实验 4：使用 MIPS 指令实现冒泡排序法

于海鑫

2017211240

版本：ℵ

更新：*2020* 年 *4* 月 *16* 日

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 实验内容和步骤

(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。
(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 冒泡排序

4.1 代码

冒泡排序几乎是刻在大家 DNA 里面的程序了吧，在此就不再展示其 C 代码了，我们的程序的签名如下：

```
void bubble(int *arr, int n);
```

我们要做的就是把 DNA 里面的冒泡排序代码转换成汇编的，结果如下（需要注意 MIPS 的调用约定）

```
.text
main:
ADDIU  $r4,$r0,a
ADDIU  $r5,$r0,n
LW     $r5, 0($r5)
BGEZAL  $r0, bubble
NOP
TEQ $r0,$r0

bubble:
ADDIU  $r7, $r5, -1
BLEZ   $r7, exit
SLL    $r5, $r5, 2
ADDIU  $r8, $r4, 4
ADDU   $r6, $r4, $r5
loop:
ADDIU  $r2, $r8, 0
run:
LW     $r3, -4($r2)
LW     $r4, 0($r2)
SLT    $r5, $r4, $r3
BEQ    $r5, $r0, end
swap:
SW     $r4, -4($r2)
SW     $r3, 0($r2)
end:
ADDIU  $r2, $r2, 4
```

```
BNE    $r6, $r2, run

ADDIU  $r7, $r7, -1
ADDIU  $r6, $r6, -4
BNE    $r7, $r0, loop

exit:
JR     $r31

.data
a:
.word 11,10,9,8,7,6,5,4,3,2,1
n:
.word 11
```

4.2 运行结果

4.2.1 未开启定向功能时

```
汇总：
  执行周期总数：981
  ID段执行了492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：365         占周期总数的百分比：37.20693%
  其中：
    load停顿：110      占所有RAW停顿的百分比：30.13699%
    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：123        占周期总数的百分比：12.53823%
```

```
    自陷停顿：0       占周期总数的百分比：0%
    停顿周期总数：488     占周期总数的百分比：49.74516%

  分支指令：
    指令条数：122         占指令总数的百分比：24.79675%
    其中：
      分支成功：56        占分支指令数的百分比：45.90164%
      分支失败：67        占分支指令数的百分比：54.91803%

  load/store指令：
    指令条数：221         占指令总数的百分比：44.9187%
    其中：
      load：111       占load/store指令数的百分比：50.22625%
      store：110          占load/store指令数的百分比：49.77375%

  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：0.203252%
```

时钟周期图如下：

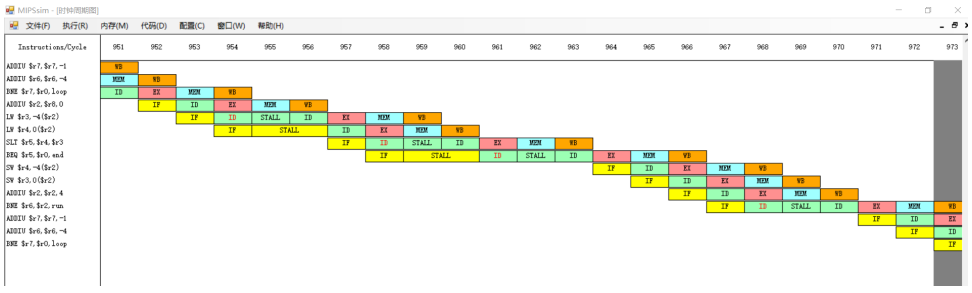

图 1: 时钟周期图

4.2.2 开启定向功能后

```
    汇总：
    执行周期总数：782
```

```
  ID段执行了492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：166         占周期总数的百分比：21.22762%
  其中：
    load停顿：55       占所有RAW停顿的百分比：33.13253%
    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：123        占周期总数的百分比：15.7289%
  自陷停顿：0      占周期总数的百分比：0%
  停顿周期总数：289    占周期总数的百分比：36.95652%

分支指令：
  指令条数：122        占指令总数的百分比：24.79675%
  其中：
    分支成功：56       占分支指令数的百分比：45.90164%
    分支失败：67       占分支指令数的百分比：54.91803%

load/store指令：
  指令条数：221        占指令总数的百分比：44.9187%
  其中：
    load：111      占load/store指令数的百分比：50.22625%
    store：110         占load/store指令数的百分比：49.77375%

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：0.203252%
```

时钟周期图如下：

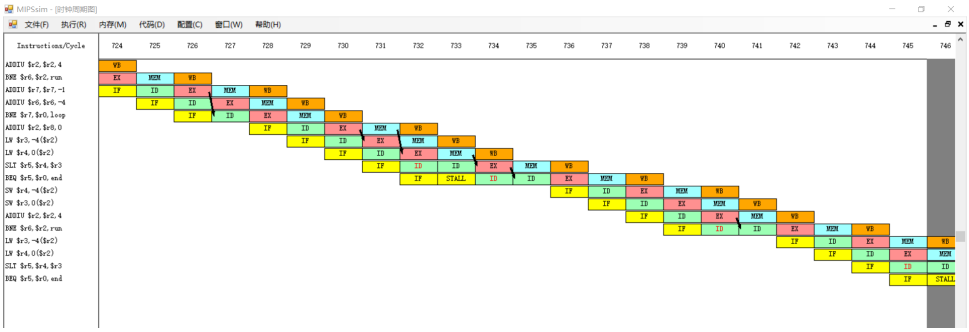

图 2: 时钟周期图

5 优化后的冒泡排序

5.1 代码

很遗憾的是，在这里我们的代码可以优化的地方寥寥无几，能修改的只有删掉之前符合语义的 `SLT` 指令。修改后的代码如下：

```
.text
main:
ADDIU   $r4,$r0,a
ADDIU   $r5,$r0,n
LW      $r5, 0($r5)
BGEZAL   $r0, bubble
NOP
TEQ $r0,$r0

bubble:
ADDIU   $r7, $r5, -1
BLEZ    $r7, exit
SLL     $r5, $r5, 2
ADDIU   $r8, $r4, 4
ADDU    $r6, $r4, $r5
loop:
ADDIU   $r2, $r8, 0
```

```
run:
LW      $r3, -4($r2)
LW      $r4, 0($r2)
BLT     $r4, $r3, end
swap:
SW      $r4, -4($r2)
SW      $r3, 0($r2)
end:
ADDIU   $r2, $r2, 4
BNE     $r6, $r2, run

ADDIU   $r7, $r7, -1
ADDIU   $r6, $r6, -4
BNE     $r7, $r0, loop

exit:
JR      $r31

.data
a:
.word 11,10,9,8,7,6,5,4,3,2,1
n:
.word 11
```

5.2 运行结果

```
汇总：
  执行周期总数：507
  ID段执行了382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：56       占周期总数的百分比：11.04537%
    其中：
      load停顿：0         占所有RAW停顿的百分比：0%
      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：68          占周期总数的百分比：13.41223%
    自陷停顿：0       占周期总数的百分比：0%
    停顿周期总数：124    占周期总数的百分比：24.45759%

  分支指令：
    指令条数：67          占指令总数的百分比：17.53927%
    其中：
      分支成功：56        占分支指令数的百分比：83.58209%
      分支失败：12        占分支指令数的百分比：17.91045%

  load/store指令：
    指令条数：221         占指令总数的百分比：57.8534%
    其中：
      load：111       占load/store指令数的百分比：50.22625%
      store：110          占load/store指令数的百分比：49.77375%

  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：0.2617801%
```

时钟周期图如下：

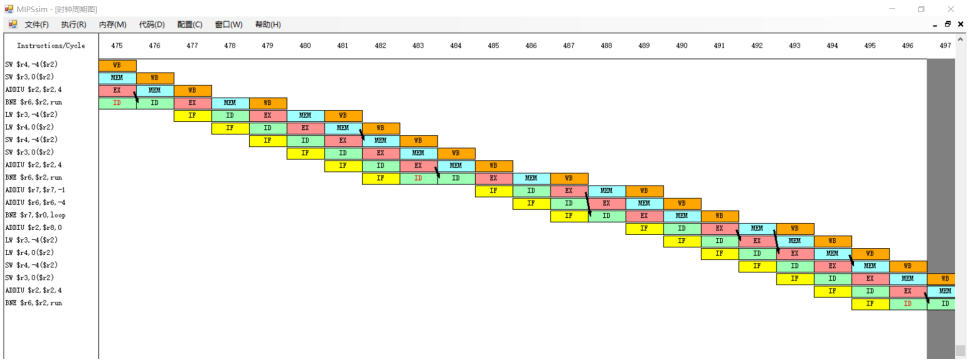

图 3: 时钟周期图

与之前的代码相比，效率大约是之前的 782/507 = 1.54 倍。

6 实验中的问题与心得

本次实验主要是人肉模拟编译器做一些简单的优化，本次实验中没有遇到新的问题。

本次的心得大约是通过丧失语义来进行一些更为激烈的优化，我们依然可以很好的进一步榨取 CPU 的性能。