实验4 使用 MIPS 指令实现冒泡排序法

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. 实验原理

（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。

注意：

1. 不要使用浮点指令及浮点寄存器！！
2. 整数减勿使用 `SUB` 指令，请使用 `DSUB` 指令代替！！

4. 程序代码及注释

```
.data
arr:
.word 10,9,8,7,6,5,4,3,2,1
len:
.word 10

.text
main:
ADDIU $r1, $r0, arr # 获取数组首地址
ADDIU $r2, $r0, len # 获取len的地址
LW $r2, 0($r2) # 获取数组长度
SLL $r2, $r2, 2 # len<<2
ADD $r2, $r2, $r1 # arr[len]的地址
outer_loop:
ADDI $r2, $r2, -4 # len--
```

```
BEQ $r1, $r2, exit # 如果arr[i] == arr[len]，则结束
ADDIU $r3, $r1, 0 # 初始 k = 0，获取arr[k]的地址
inner_loop:
LW $r4, 0($r3) # 获取arr[k]的值
LW $r5, 4($r3) # 获取arr[k+1]的值
SLT $r6, $r5, $r4 # arr[k+1] < arr[k] ?
BEQ $r6, $r0, skip # 如果arr[k+1] >= arr[k]，则跳过交换
SW $r5, 0($r3) # 存arr[k+1]到arr[k]
SW $r4, 4($r3) # 存arr[k]到arr[k+1]
skip:
ADDI $r3, $r3, 4 # k++
BNE $r3, $r2, inner_loop # 如果k != len，继续循环
BEQ $r0, $r0, outer_loop # 继续循环
exit:
TEQ $r0, $r0 # 结束
```

运行结果

汇总：
执行周期总数：741
ID段执行了315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：不采用

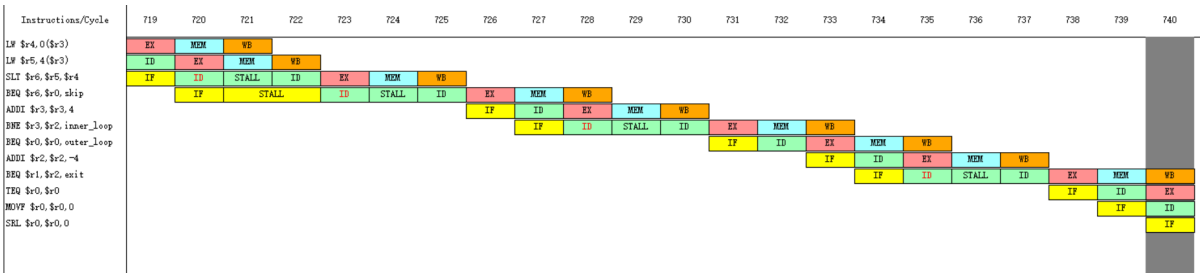
停顿（周期数）：
RAW停顿：316　　占周期总数的百分比：42.64507%
其中：
load停顿：92　　占所有RAW停顿的百分比：29.11392%
浮点停顿：0　　占所有RAW停顿的百分比：0%
WAW停顿：0　　占周期总数的百分比：0%
结构停顿：0　　占周期总数的百分比：0%
控制停顿：109　　占周期总数的百分比：14.70985%
自陷停顿：0　　占周期总数的百分比：0%
停顿周期总数：425　占周期总数的百分比：57.35493%

分支指令：
指令条数：109　　占指令总数的百分比：34.60318%
其中：
分支成功：91　　占分支指令数的百分比：83.48624%
分支失败：18　　占分支指令数的百分比：16.51376%

load/store指令：
指令条数：91　　占指令总数的百分比：28.88889%
其中：
load：91　　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：0.3174603%



其中，有相当一部分指令，发生了冲突，冲突类型是`RAW`

5. 优化后的程序代码

开启定向

汇总：
  执行周期总数：571
  ID段执行了315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

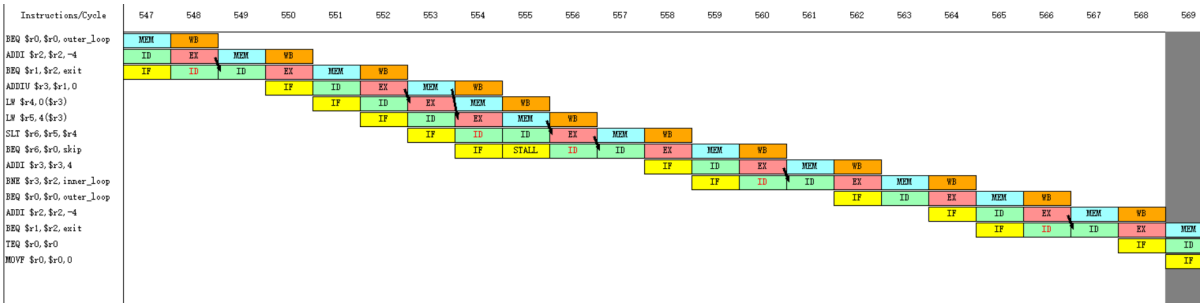
停顿（周期数）：
  RAW停顿：146　　占周期总数的百分比：25.56918%
  其中：
    load停顿：46　　占所有RAW停顿的百分比：31.50685%
    浮点停顿：0　　占所有RAW停顿的百分比：0%
  WAW停顿：0　　占周期总数的百分比：0%
  结构停顿：0　　占周期总数的百分比：0%
  控制停顿：109　　占周期总数的百分比：19.08932%
  自陷停顿：0　　占周期总数的百分比：0%
  停顿周期总数：255　占周期总数的百分比：44.65849%

分支指令：
  指令条数：109　　占指令总数的百分比：34.60318%
  其中：
    分支成功：91　　占分支指令数的百分比：83.48624%
    分支失败：18　　占分支指令数的百分比：16.51376%

load/store指令：
指令条数：91　　占指令总数的百分比：28.88889%
其中：
load：91　　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：0.3174603%



可以发现，定向功能，能够消除部分数据冲突。性能提升了 $\frac{741}{571} \approx 1.30$ 倍

使用静态调度，重排

```
.data
arr:
.word 10,9,8,7,6,5,4,3,2,1
len:
.word 10

.text
main:
ADDIU $r1, $r0, arr # 获取数组首地址
ADDIU $r2, $r0, len # 获取len的地址
LW $r2, 0($r2) # 获取数组长度
SLL $r2, $r2, 2 # len<<2
ADD $r2, $r2, $r1 # arr[len]的地址
outer_loop:
ADDI $r2, $r2, -4 # len--
ADDIU $r3, $r1, 0 # 初始 k = 0, 获取arr[k]的地址
BEQ $r1, $r2, exit # 如果arr[i] == arr[len]，则结束
inner_loop:
LW $r4, 0($r3) # 获取arr[k]的值
ADDI $r3, $r3, 4 # k++
LW $r5, 0($r3) # 获取arr[k+1]的值
SLT $r6, $r5, $r4 # arr[k+1] < arr[k] ?
BEQ $r6, $r0, skip # 如果arr[k+1] >= arr[k]，则跳过交换
```

```
SW $r5, -4($r3) # 存arr[k+1]到arr[k]
SW $r4, 0($r3) # 存arr[k]到arr[k+1]
skip:
BNE $r3, $r2, inner_loop # 如果k != len，继续循环
BEQ $r0, $r0, outer_loop # 继续循环
exit:
TEQ $r0, $r0 # 结束
```

主要是将 `LW $r5, 4($r3)` 指令移动到 `ADDI $r3, $r3, 4` 指令之后，这样可以在读取 `arr[k+1]` 的值之前先更新 `k`，从而减少数据相关性。

汇总：
执行周期总数：517
ID段执行了316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

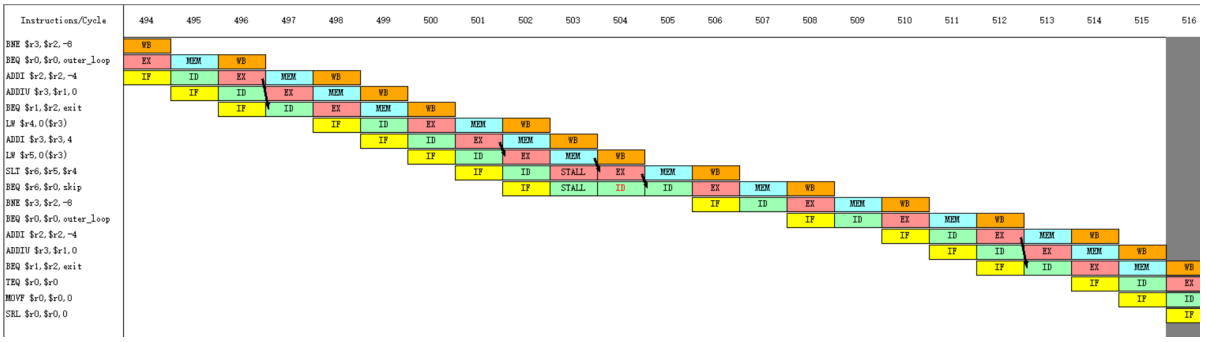
停顿（周期数）：
RAW停顿：91　　占周期总数的百分比：17.60155%
其中：
load停顿：46　　占所有RAW停顿的百分比：50.54945%
浮点停顿：0　　占所有RAW停顿的百分比：0%
WAW停顿：0　　占周期总数的百分比：0%
结构停顿：0　　占周期总数的百分比：0%
控制停顿：109　　占周期总数的百分比：21.08317%
自陷停顿：0　　占周期总数的百分比：0%
停顿周期总数：200　占周期总数的百分比：38.68472%

分支指令：
指令条数：109　　占指令总数的百分比：34.49367%
其中：
分支成功：91　　占分支指令数的百分比：83.48624%
分支失败：18　　占分支指令数的百分比：16.51376%

load/store指令：
指令条数：91　　占指令总数的百分比：28.79747%
其中：
load：91　　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：0.3164557%



性能提升了 $\frac{571}{517} \approx 1.1$ 倍