计算机系统结构实验报告

班级：2017211314　　学号：2017213508　　学生：蒋雪枫

> 完整汇编代码可以在github.com/sprinter1999/PlayGround找到

1.实验 1 MIPS 指令系统和 MIPS 体系结构

1.1 实验目的

(1). 了解和熟悉指令级模拟器
(2). 熟练掌握 MIPSsim 模拟器的操作和使用方法
(3). 熟悉 MIPS 指令系统及其特点，加深对 MIPS 指令操作语义的理解
(4). 熟悉 MIPS 体系结构

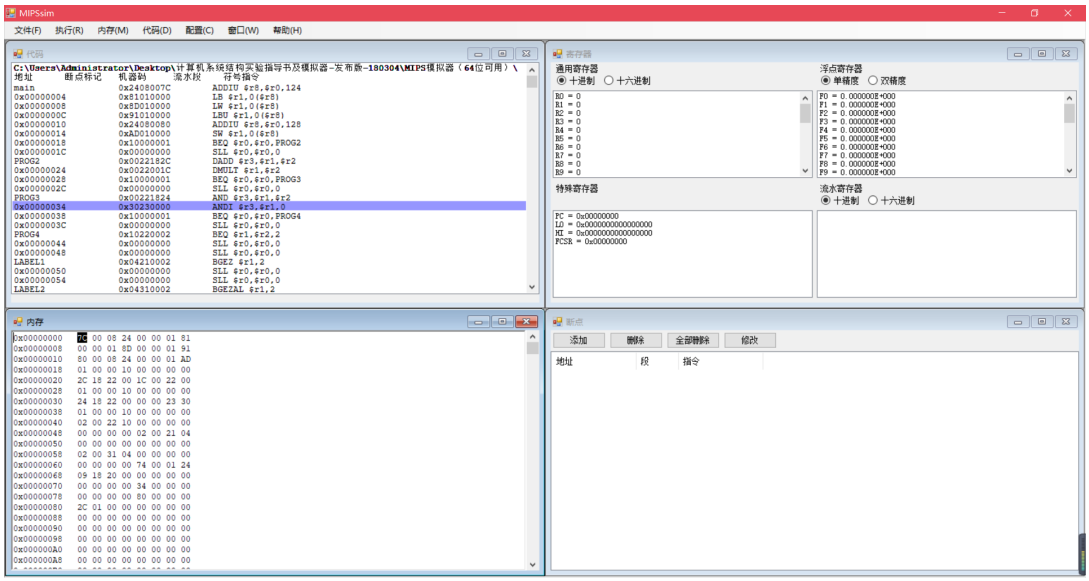
1.2 实验平台

实验平台采用指令级和流水线操作级模拟器 MIPSsim ,工作机为 Windows10 系统 64 位。

1.3 实验内容

首先要阅读 MIPSsim 模拟器的使用方法（见附录），然后了解 MIPSsim 的指令系统和汇
编语言。

(1) 启动 MIPSsim(用鼠标双击 MIPSsim．exe)。
(2) 选择“配置”—>“流水方式”选项，使模拟器工作在非流水方式下。
(3) 参照 MIPSsim 使用说明，熟悉 MIPSsim 模拟器的操作和使用方法。
可以先载人一个样例程序(在本模拟器所在的文件夹下的“样例程序”文件夹中)，然后分别以单步执行一条指令、执行多条指令、连续执行、设置断点等的方式运行程序，观察程序执行情况，观察 CPU 中寄存器和存储器的内容的变化。
(4)选择“文件”—> “载入程序”选项，加载样例程序 alltest.asm，然后查看“代码”窗口， 至看程序所在的位置(起始地址为 0x00000100)。



(5). 查看“寄存器”窗口 PC 寄存器的值：[PC] = 0x00000000。

(6). 执行 load 和 store 指令，步骤如下：

单步执行 1 条指令 (F7)。

下一条指令地址为 0x00000004，是一条 有符号载入 字节 指令。

单步执行 1 条指令 (F7)。

查看 R1 的值，[R1] = 0xFFFFFFFFFFFFFF80。

下一条指令地址为 0x00000008，是一条 有符号载入 字 指令。

单步执行 1 条指令 (F7)。

查看 R1 的值，[R1] = 0x0000000000000080。

下一条指令地址为 0x0000000C，是一条 无符号载入 字节 指令。

单步执行 1 条指令 (F7)。

查看 R1 的值，[R1] = 0x0000000000000080。

单步执行 1 条指令 (F7)。

下一条指令地址为 0x00000014，是一条保存 字 指令。

单步执行 1 条指令 (F7)。

查看内存 BUFFER 处字的值，值为 0x80。（内存 → 符号表）

(7). 执行逻辑运算类指令。步骤如下：

双击“寄存器”窗口中的 R1，将其值修改为 2。

双击“寄存器”窗口中的 R2，将其值修改为 3

2 单步执行 1 条指令。

下一条指令地址为 0x00000020，是一条加法指令。

单步执行 1 条指令。

查看 R3 的值，[R3] = 0x0000000000000005。

下一条指令地址为 0x00000024，是一条乘法指令。

单步执行 1 条指令。

查看 L0、HI 的值，[LO]= 0x0000000000000006，[HI]= 0x0000000000000000。

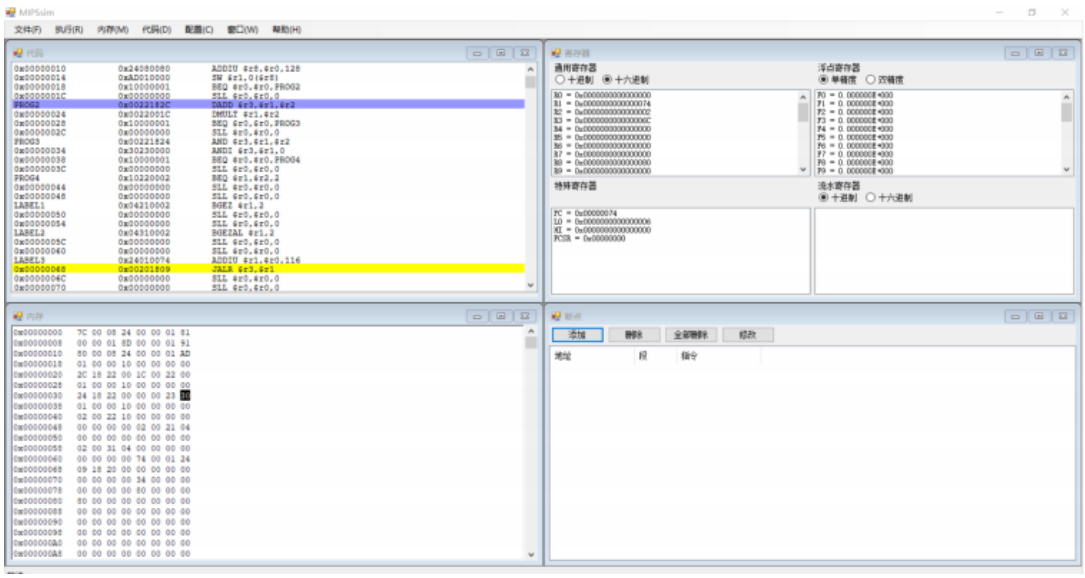
(8). 执行逻辑运算类指令。步骤如下：

双击“寄存器”窗口中的 R1，将其值修改为 0xFFFF0000。

双击“寄存器”窗口中的 R2，将其值修改为 0xFF00FF00。

单步执行 1 条指令。

下一条指令地址为 0x00000030，是一条逻辑与运算指令，第二个操作数寻址方式是 寄存器直接寻址。
单步执行 1 条指令。
查看 R3 的值，[R3] = 0x00000000FF000000。
下一条指令地址为 0x00000034，是一条逻辑与运算指令，第二个操作数寻址方式是 立即数寻址。
单步执行 1 条指令。
查看 R3 的值，[R3] = 0x0000000000000000。
(9). 执行控制转移类指令。步骤如下：
双击“寄存器”窗口中的 R1，将其值修改为 2。
双击“寄存器”窗口中的 R2，将其值修改为 2
单步执行 1 条指令。
下一条指令地址为 0x00000040，是一条 BEQ 指令，其测试条件是 $r1 == $r2，目
标地址为 0x0000004C。
单步执行 1 条指令。
查看 PC 的值，[PC]= 0x0000004C，表明分支 成功。
下一条指令是一条 BGEZ 指令，其测试条件是 $r1 ⩾ 0，目标地址为 0x00000058。
单步执行 1 条指令。
查看 PC 的值，[PC]= 0x00000058，表明分支 成功。
下一条指令是一条 BGEZAL 指令，其测试条件是 $r1 ⩾ 0，目标地址为 0x00000064。
单步执行 1 条指令。
查看 PC 的值，[PC]= 0x00000064，表明分支 成功；查看 R31 的值，[R31] = 0x0000005C。
单步执行 1 条指令。
查看 R1 的值，[R1] = 0x0000000000000074。
下一条指令地址为 0x00000068，是一条 JALR 指令，保存目标地址的寄存器为 3R1，保存返回地址的目标寄存器为 R3。
单步执行 1 条指令。
查看 PC 和 R3 的值，[PC]= 0x00000074，[R3]=0x000000000000006C。



1.4 实验问题与心得

本次实验主要目的是在于熟悉模拟器的使用以及 MIPS 体系结构的特点，实验中没有遇到任何问题。 不过实验 1 的填空太冗长了，有些空感觉不是很有必要。

2.实验 流水线以及流水线中的冲突

2.1 实验目的

（1）加深对计算机流水线基本概念的理解。
（2）理解 MIPS 结构如何用 5 段流水线来实现，理解各段的功能和基本操作。
（3）加深对数据冲突和资源冲突的理解，理解这两类冲突对 CPU 性能的影响。
（4）进一步理解解决数据冲突的方法，掌握如何应用定向技术来减少数据冲突引起的停顿。

2.2 结构冲突

流水线处理机的实际CPI

- 理想流水线的CPI加上各类停顿的时钟周期数：

$$CPI_{流水线} = CPI_{理想} + 停顿_{结构冲突} + 停顿_{数据冲突} + 停顿_{控制冲突}$$

- 理想CPI是衡量流水线最高性能的一个指标。

那什么是结构冲突呢？某些指令组合在流水线重叠执行过程中，如果硬件资源满足不了指令重叠执行的要求，便会产生资源冲突。资源份数不够。在同一个时钟周期内争用同一个功能部件，功能部件不是完全流水。

结构冲突举例 – 访存冲突

I(LOAD)	IF	ID	EX	ME	WB				
I+1		IF	ID	EX	ME	WB			
I+2			IF	ID	EX	ME	WB		
I+3				IF	ID	EX	ME	WB	

- 方法1：插入暂停周期

I(LOAD)	IF	ID	EX	ME	WB				
I+1		IF	ID	EX	ME	WB			
I+2			IF	ID	EX	ME	WB		
I+3				stall	IF	ID	EX	ME	WB

- 方法2：指令与数据的分离（分时）存取处理：
  - 分离cache、双端口RAM、Harvard architecture等

2.3 实验内容和步骤

首先要阅读 MIPSsim 模拟器的使用方法（见附录），然后了解 MIPSsim 的指令系统和汇编语言。

(1). 启动 MIPSsim.exe

(2). 进一步理解流水线窗口中各段的功能，掌握各流水寄存器的含义。（鼠标双击各段，即可看到各流水寄存器的内容）

(3). 载入一个样例程序（在本模拟器所在文件夹下的“样例程序”文件夹中），然后分别以单步执行一个周期、执行多个周期、连续执行、设置断点等方式运行程序，观察程序的执行情况，观察 CPU 中寄存器和存储器内容的变化，特别是流水寄存器内容的变化。

(4). 选择配置菜单中的“流水方式”选项，使模拟器工作于流水方式下

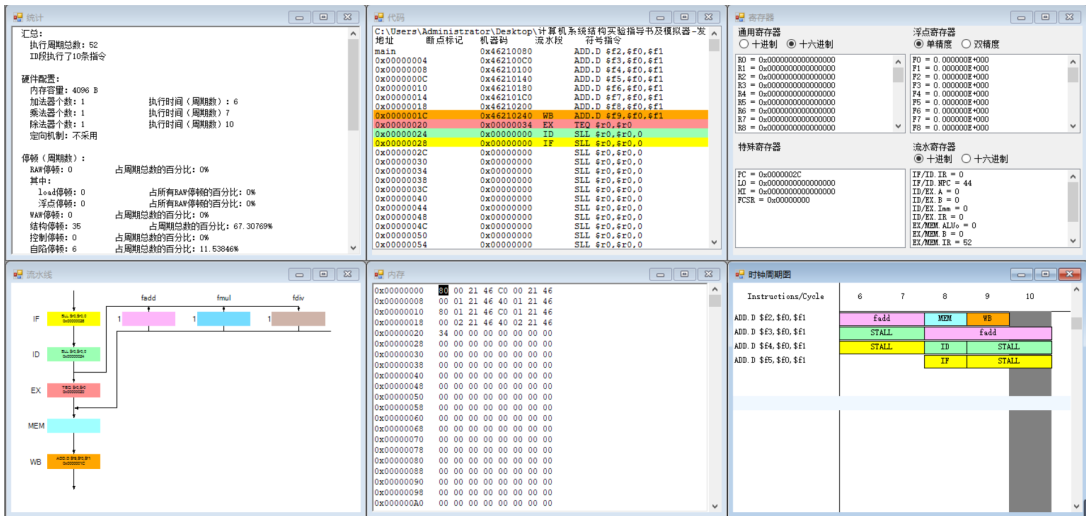
(5). 观察程序在流水方式下的执行情况

(6). 观察和分析结构冲突对 CPU 性能的影响，步骤如下：

a. 加载 structure_hz.s （在模拟器所在文件夹下的“样例程序”文件夹中）。

b. 执行该程序，找出存在结构冲突的指令对以及导致结构冲突的部件。

0x20 前的任意一条指令都会引发结构冲突，冲突部件为浮点数加法器(fadd)。



c. 记录由结构冲突引起的停顿周期数，计算停顿周期数占总执行周期数的百分比。

```
汇总：
  执行周期总数：52
  ID 段执行了 10 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：0       占周期总数的百分比：0%
  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：35      占周期总数的百分比：67.30769%
  控制停顿：0       占周期总数的百分比：0%
  自陷停顿：6       占周期总数的百分比：11.53846%
  停顿周期总数：41  占周期总数的百分比：78.84615%
分支指令：
  指令条数：0       占指令总数的百分比：0%
  其中：
    分支成功：0         占分支指令数的百分比：0%
    分支失败：0         占分支指令数的百分比：0%
load/store 指令：
  指令条数：0       占指令总数的百分比：0%
  其中：
    load：0         占 load/store 指令数的百分比：0%
    store：0        占 load/store 指令数的百分比：0%
浮点指令：
  指令条数：8       占指令总数的百分比：80%
  其中：
    加法：8         占浮点指令数的百分比：100%
    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：10%
```

把浮点加法器的个数改为 4 个。

再重复 1–3 的步骤。

停顿周期：8

占比：42.10526%

运行报告：

```
汇总：
  执行周期总数：19
  ID 段执行了 10 条指令
硬件配置：
```

```
    内存容量：4096 B
    加法器个数：4     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：0       占周期总数的百分比：0%
    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：2       占周期总数的百分比：10.52632%
    控制停顿：0       占周期总数的百分比：0%
    自陷停顿：6       占周期总数的百分比：31.57895%
    停顿周期总数：8   占周期总数的百分比：42.10526%
  分支指令：
    指令条数：0       占指令总数的百分比：0%
    其中：
      分支成功：0         占分支指令数的百分比：0%
      分支失败：0         占分支指令数的百分比：0%
  load/store 指令：
    指令条数：0       占指令总数的百分比：0%
    其中：
      load：0         占 load/store 指令数的百分比：0%
      store：0        占 load/store 指令数的百分比：0%
  浮点指令：
    指令条数：8       占指令总数的百分比：80%
    其中：
      加法：8         占浮点指令数的百分比：100%
      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：10%
```

分析结构冲突对 CPU 性能的影响，讨论解决结构冲突的方法。

影响：当发生冲突时，流水线会出现停顿从而降低 CPU 的性能

解决方式：增加部件，设置独立寄存器

(7). 观察数据冲突并用定向技术来减少停顿，步骤如下：

1.全部复位。

2.加载 data_hz.s （在模拟器所在文件夹下的“样例程序”文件夹中）。

3.关闭定向功能（在“配置”菜单下选择取消“定向”）。

4.用单步执行一个周期的方式执行该程序，观察时钟周期图，列出什么时刻发生了 RAW 冲突。

第 4, 6, 7, 9, 10, 13, 14, 17, 18, 20, 21, 25, 26, 28, 29, 32, 33, 36, 37, 39, 40, 44, 45, 47, 48, 51, 52, 55, 56, 58 周期发生了 RAW 冲突。

5.记录数据冲突引起的停顿周期数以及程序执行的总时钟周期数，计算停顿时钟周期数占总执行周期数的百分比。

停顿周期：31

总周期：65

占比：47.69231%

运行报告：

```
汇总：
  执行周期总数：65
  ID 段执行了 29 条指令
硬件配置：
  内存容量：4096 B
  加法器个数：4     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：31     占周期总数的百分比：47.69231%
  其中：
    load 停顿：12       占所有 RAW 停顿的百分比：38.70968%
    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：3      占周期总数的百分比：4.615385%
  自陷停顿：1      占周期总数的百分比：1.538462%
  停顿周期总数：35  占周期总数的百分比：53.84615%
分支指令：
  指令条数：3      占指令总数的百分比：10.34483%
  其中：
    分支成功：2         占分支指令数的百分比：66.66666%
    分支失败：1         占分支指令数的百分比：33.33333%
load/store 指令：
  指令条数：9      占指令总数的百分比：31.03448%
  其中：
    load：6        占 load/store 指令数的百分比：66.66666%
    store：3       占 load/store 指令数的百分比：33.33333%
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：3.448276%
```

1.复位 CPU。

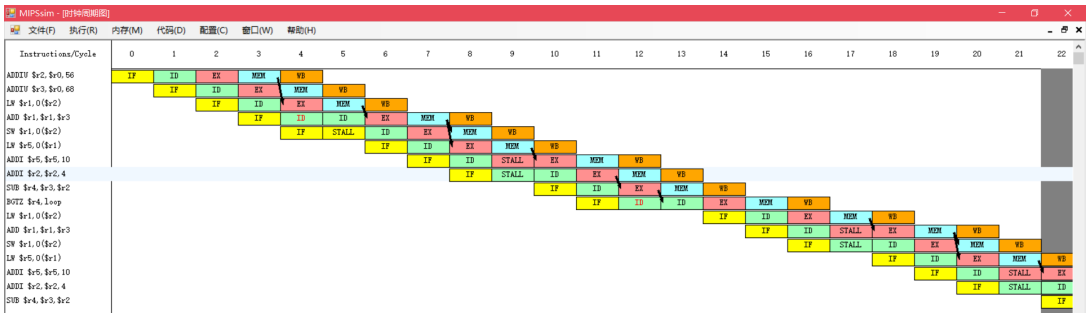
2.打开定向功能。

3.用单步执行一个周期的方式执行该程序，查看时钟周期图，列出什么时刻发生了 RAW 冲突，并与步骤 3）的结果比较。

第 5,9,13,17,21,25,29,33,37 周期发生了 RAW 冲突。

比较：通过定向技术，我们大大减少了 RAW 冲突数

时钟周期图为：



记录数据冲突引起的停顿周期数以及程序执行的总周期数。计算采用定向以后性能比原来提高多少。

停顿周期：9

总周期：43

性能提升：65 / 43 = 1.51

占比：20.93023%

运行报告：

```
汇总：
  执行周期总数：43
  ID 段执行了 29 条指令
硬件配置：
  内存容量：4096 B
  加法器个数：4      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：9        占周期总数的百分比：20.93023%
  其中：
    load 停顿：6         占所有 RAW 停顿的百分比：66.66666%
    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：3        占周期总数的百分比：6.976744%
  自陷停顿：1        占周期总数的百分比：2.325581%
  停顿周期总数：13   占周期总数的百分比：30.23256%
分支指令：
  指令条数：3        占指令总数的百分比：10.34483%
  其中：
    分支成功：2          占分支指令数的百分比：66.66666%
    分支失败：1          占分支指令数的百分比：33.33333%
load/store 指令：
  指令条数：9        占指令总数的百分比：31.03448%
  其中：
    load：6          占 load/store 指令数的百分比：66.66666%
    store：3         占 load/store 指令数的百分比：33.33333%
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：3.448276%
```

2.4 实验心得

本次实验大部分时间是在按部就班的运行别人的代码作为测试，实验中没有遇到任何问题。 至于心得，个人认为是测试程序很好地体现了旁路技术带来的性能优化。

3. 实验 3 使用 MIPS 指令实现求两个数组的点积

3.1 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

3.2 实验内容与步骤：

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：

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + \cdots + a_n b_n$$

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，载入自己编写的程序，观察流水线输出结果。
3. 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。
4. 采用静态调度方法重排指令序列，减少相关，优化程序。
5. 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。

3.3 向量点乘

向量点乘还是比较直接的。

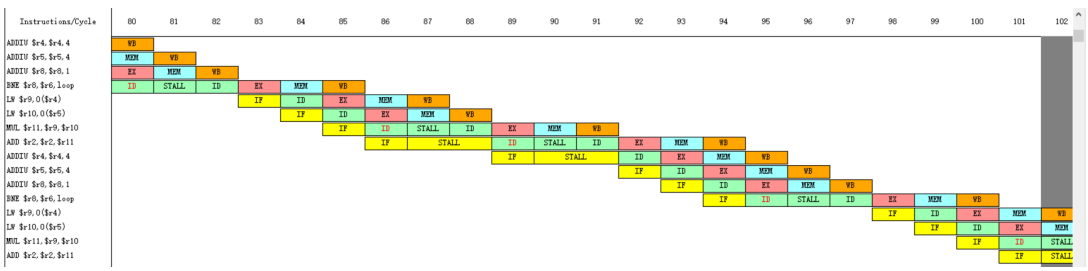
```
int prod(int *a,int *b,int n)
{
  int size=n;
  int result=0;
  for(int i=0;i<size;i++)
  {
      result+=a[i]*b[i];
  }
  return result;
}
```

然后把这段代码翻译成 MIPS 风格的汇编代码即可，结果如下（需要注意 MIPS 的函数调用约定）

```
.text
main:
ADDIU $r4,$r0,a
ADDIU $r5,$r0,b
ADDIU $r6,$r0,n
BGEZAL $r0,prod
NOP
TEQ $r0,$r0
prod:
LW $r6,0($r6)
ADD $r8,$r0,$r0
ADD $r2,$r0,$r0
loop:
LW $r9,0($r4)
LW $r10, 0($r5)
MUL $r11, $r9, $r10
ADD $r2, $r2, $r11
ADDIU $r4, $r4, 4
ADDIU $r5, $r5, 4
ADDIU $r8, $r8, 1
BNE $r8, $r6, loop
JR $r31
.data
a:
.word 1,2,3,4,5,6,7,8,9,10,11
b:
.word 1,2,3,4,5,6,7,8,9,10,11
n:
.word 11
```

3.4 运行结果

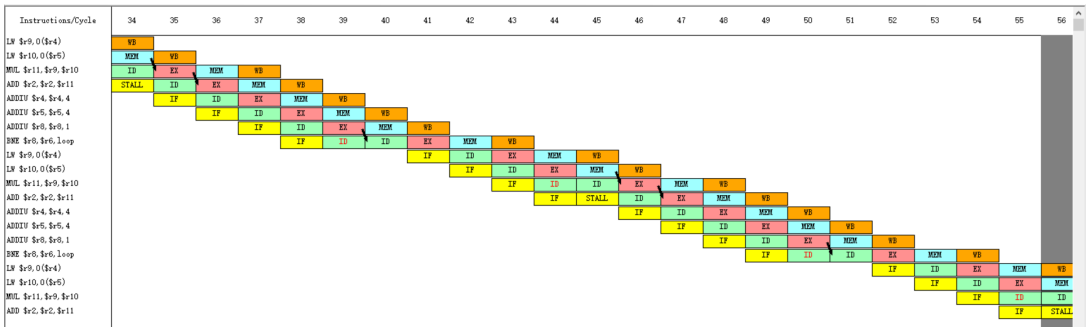
未开启定向时：




```
汇总：
    执行周期总数：178
    ID 段执行了 98 条指令
硬件配置：
    内存容量：4096 B
    加法器个数：4      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：66      占周期总数的百分比：37.07865%
    其中：
      load 停顿：22      占所有 RAW 停顿的百分比：33.33333%
      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：13      占周期总数的百分比：7.303371%
    自陷停顿：0      占周期总数的百分比：0%
    停顿周期总数：79  占周期总数的百分比：44.38202%
  分支指令：
    指令条数：12      占指令总数的百分比：12.2449%
    其中：
      分支成功：12      占分支指令数的百分比：100%
      分支失败：1      占分支指令数的百分比：8.333333%
  load/store 指令：
    指令条数：24      占指令总数的百分比：24.4898%
    其中：
      load：24      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：1.020408%
```

开启定向时：




```
  汇总：
    执行周期总数：134
    ID 段执行了 98 条指令
  硬件配置：
    内存容量：4096 B
    加法器个数：4      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
```

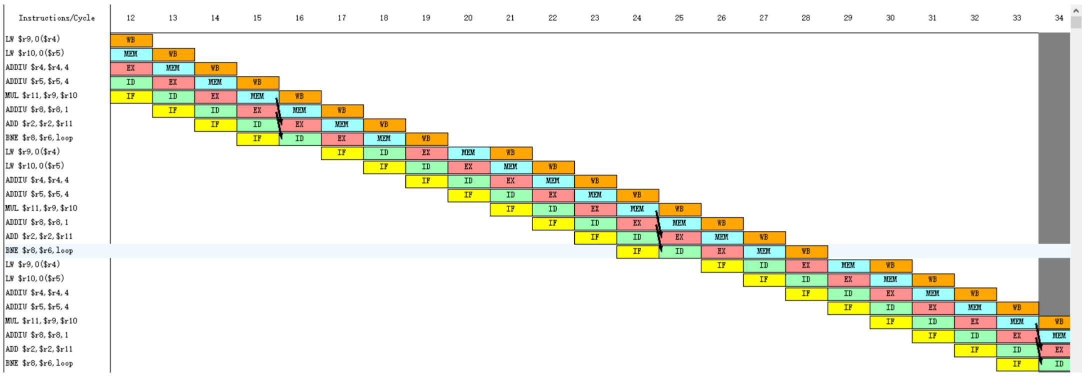
```
  停顿（周期数）：
    RAW 停顿：22      占周期总数的百分比：16.41791%
    其中：
      load 停顿：11       占所有 RAW 停顿的百分比：50%
      浮点停顿：0         占所有 RAW 停顿的百分比：0%
    WAW 停顿：0       占周期总数的百分比：0%
    结构停顿：0       占周期总数的百分比：0%
    控制停顿：13      占周期总数的百分比：9.701492%
    自陷停顿：0       占周期总数的百分比：0%
    停顿周期总数：35  占周期总数的百分比：26.1194%
  分支指令：
    指令条数：12      占指令总数的百分比：12.2449%
    其中：
      分支成功：12        占分支指令数的百分比：100%
      分支失败：1         占分支指令数的百分比：8.333333%
  load/store 指令：
    指令条数：24      占指令总数的百分比：24.4898%
    其中：
      load：24       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：1.020408%
```

3.5 消除数据相关后的向量点积

```
.text
main:
ADDIU $r4,$r0,a
ADDIU $r5,$r0,b
ADDIU $r6,$r0,n
BGEZAL $r0, prodOpt
NOP
TEQ $r0,$r0
prodOpt:
LW $r6, 0($r6)
ADD $r8, $r0, $r0
ADD $r2, $r0, $r0
loop:
LW $r9, 0($r4)
LW $r10, 0($r5)
ADDIU $r4, $r4, 4
ADDIU $r5, $r5, 4
MUL $r11, $r9, $r10
```

```
ADDIU $r8, $r8, 1
ADD $r2, $r2, $r11
BNE $r8, $r6, loop
JR $r31
.data
a:
.word 1,2,3,4,5,6,7,8,9,10,11
b:
.word 1,2,3,4,5,6,7,8,9,10,11
n:
.word 11
```

执行结果:



运行报告:

```
汇总:
    执行周期总数: 112
    ID 段执行了 98 条指令
硬件配置:
    内存容量: 4096 B
    加法器个数: 4      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: 0       占周期总数的百分比: 0%
    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: 13      占周期总数的百分比: 11.60714%
    自陷停顿: 0       占周期总数的百分比: 0%
    停顿周期总数: 13  占周期总数的百分比: 11.60714%
分支指令:
    指令条数: 12      占指令总数的百分比: 12.2449%
    其中:
        分支成功: 12         占分支指令数的百分比: 100%
        分支失败: 1          占分支指令数的百分比: 8.333333%
load/store 指令:
    指令条数: 24      占指令总数的百分比: 24.4898%
```

```
    其中：
      load：24        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：1.020408%
```

从周期上来看，与之前的代码相比，效率大约是之前的 134/112 = 1.19 倍。

3.6 实验总结

本次实验主要是模拟编译器做一些优化，本次实验中遇到的问题有一部分指令没有在现有的 MIPSsim 模拟器上实现，比如实际编译器经常产生的 BAL 指令，不过这类问题可以通过更换别的指令替代来解决。本次的心得大约是通过一些静态优化，我们也可以很好地进一步减少程序运行时间，提高代码效率。

4.实验4 使用 MIPS 指令实现冒泡排序法

4.1 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

4.2 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。

4.3 冒泡排序

冒泡排序作为大一我们就学过的基本算法，虽然我们一般都直接std::sort，但它的思想理应刻在我们的脑海中。

```
void bubbleSort(int *arr,int n)
{
    int m,i,j;
    for(i=0;i<n-1;i++)
```

```
        for(j=0;j<n-1-i;j++)
            if(arr[j]>arr[j+1])
            {
                m=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=m;
            }
}
```

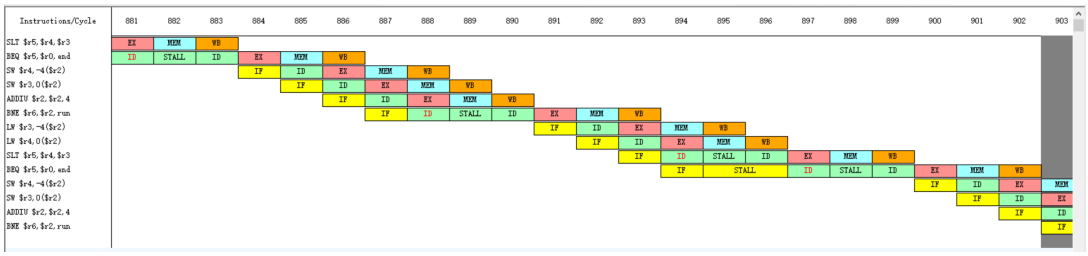
然后遵循MIPS调用约定将它转换为汇编代码：

```
.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
```

4.3 运行结果

未开启定向：

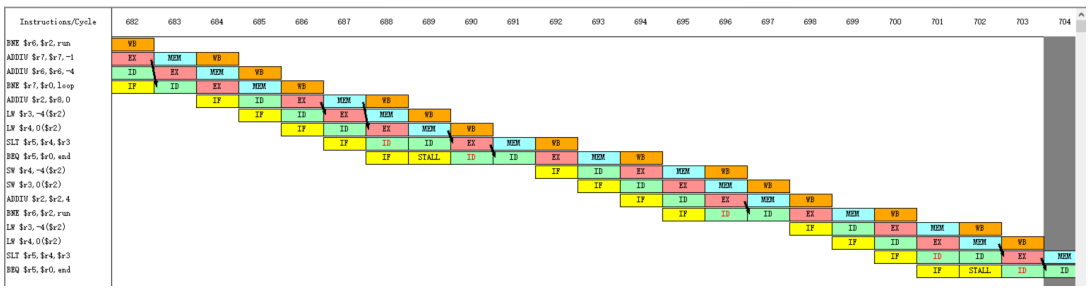
```
汇总：
    执行周期总数：981
    ID段执行了492条指令
硬件配置：
    内存容量：4096 B
    加法器个数：1       执行时间（周期数）：6
    乘法器个数：1       执行时间（周期数）7
    除法器个数：1       执行时间（周期数）10
    定向机制：不采用
停顿（周期数）：
    RAW停顿：365        占周期总数的百分比：37.20693%
    其中：
        load停顿：110       占所有RAW停顿的百分比：30.13699%
        浮点停顿：0         占所有RAW停顿的百分比：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%
```



开启定向后：

```
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%
```



4.4 优化后的冒泡排序

将两条指令用一条指令进行实现，优化SLT指令：

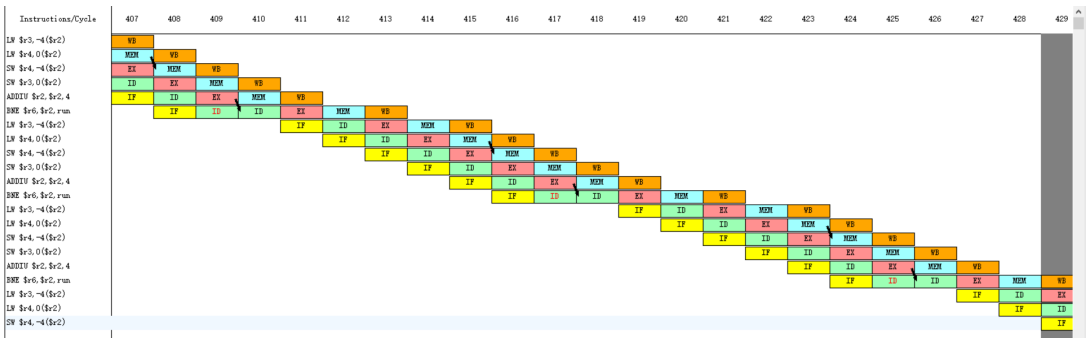
```
.text
main:
ADDIU $r4,$r0,a
```

```
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
```

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%
```



从周期看，与之前的代码相比，效率大约是之前的 782/507 = 1.54 倍。

4.5 实验心得

本次实验感觉比较难的在于编写冒泡排序代码，通过条件跳转嵌套实现嵌套for循环，同时对汇编进行一定优化，可以进一步榨取CPU性能。

5. 实验 5 指令调度与延迟分支

5.1 实验目的

（1）加深对指令调度技术的理解。
（2）加深对延迟分支技术的理解。
（3）熟练账务用指令调度技术解决流水线中的数据冲突的方法。
（4）进一步理解指令调度技术对 CPU 性能的改进。
（5） 进一步理解延迟分支技术对 CPU 性能的改进。

5.2 指令调度技术与控制冲突

指令调度：

- 让编译器重新组织指令顺序来消除冲突，这种技术称为指令调度或流水线调度。
  - 例如：采用典型的代码生成方法，表达式A＝B＋C的代码会导致暂停

LD Rb, B	IF	ID	EX	MEM	WB				
LD Rc, C		IF	ID	EX	MEM	WB			
DADD Ra, Rb, Rc			IF	ID	stall	EX	MEM	WB	
SD Ra , A				IF	stall	ID	EX	MEM	WB

- 举例：请为下列表达式生成没有暂停的指令序列：
  A＝B＋C；
  D＝E－F；

调度前的代码		调度后的代码	
LD	Rb, B	LD	Rb, B
LD	Rc, C	LD	Rc, C
DADD	Ra, Rb, Rc	LD	Re, E
SD	Ra, A	DADD	Ra, Rb, Rc
LD	Re, E	LD	Rf, F
LD	Rf, F	SD	Ra, A
DSUB	Rd, Re, Rf	DSUB	Rd, Re, Rf
SD	Rd, D	SD	Rd, D

控制相关：

- 执行分支指令的结果有两种
  - 分支成功：PC值改变为分支转移的目标地址。在条件判定和转移地址计算都完成后，才改变PC值。
  - 不成功或者失败：PC的值保持正常递增，指向顺序的下一条指令。
- 处理分支指令最简单的方法 - “冻结”或者“排空”流水线 。

BRANCH	IF	ID	EX	ME	WB				
i+1		*stall*	*stall*	*stall*	IF	ID	EX	ME	WB

Freeze

- 把由分支指令引起的延迟称为分支延迟。
- 分支指令在目标代码中出现的频度
  - 若每3～4条指令就有一条是分支指令。
    - 假设：分支指令出现的频度是30%，流水线理想CPI＝1，
    - 那么：流水线的实际 CPI ＝ ？ 。
- 可采取两种措施来减少分支延迟
  - 在流水线中尽早判断出分支转移是否成功；
  - 尽早计算出分支目标地址。

- 3种通过软件（编译器）来减少分支延迟的方法：
  - 预测分支失败-Treat every branch as not taken
  - 预测分支成功-Treat every branch as taken
  - 延迟分支- Delayed branch

5.3 实验步骤

(1). 启动 MIPSsim(用鼠标双击 MIPSsim.exe)。

(2). 根据实验 2 的相关知识中关于流水线各段操作的描述，进一步理解流水线窗口中各段的功能，掌握各流水线寄存器的含义（双击各段，就可以看到各流水线寄存器中的内容）。

(3). 选择“配置”→ “流水方式”选项，使模拟器工作在流水方式下

(4). 用指令调度技术解决流水线中的结构冲突与数据冲突：

启动 MIPSsim。

1用 MIPSsim 的“文件”→ “载入程序”选项来加载 schedule.s （在模拟器所在文件夹下的“样例程序”文件夹中）。

关闭定向功能，这是通过“配置“→ ”定向“选项来实现的。

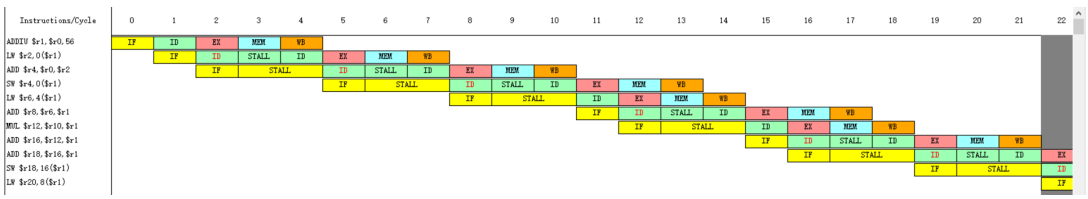
执行所载入的程序，通过查看统计数据和时钟周期图，**找出并记录程序执行过程中各种冲突发生的次数**，发生冲突的指令组合以及程序执行的总时钟周期数。

- RAW 停顿：16
- 自陷停顿：1
- 发生冲突的指令组合：
  * LW $r2,0($r1) 和 ADD $r4,$r0,$r2
  * ADD $r4,$r0,$r2 和 SW $r4,0($r1)
  * SW $r4,0($r1) 和 LW $r6,4($r1)
  * ADD $r8,$r6,$r1 和 MUL $r12,$r10,$r1
  * ADD $r16,$r12,$r1 和 ADD $r18,$r16,$r1
  * ADD $r18,$r16,$r1 和 SW $r18,16($r1)
  * SW $r18,16($r1) 和 LW $r20 8($r1)
  * MUL $r22,$r20,$r14 和 MUL $r24,$r26,$r14
- 总执行周期：33

（5）运行报告：

```
汇总：
  执行周期总数：33
  ID段执行了15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：16       占周期总数的百分比：48.48485%
  其中：
```

```
        load停顿：6          占所有RAW停顿的百分比：37.5%
        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：0         占周期总数的百分比：0%
      自陷停顿：1         占周期总数的百分比：3.030303%
      停顿周期总数：17    占周期总数的百分比：51.51515%
    分支指令：
      指令条数：0         占指令总数的百分比：0%
      其中：
        分支成功：0          占分支指令数的百分比：0%
        分支失败：0          占分支指令数的百分比：0%
    load/store指令：
      指令条数：5         占指令总数的百分比：33.33333%
      其中：
        load：3          占load/store指令数的百分比：60%
        store：2         占load/store指令数的百分比：40%
    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：6.666667%
```



5.4 应用指令调度技术

自己采用调度技术对程序进行**指令调度**，**消除冲突**（自己修改源程序）。将调度（修改）后的程序重新命名为 afer-schedule.s 。（注意：调度方法灵活多样，在保证程序正确性的前提下自己随意调度，尽量减少冲突即可，不要求要达到最优。）

原来的schedule.s:

```
.text
main:
ADDIU  $r1,$r0,A
LW     $r2,0($r1)
ADD    $r4,$r0,$r2
SW     $r4,0($r1)
LW     $r6,4($r1)
ADD    $r8,$r6,$r1
MUL    $r12,$r10,$r1
ADD    $r16,$r12,$r1
```

```
ADD    $r18,$r16,$r1
SW     $r18,16($r1)
LW     $r20,8($r1)
MUL    $r22,$r20,$r14
MUL    $r24,$r26,$r14
TEQ $r0,$r0
.data
A:
.word 4,6,8
```

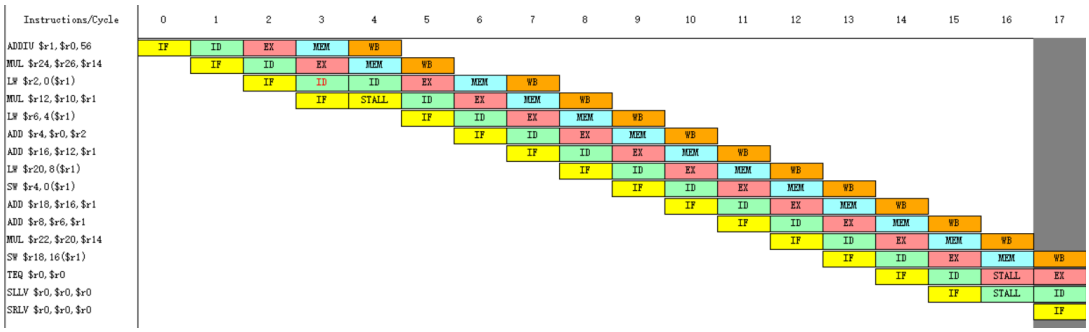
优化后的after-schedule.s：

```
.text
main:
ADDIU $r1,$r0,A
MUL $r24,$r26,$r14
LW $r2,0($r1)
MUL $r12,$r10,$r1
LW $r6,4($r1)
ADD $r4,$r0,$r2
ADD $r16,$r12,$r1
LW $r20,8($r1)
SW $r4,0($r1)
ADD $r18,$r16,$r1
ADD $r8,$r6,$r1
MUL $r22,$r20,$r14
SW $r18,16($r1)
TEQ $r0,$r0
.data
A:
.word 4,6,8
```

运行报告：

```
  汇总：
    执行周期总数：18
    ID段执行了15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：1        占周期总数的百分比：5.555555%
    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：0       占周期总数的百分比：0%
```

```
    自陷停顿：1          占周期总数的百分比：5.555555%
    停顿周期总数：2      占周期总数的百分比：11.11111%
  分支指令：
    指令条数：0          占指令总数的百分比：0%
    其中：
      分支成功：0            占分支指令数的百分比：0%
      分支失败：0            占分支指令数的百分比：0%
  load/store指令：
    指令条数：5          占指令总数的百分比：33.33333%
    其中：
      load：3            占load/store指令数的百分比：60%
      store：2           占load/store指令数的百分比：40%
  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：6.666667%
```



比较调度前后性能：

调度前的执行周期为 33，调度后的执行周期数为 18。

指令调度可以消除部分的数据冲突，通过使用指令调度提高了 CPU 的使用率，大大减少了指令冲突的次数，提高了 CPU 性能。

5.5 用延迟分支技术减少分支指令对性能的影响：

在 MIPSsim 中载入 branch.s 样例程序（在本模拟器目录的“样例程序”文件夹中。 关闭延迟分支功能。这是通过在“配置”→ “延迟槽”选项来实现的。执行该程序，观察并记录发生分支延迟的时刻，记录该程序执行的总时钟周期数。

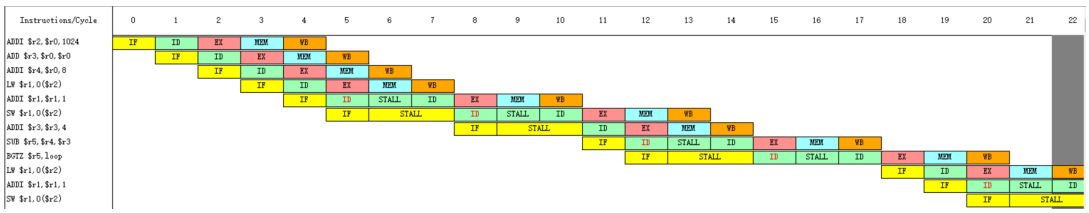
总执行周期：38

第 6, 9, 13, 21, 24, 28 周期发生了分支延迟

运行报告：

```
  汇总：
    执行周期总数：38
    ID段执行了18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：16        占周期总数的百分比：42.10526%
    其中：
      load停顿：4          占所有RAW停顿的百分比：25%
      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：2        占周期总数的百分比：5.263158%
    自陷停顿：1        占周期总数的百分比：2.631579%
    停顿周期总数：19   占周期总数的百分比：50%
  分支指令：
    指令条数：2        占指令总数的百分比：11.11111%
    其中：
      分支成功：1          占分支指令数的百分比：50%
      分支失败：1          占分支指令数的百分比：50%
  load/store指令：
    指令条数：4        占指令总数的百分比：22.22222%
    其中：
      load：2          占load/store指令数的百分比：50%
      store：2         占load/store指令数的百分比：50%
  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：5.555555%
```



假设延迟槽为一个，自己对 branch.s 程序进行指令调度（自己修改源程序），将调度后的程序重新命名为 delayed-branch.s：

原始branch.s:

```
.text
main:
ADDI   $r2,$r0,1024
ADD    $r3,$r0,$r0
ADDI   $r4,$r0,8
loop:
LW     $r1,0($r2)
ADDI   $r1,$r1,1
```

```
SW    $r1,0($r2)
ADDI  $r3,$r3,4
SUB   $r5,$r4,$r3
BGTZ  $r5,loop
ADD   $r7,$r0,$r6
TEQ   $r0,$r0
```

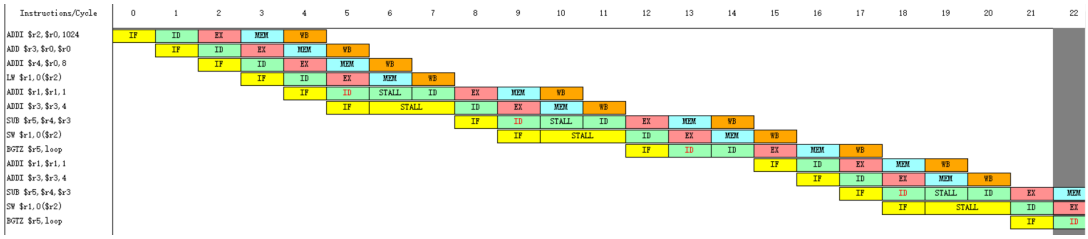
修改后的delayed-branch.s:

```
.text
main:
ADDI $r2,$r0,1024
ADD $r3,$r0,$r0
ADDI $r4,$r0,8
LW $r1,0($r2)
loop:
ADDI $r1,$r1,1
ADDI $r3,$r3,4
SUB $r5,$r4,$r3
SW $r1,0($r2)
BGTZ $r5,loop
LW $r1,0($r2)
ADD $r7,$r0,$r6
TEQ $r0,$r0
```

载入 delayed-branch.s，打开延迟分支功能，执行该程序，观察其时钟周期图，记录程序执行的总时钟周期数。总执行30个时钟周期，运行报告如下。

```
  汇总：
    执行周期总数：30
    ID段执行了18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：8        占周期总数的百分比：26.66667%
    其中：
      load停顿：2         占所有RAW停顿的百分比：25%
      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：2       占周期总数的百分比：6.666667%
    自陷停顿：1       占周期总数的百分比：3.333333%
    停顿周期总数：11  占周期总数的百分比：36.66667%
  分支指令：
    指令条数：2       占指令总数的百分比：11.11111%
    其中：
      分支成功：1         占分支指令数的百分比：50%
```

```
        分支失败：1          占分支指令数的百分比：50%
    load/store指令：
      指令条数：4        占指令总数的百分比：22.22222%
      其中：
        load：2          占load/store指令数的百分比：50%
        store：2         占load/store指令数的百分比：50%
    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：5.555555%
```



5.6 实验中的问题与心得

本次实验的心得是，分支延迟作为 MIPS 当年提出时的特性之一，在大家都是顺序流水线时对性能做出了很好的优化。但是随着乱序流水线等其他技术的兴起，分支延迟逐渐不被大家看好，成为程序运行的负担，包括RISC-V的官方设计文档也把这个拿出来批评了一番，不过我们都应该尊重当年这个优秀的技术的设计思想。