# 实验3 使用MIPS指令实现求两个数组的点积

陈朴炎 2021211138

## 1 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

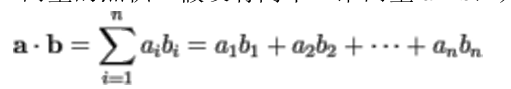
（4）了解对代码进行优化的方法

## 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）自行编写一个计算两个向量点积的汇编程序，该程序要求可以实现求两个向量点 积计算后的结果。 向量的点积：假设有两个n维向量a、b，则a与b的点积为：



两个向量元素使用数组进行数据存储，**要求向量的维度不得小于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

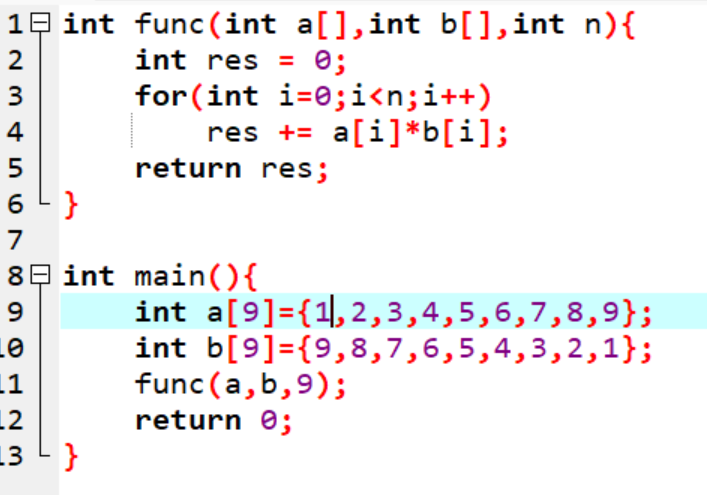
（6）对优化后的程序使用定向功能执行，与刚才执行结果进行比较，观察执行效率的 不同。

注意： 不要使用浮点指令及浮点寄存器！！

使用TEQ $r0 $r0 结束程序！！

## 4 实验内容及步骤

向量点积的函数代码如下所示



1. **int** func(**int** a[],**int** b[],**int** n){
2. **int** res = 0;
3. **for**(**int** i=0;i<n;i++)
4. res += a[i]\*b[i];
5. **return** res;
6. }
8. **int** main(){
9. **int** a[9]={1,2,3,4,5,6,7,8,9};
10. **int** b[9]={9,8,7,6,5,4,3,2,1};
11. func(a,b,9);
12. **return** 0;
13. }

之后开始编写汇编程序。

为a、b、n变量进行赋值：

|  |
| --- |
| .data  a: .word 1, 2, 3, 4, 5, 6, 7, 8, 9 # 数组 a[]  b: .word 9, 8, 7, 6, 5, 4, 3, 2, 1 # 数组 b[]  n: .word 9 # 给 n赋值  res: .word 0 # 给 res初始化 |

在main函数中，要将数组a和数组b的地址和向量维度用三个寄存器存起来。然后就可以调用func函数，最后程序终止。如下

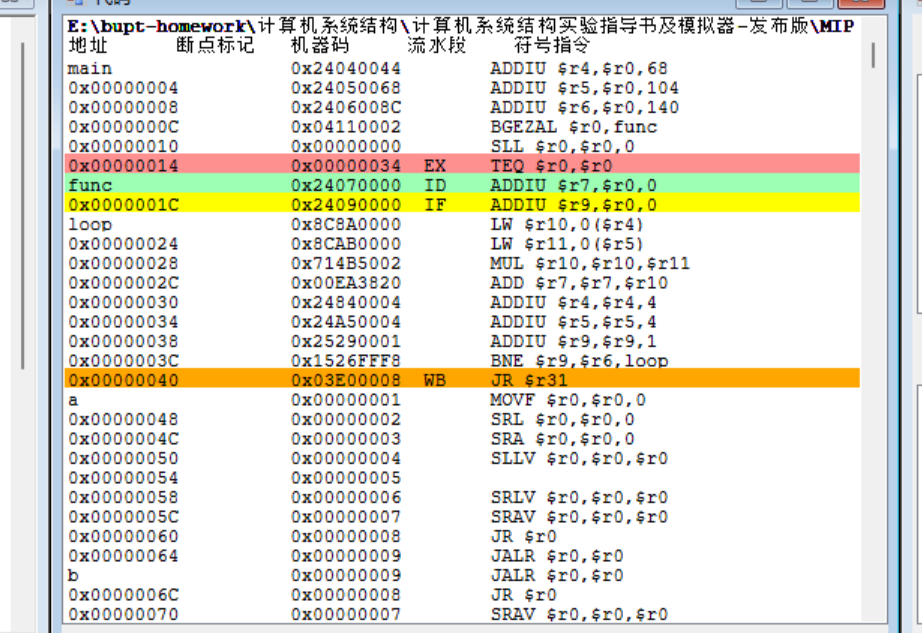
|  |
| --- |
| .text  main:  ADDIU $r4, $r0, a # $r4 = &a[0]  ADDIU $r5, $r0, b # $r5 = &b[0]  ADDIU $r6, $r0, n # $r6 = n  BGEZAL $r0, func # 函数调用  NOP  TEQ $r0, $r0 |

接着要调用func函数，func函数内部如下：

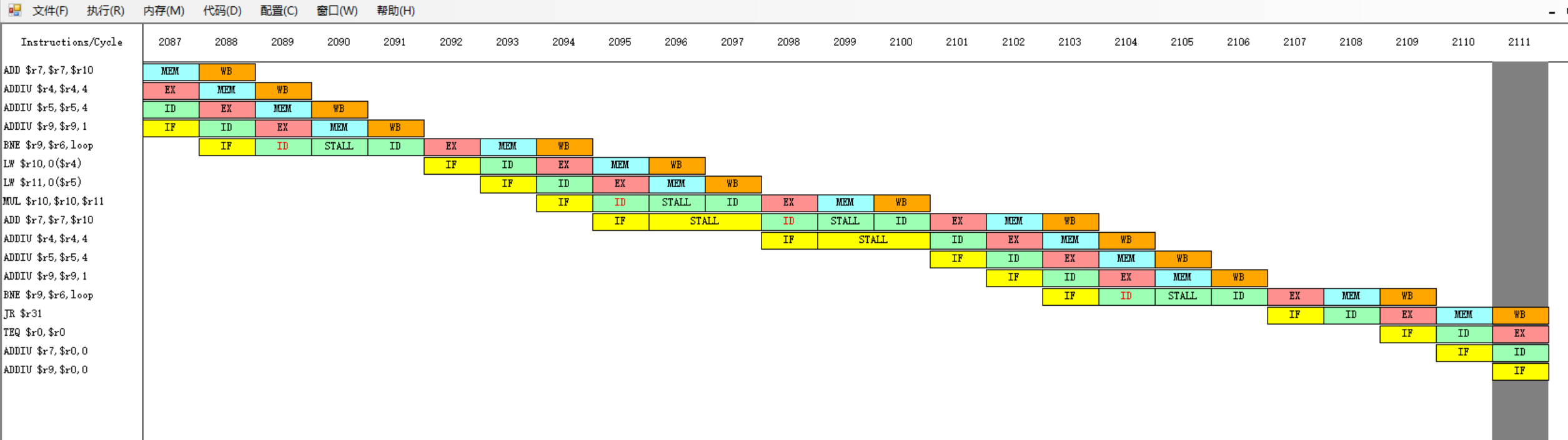
|  |
| --- |
| func:  # initial res  ADDIU $r7, $r0, 0 # $r7 = res  ADDIU $r9, $r0, 0 # 循环的下标  loop:  LW $r10, 0($r4) # $r10 = a[i]  LW $r11, 0($r5) # $r11 = b[i]  MUL $r10, $r10, $r11 # $r10 = a[i] \* b[i]  ADD $r7, $r7, $r10 # res += a[i] \* b[i]  ADDIU $r4, $r4, 4 # r4地址指向下一个  ADDIU $r5, $r5, 4 # r5指向下一个地址  ADDIU $r9, $r9, 1 # 索引自增  BNE $r9, $r6, loop  JR $r31 # 返回值 |

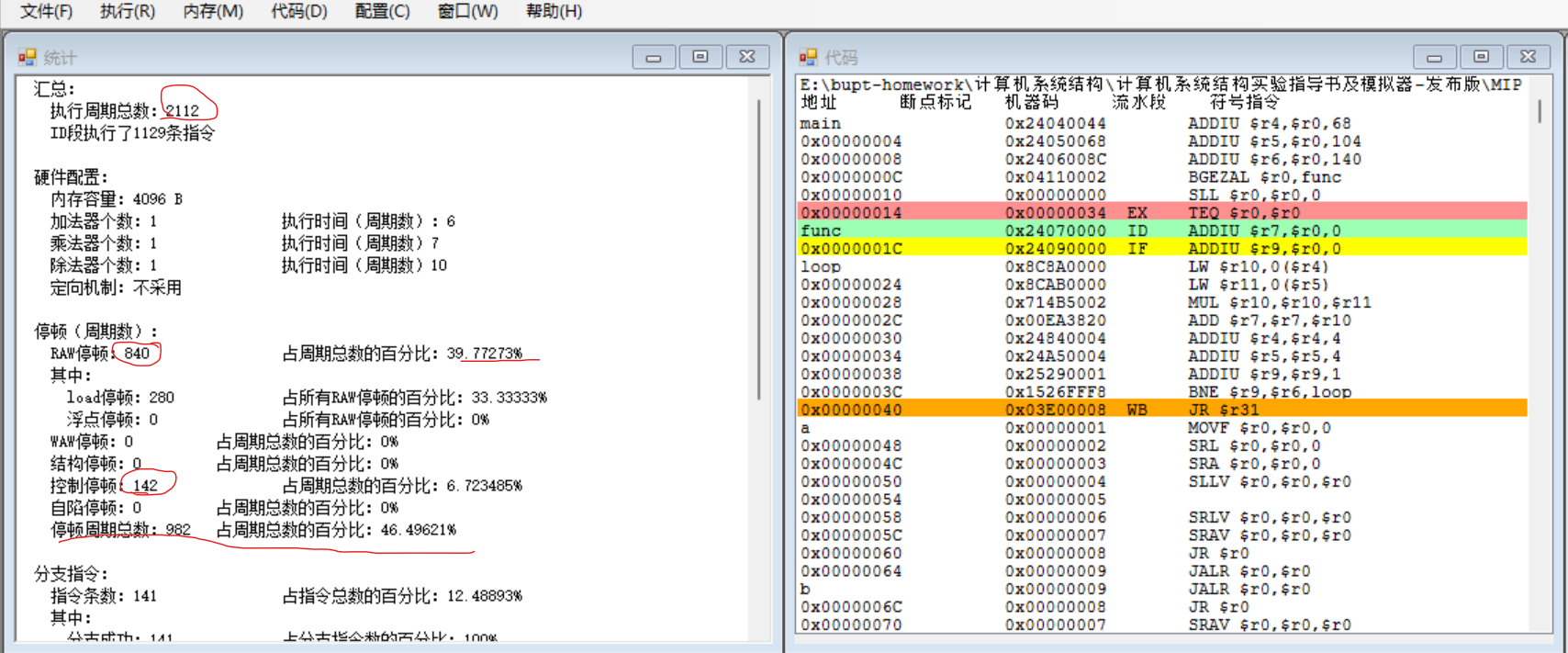
1. .text
2. main:
3. ADDIU $r4, $r0, a    # $r4 = &a[0]
4. ADDIU $r5, $r0, b    # $r5 = &b[0]
5. ADDIU $r6, $r0, n    # $r6 = n
6. BGEZAL $r0, func     # 函数调用
7. NOP
8. TEQ $r0, $r0
10. func:
11. ADDIU $r7, $r0, 0    # $r7 = res
12. ADDIU $r9, $r0, 0    # 循环的下标
13. loop:
14. LW $r10, 0($r4)         # $r10 = a[i]
15. LW $r11, 0($r5)         # $r11 = b[i]
16. MUL $r10, $r10, $r11    # $r10 = a[i] \* b[i]
17. ADD $r7, $r7, $r10      # res += a[i] \* b[i]
18. ADDIU $r4, $r4, 4       # r4地址指向下一个
19. ADDIU $r5, $r5, 4       # r5指向下一个地址
20. ADDIU $r9, $r9, 1       # 索引自增
21. BNE $r9, $r6, loop
22. JR $r31                 # 返回值
24. .data
25. a:.word 1, 2, 3, 4, 5, 6, 7, 8, 9   # 数组 a[]
26. b: .word 9, 8, 7, 6, 5, 4, 3, 2, 1   # 数组 b[]
27. n: .word 9                           # 给 n赋值
28. res: .word 0                         # 给 res初始化

写好程序，之后导入。如下所示

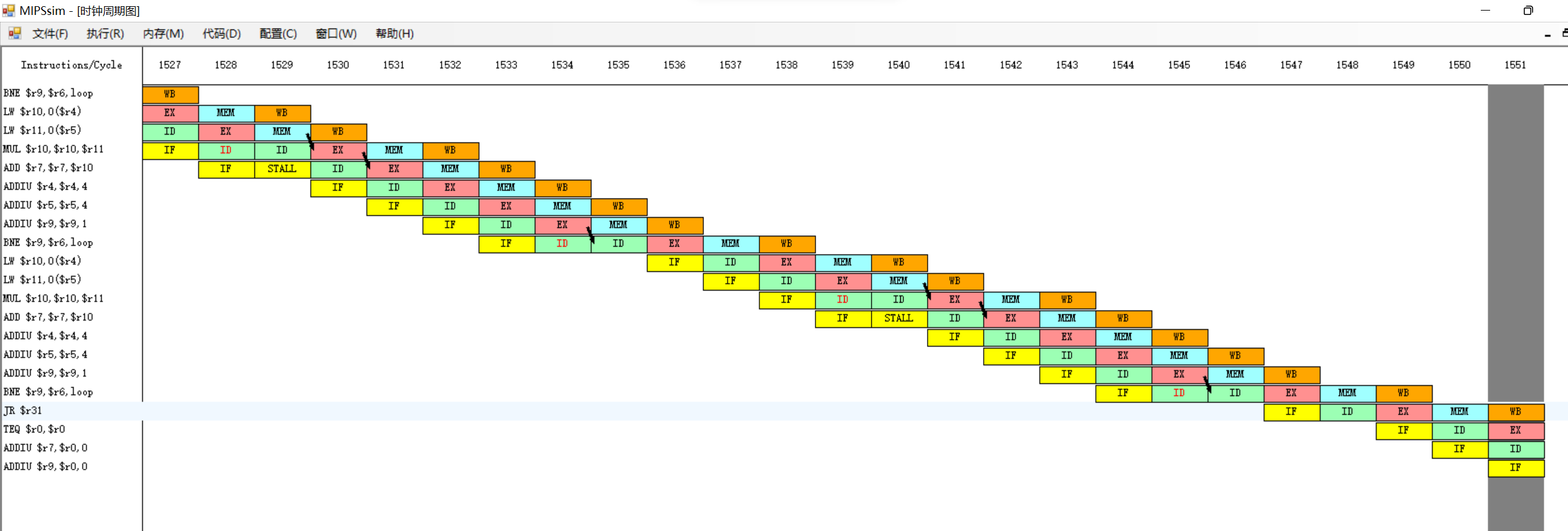


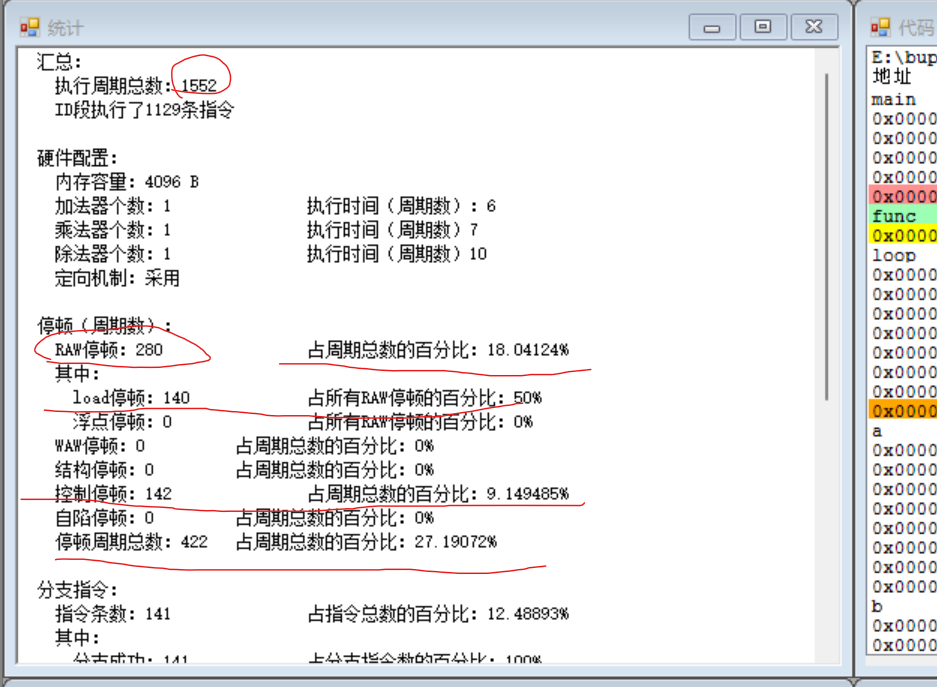
第一次，不开启定向功能，观察流水线输出结果，如下：





使用定向功能再次执行代码，与刚才执行结果相比，如下

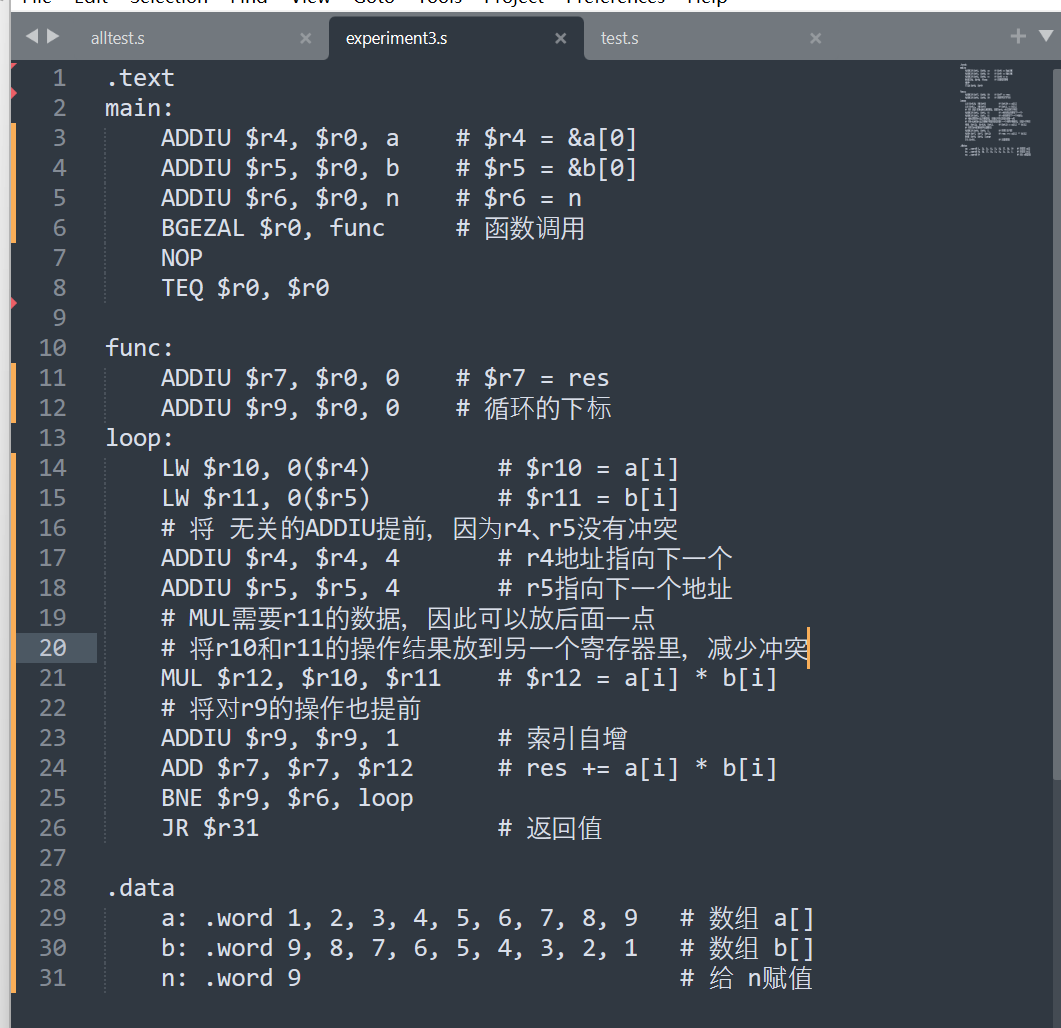




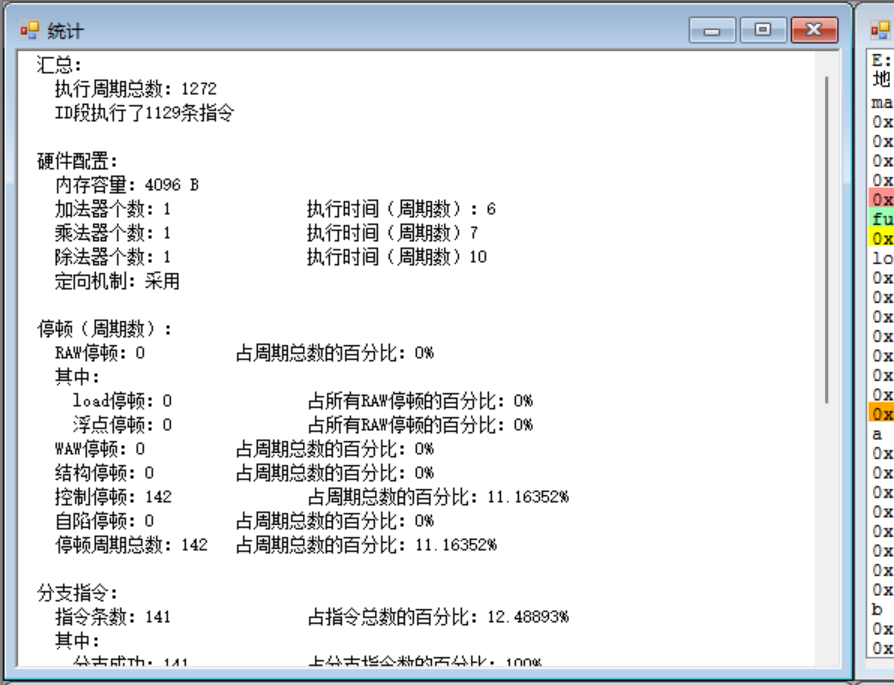
如上所示，执行周期从2112下降到了1552，停顿占比从39.7723%下降到了18.04142%。说明采用了定向技术将会使性能大大提升。

下面采用静态调度方法重排指令序列，减少相关，优化程序，如下：

1. .text
2. main:
3. ADDIU $r4, $r0, a    # $r4 = &a[0]
4. ADDIU $r5, $r0, b    # $r5 = &b[0]
5. ADDIU $r6, $r0, n    # $r6 = n
6. BGEZAL $r0, func     # 函数调用
7. NOP
8. TEQ $r0, $r0
10. func:
11. ADDIU $r7, $r0, 0    # $r7 = res
12. ADDIU $r9, $r0, 0    # 循环的下标
13. loop:
14. LW $r10, 0($r4)         # $r10 = a[i]
15. LW $r11, 0($r5)         # $r11 = b[i]
16. # 将 无关的ADDIU提前，因为r4、r5没有冲突
17. ADDIU $r4, $r4, 4       # r4地址指向下一个
18. ADDIU $r5, $r5, 4       # r5指向下一个地址
19. # MUL需要r11的数据，因此可以放后面一点
20. # 将r10和r11的操作结果放到另一个寄存器里，减少冲突
21. MUL $r12, $r10, $r11    # $r12 = a[i] \* b[i]
22. # 将对r9的操作也提前
23. ADDIU $r9, $r9, 1       # 索引自增
24. ADD $r7, $r7, $r12      # res += a[i] \* b[i]
25. BNE $r9, $r6, loop
26. JR $r31                 # 返回值
28. .data
29. a: .word 1, 2, 3, 4, 5, 6, 7, 8, 9   # 数组 a[]
30. b: .word 9, 8, 7, 6, 5, 4, 3, 2, 1   # 数组 b[]
31. n: .word 9                           # 给 n赋值



执行结果如下：



可以看到，已经没有RAW冲突，只有控制停顿，并且执行周期又下降到了1272个，性能大大提升。停顿周期总数也只有142，占比11.16352%。