

**实验报告**



**题目： 使用 MIPS 指令实现冒泡排序法**

|  |  |  |  |
| --- | --- | --- | --- |
| **计算机系统结构实验小组成员信息** | | | |
| **班级** | **姓名** | **学号** | **学院** |
| **2020211314** | **王小龙** | **2020211502** | **计算机学院** |
| **2020211314** | **闻奔放** | **2020211505** | **计算机学院** |
| **2020211314** | **黄洪健** | **2020211371** | **计算机学院** |

**注：红色标出的成员为本次实验的完成者**

**2023年 5月 9日**

一、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

1. 实验原理
2. 冒泡算法的运作原理

①比较相邻的元素。如果第一个比第二个大，就交换他们两个。

②对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。

③针对所有的元素重复以上的步骤，除了最后一个。

④持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

1. 定向技术的原理

在某条指令产生一个计算结果之前，其他指令并不真正需要该计算结构，如果能将该计算结果从其产生的地方直接送到其他指令需要它的地方，就可以避免暂停。

1. 静态调度的优化原理

依靠编译器确定并分理处程序中存在相关的指令，通过调整指令的顺序来减少流水线的停顿,提高程序的执行速度。

1. 冒泡排序代码清单及注释说明

.text

main:

ADDIU $r1, $r0, 15 #保存数组大小

ADDIU $r2, $r0, 14 #外循环计数 i

LOOP1: #外循环

ADDIU $r3, $r0, array #数组array

ADDIU $r4, $r0, 0 #内循环计数 j

LOOP2: #内循环

LW $r5, 0($r3) # array[j]

LW $r6, 4($r3) # array[j+1]

DSUB $r7, $r5, $r6 #array[j] - array[j+1]

BLTZ $r7, bk #若array[j] < array[j+1],则跳转到bk

SW $r6, 0($r3) #若array[j] > array[j+1],则交换位置

SW $r5, 4($r3)

bk:

ADDIU $r4, $r4, 1 #j=j+1

ADDIU $r3, $r3, 4 #下一个数

DSUB $r8, $r2, $r4 #i - j

BGTZ $r8, LOOP2

ADDIU $r2, $r2, -1 #i=i-1

BGTZ $r2, LOOP1 #i > 0 继续外循环

TEQ $r0, $r0 #End

.data

array:

.word 5,30,10,24,29,67,28,81,34,17,13,7,50,4,1

1. 优化后的程序代码清单

.text

main:

ADDIU $r1, $r0, 15 #保存数组大小

ADDIU $r2, $r0, 14 #外循环计数 i

LOOP1: #外循环

ADDIU $r3, $r0, array #数组array

ADDIU $r4, $r0, 0 #内循环计数 j

LOOP2: #内循环

LW $r5, 0($r3) # array[j]

LW $r6, 4($r3) # array[j+1]

ADDIU $r4, $r4, 1 #j=j+1

DSUB $r7, $r5, $r6 #array[j] - array[j+1]

DSUB $r8, $r2, $r4 #i - j

BLTZ $r7, bk #若array[j] < array[j+1],则跳转到bk

SW $r6, 0($r3) #若array[j] > array[j+1],则交换位置

SW $r5, 4($r3)

bk:

ADDIU $r3, $r3, 4 #下一个数

BGTZ $r8, LOOP2

ADDIU $r2, $r2, -1 #i=i-1

BGTZ $r2, LOOP1 #i > 0 继续外循环

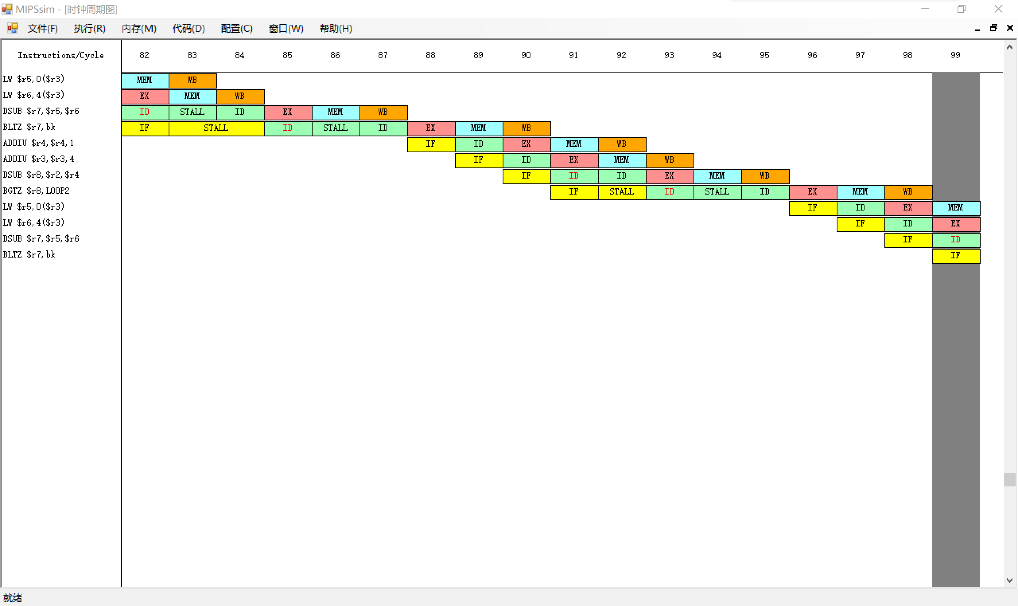
TEQ $r0, $r0 #End

.data

array:

.word 5,30,10,24,29,67,28,81,34,17,13,7,50,4,1

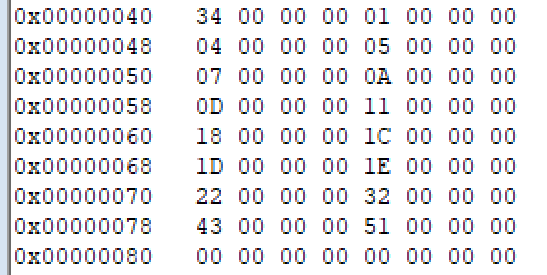
五、未优化代码和优化代码性能分析比较结果



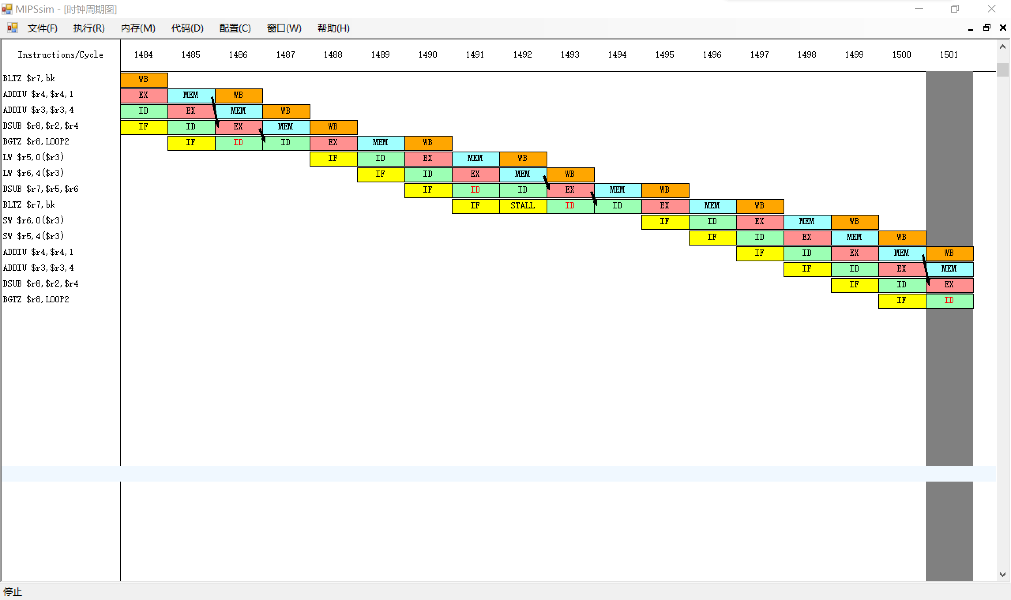
未优化代码未使用定向技术的流水图



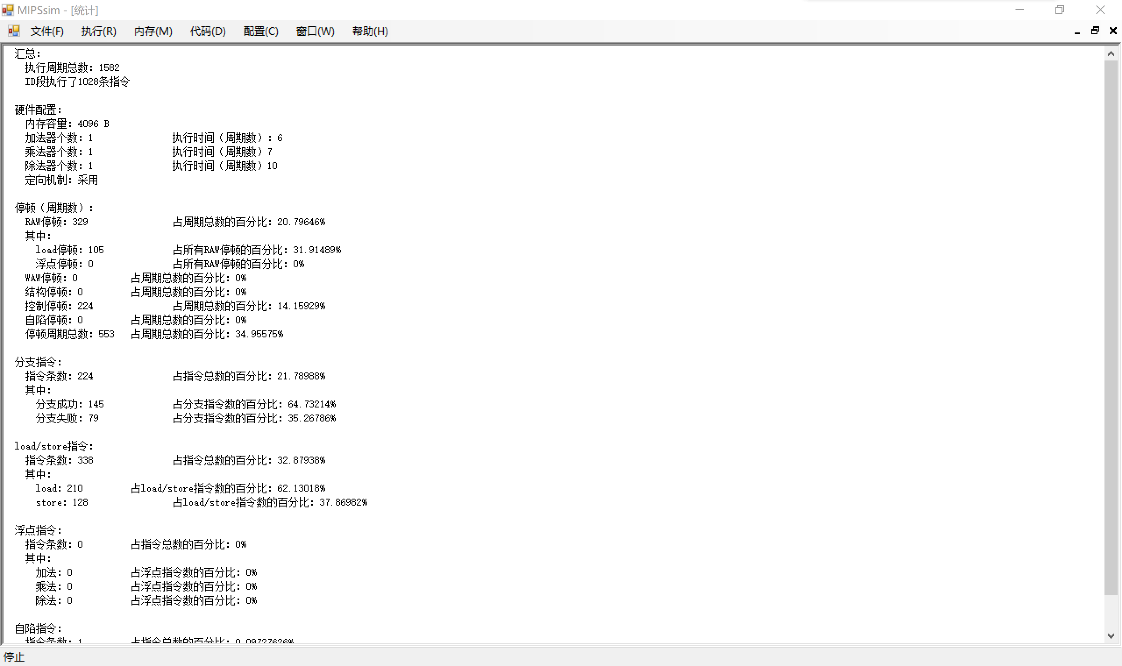
未优化代码未使用定向技术的停顿数



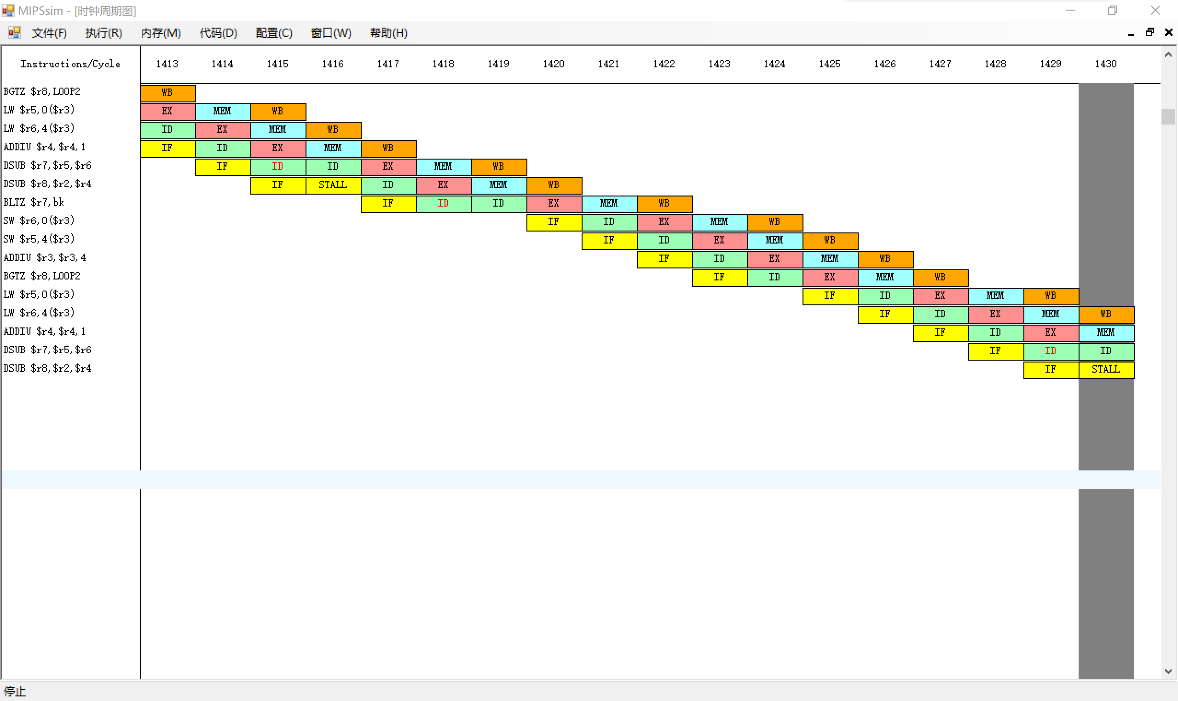
运行结果



未优化代码使用定向技术的流水图



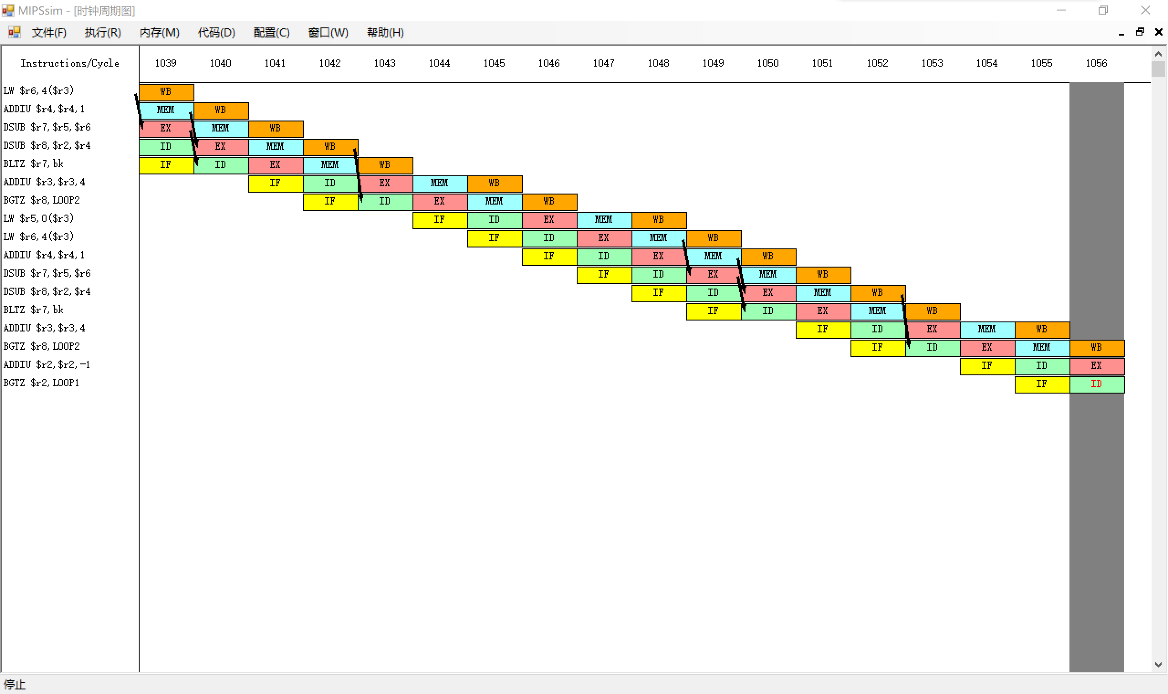
未优化代码使用定向技术的停顿数



优化代码未使用定向技术的流水图



优化代码未使用定向技术的停顿数



优化代码使用定向技术的流水图



优化代码使用定向技术的停顿数

通过上述图片可以发现，未优化代码未使用定向技术的停顿数为777，占周期总数38%，使用定向技术后停顿数减少至329，占周期总数20%，优化了约57%。优化后停顿数为252，占周期总数16%，优化效果比仅使用定向技术好。优化后再使用定向技术可以将停顿数锐减至14，占周期总数1.2%，效果非常好。