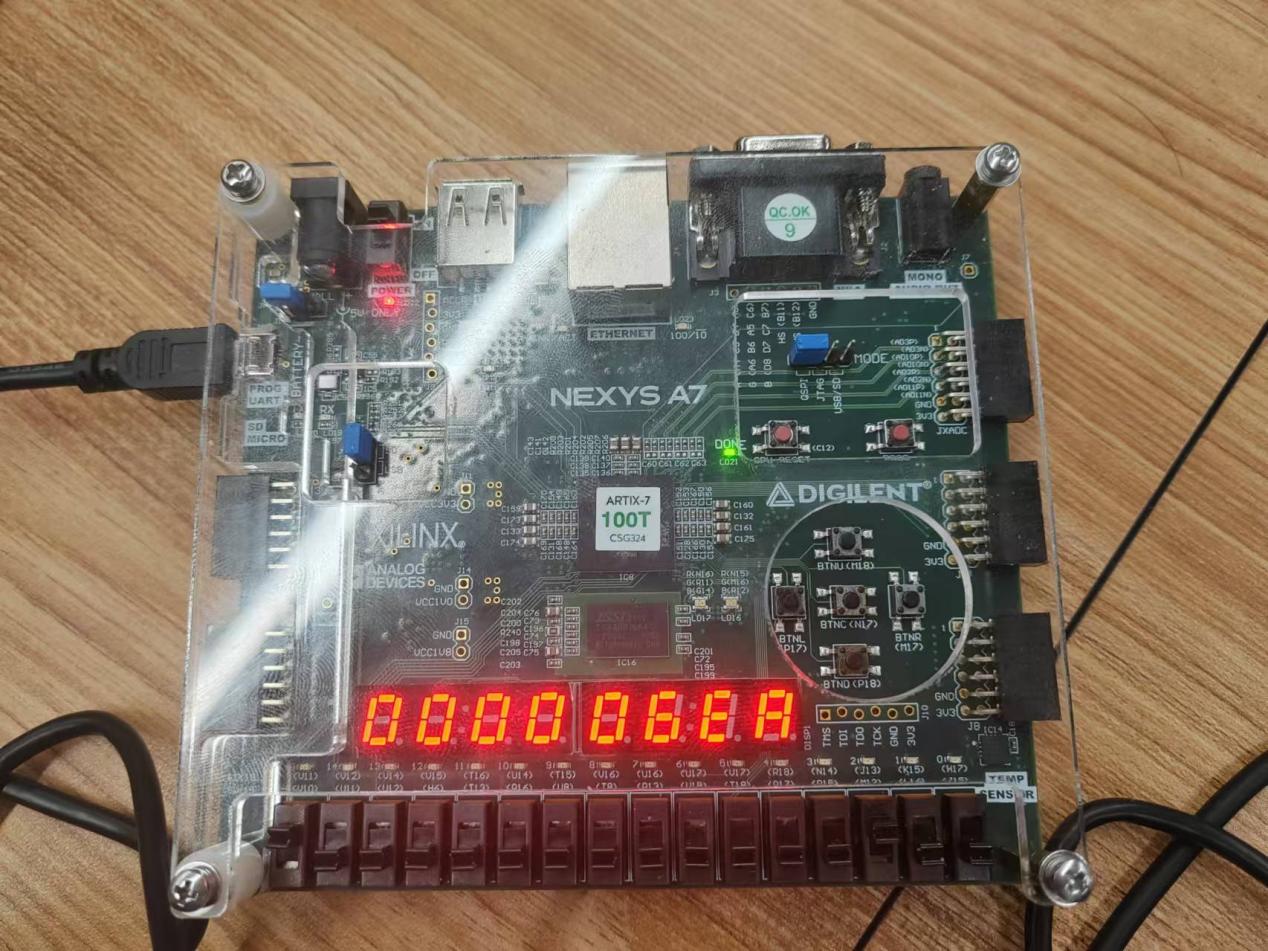


程序运行时流动的pc值

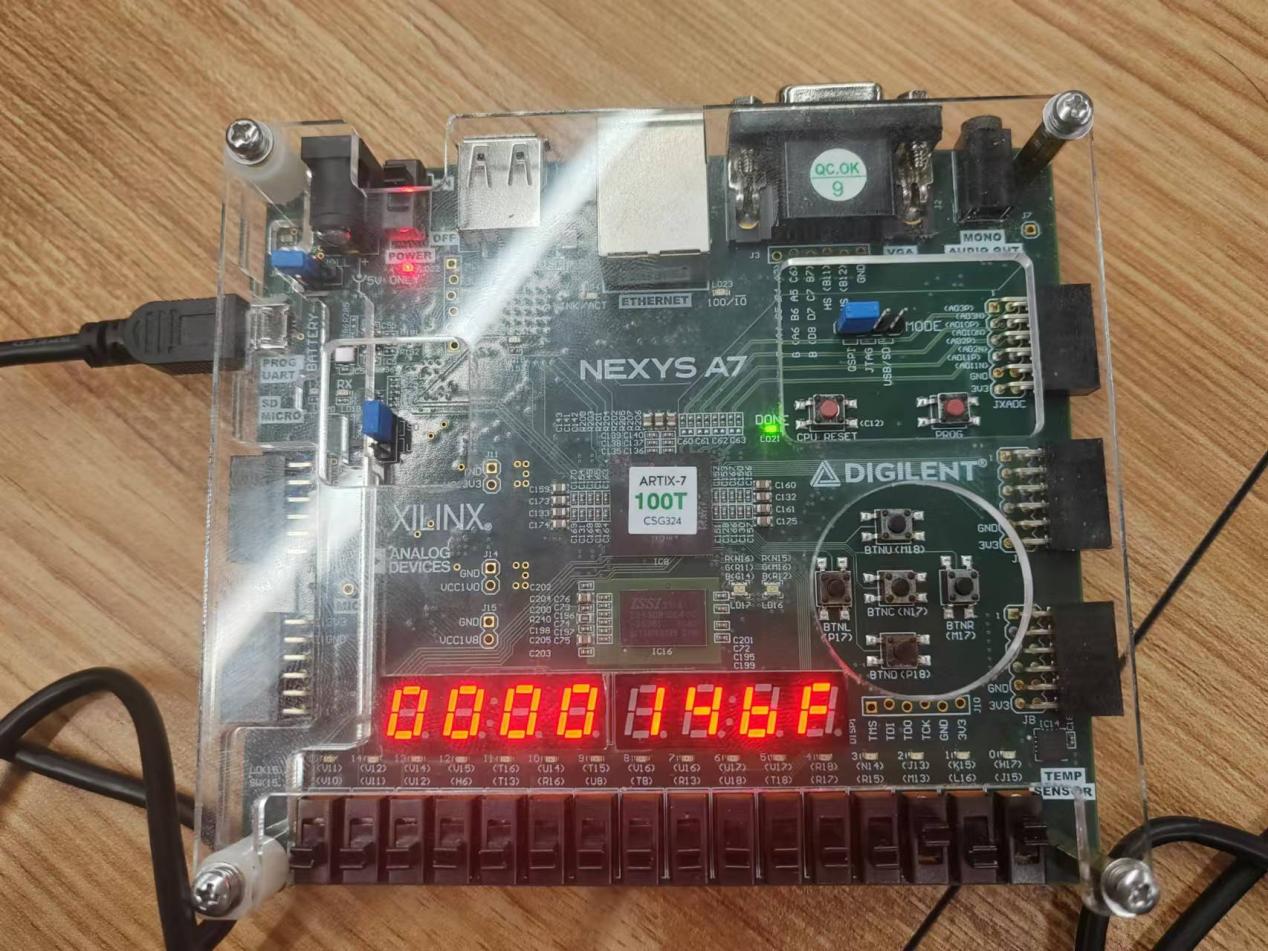


程序运行时流动的instruction值

当程序运行结束时将ena拉杆拉回中断程序，可以看到此时最终的不同的数值：



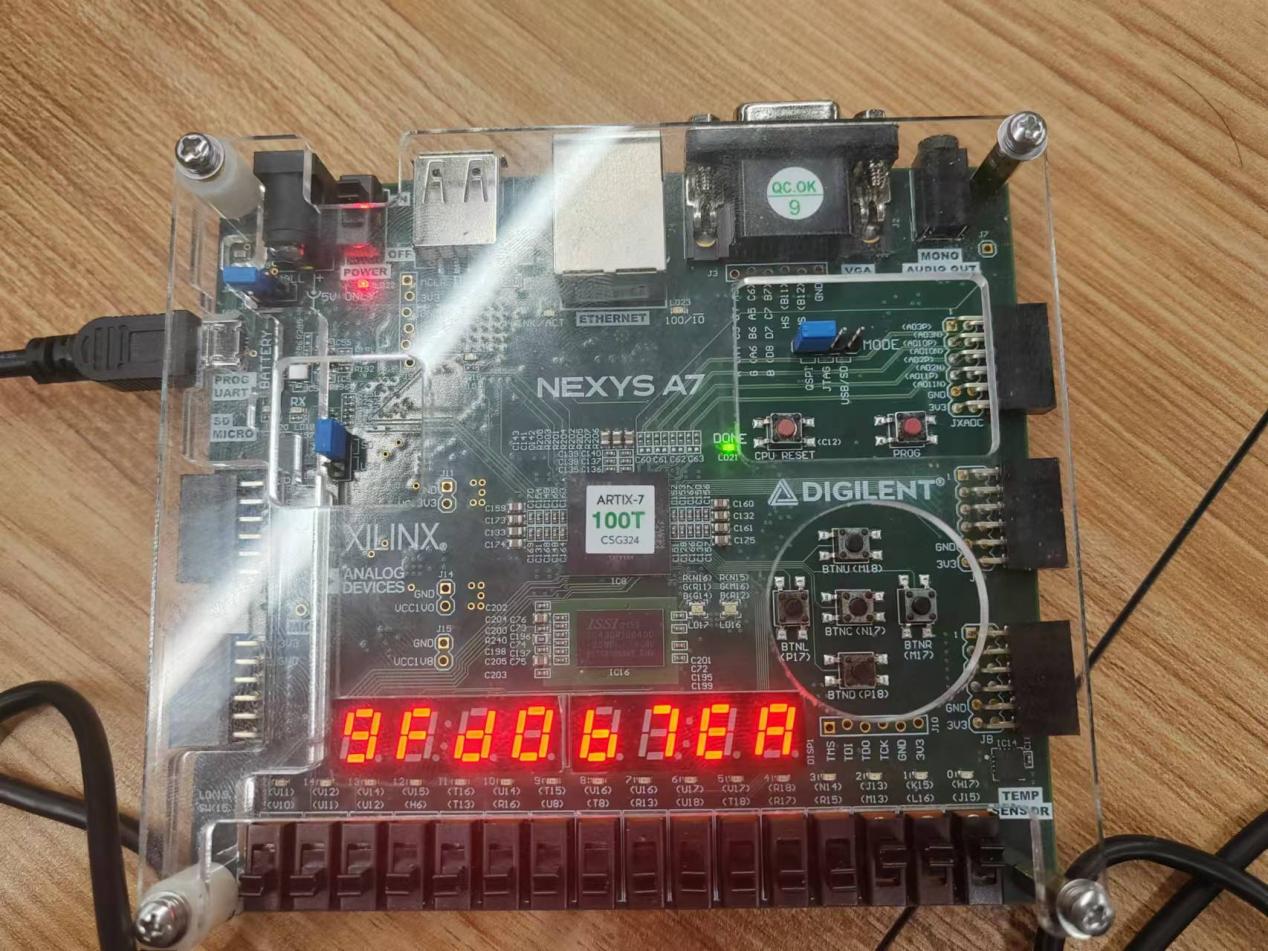
a[i]: 0x6ea



b[i]: 0x14bf

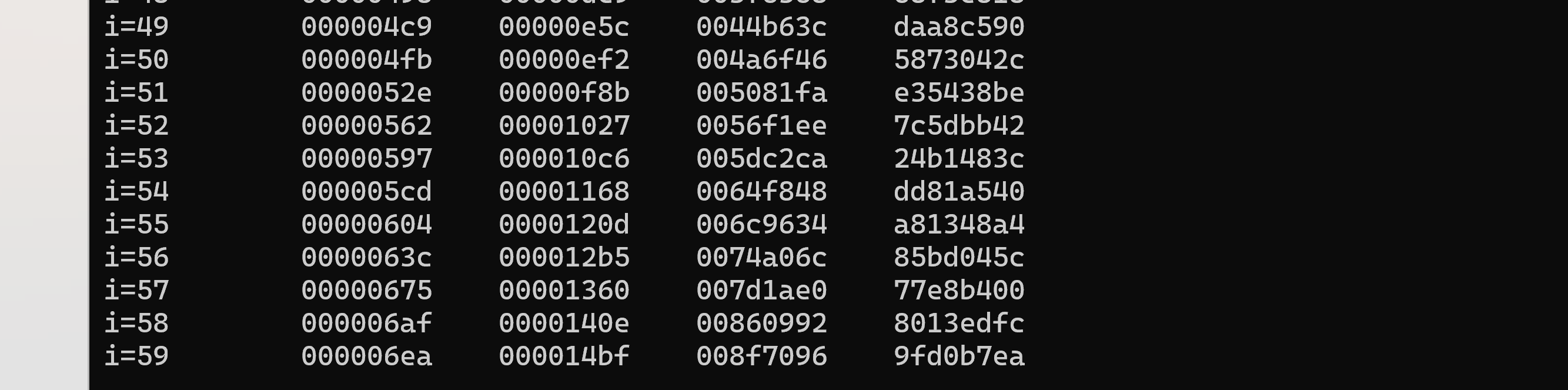


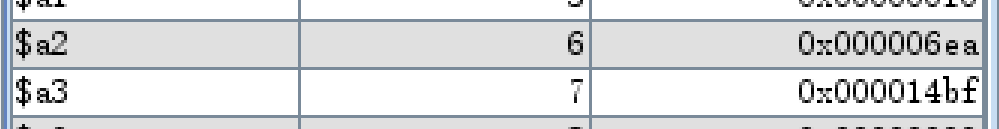
c[i]: 0x8f7096

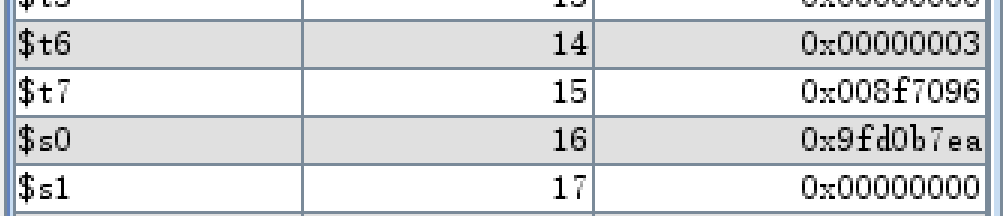


d[i]: 9fd067ea

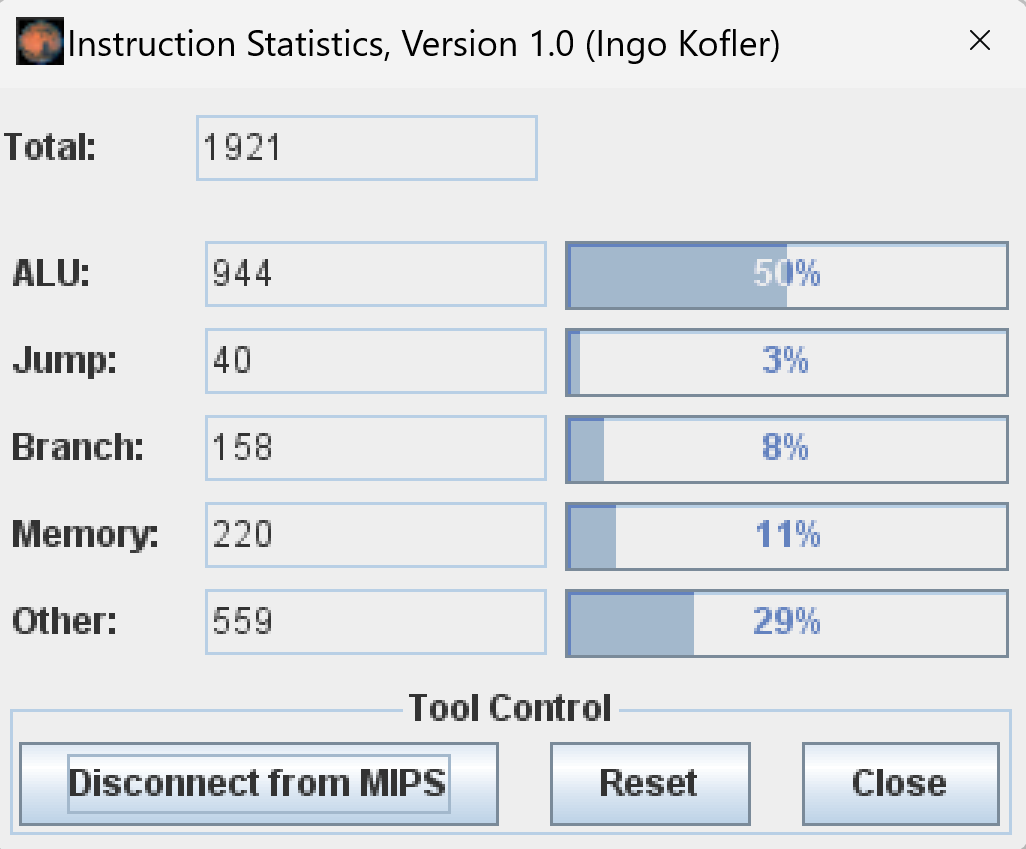
与c程序和Mars跑出来的结果是一致的：







# 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）



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：总的数据如图所示，其中总的指令数为1921条。

## 吞吐率分析

由波形图可得总的时钟周期数为2081，则吞吐率为：

## 加速比分析

指令类型汇总：

共有ALU指令944条，还有40条跳转指令、158 条分支指令，访存指令220条，其余 559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：

## 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：

## 指令相关与冲突分析

### 数据相关

读后写（RAW）：

*srl $12, $5, 2 # 计算逻辑索引 $12 = $5 >> 2*

*add $6, $6, $12 # 更新 a[i]，依赖 $12 的值*

$12 的值依赖于前面的指令 srl $12, $5, 2。由于 srl 是一个位移操作，在 ALU 阶段完成，但随后指令需要 $12 的结果，这会导致 RAW 数据冒险。可以采用内部前推解决，在 ID 级输出EX级所需源操作数时增加多路选择器，将之前周期EX的输出结果或MEM级的输出结果定向前推至该多路选择器的输入中，通过控制信号进行数据传递的判断。

写后读（WAR）：

*mul $15, $6, $7 # c[i] = a[i] \* b[i]*

*add $15, $15, $7 # 覆盖 $15，计算 c[i] + b[i]*

$15 在第二条指令中被覆盖，可能导致第一条 mul 指令的结果未被正确使用。可以通过使用不同的寄存器存储中间结果，避免覆盖：

*mul $15, $6, $7*

*add $17, $15, $7 # 使用 $17 保存新的结果*

或者在寄存器堆模块中添加多路选择器，当同时读写同一单元时，将写入的值作为读到的值输出，便可解决该冲突。

### 控制相关：

在条件分支跳转出现控制相关：

*slti $10, $5, 80 # 判断是否小于 80*

*bne $10, 1, c1 # 根据结果跳转到 c1*

bne 指令需要 slti 的结果才能决定跳转方向。在流水线中，分支目标地址和跳转条件可能直到 EX 阶段才能确定。如果跳转预测失败，流水线需要清空误取指令，造成性能损失。

可使用分支预测的方法，使用硬件分支预测器预测跳转方向。如果预测正确，可以避免控制冒险。或者添加延迟槽，在 bne 指令后插入一条无关指令作为延迟槽，即便预测失败，也不会浪费指令周期，但若延迟槽内的指令也是跳转 或分支语句，还需要禁止其对 PC 的修改。

# 总结与体会

通过本次实验，我深入地学习了动态流水线的设计与实现过程，并完成了基于MIPS指令集的五段流水线CPU。实验让我全面理解了流水线各阶段：取指、译码、执行、访存和写回的功能分工及其在处理指令流中的协同工作机制。在实际搭建流水线的过程中，数据相关和控制相关问题是设计的主要挑战，通过引入数据前推、延迟槽和多路选择器等技术手段，我逐步解决了这些问题，也对流水线技术的核心思想有了更深刻的认识。

这次实验让我感受到了模块化设计的重要性。流水线的各个功能阶段被设计为独立的模块，不仅让复杂系统的实现变得更加清晰和条理化，还便于后续的调试和优化。在仿真阶段，通过逐级检查流水线的信号传递和指令执行结果，我能够及时发现问题所在，并调整设计以改进性能。例如，针对ALU运算和访存指令之间的数据相关问题，我通过优化数据前推机制有效减少了流水线停顿次数，从而显著提升了CPU的吞吐率。

此外，我对性能指标的分析也让我认识到流水线设计的价值和局限性。通过计算吞吐率、加速比和效率，我验证了设计的动态流水线在多种场景下的表现，并明确了硬件资源和执行效率之间的平衡点。在实验中，虽然通过优化减少了冲突，但对于复杂的分支跳转指令，预测错误仍会影响整体性能，这提示我未来可以研究更加智能的分支预测机制。

本次实验是一次理论与实践结合的宝贵经历，不仅帮助我加深了对计算机体系结构的理解，还提升了我在逻辑设计、代码实现和问题调试方面的能力。这些收获为我今后学习更复杂的体系结构奠定了基础，同时让我意识到性能优化与硬件复杂度之间的权衡是计算机系统设计中的永恒课题。我期待未来能够应用更先进的技术，进一步优化流水线设计，探索更高效的计算方法。

# 附件（所有程序见压缩包）