



**实验三：**

**使用MIPS指令实现求两个数组的点积**

**学院：计算机学院（国家示范性软件学院）**

**专业： 计算机科学与技术**

**班级： 2022211305**

**学号： 2022211683**

**姓名： 张晨阳**

**2025年4月21号**

**目录**

[1. 实验目的 1](#_Toc196151134)

[2. 实验平台 1](#_Toc196151135)

[3. 实验原理 1](#_Toc196151136)

[3.1. MIPS 汇编语言实现 1](#_Toc196151137)

[3.2. 定向功能与静态调度 1](#_Toc196151138)

[3.3. 优化原理 2](#_Toc196151139)

[4. 源程序代码及注释说明 3](#_Toc196151140)

[5. 优化后程序说明 5](#_Toc196151141)

[6. 性能分析与比较 8](#_Toc196151142)

[6.1. 优化前-未开启定向 8](#_Toc196151143)

[6.2. 开启定向功能 10](#_Toc196151144)

[6.3. 优化后程序 12](#_Toc196151145)

[7. 实验总结 14](#_Toc196151146)

# 实验目的

1. 通过实验熟悉实验 1 和实验 2 的内容
2. 增强汇编语言编程能力
3. 学会使用模拟器中的定向功能进行优化
4. 了解对代码进行优化的方法

# 实验平台

实验平台采用指令级和流水线操作级模拟器 MIPSsim。

# 实验原理

## MIPS 汇编语言实现

MIPS 汇编语言是一种简化的 RISC 架构，用于进行低级编程。在本实验中，我们将编写一个 MIPS 汇编程序，计算两个向量的点积。

具体步骤如下：

* **数据加载**：程序开始时，将向量 A 和 B 的地址加载到寄存器中。
* **循环计算**：通过一个循环，程序按顺序将向量的每一对元素加载到寄存器中，计算它们的乘积，并将结果累加到最终的点积中。
* **结束计算**：当循环遍历完所有向量元素时，将累加结果存储到指定的内存位置，并结束程序执行。

## 定向功能与静态调度

定向功能用于跟踪程序的每个阶段，确保程序执行的每个步骤都能按预期进行。通过定向功能，我们可以识别出程序中的潜在瓶颈，尤其是指令间的数据依赖。

静态调度是通过手动重排指令序列，避免不必要的等待和资源冲突，从而减少相关性。例如，可以将不相关的指令移动到其他指令之间，以减少处理器的空闲时间，从而提高程序的执行效率。

## 优化原理

在实验中，我们使用静态调度方法优化程序。通过分析程序中的指令依赖性，识别并消除不必要的延迟，使得程序在执行时更高效。优化后的程序通过减少流水线冲突和空闲时间，实现了更快的执行速度。

* **优化目标**：减少指令间的等待时间，优化数据访问顺序，避免数据冒险。
* **优化策略**：通过重排指令，确保在计算当前向量元素的乘积时，其他操作不会干扰执行。通过调整循环内的指令顺序，优化向量元素的访问顺序，减少不必要的内存访问和寄存器操作。

# 源程序代码及注释说明

1. .text

2. main:

3. ADDIU $r1, $r0, A # 将向量A的地址加载到寄存器$r1

4. ADDIU $r2, $r0, B # 将向量B的地址加载到寄存器$r2

5. ADDIU $r3, $r0, ans # 将结果存储地址加载到寄存器$r3

6. ADDIU $r4, $r0, n # 将向量维度n的地址加载到寄存器$r4

7. LW $r4, 0($r4) # 从内存中加载向量的维度（n值）

8. ADDIU $r5, $r0, 0 # 初始化循环索引i为0

9. ADDIU $r6, $r0, 0 # 初始化累加器$r6为0，用于存储结果

10.

11. loop:

12. LW $r7, 0($r1) # 从向量A中加载第i个元素到$r7

13. LW $r8, 0($r2) # 从向量B中加载第i个元素到$r8

14. MUL $r9, $r7, $r8 # 计算A[i] \* B[i]，结果存储到$r9

15. ADD $r6, $r6, $r9 # 将计算结果累加到累加器$r6中

16.

17. ADDI $r1, $r1, 4 # 将$r1移动到向量A的下一个元素

18. ADDI $r2, $r2, 4 # 将$r2移动到向量B的下一个元素

19. ADDI $r5, $r5, 1 # 索引i加1

20.

21. BNE $r5, $r4, loop # 如果索引i不等于向量的维度n，继续循环

22.

23. SW $r6, 0($r3) # 将最终的点积结果存储到内存地址ans中

24.

25. # 程序结束标志

26. TEQ $r0, $r0 # 比较寄存器$r0与自身，结束程序

27.

28. # 数据部分

29. .data

30. A: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 # 向量A的数据

31. B: .word 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 # 向量B的数据

32. n: .word 10 # 向量的维度n

33. ans: .word 0 # 用于存储计算结果的内存地址

**1. 程序概述**

本程序使用 MIPS 汇编语言编写，目的是计算两个向量 A 和 B 的点积。程序假设向量 A 和 B 的维度 n = 10。向量数据存储在内存中，每个向量元素占用 4 字节（32 位）。程序通过循环逐个计算每个元素的乘积并累加，最终将点积结果存储在内存中的指定位置。

**2. 程序结构**

**(1) 初始化阶段**

* **加载向量数据地址**：首先，程序通过 ADDIU 指令将向量 A 和 B 的地址加载到寄存器 $r1 和 $r2 中，准备访问向量元素。$r3 被用来存储点积的结果地址。
* **加载维度**：程序使用 LW 指令从内存中加载向量维度 n 的值到寄存器 $r4，这个值将用于控制循环次数。

**(2) 循环计算**

* **加载向量元素**：每次循环，程序会使用 LW 指令将向量 A和 B 的当前元素加载到寄存器 $r7 和 $r8。
* **乘法计算**：程序使用 MUL 指令计算当前元素的乘积 ，结果存储在寄存器 $r9。
* **累加结果**：将每次的乘积结果加到累加器 $r6 中，最终 $r6 将存储所有乘积之和，即点积结果。

**(3) 循环控制**

* **更新指针和索引**：通过 ADDI 指令更新指针 $r1 和 $r2，移动到下一个向量元素。寄存器 $r5 用来记录当前元素的索引，每次循环加 1。
* **判断是否继续**：BNE 指令判断索引 $r5 是否等于维度 $r4，如果不相等，则继续执行循环。

**(4) 结果存储与结束**

* **存储结果**：在循环结束后，程序通过 SW 指令将最终的点积结果存储到内存中的 ans 位置。
* **程序结束**：通过 TEQ 指令比较寄存器 $r0 与自身，作为程序的结束标志。

**3. 数据部分**

* **向量 A**：存储了 10 个整数，表示向量 A 的元素。
* **向量 B**：存储了 10 个整数，表示向量 B 的元素。
* **维度 n**：存储向量的维度，值为 10。
* **结果存储区 ans**：用于存储计算后的点积结果。

# 优化后程序说明

1. .text

2. main:

3. ADDIU $r1, $r0, A # 将向量A的地址加载到寄存器$r1

4. ADDIU $r2, $r0, B # 将向量B的地址加载到寄存器$r2

5. ADDIU $r3, $r0, ans # 将结果存储的地址加载到寄存器$r3

6. ADDIU $r4, $r0, n # 将向量维度n的地址加载到寄存器$r4

7. LW $r4, 0($r4) # 从内存中加载向量维度n的值到寄存器$r4

8. ADDIU $r5, $r0, 0 # 初始化循环索引i为0，存储在寄存器$r5

9. ADDIU $r6, $r0, 0 # 初始化结果寄存器$r6为0，用于存储点积结果

10.

11. loop:

12. LW $r7, 0($r1) # 从向量A中加载第i个元素（A[i]）到寄存器$r7

13. LW $r8, 0($r2) # 从向量B中加载第i个元素（B[i]）到寄存器$r8

14.

15. MUL $r9, $r7, $r8 # 计算A[i] \* B[i]，结果存储在寄存器$r9

16.

17. ADDI $r1, $r1, 4 # 移动到向量A的下一个元素，4字节为一个整数的大小

18. ADDI $r2, $r2, 4 # 移动到向量B的下一个元素，同样是4字节

19. ADDI $r5, $r5, 1 # 索引i加1

20.

21. ADD $r6, $r6, $r9 # 将计算的A[i] \* B[i]的结果累加到寄存器$r6中，得到点积结果

22.

23. BNE $r5, $r4, loop # 如果索引i不等于向量维度n，继续循环，执行下一次迭代

24.

25. SW $r6, 0($r3) # 将点积结果存储到内存地址ans中

26.

27. # 结束程序

28. TEQ $r0, $r0 # 比较寄存器$r0与自身，作为程序结束的标志

29.

30. # 数据部分

31. .data

32. A: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 # 向量A的元素

33. B: .word 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 # 向量B的元素

34. n: .word 10 # 向量的维度n

35. ans: .word 0 # 存储点积结果的内存地址

**1. 程序结构**

**(1) 初始化阶段**

* **加载向量数据地址**：程序首先使用 ADDIU 指令将向量 A 和 B 的地址加载到寄存器 $r1 和 $r2 中。寄存器 $r3 用于存储最终的点积结果的内存地址。
* **加载向量维度**：通过 LW 指令将向量的维度 n 从内存中加载到寄存器 $r4，用来控制循环的次数。

**(2) 循环计算**

优化前的程序会在每次循环迭代中，等待乘法结果才能进行加法计算，并且每次都需要更新内存地址指针。

优化后的程序将加载数据、计算乘法、累加结果以及指针更新进行了合理安排，尽量减少指令之间的等待时间。

* **加载向量元素**：每次循环中，使用 LW 指令分别从向量 A和 B 中加载当前元素到寄存器 $r7 和 $r8。
* **乘法计算**：计算当前元素的乘积 ，结果存储在寄存器 $r9。
* **指针更新与索引增加**：通过 ADDI 指令分别更新向量 A 和 B的地址，移动到下一个元素，同时更新索引寄存器 $r5。
* **累加结果**：将每次计算的乘积加到结果寄存器 $r6 中，通过 ADD 指令将乘积累加到之前的结果中。

**(3) 循环控制**

* **继续循环判断**：通过 BNE 指令检查索引 $r5 是否与维度 $r4 相等。如果不相等，继续执行循环，直到所有元素计算完毕。

**(4) 结果存储与程序结束**

* **存储结果**：在循环结束后，程序通过 SW 指令将计算的点积结果存储到内存地址 ans 中。
* **结束程序**：通过 TEQ 指令比较寄存器 $r0 与自身，标记程序结束。

**2. 数据部分**

* **向量 A**：存储了 10 个整数，表示向量 A 的元素。
* **向量 B**：存储了 10 个整数，表示向量 B 的元素。
* **维度 n**：存储向量的维度，值为 10。
* **结果存储区 ans**：用于存储计算后的点积结果。

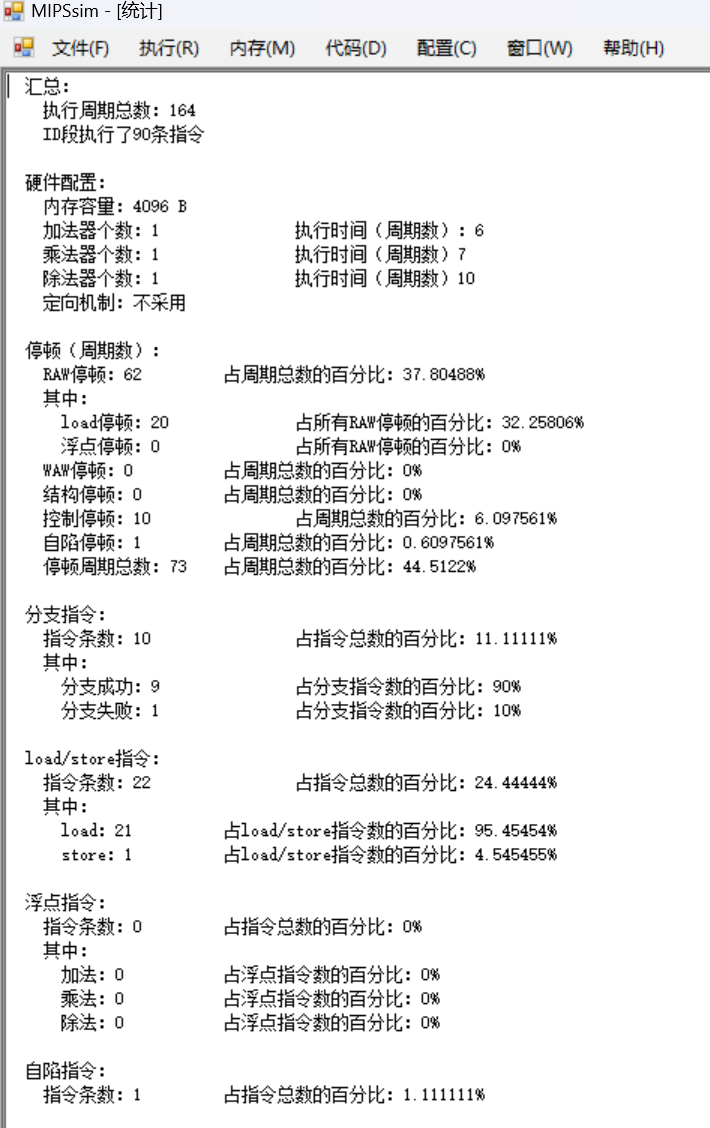
**3. 优化内容与策略**

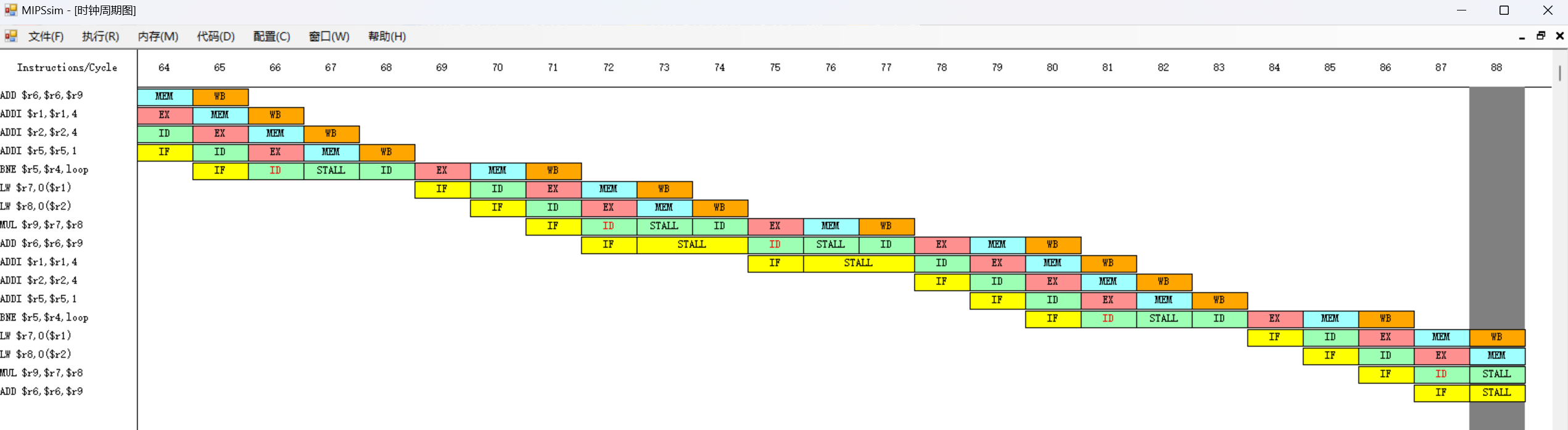
优化后的代码在指令的重排上做了以下优化：

* **减少数据依赖和等待**：通过将加载数据、乘法计算和累加等操作分开并重排顺序，尽量避免了指令间的依赖等待。例如，在计算乘积的同时更新指针，使得下一轮的计算可以尽快进行，减少了流水线等待。
* **优化指令顺序**：通过将不相关的指令重排，减少了乘法操作与加法操作之间的依赖，使得计算过程更加流畅，提高了指令的并行性。
* **指令间并行执行**：通过合理安排指令，允许不同的指令在不同的流水线阶段并行执行，最大程度地减少了流水线冲突和指令延迟。

# 性能分析与比较

## 优化前-未开启定向

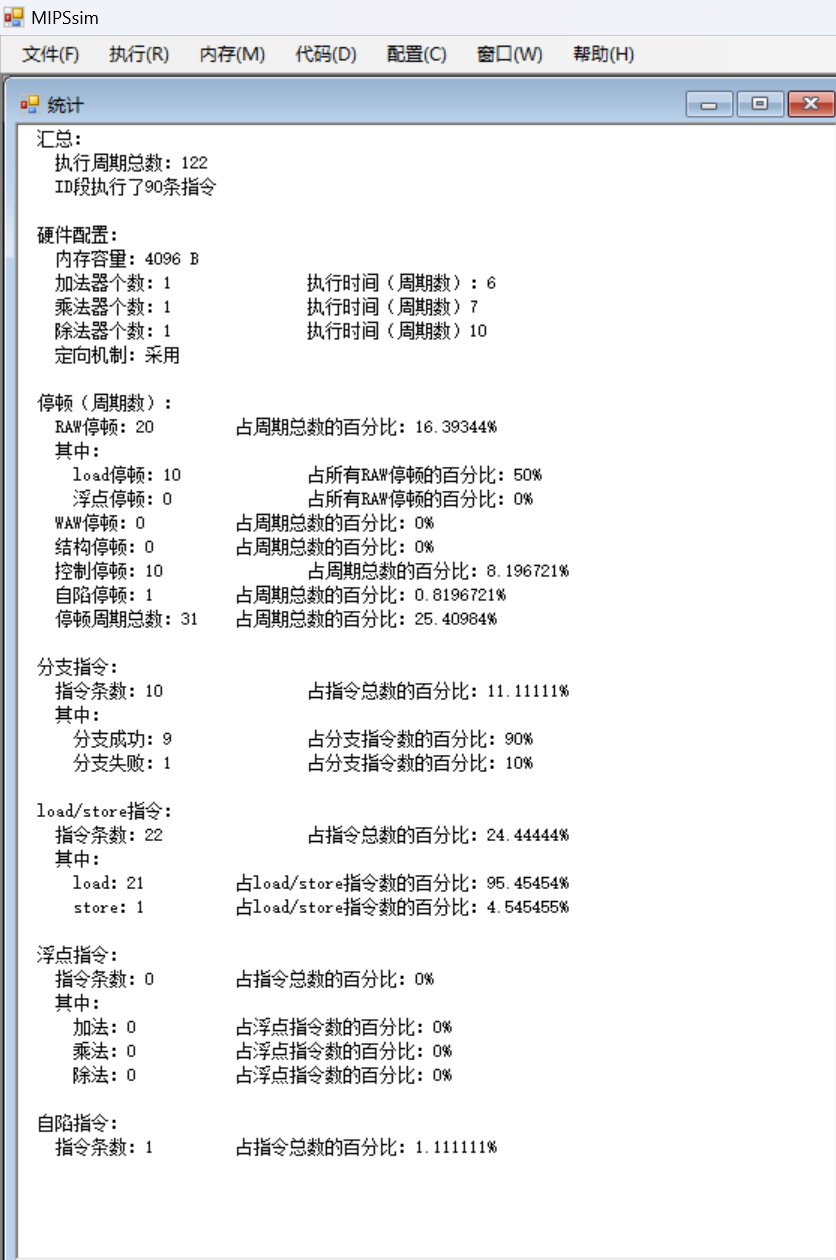


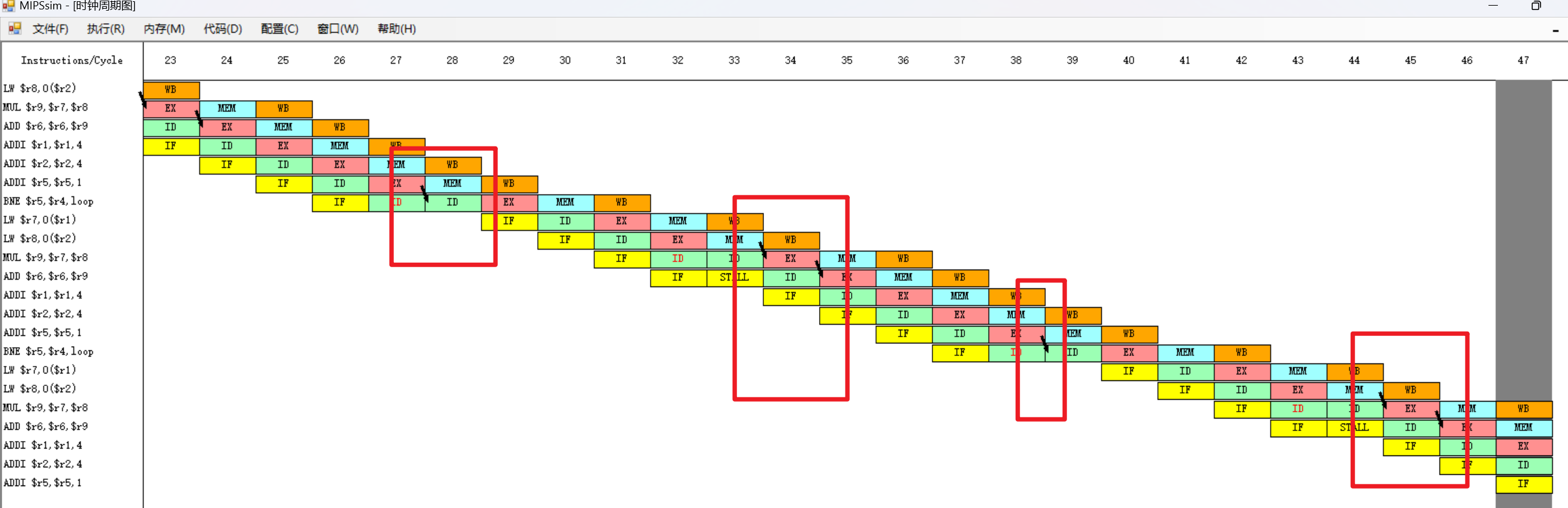


根据STALL的时钟周期，不难发现，主要的数据相关来自：

* MUL $r9, $r7, $r8 和 ADD $r6, $r6, $r9
* LW $r8, 0($r2) 和 MUL $r9, $r7, $r8

## 开启定向功能

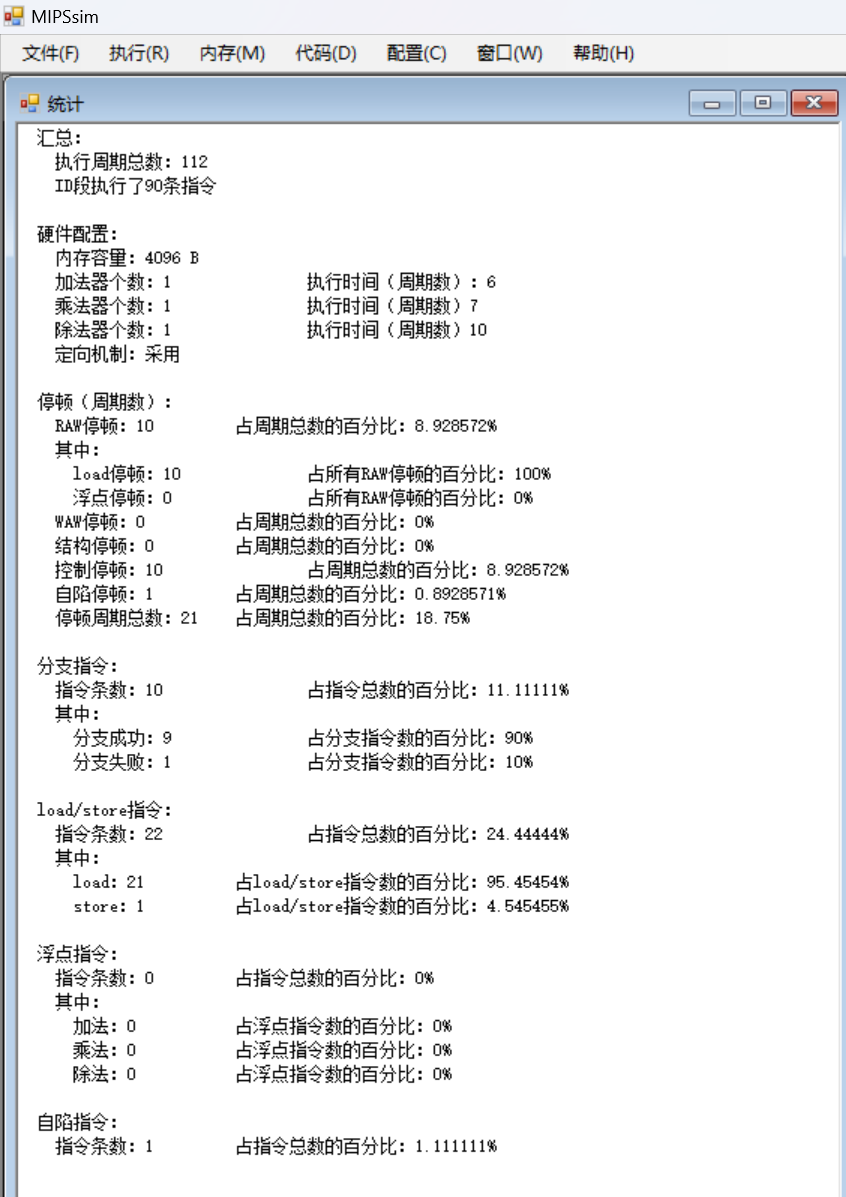


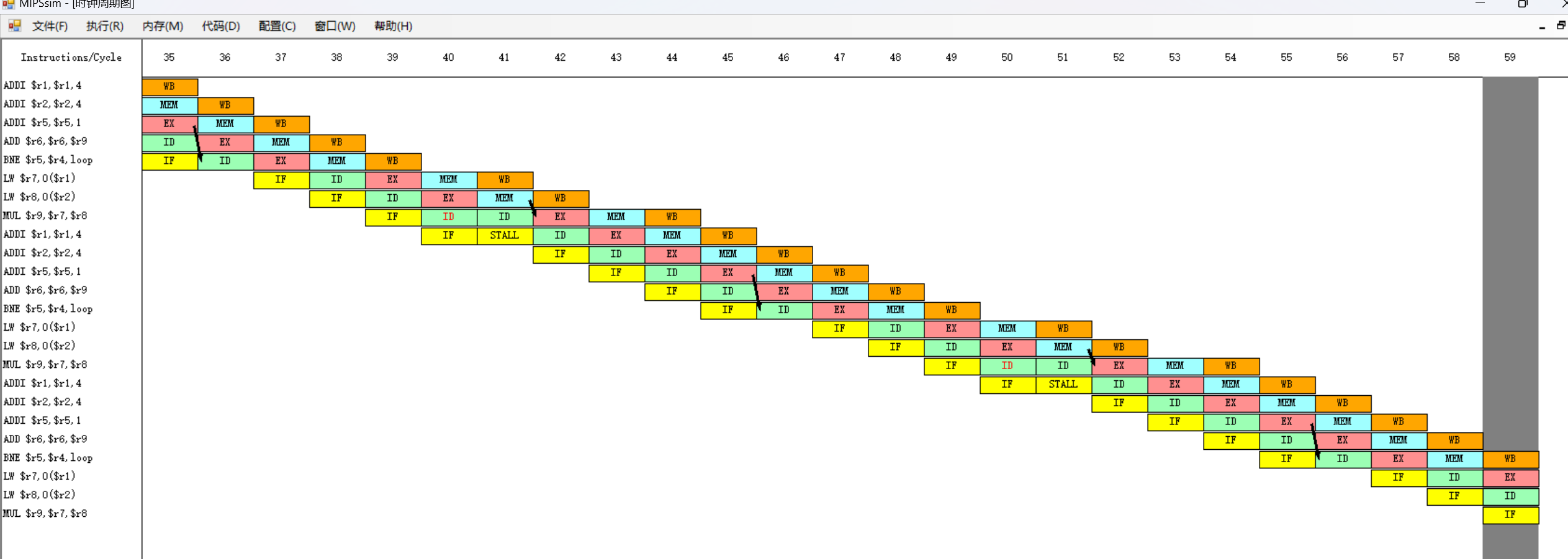


主要的定向内容已在图中标出。

定向功能消除了部分数据冲突，性能提升了约 倍。

## 优化后程序





与最开始的程序相比，效率大概是 倍，

与仅仅开启定向的程序相比，效率大概是 倍。

从三个执行过程的时钟周期图可以发现，性能提升的原因主要有两点：

（1） 定向技术的使用优化了数据的流动过程；

（2） 数据冲突的避免减少了执行过程中的STALL。

# 实验总结

通过本次实验，我对 MIPS 汇编语言的编程、流水线模拟以及程序优化有了更深刻的理解。整个实验过程不仅帮助我掌握了如何实现向量点积计算，还使我认识到程序执行效率和优化的重要性。

总之，本次实验不仅提升了我的汇编编程能力，还让我认识到优化对程序性能的影响。在今后的学习中，我将继续深入研究程序优化，力求在设计和开发过程中最大化地提高程序的运行效率。