

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/软件工程

指导教师: 王 毅

报告人: 黄亮铭 学号: 2022155028 班级: 腾班

实 验 时 间: 2024 年 10 月 11 日

实验报告提交时间: 2024 年 10 月 25 日

一、实验目标：

1. 实际运用 WinMIPS64 进行实验，以期更了解 WinMIPS64 的操作；
2. 更加深入地了解 MIPS 程序的语法；
3. 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

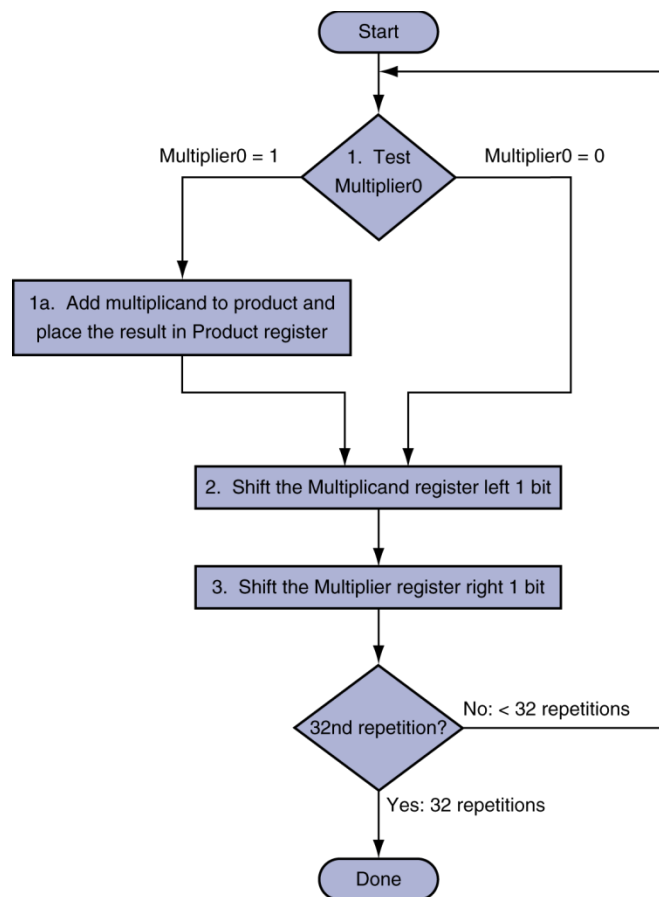
本次试验分为两个部分：第一部分、用加法器设计一个不考虑溢出的乘法器；第二部分、用加法器设计一个考虑溢出的乘法器（编程熟练的同学，也可以用除法器、浮点加法器等替代）。

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：

```

Terminal
please enter two numbers:
12
12
result:
144
  
```

第二种情况截图：

```

Terminal
please enter two numbers:
10000000
10000000
result:
276447232
  
```

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。

1.1 设置初始信息

首先设置数据交互和控制信息在内存的位置，然后在内存开辟两个字的大小，用于存储用户输入的两个乘数，再设置栈的大小以及栈指针，最后设置提示语句。具体代码如下图所示。

```
.data
DATA: .word 0x10008
CONTROL: .word 0x10000
NUM1: .word 0
NUM2: .word 0
STACK_SIZE: .space 300
STACK_POINTER: .word 0
INPUT_PROMPT: .asciiz "please enter two numbers:\n"
RESULT_PROMPT: .asciiz "result:\n"
WARNING_PROMPT: .asciiz "warning: result overflow\n"
```

图 1：初始信息

1.2 输出函数

输出函数一共有两个，一个负责输出字符串，另一个负责输出整数，两者分别命名为 writeString 和 writeInt。

1.2.1 writeString

实现的步骤如下所示。

1. 将寄存器中的内容保存到栈帧。
2. 将 a0 寄存器中相应字符串的地址传送到内存中数据交互的位置。
3. 设置控制信息为输出字符串。
4. 将栈帧中保存的内容恢复到寄存器。

具体代码如图 2 所示。

```
writeString:
    daddi $sp, $sp, -4
    sw $ra, ($sp)

    sw $a0, ($a1)
    daddi $t0, $zero, 4
    sw $t0, ($a2)

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

图 2：writeString 函数

1.2.2 writeInt

该函数的实现步骤与 writeString 函数的实现步骤类似。主要的区别为步骤 2 时会将需要输出的整数直接传送到数据交互的位置，而非传送地址。另一个不同之处为步骤 3 的控制信息更改为输出一个字。具体实现代码如图 3 所示。

```
writeInt:
    daddi $sp, $sp, -4
    sw $ra, ($sp)

    sw $a0, ($a1)
    daddi $t0, $zero, 2
    sw $t0, ($a2)

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

图 3: writeInt 函数

1.3 输入函数

输入函数只有一个，负责读入整数到设置初始信息中开辟的存放乘数的位置。此函数命名为 readInt。

1.3.1 readInt

实现的步骤如下。

1. 将寄存器中的内容保存到栈帧中。
2. 设置控制信息为读入一个字。
3. 获取数据交互位置的内容。
4. 将步骤 3 获取到的内容存储到相应的位置。
5. 将栈帧中保存的内容恢复到寄存器中。

具体实现代码如图 4 所示。

```
readInt:
    daddi $sp, $sp, -4
    sw $ra, ($sp)

    daddi $t0, $zero, 8
    sw $t0, ($a2)

    lw $t1, ($a1)
    sw $t1, ($a0)

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

图 4: readInt 函数

1.4 主函数 main

代码实现的具体步骤如下所示。

1. 初始化：将数据交互和控制信息在内存中的地址存储到寄存器中，同时初始化栈指针 `sp`。
2. 输出提示词，提示用户输入两个乘数。
3. 读取用户输入的乘数。
4. 从内存中加载用户输入的乘数到临时寄存器中。
5. 根据实验报告给出的乘法器实现步骤实现两个数相乘。
6. 输出提示词同时输出结果。
7. 中断程序。

具体实现代码如下图所示。

```
.text
main:
    lw $a1, DATA($zero)
    lw $a2, CONTROL($zero)
    daddi $sp, $zero, STACK_SIZE

    daddi $a0, $zero, INPUT_PROMPT
    jal writeString

    daddi $a0, $zero, NUM1
    jal readInt
    daddi $a0, $zero, NUM2
    jal readInt

    daddi $t0, $zero, 32
    lw $t1, NUM1($zero)
    lw $t2, NUM2($zero)
loop:
    beq $t0, $zero, end
    andi $t3, $t1, 1
    beq $t3, $zero, noAdd
    dadd $t4, $t4, $t2
noAdd:
    dsrl $t1, $t1, 1
    dsll $t2, $t2, 1
    daddi $t0, $t0, -1
    j loop
end:

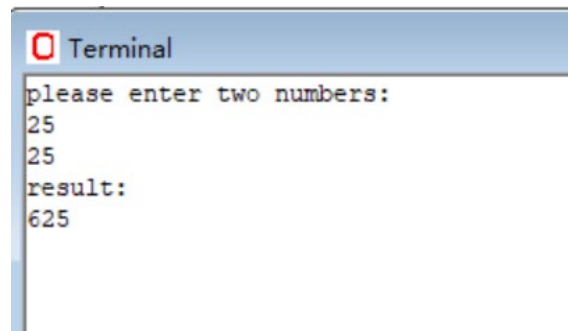
    daddi $a0, $zero, RESULT_PROMPT
    jal writeString
    daddi $a0, $t4, 0
    jal writeInt

    halt
```

图 5: main 函数

1.5 结果展示

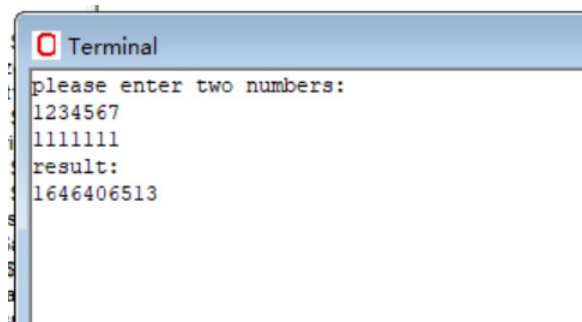
第一种情况：结果小于 32 位的截图。



```
Terminal
please enter two numbers:
25
25
result:
625
```

图 6：没有溢出

第二种情况：结果大于 32 位的截图。可以看到，程序只显示了结果的低 32 位，高 32 位直接被舍弃。



```
Terminal
please enter two numbers:
1234567
1111111
result:
1646406513
```

图 7：存在溢出

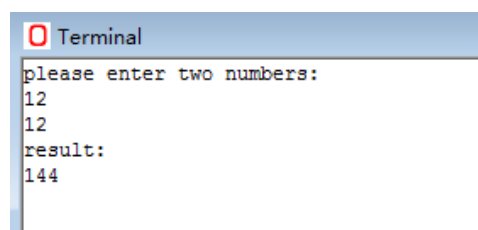
2、溢出提示的乘法器

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。

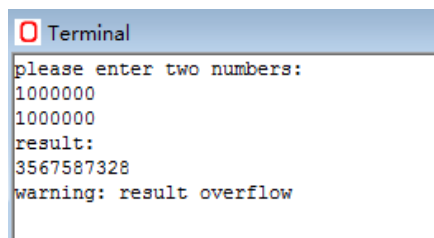
上述代码运行结果也有两个，一个是没有溢出的情况下的结果，一个是溢出了的情况下的结果。

首先，我们看没有溢出的情况结果：



```
Terminal
please enter two numbers:
12
12
result:
144
```

结果正确，其次，我们看溢出的情况结果如何：

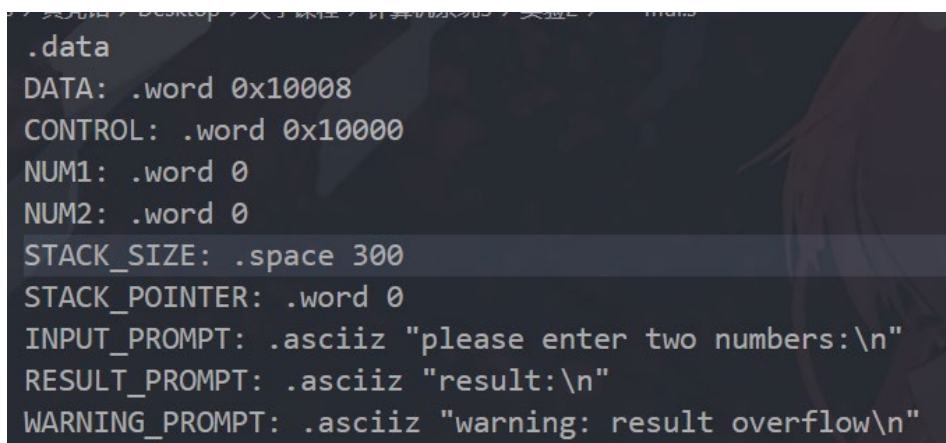


```
Terminal
please enter two numbers:
1000000
1000000
result:
3567587328
warning: result overflow
```

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

2.1 设置初始信息

首先设置数据交互和控制信息在内存的位置，然后在内存开辟两个字的大小，用于存储用户输入的两个乘数，再设置栈的大小以及栈指针，最后设置提示语句。具体代码如下图所示。



```
.data
DATA: .word 0x10008
CONTROL: .word 0x10000
NUM1: .word 0
NUM2: .word 0
STACK_SIZE: .space 300
STACK_POINTER: .word 0
INPUT_PROMPT: .ascii "please enter two numbers:\n"
RESULT_PROMPT: .ascii "result:\n"
WARNING_PROMPT: .ascii "warning: result overflow\n"
```

图 8：初始信息

2.2 输出函数

输出函数一共有两个，一个负责输出字符串，另一个负责输出整数，两者分别命名为 writeString 和 writeInt。

2.2.1 writeString

实现的步骤如下所示。

1. 将寄存器中的内容保存到栈帧。
2. 将 a0 寄存器中相应字符串的地址传送到内存中数据交互的位置。
3. 设置控制信息为输出字符串。
4. 将栈帧中保存的内容恢复到寄存器。

具体代码如图 9 所示。


```

writeString:
    daddi $sp, $sp, -4
    sw $ra, ($sp)

    sw $a0, ($a1)
    daddi $t0, $zero, 4
    sw $t0, ($a2)

    lw $ra, ($sp)
    daddi $sp, $sp, 4
    jr $ra # return

```

图 9: writeString 函数

2.2.2 writeInt

该函数的实现步骤与 writeString 函数的实现步骤类似。主要的区别为步骤 2 时会将需要输出的整数直接传送到数据交互的位置，而非传送地址。另一个不同之处为步骤 3 的控制信息更改为输出一个字。具体实现代码如图 10 所示。

```

writeInt:
    daddi $sp, $sp, -4
    sw $ra, ($sp)

    sw $a0, ($a1)
    daddi $t0, $zero, 2
    sw $t0, ($a2)

    lw $ra, ($sp)
    daddi $sp, $sp, 4
    jr $ra # return

```

图 10: writeInt 函数

2.3 输入函数

输入函数只有一个，负责读入整数到设置初始信息中开辟的存放乘数的位置。此函数命名为 readInt。

2.3.1 readInt

实现的步骤如下。

1. 将寄存器中的内容保存到栈帧中。
2. 设置控制信息为读入一个字。
3. 获取数据交互位置的内容。
4. 将步骤 3 获取到的内容存储到相应的位置。

5. 将栈帧中保存的内容恢复到寄存器中。

具体实现代码如图 11 所示。

```
readInt:
    daddi $sp, $sp, -4
    sw $ra, ($sp)

    daddi $t0, $zero, 8
    sw $t0, ($a2)

    lw $t1, ($a1)
    sw $t1, ($a0)

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

图 11: readInt 函数

2.4 主函数 main

代码实现的具体步骤如下所示。

1. 初始化：将数据交互和控制信息在内存中的地址存储到寄存器中，同时初始化栈指针 `sp`。

```
lw $a1, DATA($zero)
lw $a2, CONTROL($zero)
daddi $sp, $zero, STACK_SIZE
```

图 12: 初始化

2. 输出提示词，提示用户输入两个乘数。

```
daddi $a0, $zero, INPUT_PROMPT
jal writeString
```

图 13: 提示词输出

3. 读取用户输入的乘数。

```
daddi $a0, $zero, NUM1
jal readInt
daddi $a0, $zero, NUM2
jal readInt
```

图 14: 读取数据

4. 从内存中加载用户输入的乘数到临时寄存器中。

```
daddi $t0, $zero, 32
lw $t1, NUM1($zero)
lw $t2, NUM2($zero)
```

图 15: 加载数据

5. 根据实验报告给出的乘法器实现步骤实现两个数相乘。

```
loop:
    beq $t0, $zero, end
    andi $t3, $t1, 1
    beq $t3, $zero, noAdd
    dadd $t4, $t4, $t2
noAdd:
    dsrl $t1, $t1, 1
    dsll $t2, $t2, 1
    daddi $t0, $t0, -1
    j loop
end:
```

图 16: 乘法器实现

6. 输出提示词同时输出结果。

```
daddi $a0, $zero, RESULT_PROMPT
jal writeString
daddi $a0, $t4, 0
jal writeInt
```

图 17: 提示词输出

7. 将结果右移 32 位，检查结果是否为 0，如不为 0，则发生溢出；如为 0，则没有溢出。

```
dsrl $t4, $t4, 31
dsrl $t4, $t4, 1
beq $t4, $zero, return
daddi $a0, $zero, WARNING_PROMPT
jal writeString
```

图 18: 检查溢出

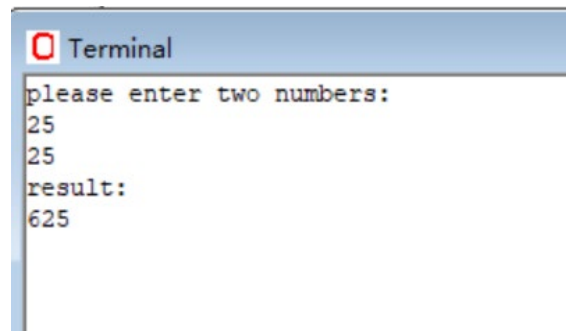
8. 中断程序。

```
return:
    halt
```

图 19: 中断程序

2.5 结果展示

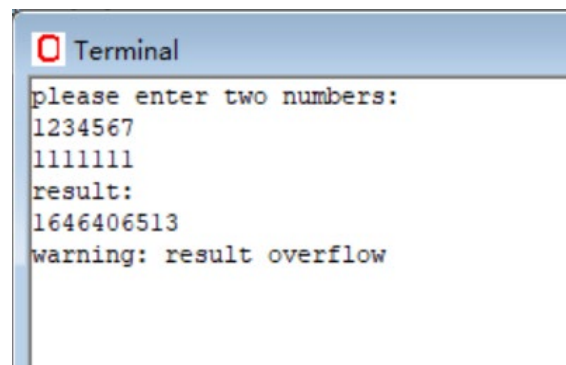
第一种情况：结果小于 32 位的截图。



```
Terminal
please enter two numbers:
25
25
result:
625
```

图 20：没有溢出

第二种情况：结果大于 32 位的截图。可以看到，程序只显示了结果的低 32 位，高 32 位直接被舍弃，但是会给出提示，提醒用户存在数据溢出情况。



```
Terminal
please enter two numbers:
1234567
1111111
result:
1646406513
warning: result overflow
```

图 21：存在溢出

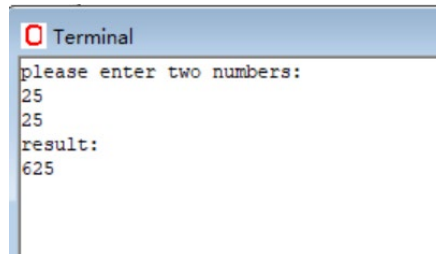
4 结束语

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。

五、实验结果

5.1 没有溢出处理的乘法器的运行结果

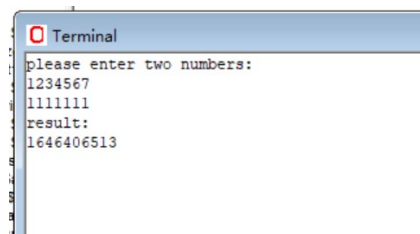
第一种情况：结果小于 32 位的截图。



```
Terminal
please enter two numbers:
25
25
result:
625
```

图 22：没有溢出

第二种情况：结果大于 32 位的截图。可以看到，程序只显示了结果的低 32 位，高 32 位直接被舍弃。

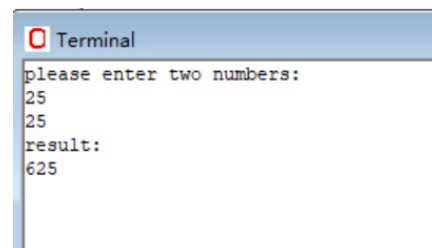


```
Terminal
please enter two numbers:
1234567
1111111
result:
1646406513
```

图 23：存在溢出

5.2 拥有溢出处理的乘法器的运行结果

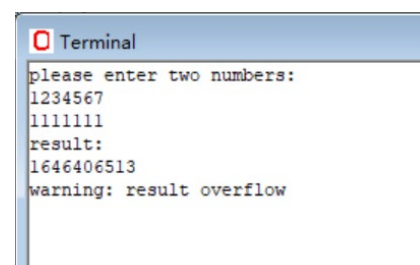
第一种情况：结果小于 32 位的截图。



```
Terminal
please enter two numbers:
25
25
result:
625
```

图 24：没有溢出

第二种情况：结果大于 32 位的截图。可以看到，程序只显示了结果的低 32 位，高 32 位直接被舍弃，但是会给出提示，提醒用户存在数据溢出情况。



```
Terminal
please enter two numbers:
1234567
1111111
result:
1646406513
warning: result overflow
```

图 25：存在溢出

六、实验总结与体会

1. 本次实验的主要内容为在 WinMIPS64 上使用加法模拟乘法，实现两种乘法器，一种为不检查溢出，另一种为检查溢出。检查溢出的乘法器能让用户更加容易排查溢出导致的无法达到预期结果的问题。
2. 通过本次 MIPS64 乘法器模拟实验，我对计算机中乘法的实现有了更深入的理解。
3. 本次实验我实现了一个 32 位乘法器。这一过程不仅加深了我对 MIPS 程序语法的理解，也让我对计算机硬件中的乘法器工作原理有了更加直观的认识。
4. 在实验过程中，我遇到了一些 bug，通过排查，发现都是一些很小的错误，例如 daddi 少写了一个 i，导致程序报错。这些错误让我更加意识到了细节在编程中的重要性。

指导教师批阅意见：

成绩评定：

指导教师签字： **王毅**

2024 年 11 月 1 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。