**深 圳 大 学 实 验 报 告**

**课 程 名 称： 计算机系统(3)**

**实验项目名称： MIPS64乘法器模拟实验**

**学 院： 计算机与软件学院**

**专 业： 计算机科学与技术**

**指 导 教 师： 刘刚**

**报告人： 吴嘉楷 学号： 2022150168 班级： 国际班**

**实 验 时 间： 2024年10月10日~10月17日**

**实验报告提交时间： 2024年10月12日**

**教务处制**

**一、 实验目标：**

实际运用WinMIPS64进行试验，以期更了解WinMIPS64的操作；

更加深入地了解MIPS程序的语法；

深入地了解在计算机中乘法的实现以及加法与乘法之间的关系。

**二、实验内容**

按照下面的实验步骤及说明，完成相关操作记录实验过程的截图：

首先，我们使用加法操作设计一个不检测溢出的乘法操作；完成后，我们对此进行优化，以期获得一个可以对溢出进行检测的乘法操作。（100分）

**三、实验环境**

硬件：桌面PC

软件：Windows，WinMIPS64仿真器

**四、****实验步骤及说明**

本次试验分为两个部分：第一部分、用加法器设计一个不考虑溢出的乘法器；第二部分、用加法器设计一个考虑溢出的乘法器（编程熟练的同学，也可以用除法器、浮点加法器等替代）。

1、忽略溢出的乘法器

首先，我们得了解乘法器如何由加法器设计得到，此处，我们以32位乘法为例。

总共分为4步：

1. 测试乘数**最低位**是否为1，是则给乘积加上被乘数，将结果写入乘积寄存器；
2. 被乘数寄存器左移1位；
3. 乘数寄存器右移一位；
4. 判断是否循环了32次，如果是，则结束，否则返回步骤1。

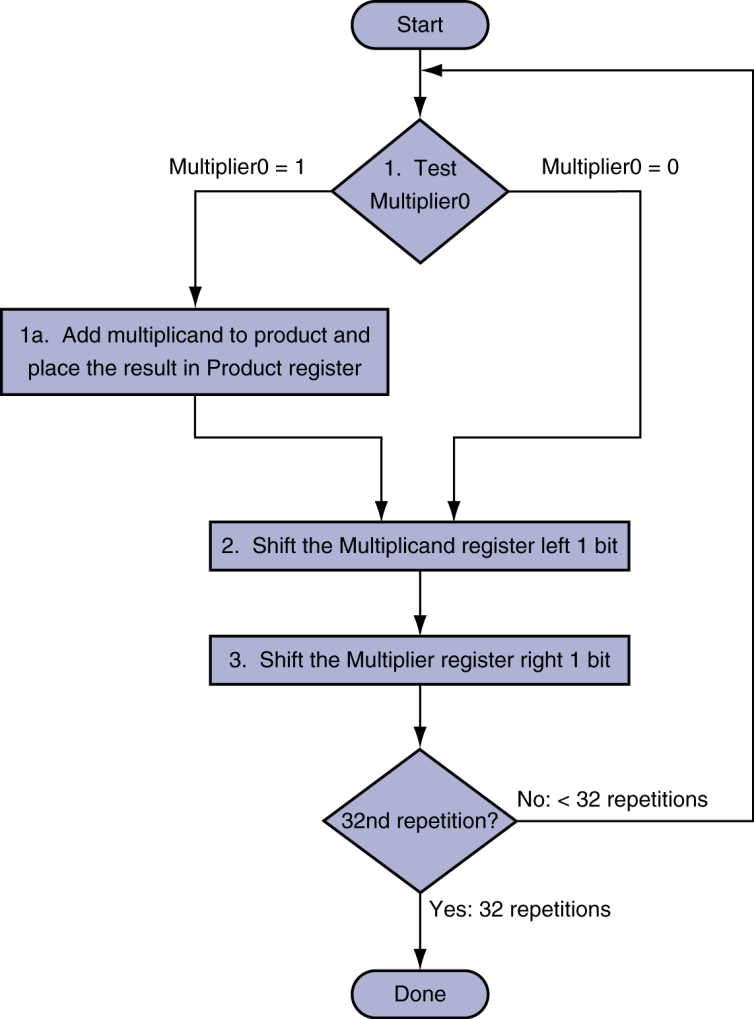


图1 忽略溢出的乘法器设计流程图

**编写MIPS64代码程序**：

1. 编写数据段

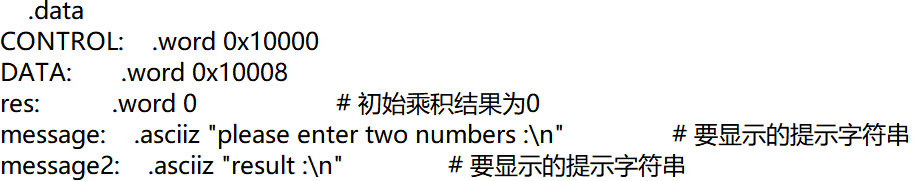


图2 数据段详情

CONTROL 和 DATA 是用来存储 IO 控制和数据寄存器地址的变量。

res 用于初始化乘法运算的乘积结果。

message 和 message2 是程序运行期间要显示的提示信息字符串。

1. 输出提示信息

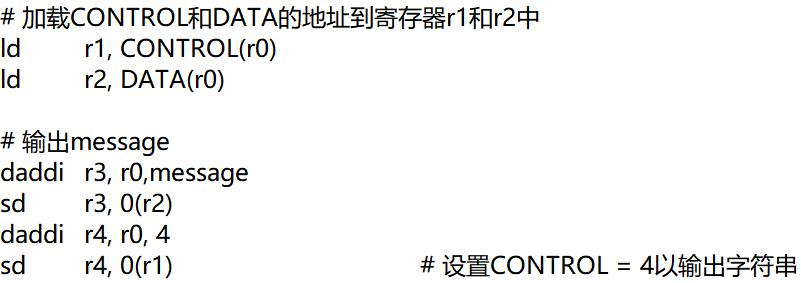


图3 输出提示信息的代码

首先，将 CONTROL 地址加载到 r1 寄存器中，将 DATA 地址加载到 r2 中，以便后续访问和控制输出、输入。然后，将 message 地址存储到 DATA 中并设置 CONTROL 为 4，触发输出操作，显示提示信息 "please enter two numbers :\n"。

1. 读取两个整数

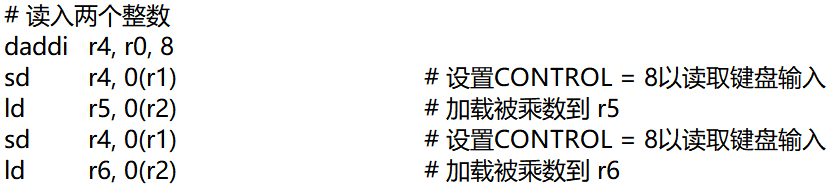


图4 输入两个乘数的代码

通过设置 CONTROL 为 8，触发从键盘输入两个整数。第一个数加载到 r5（被乘数），第二个数加载到 r6（乘数）。

1. 按位乘法运算（乘法器核心逻辑）

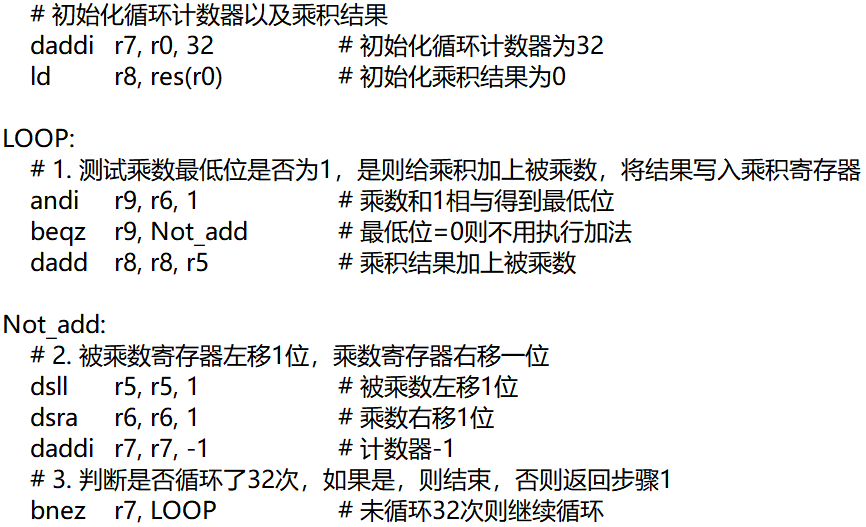


图5 乘法器核心逻辑代码

乘法器的核心逻辑通常基于移位加法法，这种方法是通过逐位检测乘数的每一位是否为 1 来决定是否将被乘数累加到乘积上，同时通过移位来处理每一位。以下是详细的核心逻辑描述：

1. **初始化：**

设置循环计数器，通常等于乘数的位数（32位），表示需要执行多少次移位。**被乘数**保存在一个寄存器中（**r5**），**乘数**保存在另一个寄存器中（**r6**），**乘积结果**初始为 0（存放在**r8**）。

1. **按位处理乘法：**

每次循环时，检查乘数的**最低位**（即 **r6 & 1** 判断最低位是否为 1）。如果最低位为 1，则将被乘数**累加**到当前的乘积结果中（累加到 **r8**），因为这代表当前位对应的被乘数应该参与累加。

1. **移位操作：**

被乘数**左移一位**，准备在下一次累加时使用。乘数**右移一位**（相当于将最低位移出），准备在下一次循环中检测下一个最低位。每次循环后，**循环计数器减1**，直到计数器为0为止。

1. **结束条件：**

当循环计数器为 0 时，表示所有位都已处理，乘积结果即为最终结果。

1. 输出结果提示和乘积

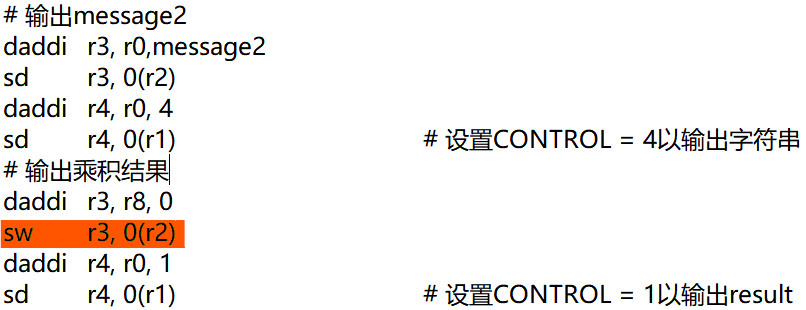


图6 输出结果提示和乘积结果的代码

输出 message2 字符串，提示结果输出。将乘积结果加载到 r3 并写入 DATA，将 CONTROL 设置为1，触发输出结果。注意：在存乘积结果进入DATA时，需要使用32位存储指令，否则高32位不会溢出。

**对编写的代码进行语法测试**：如图7所示，结果显示代码语法正确。

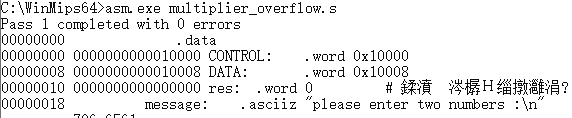


图7 语法测试结果

**运行结果**：

运行显示运行结果的例子如下，由于我们这里展示的是忽略了溢出的乘法，所以结果有两种：1、小于32位；2、大于32位。

第一种情况截图：左图为测试结果，右图为正确结果。可见测试结果正确。

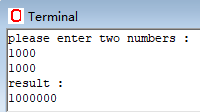
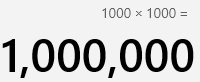
 

图8 小于32位的测试结果

第二种情况截图：左图为测试结果，右图为正确结果。可见测试结果溢出了。

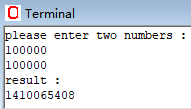
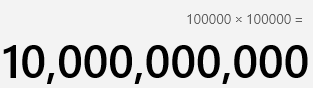
 

图9 大于32位的测试结果

根据上面的程序代码和截图，我们可以很清楚的看出，当结果小于32位时，结果正常；当结果大于32位时，结果只截取了低32位的结果，而高32位的结果直接忽略掉了。

2、溢出提示的乘法器

上述的程序，用加法实现了32位乘法，但是，其中，对溢出情况没有进行考虑是其中的弊端。这里，我们来完善上述的乘法器，使得该乘法器会在结果溢出时候提示。

其实，这个小优化是十分简单的，只需要对64位的寄存器中的高32位进行检测即可。当高32位为0时，说明结果没有溢出，否则，结果溢出。

为了优化代码并检测64位寄存器中的高32位是否为零（从而判断结果是否溢出），可以通过将寄存器内容右移32位并与零进行比较来实现。下面是优化后的代码逻辑描述及实现：

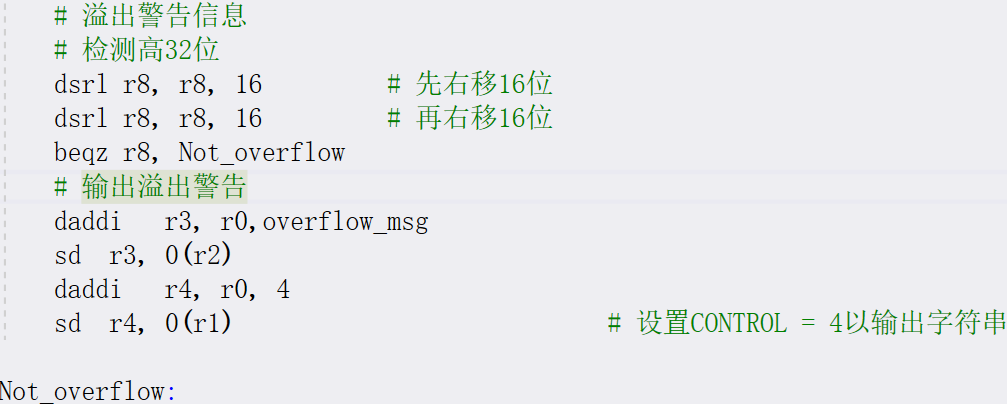


图10 新增溢出提示的代码

**代码说明**：

1. 将64位寄存器的内容分两次右移16位。因为MIPS64指令最多只能移动31位。
2. 检查右移后的结果（高32位）是否为0。
3. 如果为0，则结果没有溢出；否则，结果溢出。

**语法测试：**对编写的代码进行语法测试，如图11所示，结果显示代码语法正确：

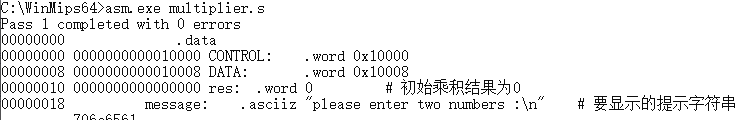


图11 代码语法检测结果

上述代码运行结果也有两个，一个是没有溢出的情况下的结果，一个是溢出了的情况下的结果。

首先，我们看没有溢出的情况结果：左图为测试结果，右图为正确结果。

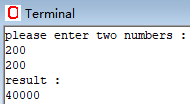
 

图12 无溢出结果(2)

结果正确，其次，我们看溢出的情况结果如何：左图为测试结果，右图为正确结果。

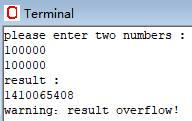
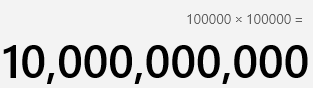
 

图13 有溢出结果(2)

可以看到，当结果溢出时，程序会给出提示“warning：result overflow！”。

3、结束语

本实验介绍了通过加法器来设计乘法器的原理，并且在编写该实验程序的时候，我们更加了解了：1、计算机乘法器工作原理的内容；2、进一步熟练MIPS的编程方法；3、WinMIPS64的使用方法。当然，如果想要更加深入的学习，我们也可以课外继续编写对除法的模拟。Perf软件的使用让学生初步熟悉性能测评的主要工具。

**五、实验结果**

1. 成功利用加法器设计出忽略溢出的乘法器，对其进行测试，无溢出和有溢出两种测试结果如图14，图15所示。

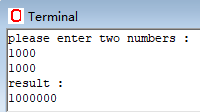
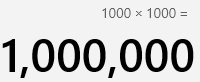
 

图14 无溢出结果（1）

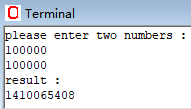
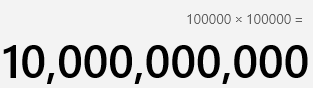
 

图15 有溢出结果（1）

1. 在忽略溢出乘法器的基础上，优化代码，增添提示信息。设计出有溢出提示的乘法器。对其进行测试，无溢出和有溢出两种测试结果如图16，图17所示。

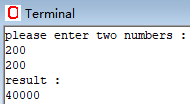
 

图16 无溢出结果(2)

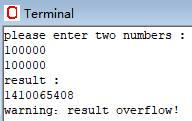
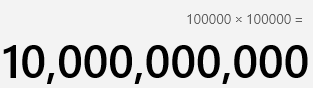
 

图17 有溢出结果(2)

**六、实验总结与体会**

**实验挑战：**

挑战一：如何使用键盘输入两个乘数？

解决一：回顾实验一的IO操作可知，设置CONTROL = 8可将键盘输入的整数写到DATA中。

挑战二：64位的寄存器如何实现忽略溢出的乘法器？

解决二：如果使用64位指令，乘法器是不会发生高32位溢出的。于是，需要在将结果输出的过程中，使用32位指令，忽略高32位的数值，从而发生溢出现象。

挑战三：MIPS64指令集的移位指令最多移动31位，如何移动32位？

解决三：分两次进行移动，如：每次移动16位。

**实验总结：**

1. 本次实验增强了我对WinMIPS64软件的基本功能、作用及其各窗口结构与功能的认识。
2. 我更熟练地掌握了利用slt, slti, beq, bne和$zero创建比较条件，并结合跳转指令j和jal实现循环结构的技巧。例如，在设计乘法器时，我使用了beq指令来判断乘数是否为零，从而决定是否继续循环或结束乘法操作。通过这些指令，我能够创建出精确控制程序流程的条件，这对于实现复杂的算法逻辑至关重要。
3. 通过学习计算机乘法器的工作原理，我利用MIPS指令和加法器成功设计了乘法器，并对乘法溢出判断有了新见解。我通过按位与操作and来判断乘数的最后一位是否为1，如果是，则将被乘数累加到乘积中。
4. 实验加深了我对左移（sll）和右移（srl）操作的掌握。在乘法器的设计中，我频繁地使用这些指令来实现乘数和被乘数的位移操作，这对于理解位级操作和数据表示非常有帮助。
5. 在代码编写过程中遇到的格式和指令错误，加深了我对MIPS64指令的理解和应用能力。例如：在编写乘法器程序时，使用asm.exe检查语法错误发生了报错，检查后才发现是在定义数据段的时候缺少了“：”，导致语法错误。并且，我还发现MIPS64指令没有32位的add指令。
6. 实验中，我学会了通过按位与操作判断一个数的最后一位是否为1，这一技巧启发了我，认识到位运算在提升程序效率方面的巨大潜力。

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字： 刘刚  2024年10月 日 |
| 备注： |