实验3 使用 MIPS 指令实现求两个数组的点积

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. 实验内容和步骤：

（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：

$$a \cdot b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

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。

（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. 实验原理

这段代码是用于计算两个向量的点积的 MIPS 汇编程序。以下是这个程序的实验原理：

1. **数据定义**：在`.data`段中，定义了两个向量`vectorA`和`vectorB`，以及一个变量`size`用于存储向量的维度，还有一个变量`result`用于存储点积的结果。
2. **初始化**：在`main`函数中，首先通过`ADDIU`指令将向量和结果的地址加载到寄存器中，然后通过`LW`指令将向量的维度加载到寄存器中。接着，初始化索引`i`和结果为 0。
3. **循环计算**：在`loop`标签下，通过`LW`指令加载向量的第`i`个元素，然后通过`MUL`指令计算两个元素的乘积，接着通过`ADD`指令将乘积累加到结果中。然后，通过`ADDI`指令移动到向量的下一个元素，并将索引`i`加 1。最后，通过`BNE`指令判断`i`是否等于向量的维度，如果不等于，则跳转到`loop`继续循环。
4. **存储结果**：当`i`等于向量的维度时，循环结束，通过`SW`指令将结果存储到内存中。
5. **结束程序**：通过`TEQ`指令结束程序。

这个程序的主要目的是通过汇编语言实现向量的点积运算，以此来理解和掌握汇编语言的基本语法和指令，以及计算机的运算原理。

5. 程序代码及注释

```
.data
    vectorA: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10  # 向量A
    vectorB: .word 10, 9, 8, 7, 6, 5, 4, 3, 2, 1  # 向量B
    size: .word 10  # 向量的维度
    result: .word 0  # 存储点积结果
.text
main:
    ADDIU $r1, $r0, vectorA  # 加载向量A的地址
    ADDIU $r2, $r0, vectorB  # 加载向量B的地址
    ADDIU $r3, $r0, result  # 加载结果的地址
    ADDIU $r4, $r0, size # 加载 size 的地址
    LW $r4, 0($r4)  # 加载向量的维度
    ADDIU $r5, $r0, 0  # 初始化索引i为0
    ADDIU $r6, $r0, 0  # 初始化结果为0

loop:
    LW $r7, 0($r1)  # 加载向量A的第i个元素
    LW $r8, 0($r2)  # 加载向量B的第i个元素
    MUL $r9, $r7, $r8  # 计算A[i]*B[i]
    ADD $r6, $r6, $r9  # 累加到结果中

    ADDI $r1, $r1, 4  # 移动到向量A的下一个元素
    ADDI $r2, $r2, 4  # 移动到向量B的下一个元素
    ADDI $r5, $r5, 1  # 索引i加1

    BNE $r5, $r4, loop  # 如果i不等于向量的维度，继续循环

    SW $r6, 0($r3)  # 将结果存储到内存中

    # 结束程序
    TEQ $r0, $r0
```

运行结果

汇总：
执行周期总数：164
ID段执行了90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：62　　占周期总数的百分比：37.80488%
其中：
load停顿：20　　占所有RAW停顿的百分比：32.25806%

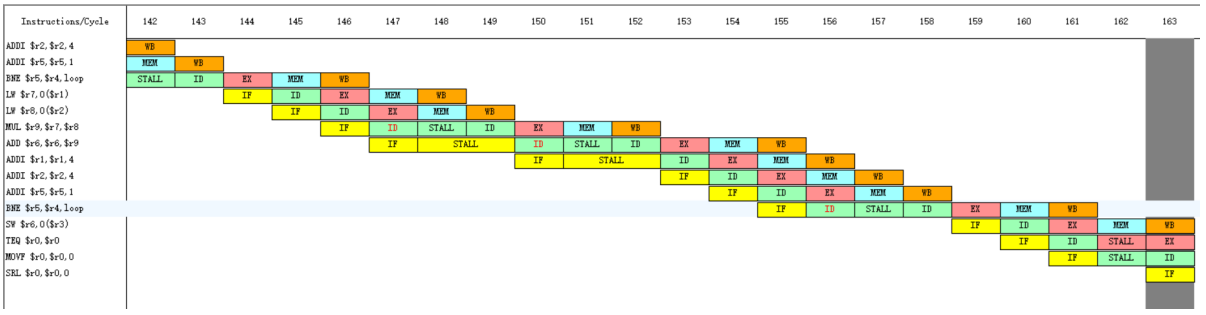
浮点停顿：0　　占所有RAW停顿的百分比：0%
WAW停顿：0　　占周期总数的百分比：0%
结构停顿：0　　占周期总数的百分比：0%
控制停顿：10　　占周期总数的百分比：6.097561%
自陷停顿：1　　占周期总数的百分比：0.6097561%
停顿周期总数：73　　占周期总数的百分比：44.5122%

分支指令：
指令条数：10　　占指令总数的百分比：11.11111%
其中：
分支成功：9　　占分支指令数的百分比：90%
分支失败：1　　占分支指令数的百分比：10%

load/store指令：
指令条数：22　　占指令总数的百分比：24.44444%
其中：
load：21　　占load/store指令数的百分比：95.45454%
store：1　　占load/store指令数的百分比：4.545455%

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：1.111111%



其中，主要有：

`MUL $r9, $r7, $r8` 和 `ADD $r6, $r6, $r9` 的数据相关

`LW $r8, 0($r2)` 和 `MUL $r9, $r7, $r8` 的数据相关

开启定向后，再次执行

结果如下

汇总：
执行周期总数：122
ID段执行了90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：20　　占周期总数的百分比：16.39344%

其中：

load停顿：10　　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：10　　占周期总数的百分比：8.196721%

自陷停顿：1　　占周期总数的百分比：0.8196721%

停顿周期总数：31　　占周期总数的百分比：25.40984%

分支指令：

指令条数：10　　占指令总数的百分比：11.11111%

其中：

分支成功：9　　占分支指令数的百分比：90%

分支失败：1　　占分支指令数的百分比：10%

load/store指令：

指令条数：22　　占指令总数的百分比：24.44444%

其中：

load：21　　占load/store指令数的百分比：95.45454%

store：1　　占load/store指令数的百分比：4.545455%

浮点指令：

指令条数：0　　占指令总数的百分比：0%

其中：

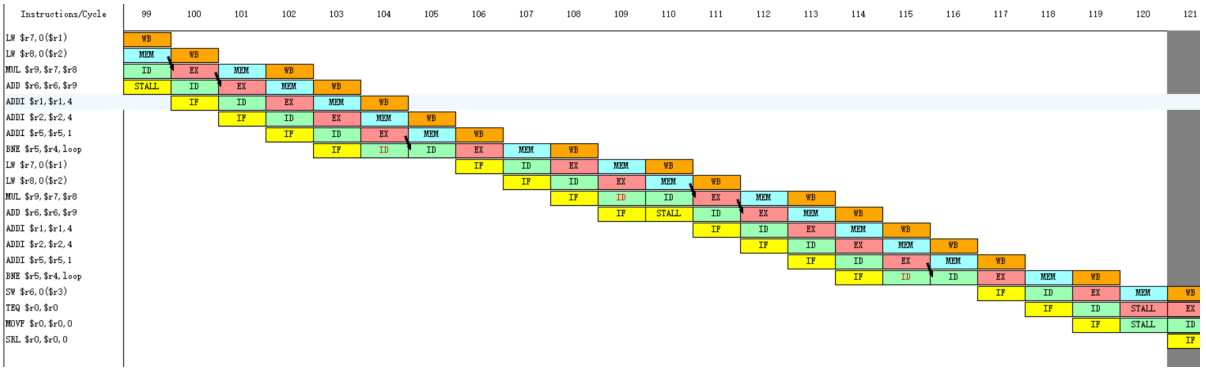
加法：0　　占浮点指令数的百分比：0%

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：1.111111%



定向功能消除了部分数据冲突，性能提升了 $164/122 \approx 1.344$倍

6. 优化后的程序代码

采用静态调度的方法，重排指令序列，减少相关，优化后的程序

```
.data
    vectorA: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10  # 向量A
    vectorB: .word 10, 9, 8, 7, 6, 5, 4, 3, 2, 1  # 向量B
    size: .word 10  # 向量的维度
    result: .word 0  # 存储点积结果
.text
main:
    ADDIU $r1, $r0, vectorA  # 加载向量A的地址
    ADDIU $r2, $r0, vectorB  # 加载向量B的地址
    ADDIU $r3, $r0, result  # 加载结果的地址
    ADDIU $r4, $r0, size # 加载 size 的地址
    LW $r4, 0($r4)  # 加载向量的维度
    ADDIU $r5, $r0, 0  # 初始化索引i为0
    ADDIU $r6, $r0, 0  # 初始化结果为0

loop:
    LW $r7, 0($r1)  # 加载向量A的第i个元素
    LW $r8, 0($r2)  # 加载向量B的第i个元素

    MUL $r9, $r7, $r8  # 计算A[i]*B[i]

    ADDI $r1, $r1, 4  # 移动到向量A的下一个元素
    ADDI $r2, $r2, 4  # 移动到向量B的下一个元素
    ADDI $r5, $r5, 1  # 索引i加1

    ADD $r6, $r6, $r9  # 累加到结果中

    BNE $r5, $r4, loop  # 如果i不等于向量的维度，继续循环

    SW $r6, 0($r3)  # 将结果存储到内存中

    # 结束程序
    TEQ $r0, $r0
```

7. 优化前后的性能比较

对优化后的程序使用定向功能执行，与刚刚的结果进行比较。

汇总：
执行周期总数：112
ID段执行了90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

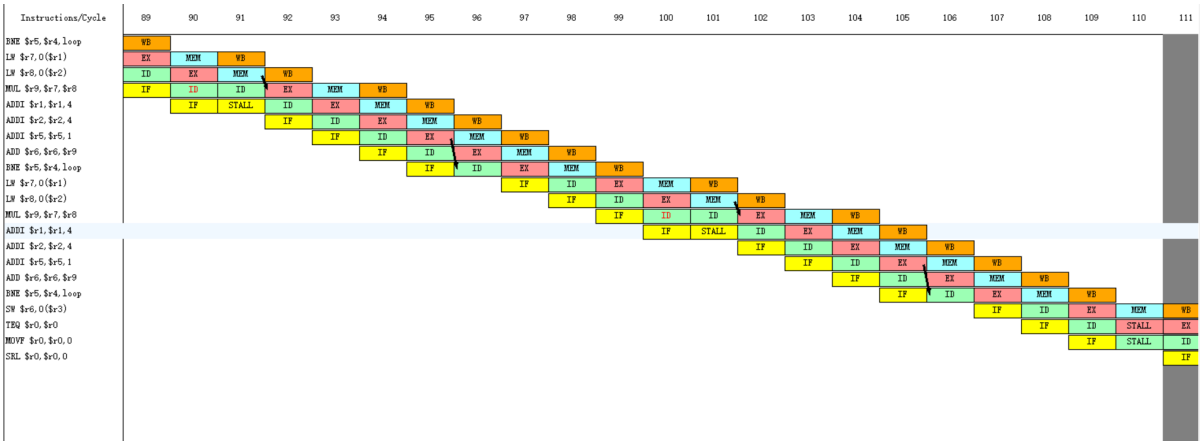
停顿（周期数）：
RAW停顿：10　　占周期总数的百分比：8.928572%
其中：
load停顿：10　　占所有RAW停顿的百分比：100%
浮点停顿：0　　占所有RAW停顿的百分比：0%
WAW停顿：0　　占周期总数的百分比：0%
结构停顿：0　　占周期总数的百分比：0%
控制停顿：10　　占周期总数的百分比：8.928572%
自陷停顿：1　　占周期总数的百分比：0.8928571%
停顿周期总数：21　　占周期总数的百分比：18.75%

分支指令：
指令条数：10　　占指令总数的百分比：11.11111%
其中：
分支成功：9　　占分支指令数的百分比：90%
分支失败：1　　占分支指令数的百分比：10%

load/store指令：
指令条数：22　　占指令总数的百分比：24.44444%
其中：
load：21　　占load/store指令数的百分比：95.45454%
store：1　　占load/store指令数的百分比：4.545455%

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：1.111111%



从中可以看出，消除了一部分的数据冲突，性能提升了$122/112 \approx 1.08$倍