



**实验四：**

**使用MIPS指令实现冒泡排序法**

**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5月16号**

**目录**

[1. 实验目的 1](#_Toc198309808)

[2. 实验平台 1](#_Toc198309809)

[3. 实验原理 1](#_Toc198309810)

[3.1. MIPS 汇编语言实现 1](#_Toc198309811)

[3.2. 定向功能与静态调度 1](#_Toc198309812)

[3.3. 优化原理 2](#_Toc198309813)

[4. 源程序代码及注释说明 3](#_Toc198309814)

[5. 优化后程序说明 5](#_Toc198309815)

[6. 性能分析与比较 7](#_Toc198309816)

[6.1. 优化前-未开启定向 7](#_Toc198309817)

[6.2. 开启定向功能 9](#_Toc198309818)

[6.3. 优化后程序 11](#_Toc198309819)

[7. 实验总结 13](#_Toc198309820)

# 实验目的

1. 掌握静态调度方法
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

# 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，我们编写了一个 MIPS 汇编程序，完成对一维整数数组的冒泡排序。冒泡排序的基本思想是不断比较相邻元素，并将较大的元素逐步“冒泡”至数组末端。

具体步骤如下：

* **数据初始化**：在数据段定义一个长度不少于 10 的整数数组，并在程序开始时将数组首地址和长度加载到寄存器中。
* **双重循环控制**：使用嵌套循环结构实现冒泡排序的逻辑，外层控制排序轮数，内层进行相邻元素的比较与交换。
* **元素比较与交换**：将相邻两个元素加载至寄存器，进行大小比较，若顺序不正确则交换两者在内存中的位置。
* **循环终止判断**：外层循环逐步减少比较次数，直到所有元素有序。

## 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, arr # 获取数组首地址

4. ADDIU $r2, $r0, len # 获取len的地址

5. LW $r2, 0($r2) # 获取数组长度

6. SLL $r2, $r2, 2 # len<<2

7. ADD $r2, $r2, $r1 # arr[len]的地址

8.

9. outer\_loop:

10. ADDI $r2, $r2, -4 # 每轮减少比较范围

11. BEQ $r1, $r2, exit # 如果起始地址等于末尾地址，排序完成

12. ADDIU $r3, $r1, 0 # 初始 k = 0, 获取arr[k]的地址

13.

14. inner\_loop:

15. LW $r4, 0($r3) # 获取arr[k]的值

16. LW $r5, 4($r3) # 获取arr[k+1]的值

17. SLT $r6, $r5, $r4 # arr[k+1] < arr[k] ?

18. BEQ $r6, $r0, skip # 如果不需要交换，跳过

19. SW $r5, 0($r3) # arr[k] = arr[k+1]

20. SW $r4, 4($r3) # arr[k+1] = arr[k]

21.

22. skip:

23. ADDI $r3, $r3, 4 # k++：前移一个元素

24. BNE $r3, $r2, inner\_loop # 如果还未到达末尾，继续比较下一对

25. BEQ $r0, $r0, outer\_loop # 回到外层循环，进行下一轮

26.

27. exit:

28. TEQ $r0, $r0 # 结束

29.

30. .data

31. arr:

32. .word 10,9,8,7,6,5,4,3,2,1

33. len:

34. .word 10

**（1）程序结构**

程序由以下几个主要部分组成：

* **数据加载部分**：加载数组 arr 的起始地址与长度 len。
* **外层循环 outer\_loop**：控制每一轮冒泡排序的比较范围，每轮“冒泡”最大的元素到当前末尾。
* **内层循环 inner\_loop**：执行每一轮中所有相邻元素的比较和必要的交换操作。
* **程序结束标志**：排序完成后触发伪指令 TEQ 表示程序终止。

**（2）寄存器用途说明**

|  |  |
| --- | --- |
| **寄存器** | **含义或用途** |
| $r1 | 数组 arr 的起始地址 |
| $r2 | 当前一轮排序的末尾地址（逐轮缩减） |
| $r3 | 当前比较元素 arr[k] 的地址 |
| $r4 | 当前元素 arr[k] 的值 |
| $r5 | 相邻元素 arr[k+1] 的值 |
| $r6 | 比较标志（arr[k+1] < arr[k]） |

**（3）排序逻辑说明**

* 程序首先通过 SLL 将 len 转换为字节偏移，并与数组起始地址相加得到当前数组末尾地址。
* 外层循环 outer\_loop 控制冒泡排序的轮次，每轮都将末尾地址减去一个元素宽度（4字节），缩小比较范围。
* 内层循环 inner\_loop 通过连续读取当前元素和下一个元素，利用 SLT 指令判断是否满足交换条件。
* 若 arr[k+1] < arr[k]，则交换两者的位置；否则跳过。
* 内层循环每次推进一个元素地址（即 k++），直到达到当前外层设定的边界。
* 当所有元素排序完成（即 r1 == r2），程序跳出循环并终止。

# 优化后程序说明

1. .text

2. main:

3. ADDIU $r1, $r0, arr # 获取数组首地址

4. ADDIU $r2, $r0, len # 获取len的地址

5. LW $r2, 0($r2) # 获取数组长度

6. SLL $r2, $r2, 2 # len<<2

7. ADD $r2, $r2, $r1 # arr[len]的地址

8.

9. outer\_loop:

10. ADDI $r2, $r2, -4 # 缩短排序范围：相当于 len--

11. ADDIU $r3, $r1, 0 # 初始 k = 0, 获取arr[k]的地址

12. BEQ $r1, $r2, exit # 如果只剩一个元素，排序完成，跳出

13.

14. inner\_loop:

15. LW $r4, 0($r3) # 获取arr[k]的值

16. ADDI $r3, $r3, 4 # k++

17. LW $r5, 0($r3) # 获取arr[k+1]的值

18. SLT $r6, $r5, $r4 # if (arr[k+1] < arr[k]) -> $r6 = 1

19. BEQ $r6, $r0, skip # 如果不需要交换，跳转到 skip

20. SW $r5, -4($r3) # arr[k] = arr[k+1]

21. SW $r4, 0($r3) # arr[k+1] = arr[k]

22.

23. skip:

24. BNE $r3, $r2, inner\_loop # 如果还未到达当前排序边界，继续循环

25. BEQ $r0, $r0, outer\_loop # 回到外层循环，开始新一轮冒泡

26.

27. exit:

28. TEQ $r0, $r0 # 结束

29.

30. .data

31. arr:

32. .word 10,9,8,7,6,5,4,3,2,1

33. len:

34. .word 10

**（1）程序结构**

程序结构与原始版本相同，由三大部分构成：

* **初始化部分**：加载数组起始地址、长度等必要信息。
* **双重循环结构**：外层循环控制排序轮次，内层循环完成元素比较和交换。
* **程序终止部分**：排序完成后通过 TEQ 停止执行。

**（2）寄存器用途**

|  |  |
| --- | --- |
| **寄存器** | **含义或用途** |
| $r1 | 数组起始地址 arr |
| $r2 | 当前一轮排序的终止地址（随轮次递减） |
| $r3 | 当前内层循环的指针地址 arr[k+1] |
| $r4 | arr[k] 的值（通过 $r3 - 4 获取） |
| $r5 | arr[k+1] 的值（当前 $r3 地址） |
| $r6 | 比较结果标志：arr[k+1] < arr[k] 时为1 |

**（3）排序实现逻辑**

* 首先通过 SLL 将数组长度转换为字节数，与数组首地址相加得到数组尾地址。
* 外层循环 outer\_loop 每轮将数组尾地址递减 4（即减少一个元素的比较范围）。
* 内层循环 inner\_loop 逐对比较相邻元素，具体流程如下：
  + 先取出当前元素 arr[k]，再提前将指针加 4，指向 arr[k+1]；
  + 然后取出 arr[k+1]，与 arr[k] 进行大小比较；
  + 若 arr[k+1] < arr[k]，执行交换操作；
  + 继续将指针向后推进一位，重复以上操作直到本轮结束。
* 当 r1 == r2 时，说明只剩下一个元素，无需再排序，程序跳出。

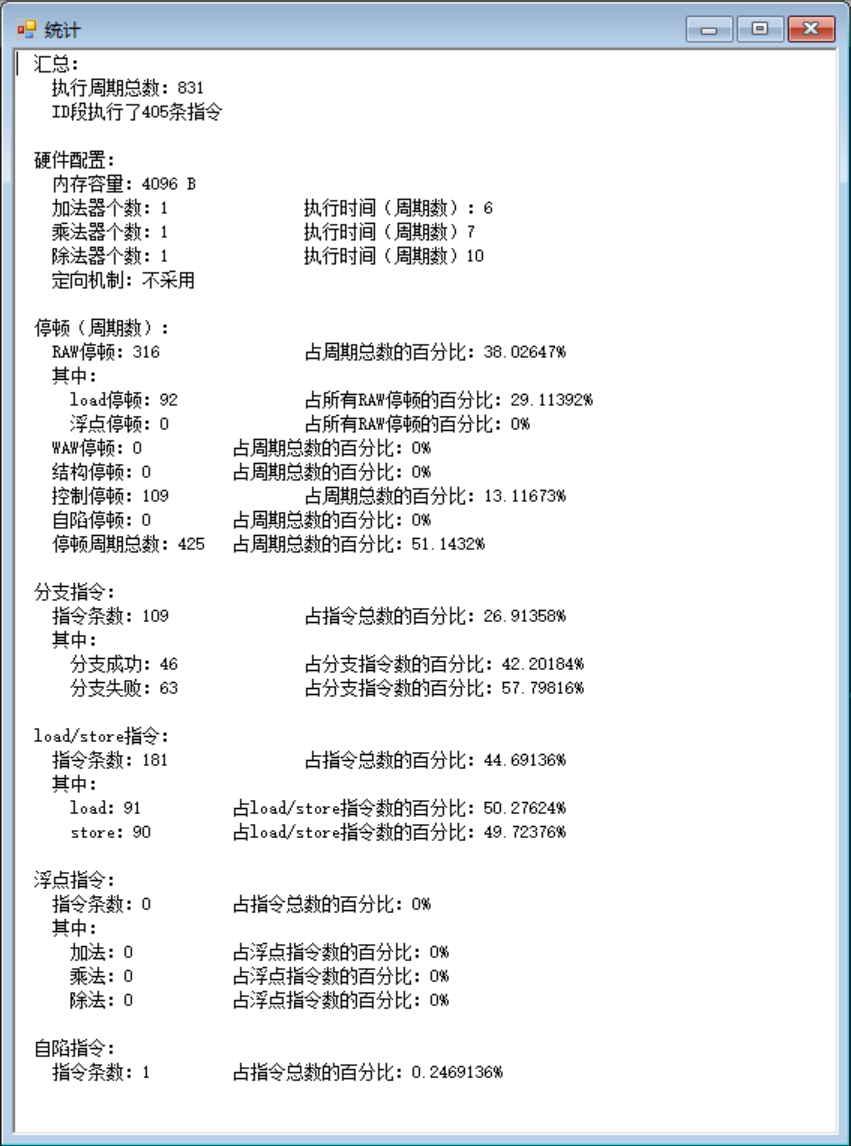
**（4）优化点分析**

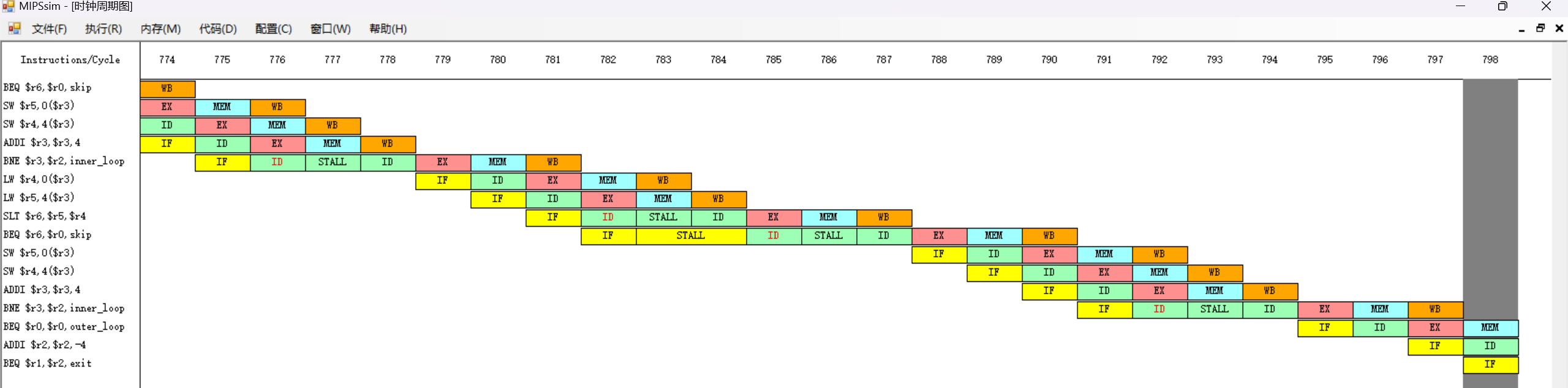
本程序通过以下静态调度策略进行了流水线友好的重排优化：

* **将 ADDI $r3, $r3, 4 移至两次 LW 之间**，避免了原始版本中 LW 后立即使用引起的数据冒险；
* **交换条件后执行无关指令**，填补潜在流水线空隙；
* **统一使用 r3 表示当前位置 arr[k+1] 的地址**，使得前一个元素 arr[k] 可通过 -4($r3) 访问，简化地址运算。

# 性能分析与比较

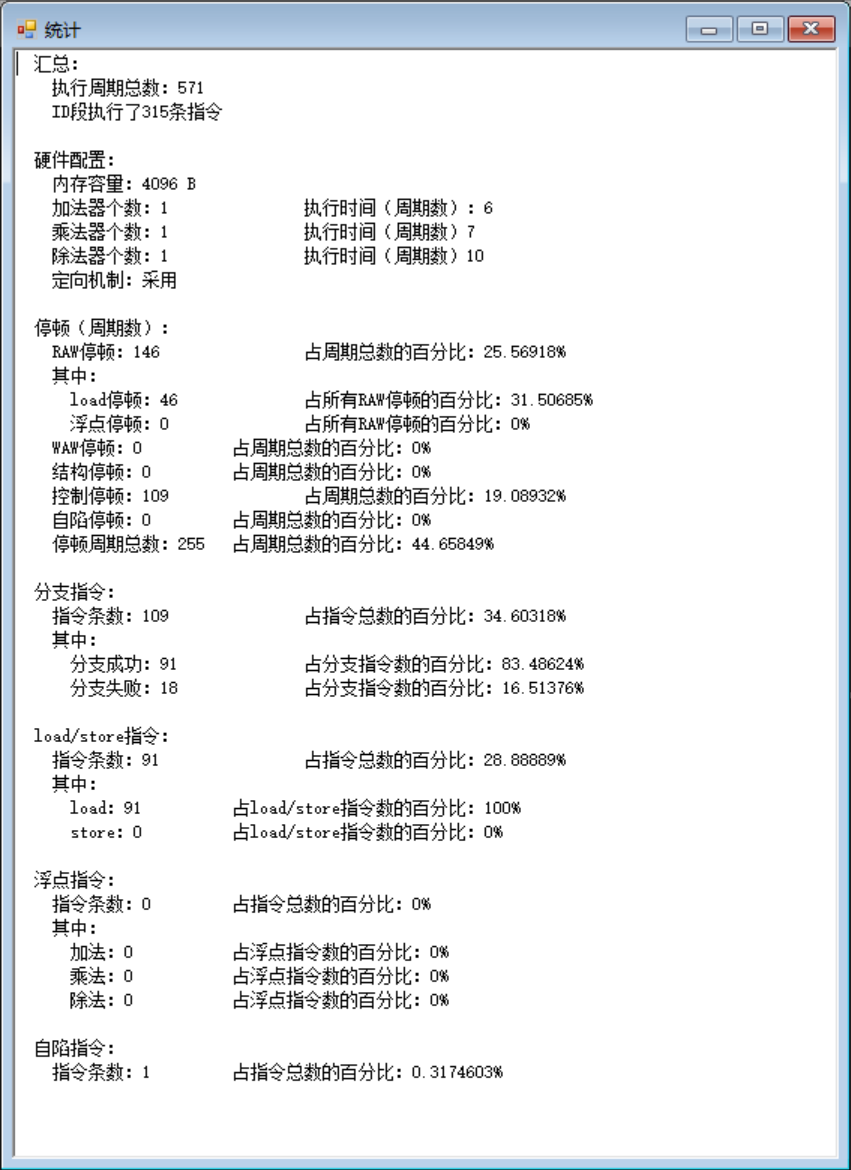
## 优化前-未开启定向

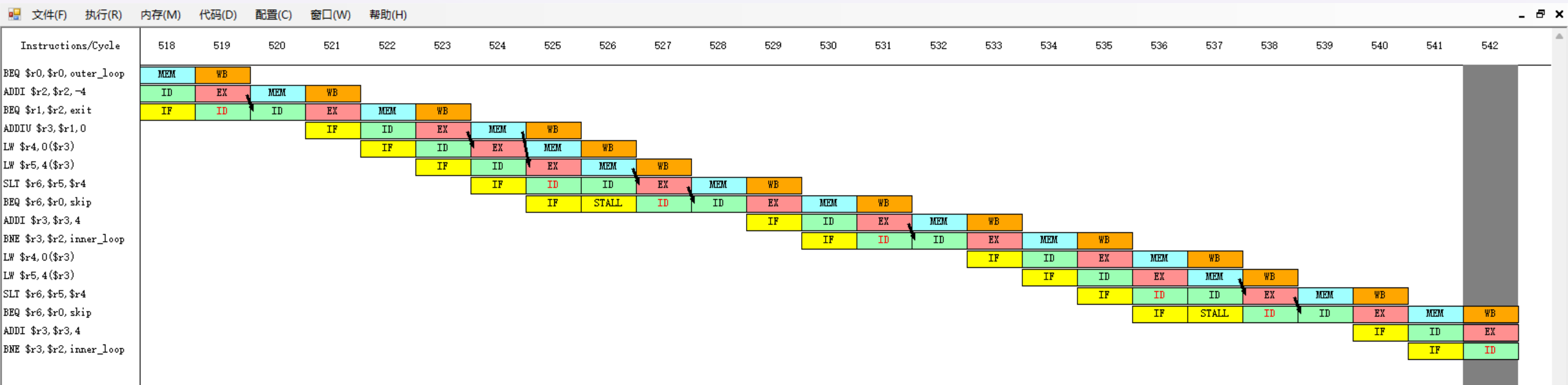




可以发现有相当一部分指令发生了冲突，冲突类型是 RAW。

## 开启定向功能



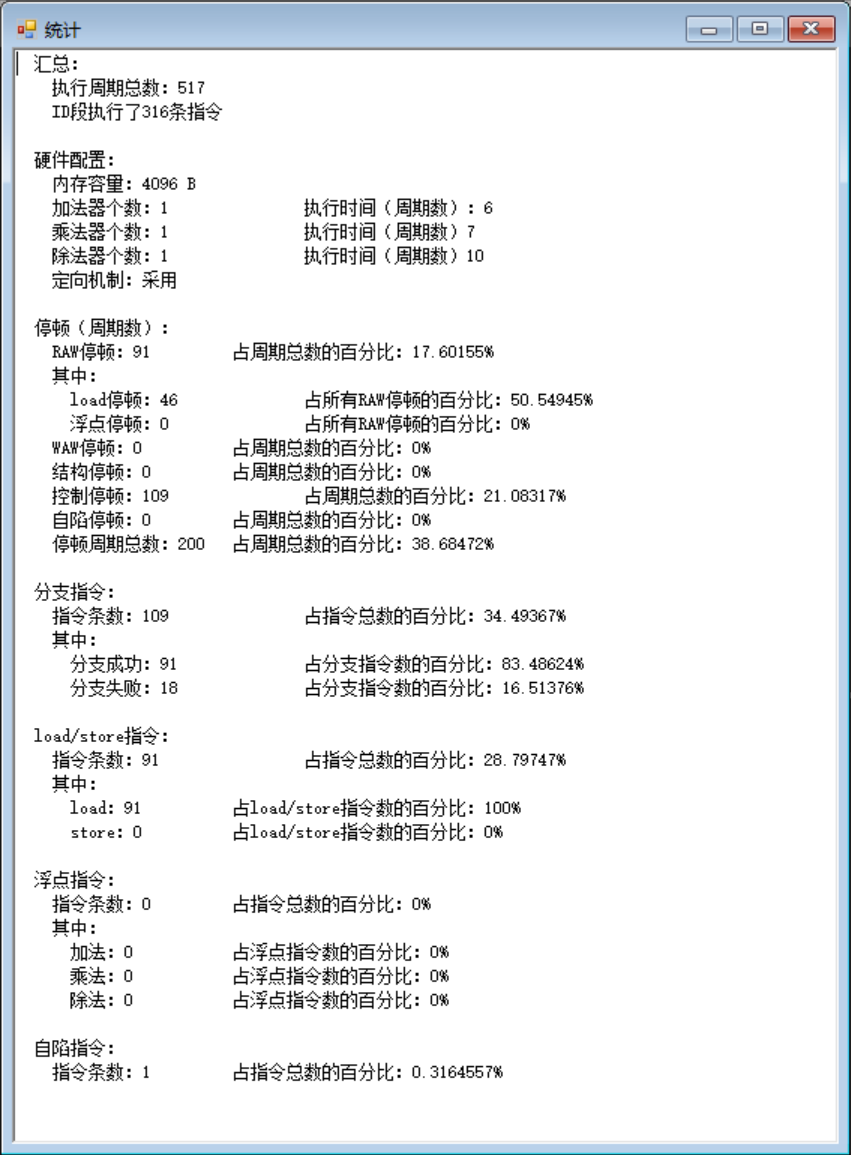


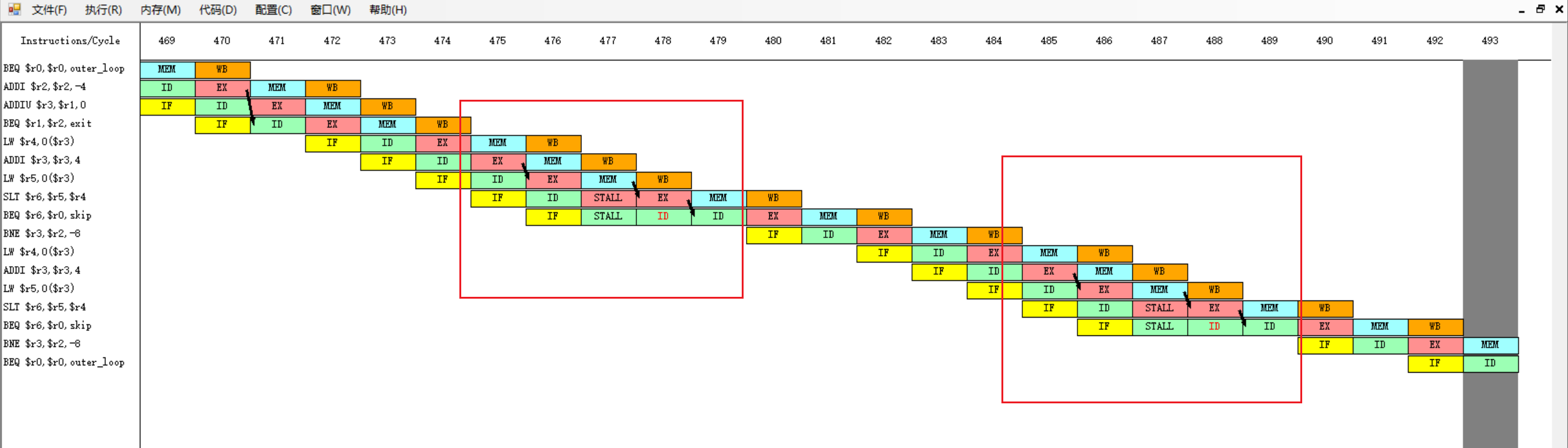
可以发现开启定向功能，能够消除部分数据冲突。

但当相邻指令发生冲突时，仍然需要stall。

性能提升了倍。

## 优化后程序





消除了 **LW** $r5**,** 4**(**$r3**)** 和 **SLT** $r6**,** $r5**,** $r4 以外的所有 RAW 冲突，**减少数据相关性。**

性能提升了倍。

# 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冒泡排序，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。