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：计算机与软件学院所有专业

指 导 教 师：刘刚

报告人：　学号：　班级：

实 验 时 间：2025 年 10 月 15 日~10 月 22 日

实验报告提交时间：2025 年 10 月 21 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

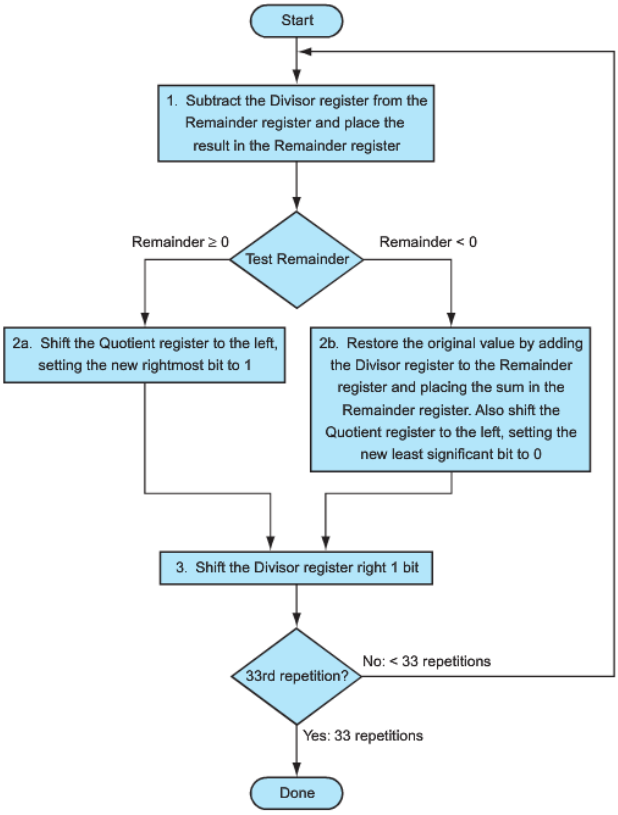
本次试验分为两个部分：第一部分、用加法器设计一个不考虑溢出的乘法器；第二部分、用加法器设计一个考虑溢出的乘法器（编程熟练的同学，也可以用除法器、浮点加法器等替代）。

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。

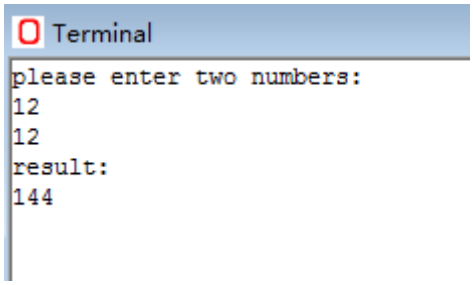
总共分为 4 步：

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。

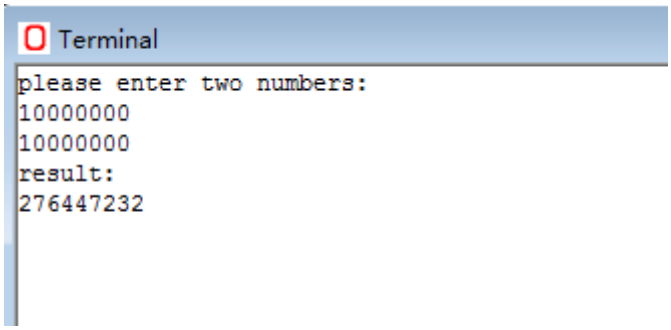


运行显示运行结果的例子如下，由于我们这里展示的是忽略了溢出的乘法，所以结果有两种：1、小于 32 位；2、大于 32 位。

第一种情况截图：



第二种情况截图：



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。

1.编写C++伪代码：

利用位运算实现两个数的乘法运算。具体实现逻辑是将第一个数分解为二进制的每一位，如果该位为1，则将第二个数加到结果上，然后第一个数右移一位（相当于除以2），第二个数左移一位（相当于乘以2），循环32次（因为32位整数），最后得到乘法的结果。

```
int main() {
    for (i = 32; i; x1>>= 1, x2 <<= 1, i--)
    if (x1 & 1) res += x2;
}
```

2.WinMIPS代码：

①设置栈指针以及加载控制变量和数据变量。输出提示字符串。读取第一个输入数字，并存储在内存中。读取第二个输入数字，并存储在内存中。执行一个简单的乘法运算，将两个输入数字相乘。输出乘法结果的字符串。输出乘法的结果。

```
.text
main:
 daddi $sp, $zero, STACK_SIZE
 #设置控制变量和DATA
 lw $a2, DATA($zero)
 lw $a3, CONTROL($zero)
 #读入一开始提示的字符串，并输出
 daddi $a1, $zero, str1
 jal soutStr
 #获取输入的第一个数字
 daddi $a1, $zero, x1
 daddi $t1, $zero, 8
 sw $t1, ($a3)
 lw $t2, ($a2)
 sw $t2, ($a1)
 #获取输入的第二个数字
 daddi $a1, $zero, x2
 daddi $t1, $zero, 8
```

```
 sw $t1, ($a3)
 lw $t2, ($a2)
 sw $t2, ($a1)


#实现两个数的乘法运算
#t2和t3代表两个数
#t1代表计数器
#t4代表最终结果
 daddi $t1, $zero, 32
 lw $t2, x1($zero)
 lw $t3, x2($zero)
loop:
 beq $t1, $zero, over
 andi $t4, $t2, 1
 beq $t4, $zero, skipAdd
 dadd $t5, $t5, $t3
skipAdd:
 dsrl $t2, $t2, 1
 dsll $t3, $t3, 1
daddi $t1, $t1, -1
 j loop
over:
 # 输出 str2
 daddi $a1, $zero, str2
 jal soutStr
 # 输出 res
 daddi $a1, $t5, 0
 sw $a1, ($a2)
```

```
 daddi $t1, $zero, 2
 sw $t1, ($a3)
     halt
```

② 使用x1 和 x2保存两个数字 ，str1，str2，str3保存输出的字符串。

```
.data
DATA: .word 0x10008
CONTROL: .word 0x10000
STACK_SIZE: .space 20
STACK_POINTER: .word 0
x1: .word 0
x2: .word 0
str1: .asciiz "please enter two numbers:\n"
str2: .asciiz "result:\n"
 str3: .asciiz "warning: result overflow\n"
```

③由于要多次输出字符串，这里提取方法输出字符串

```
#用于输出字符串
soutStr:
 daddi $sp, $sp, -4
 sw $ra, ($sp)
 #a1保存了字符串，将a1保存到a2既DATA里面，然后设置控制变量，输出字符串
 sw $a1, ($a2)
 daddi $t1, $zero, 4
 sw $t1, ($a3)

 lw $ra, ($sp)
 daddi $sp, $sp, 4
 jr $ra
```

3.运行检查

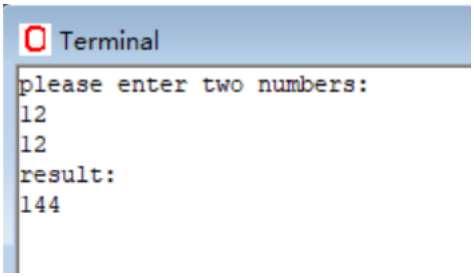
①检查语法

```
E:\WinMIPS64>asm.exe Mul0.s
Pass 1 completed with 0 errors
00000000           .data
00000000 0000000000010000 CONTROL:  .word 0x10000
00000008 0000000000010008 DATA:  .word 0x10008
```

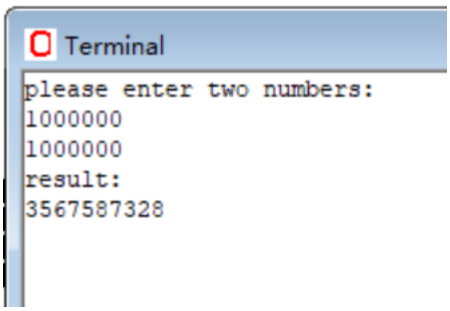
```
Pass 2 completed with 0 errors
Code Symbol Table
                main = 00000000
                loop = 00000030
              notAdd = 00000040
             endLoop = 00000050
         writeString = 00000064
             readInt = 00000084
            writeInt = 000000a8
Data Symbol Table
             CONTROL = 00000000
                DATA = 00000008
                NUM1 = 00000010
                NUM2 = 00000018
          STACK_SIZE = 00000020
       STACK_POINTER = 00000038
        INPUT_STRING = 00000040
       RESULT_STRING = 00000060
      WARNING_STRING = 00000070
```

②运行截图

不发生溢出：



发生溢出：



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。

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。

1.C++伪代码：

```
int main() {
    for (i = 32; i; x1>>= 1, x2 <<= 1, i--)
    if (x1 & 1) res += x2;
    ans >>= 32;
    if (ans) 溢出
}
```

2.新增溢出检测代码：

```
over:
    # 输出 str2
    daddi $a1, $zero, str2
    jal soutStr
    # 输出 res
    daddi $a1, $t5, 0
    sw $a1, ($a2)
    daddi $t1, $zero, 2
    sw $t1, ($a3)
    #------------做检测------------
    dsrl $t5, $t5, 16
    dsrl $t5, $t5, 16
    beq $t5, $zero, end
    #------------做检测------------
```

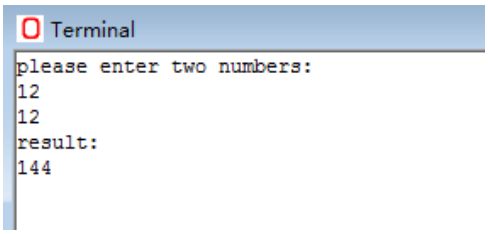
```
# 输出溢出提示
daddi $a1, $zero, str3
jal soutStr

end:
    halt
```

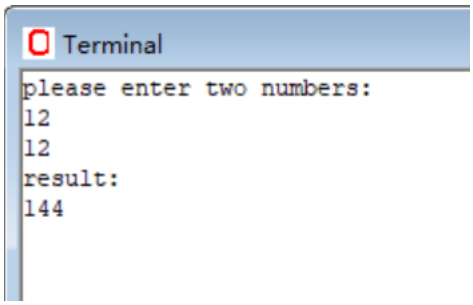
3.运行结果
①语法检查

```
E:\WinMIPS64>asm.exe Mul1.s
Pass 1 completed with 0 errors
00000000           .data
00000000 0000000000010000 CONTROL: .word 0x10000
00000008 0000000000010008 DATA: .word 0x10008
```

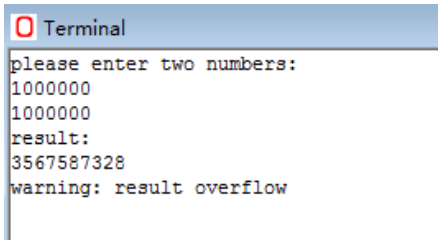
```
Pass 2 completed with 0 errors
Code Symbol Table
                main = 00000000
                loop = 00000030
              notAdd = 00000040
             endLoop = 00000050
                halt = 00000074
         writeString = 00000078
             readInt = 00000098
            writeInt = 000000bc
Data Symbol Table
             CONTROL = 00000000
                DATA = 00000008
                NUM1 = 00000010
                NUM2 = 00000018
          STACK_SIZE = 00000020
       STACK_POINTER = 00000038
        INPUT_STRING = 00000040
       RESULT_STRING = 00000060
      WARNING_STRING = 00000070
```

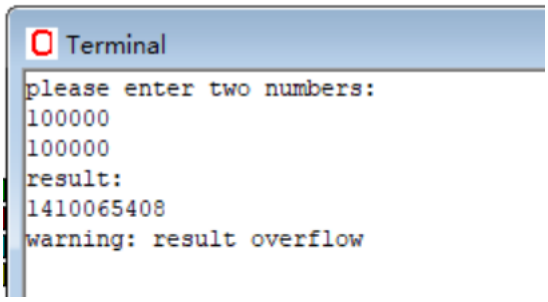
②运行截图
首先，我们看没有溢出的情况结果：





结果正确，其次，我们看溢出的情况结果如何：





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

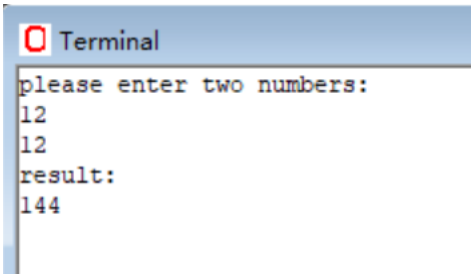
4 结束语

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。

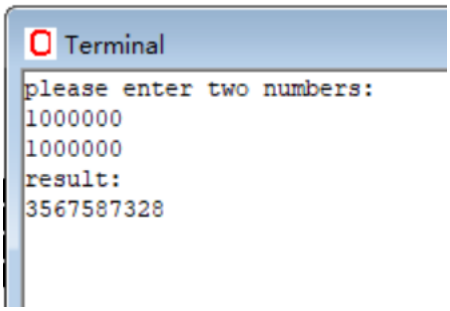
五、实验结果

1. 无溢出检测：

不发生溢出：

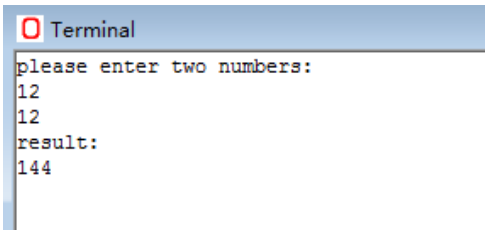


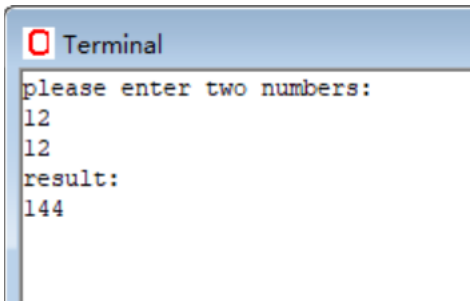
发生溢出：



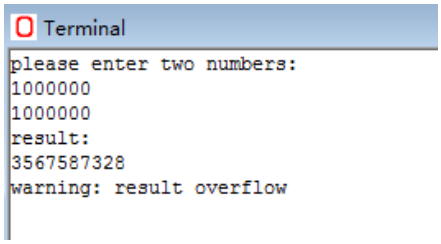
2. 有溢出检测

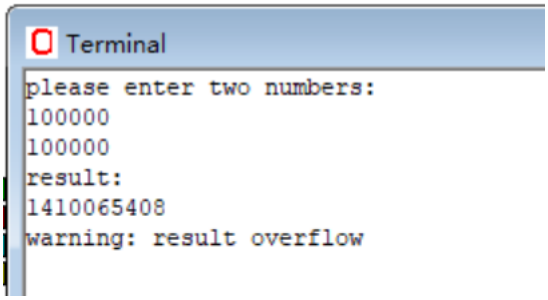
不发生溢出：





发生溢出：





五、实验总结与体会

在这个 MIPS64 乘法器模拟实验中，通过实际操作 WinMIPS64 仿真器，我更加深入地了解了 MIPS 程序的语法和计算机中乘法的实现。在设计不考虑溢出的乘法器和考虑溢出的乘法器时，我逐步了解了乘法器如何由加法器设计得到，并通过实验步骤和代码实现了这一过程。

通过实验，我掌握了在 MIPS64 架构下实现乘法运算的基本原理，以及如何实现乘法器的设计和优化。同时，通过调试代码和观察运行结果，我进一步加深了对计算机中乘法运算的理解，包括加法与乘法之间的关系以及溢出情况的处理。

这次实验不仅让我熟悉了 MIPS64 指令集和程序设计的过程，还提高了我的逻辑思维能力和对计算机系统的整体认识。通过实践操作，我对计算机系统的底层运行机制有了更深入的了解，为我未来的学习和研究打下了坚实的基础。

指导教师批阅意见： **成绩评定：** 指导教师签字： 刘刚 2025 年 10 月 日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。