

《计算机系统结构》

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）

班级： 2018211314

姓名： 李志毅

学号： 2018211582

实验四 使用 MIPS 指令实现冒泡排序法

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

二、实验环境

指令级和流水线操作级模拟器 MIPSsim

三、实验原理

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。

采用静态调度方法重排指令的顺序，从而使得 RAW 等冲突所导致的空操作得以利用，减少因为 RAW 等冲突而占用的无用周期，可以通过将不相关指令前移等方式来进行静态调度优化。

四、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 7,34,8,22,28,49,25,44,35,14,11,4,42,3,2

```

五、优化后的代码清单

```
.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 7,34,8,22,28,49,25,44,35,14,11,4,42,3,2

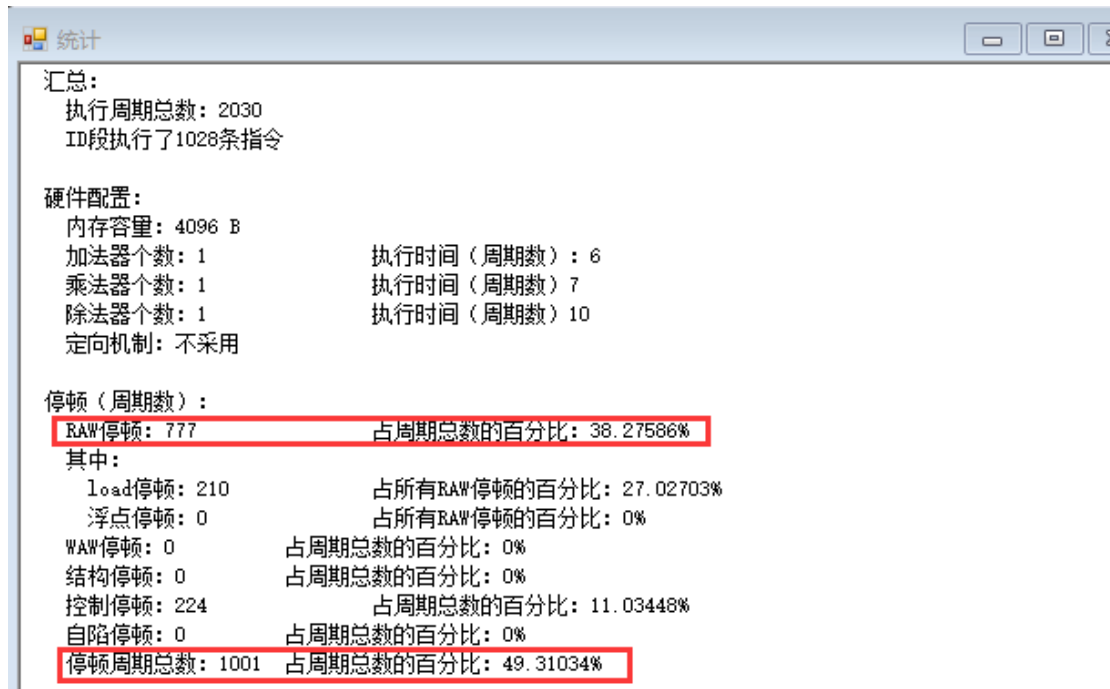
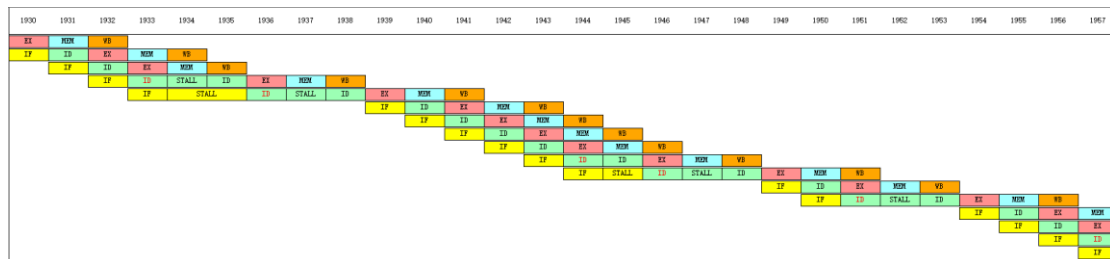
六、实验步骤

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。

要求数组长度不得小于 10

源代码见附录

2. 启动 MIPSsim
3. 载入自己编写的程序，观察流水线输出结果
流水线的部分执行情况：



分支指令：
 指令条数：224 占指令总数的百分比：21.78988%
 其中：
 分支成功：145 占分支指令数的百分比：64.73214%
 分支失败：79 占分支指令数的百分比：35.26786%

load/store指令：
 指令条数：338 占指令总数的百分比：32.87938%
 其中：
 load: 210 占load/store指令数的百分比：62.13018%
 store: 128 占load/store指令数的百分比：37.86982%

浮点指令：
 指令条数：0 占指令总数的百分比：0%
 其中：
 加法：0 占浮点指令数的百分比：0%
 乘法：0 占浮点指令数的百分比：0%
 除法：0 占浮点指令数的百分比：0%

自陷指令：
 指令条数：1 占指令总数的百分比：0.09727626%

程序计算的是 7,34,8,22,28,49,25,44,35,14,11,4,42,3,2 的冒泡排序结果

程序执行 2030 个周期，其中 RAW 冲突占用 777 个周期，占比 38.27586%

程序执行前未排序的数组：

内存	
0x00000000	0F 00 01 24 0E 00 02 24
0x00000008	44 00 03 24 00 00 04 24
0x00000010	00 00 65 8C 04 00 66 8C
0x00000018	2E 38 A6 00 02 00 E0 04
0x00000020	00 00 66 AC 04 00 65 AC
0x00000028	01 00 84 24 04 00 63 24
0x00000030	2E 40 44 00 F6 FF 00 1D
0x00000038	FF FF 42 24 F2 FF 40 1C
0x00000040	34 00 00 00 07 00 00 00
0x00000048	22 00 00 00 08 00 00 00
0x00000050	16 00 00 00 1C 00 00 00
0x00000058	31 00 00 00 19 00 00 00
0x00000060	2C 00 00 00 23 00 00 00
0x00000068	0E 00 00 00 0B 00 00 00
0x00000070	04 00 00 00 2A 00 00 00
0x00000078	03 00 00 00 02 00 00 00
0x00000080	00 00 00 00 00 00 00 00
0x00000088	00 00 00 00 00 00 00 00
0x00000090	00 00 00 00 00 00 00 00

可以看到 7,34,8,22,28,49,25,44,35,14,11,4,42,3,2 依次存储在内存中
 程序执行后排序后的数组：

内存	
0x00000000	0F 00 01 24 0E 00 02 24
0x00000008	44 00 03 24 00 00 04 24
0x00000010	00 00 65 8C 04 00 66 8C
0x00000018	2E 38 A6 00 02 00 E0 04
0x00000020	00 00 66 AC 04 00 65 AC
0x00000028	01 00 84 24 04 00 63 24
0x00000030	2E 40 44 00 F6 FF 00 1D
0x00000038	FF FF 42 24 F2 FF 40 1C
0x00000040	34 00 00 00 02 00 00 00
0x00000048	03 00 00 00 04 00 00 00
0x00000050	07 00 00 00 08 00 00 00
0x00000058	0B 00 00 00 0E 00 00 00
0x00000060	16 00 00 00 19 00 00 00
0x00000068	1C 00 00 00 22 00 00 00
0x00000070	23 00 00 00 2A 00 00 00
0x00000078	2C 00 00 00 31 00 00 00

排序后的数组为 2,3,4,7,8,11,14,22,25,28,34,35,42,44,49

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。

统计	
汇总:	
执行周期总数: 1454	
ID段执行了900条指令	
硬件配置:	
内存容量: 4096 B	
加法器个数: 1	执行时间 (周期数): 6
乘法器个数: 1	执行时间 (周期数): 7
除法器个数: 1	执行时间 (周期数): 10
定向机制: 采用	
停顿 (周期数):	
RAW停顿: 329	占周期总数的百分比: 22.62724%
其中:	
load停顿: 105	占所有RAW停顿的百分比: 31.91489%
浮点停顿: 0	占所有RAW停顿的百分比: 0%
WAW停顿: 0	占周期总数的百分比: 0%
结构停顿: 0	占周期总数的百分比: 0%
控制停顿: 224	占周期总数的百分比: 15.40578%
自陷停顿: 0	占周期总数的百分比: 0%
停顿周期总数: 553	占周期总数的百分比: 38.03301%

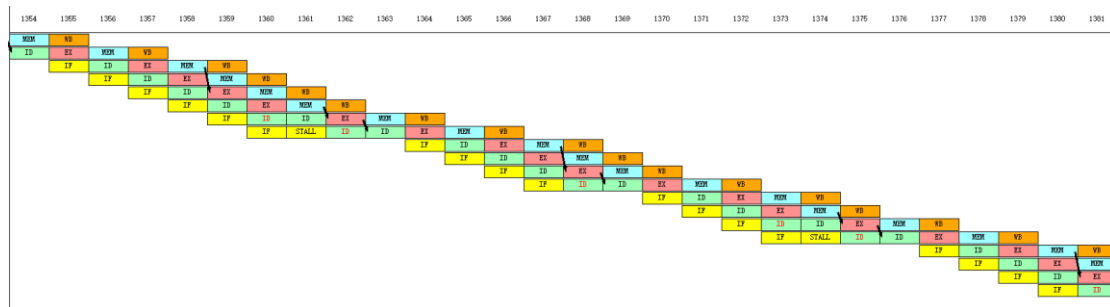
分支指令：
 指令条数：224 占指令总数的百分比：24.88889%
 其中：
 分支成功：209 占分支指令数的百分比：93.30357%
 分支失败：15 占分支指令数的百分比：6.696429%

load/store指令：
 指令条数：210 占指令总数的百分比：23.33333%
 其中：
 load: 210 占load/store指令数的百分比：100%
 store: 0 占load/store指令数的百分比：0%

浮点指令：
 指令条数：0 占指令总数的百分比：0%
 其中：
 加法：0 占浮点指令数的百分比：0%
 乘法：0 占浮点指令数的百分比：0%
 除法：0 占浮点指令数的百分比：0%

自陷指令：
 指令条数：1 占指令总数的百分比：0.1111111%

可以看到，采用定向技术后，执行总周期数变为 **1454**，其中 RAW 停顿
329 个周期，占比为 **22.62724%**，执行的效率变为原来的 $2030/1454=1.39$ 倍



5. 采用静态调度方法重排指令序列，减少相关，优化程序

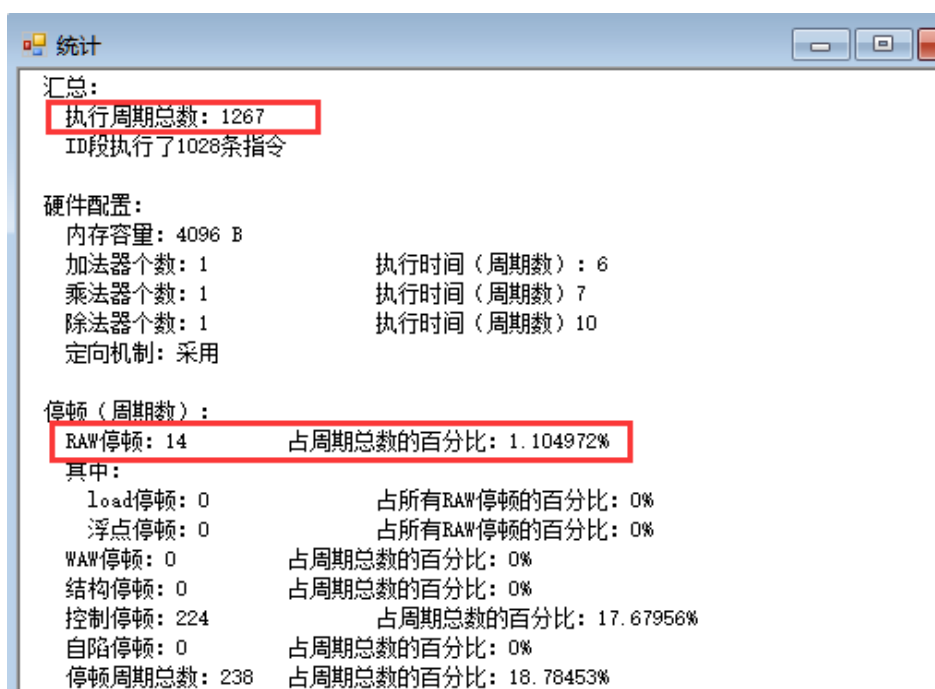
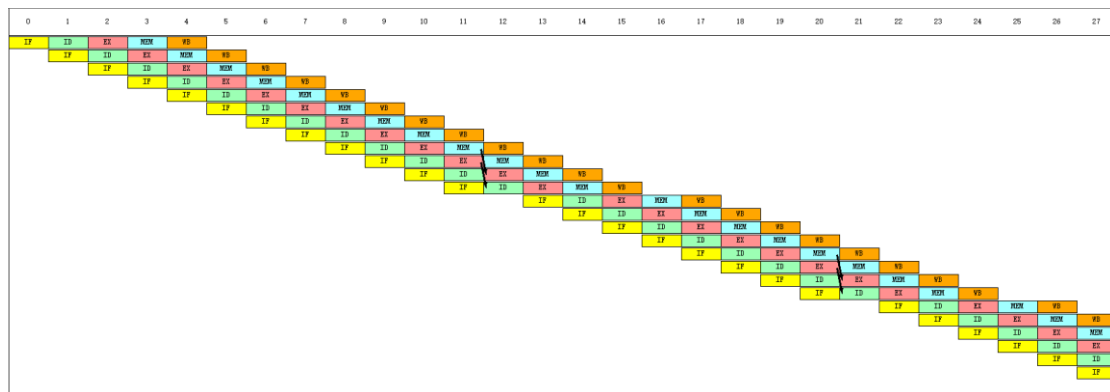
main	0x2401000F	ADDIU \$r1,\$r0,15
0x00000004	0x2402000E	ADDIU \$r2,\$r0,14
LOOP1	0x24030044	ADDIU \$r3,\$r0,68
0x0000000C	0x24040000	ADDIU \$r4,\$r0,0
LOOP2	0x8C650000	LW \$r5,0(\$r3)
0x00000014	0x8C660004	LW \$r6,4(\$r3)
0x00000018	0x00A6382E	DSUB \$r7,\$r5,\$r6
0x0000001C	0x04E00002	BLTZ \$r7,bk
0x00000020	0xAC660000	SW \$r6,0(\$r3)
0x00000024	0xAC650004	SW \$r5,4(\$r3)
bk	0x24840001	ADDIU \$r4,\$r4,1
0x0000002C	0x24630004	ADDIU \$r3,\$r3,4
0x00000030	0x0044402E	DSUB \$r8,\$r2,\$r4
0x00000034	0x1D00FFF6	BGTZ \$r8,LOOP2
0x00000038	0x2442FFFF	ADDIU \$r2,\$r2,-1
0x0000003C	0x1C40FFF2	WB BGTZ \$r2,LOOP1
0x00000040	0x00000034	EX TEQ \$r0,\$r0
array	0x00000002	ID SRL \$r0,\$r0,0
0x00000048	0x00000003	IF SRA \$r0,\$r0,0
0x0000004C	0x00000004	SLLV \$r0,\$r0,\$r0
0x00000050	0x00000007	SRAV \$r0,\$r0,\$r0

可以看到，LW \$r5, 0(\$r3)指令和 LW \$r6, 4(\$r3)指令和 DSUB \$r7, \$r5, \$r6 和

BLTZ \$r7, bk 存在数据冲突, DSUB \$r8, \$r2, \$r4 和 BGTZ \$r8, LOOP2 存在数据冲突, 将无关指令穿插中再其中, 减少 RAW 相关

LOOP2	0x8C650000	LW \$r5,0(\$r3)
0x00000014	0x8C660004	LW \$r6,4(\$r3)
0x00000018	0x24840001	ADDIU \$r4,\$r4,1
0x0000001C	0x00A6382E	DSUB \$r7,\$r5,\$r6
0x00000020	0x0044402E	DSUB \$r8,\$r2,\$r4
0x00000024	0x04E00002	BLTZ \$r7,bk
0x00000028	0xAC660000	SW \$r6,0(\$r3)
0x0000002C	0xAC650004	SW \$r5,4(\$r3)
bk	0x24630004	ADDIU \$r3,\$r3,4
0x00000034	0x1D00FFF6	BGTZ \$r8,LOOP2
0x00000038	0x2442FFFF	ADDIU \$r2,\$r2,-1
0x0000003C	0x1C40FFF2	BGTZ \$r2,LOOP1
0x00000040	0x00000034	TEQ \$r0,\$r0
array	0x00000007	SRAV \$r0,\$r0,\$r0
0x00000048	0x00000022	SUB \$r0,\$r0,\$r0
0x0000004C	0x00000000	

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。



可以看到采用静态调度的方法优化程序，并使用定向功能后，执行总周期数为 **1267**，RAW 停顿为 **14**，停顿数明显减少，效率变为原来的 $1454/1267=1.1476$ 倍。执行结果也正确。

```
0x00000040  34 00 00 00 02 00 00 00
0x00000048  03 00 00 00 04 00 00 00
0x00000050  07 00 00 00 08 00 00 00
0x00000058  0B 00 00 00 0E 00 00 00
0x00000060  16 00 00 00 19 00 00 00
0x00000068  1C 00 00 00 22 00 00 00
0x00000070  23 00 00 00 2A 00 00 00
0x00000078  2C 00 00 00 31 00 00 00
-----
```

七、实验问题与心得

本次实验使用指令级和流水线操作级模拟器 MIPSsim 分析了冒泡排序程序优化的过程，加深了我对于计算机流水线基本概念的理解，理解了 MIPS 结构是如何使用 5 段流水线来实现的，理解了各段的功能和基本操作，加深了我对数据冲突和结构冲突的理解，以及采用定向技术解决数据冲突带来的好处和性能的提升，进一步掌握了解决数据冲突的方法，掌握了如何应用定向技术来减少数据冲突引起的停顿，增强汇编语言编程能力，了解了对代码进行优化的方法。