# 实验4 使用MIPS指令实现冒泡排序

## 1 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

## 2 实验平台

实验平台采用指令级和流水线操作级模拟器MIPSsim。

## 3 实验说明

（1） 自行编写一个实现冒泡排序的汇编程序，该程序要求可以实现对一维整数数组 进行冒泡排序。 冒泡排序算法的运作如下：

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。 要求数组长度不得小于10

（2）启动MIPSsim。

（3）载入自己编写的程序，观察流水线输出结果。

（4）使用定向功能再次执行代码，与刚才执行结果进行比较，观察执行效率的不同。

（5）采用静态调度方法重排指令序列，减少相关，优化程序

（6）对优化后的程序使用定向功能执行，与刚才执行结果进行比较，观察执行效率的 不同。

## 4 实验内容和步骤

冒泡排序的函数和主函数如下所示

1. void bubble(int a[], int n){
2. int temp;
3. **for**(int i=0;i<n-1;i++){
4. **for**(int j=0;j<n-1-i;j++){
5. **if**(a[j] > a[j+1]){
6. temp = a[j];
7. a[j] = a[j+1];
8. a[j+1] = temp;
9. }
10. }
11. }
12. }
13. int main(){
14. int a[10] = {5,7,10,8,4,2,1,3,6,9};
15. int n = 10;
16. bubble(a,n);
17. **return** 0;
18. }

根据上述代码，先完成主函数代码。

|  |
| --- |
| main:  ADDIU $r4, $r0, a # r4放入a的地址  ADDIU $r5, $r0, n # r5存n的地址  LW $r5, 0($r5) # r5存n的值  BEGZAL $r0, bubble # 跳到bubble排序  NOP  TEQ $r0, $r0 # 结束 |

主函数中，先将数组a和n的值存放到寄存器中，然后调用了bubble函数。

在bubble函数中要处理两层循环。

首先计算数组长度 n 减 1，并将结果存储在 $r7 中。

如果数组长度小于等于 1，则直接返回，不需要进行排序。

将数组长度左移两位，相当于乘以 4，以便后续计算数组的末尾地址。

计算数组中第二个元素的地址，并将其存储在 $r8 中。

计算数组的末尾地址，并将其存储在 $r6 中，用于检查是否越界。

外循环下，将当前循环的起始地址存储在 $r2 中。

内循环中，要从内存中加载当前位置和下一个位置的数组元素到 $r3 和 $r4 中。

比较当前位置和下一个位置的数组元素，如果前者小于后者，则交换它们。

结束本次内部循环，继续下一次循环。

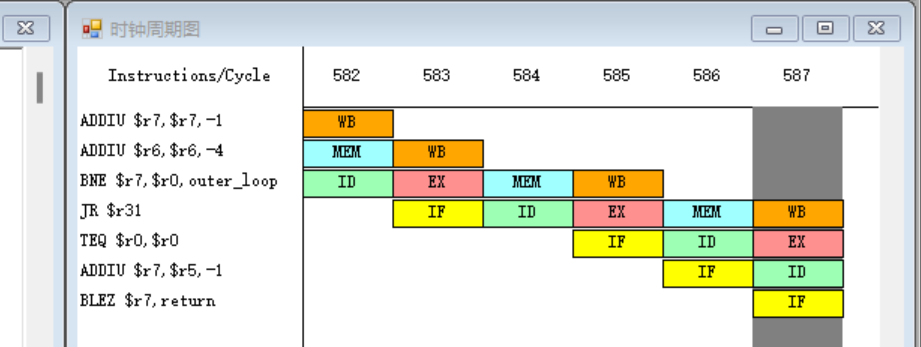
循环结束后，检查是否已经遍历完整个数组，如果没有，则继续内部循环。

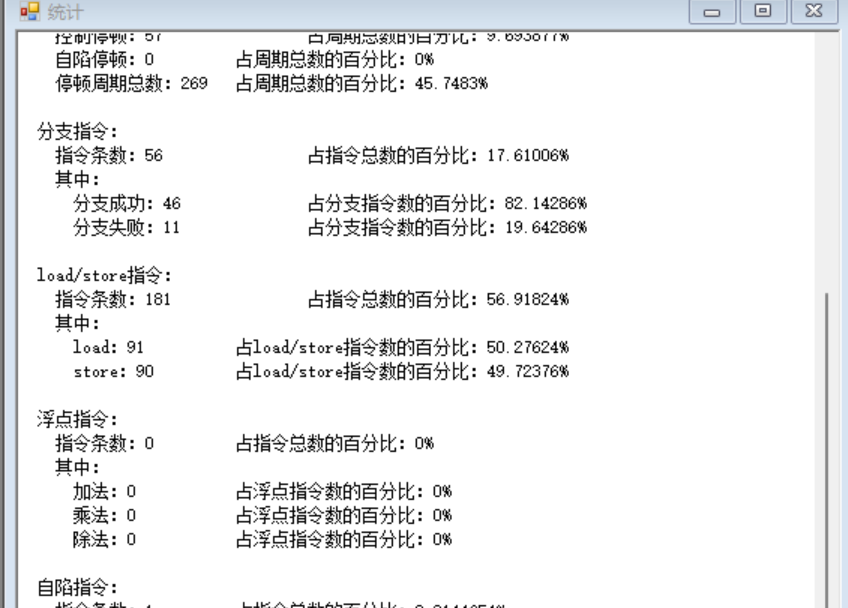
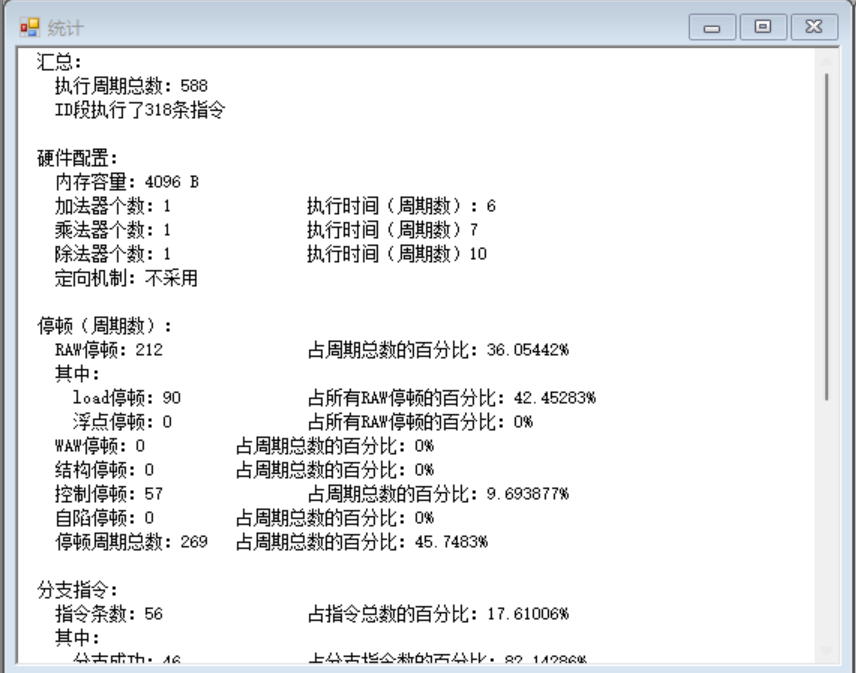
更新外部循环的计数器和数组的末尾地址。

检查是否还需要进行下一轮外部循环，如果需要，则跳转到 outer\_loop 标签处。

1. .text
2. main:
3. ADDIU $r4, $r0, a    # r4放入a的地址
4. ADDIU $r5, $r0, n    # r5存n的地址
5. LW $r5, 0($r5)       # r5存n的值
6. BEGZAL $r0, bubble   # 跳到bubble排序
7. NOP
8. TEQ $r0, $r0         # 结束
10. bubble:
11. ADDIU $r7, $r5, -1   # r7 先存n-1
12. BLEZ $r7, **return**       # if n<=1 return
13. SLL $r5, $r5, 2      # r5 左移两位, 因为后面要用到长度\*4
14. ADDIU $r8, $r4, 4    # r8 = a[1]
15. ADDU $r6, $r4, $r5   # r6存最后的地址，用来检查是否越界
16. loop:
17. ADDIU $r2, $r8, 0    # r2存a[1]
18. inner\_loop:
19. LW $r3, -4($r2)      # r3 存a[j]
20. LW $r4, 0($r2)       # r4 存a[j+1]
21. SLT $r5, $r4, $r3    # r5 = r3<r4 ? 1 : 0
22. BEQ $r5, $r0, check
23. SW $r4, -4($r2)      # 交换元素
24. SW $r3, 0($r2)       # 交换
25. check:
26. ADDIU $r2, $r2, 4    # 索引++
27. BNE $r6, $r2, inner\_loop # 判断内循环
28. ADDIU $r7, $r7, -1
29. ADDIU $r6, $r6, -4
30. BNE $r7, $r0, loop # 外循环
31. **return**:
32. JR $r31
33. .data
34. a: .word 5,7,10,8,4,2,1,3,6,9
35. n: .word 10

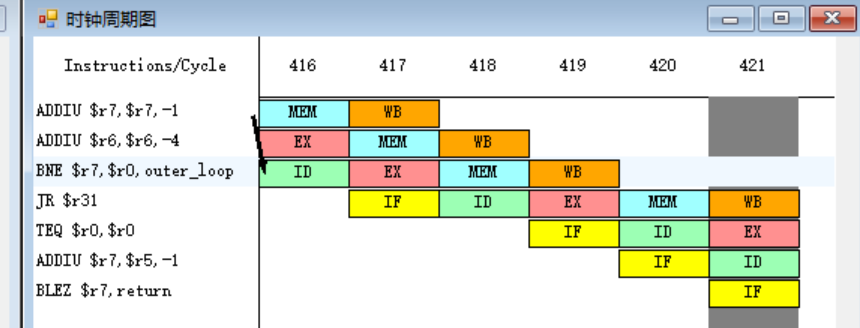
首先不开定向，执行结果如下：

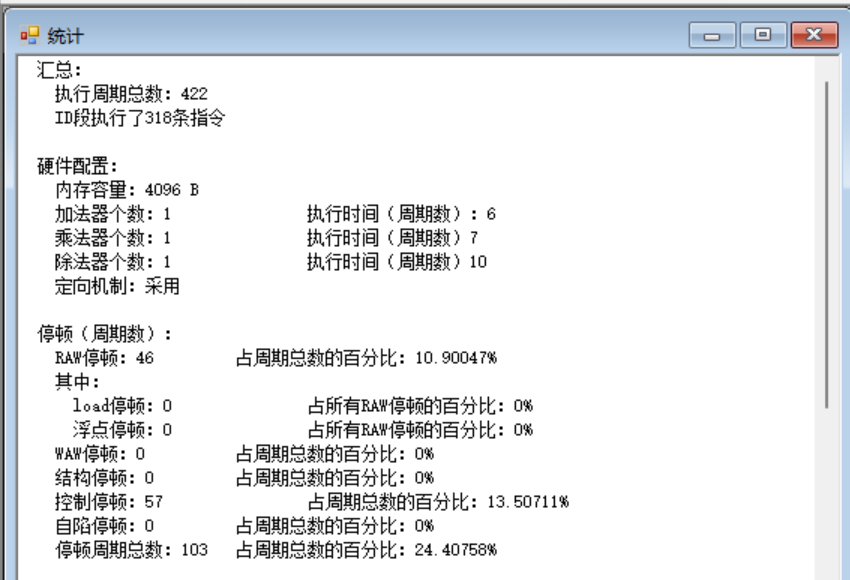


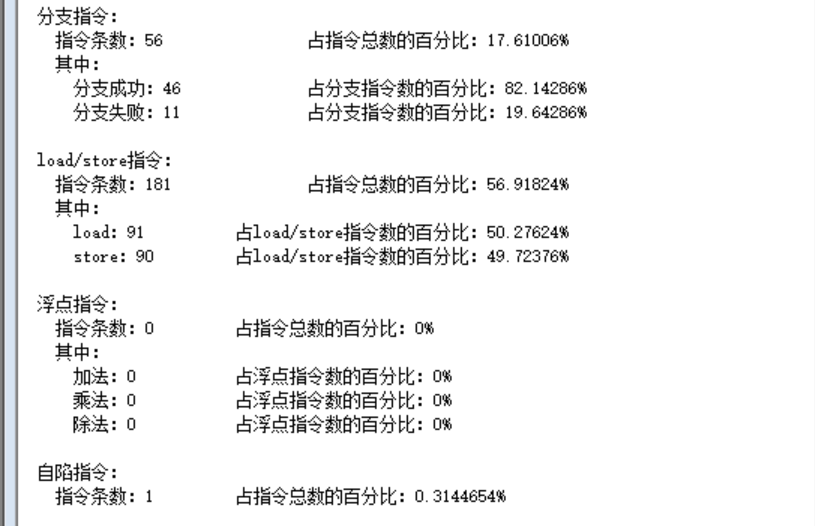


可以看到，总共执行了588个周期，RAW数据冲突停顿了212个，占总周期的36.05442%，控制停顿有57个。总停顿数占总周期数的45.7483%。

下面开启定向进行运行，结果如下：



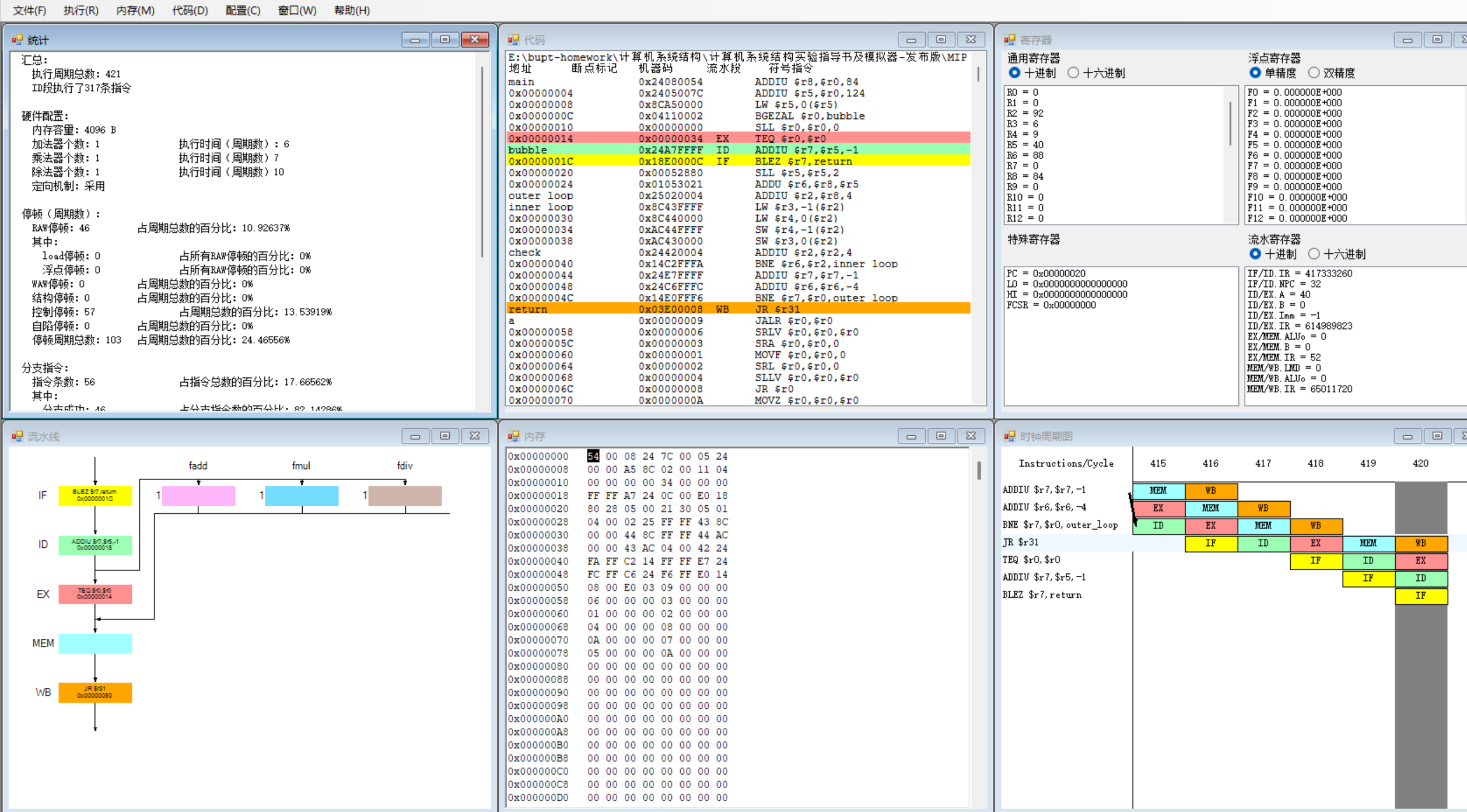


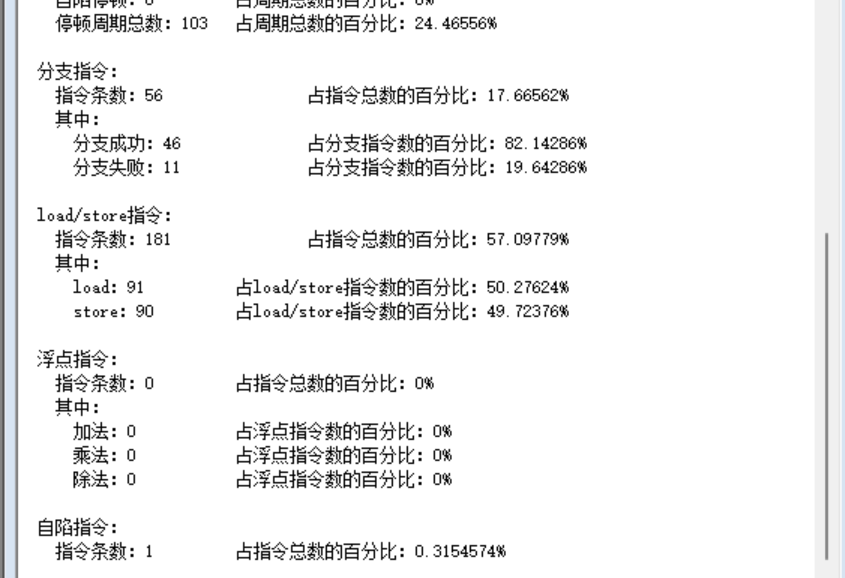


这次，只执行了422个周期，并且RAW停顿下降到了46个，只占总周期的10.90047%。总停顿也只占比24.40758%。

但是由于代码相邻指令的冲突还是太多，所以导致性能不能提升太多。下面我要对这段代码进行静态优化，如下所示：

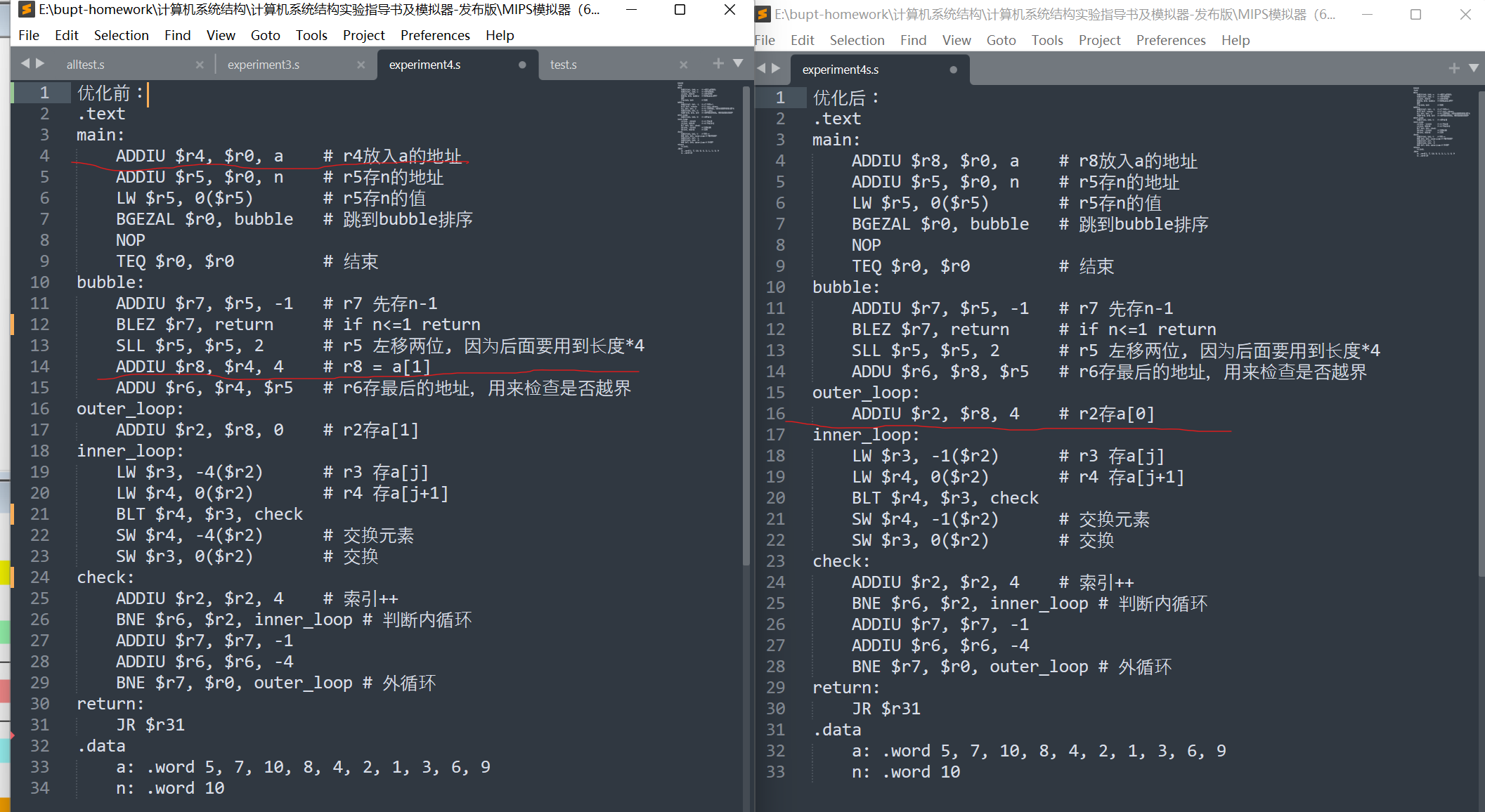
1. .text
2. main:
3. ADDIU $r8, $r0, a    # r8放入a的地址
4. ADDIU $r5, $r0, n    # r5存n的地址
5. LW $r5, 0($r5)       # r5存n的值
6. BGEZAL $r0, bubble   # 跳到bubble排序
7. NOP
8. TEQ $r0, $r0         # 结束
9. bubble:
10. ADDIU $r7, $r5, -1   # r7 先存n-1
11. BLEZ $r7, **return**     # if n<=1 return
12. SLL $r5, $r5, 2      # r5 左移两位, 因为后面要用到长度\*4
13. ADDU $r6, $r8, $r5   # r6存最后的地址，用来检查是否越界
14. outer\_loop:
15. ADDIU $r2, $r8, 4    # r2存a[0]
16. inner\_loop:
17. LW $r3, -1($r2)      # r3 存a[j]
18. LW $r4, 0($r2)       # r4 存a[j+1]
19. BLT $r4, $r3, check
20. SW $r4, -1($r2)      # 交换元素
21. SW $r3, 0($r2)       # 交换
22. check:
23. ADDIU $r2, $r2, 4    # 索引++
24. BNE $r6, $r2, inner\_loop # 判断内循环
25. ADDIU $r7, $r7, -1
26. ADDIU $r6, $r6, -4
27. BNE $r7, $r0, outer\_loop # 外循环
28. **return**:
29. JR $r31
30. .data
31. a: .word 5, 7, 10, 8, 4, 2, 1, 3, 6, 9
32. n: .word 10





可以看到，优化后，执行周期数下降到了421个周期。

优化前和优化后的代码比较如下所示：



如上图所示，左边为优化前的代码，右边为优化后的代码。由于代码每隔几步就要跳转一次，导致能够通过调整顺序优化的地方几乎没有，只能将存放数组a的步骤优化了，原本r4存放了数组a的地址，又放到r8里，优化后直接放到r8里，并且在赋值给r2的时候直接用了r8+4来代表a[1]。

要说明的是，按照C语言源程序来说r2应该存放a[0]的值，但是由于最后判断的时候要判断r6和r2的大小，r2如果存a[0]的话，会导致r4存的元素是越界的元素，并使得运行次数增加，所以要r2才去存a[1]的值。