**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： 廖好**

**报告人： 沈晨玙 学号： 2019092121 班级： 19计科04**

**实 验 时 间： 2021年11月4日星期四**

**实验报告提交时间： 2021年11月4日星期四**

**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**

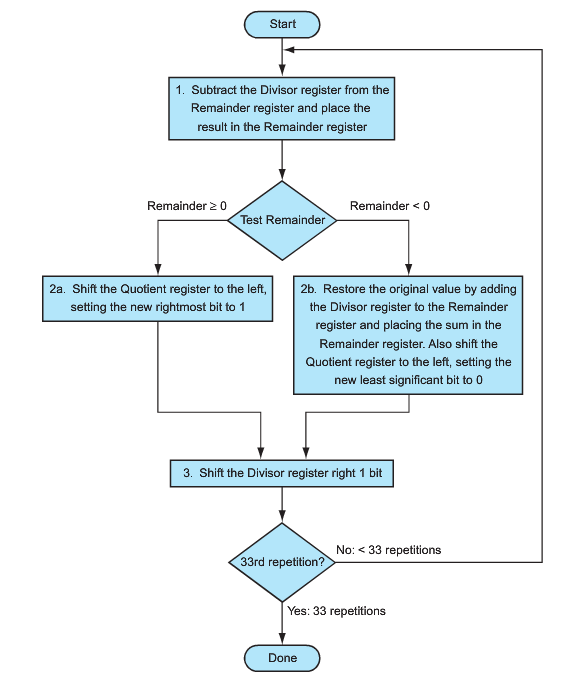
本次试验分为两个部分：第一部分、用加法器设计一个不考虑溢出的乘法器；第二部分、用加法器设计一个考虑溢出的乘法器（编程熟练的同学，也可以用除法器、浮点加法器等替代）。

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。

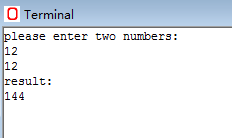
总共分为4步：

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。

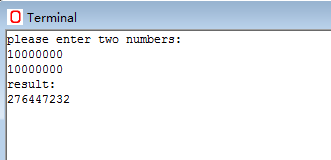


运行显示运行结果的例子如下，由于我们这里展示的是忽略了溢出的乘法，所以结果有两种：1、小于32位；2、大于32位。

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。

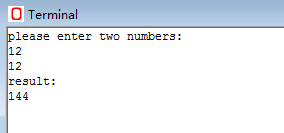
2、溢出提示的乘法器

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。

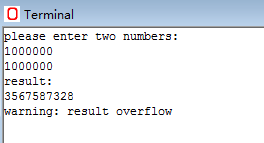
其实，这个小优化是十分简单的，只需要对64位的寄存器中的高32位进行检测即可。当高32位为0时，说明结果没有溢出，否则，结果溢出。

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：



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、忽略溢出的乘法器

首先编写.data 节的数据并且分配一些栈空间：

1. .data
2. CONTROL: .word 0x10000
3. DATA: .word 0x10008
4. NUM1: .word 0
5. NUM2: .word 0
6. STACKBOTTOM: .space 20
7. STACKTOP: .word 0
8. OFSTR: .asciiz "warning: result overflow\n"
9. STR: .asciiz "please enter two numbers:\n"
10. RES: .asciiz "result:\n"

然后编写一些帮助函数，分别是读取 int，打印 int 和打印字符串。

1. # @function readInt : 读入一个int
2. # @param arg0 : int的地址
3. # @return : ----
4. readInt:
5. daddi $sp, $sp, -4 # 创建栈
6. sw $ra, ($sp) # 保存返回地址
7. daddi $t0, $zero, 8 # t0 = 8
8. lw $t1, DATA($zero) # t1 = 0x10008
9. lw $t2, CONTROL($zero) # t2 = 0x10000
10. sw $t0, ($t2) # CONTROL = 8
11. lw $t1, ($t1) # t1 = DATA
12. sw $t1, ($a0) # M[arg0] = DATA
13. lw $ra, ($sp) # 恢复返回地址
14. daddi $sp, $sp, 4 # 退栈
15. jr $ra
16. # @function printInt :
17. # @param arg0 : 打印的int的地址
18. # @return : ----
19. printInt:
20. daddi $sp, $sp, -4 # sp -= 4
21. sw $ra, ($sp) # 保存返回地址
22. daddi $t0, $zero, 2 # t0 = 2
23. lw $t1, DATA($zero) # t1 = 0x10008
24. lw $t2, CONTROL($zero) # t2 = 0x10000
25. sw $a0, ($t1) # DATA = arg0
26. sw $t0, ($t2) # CONTROL = 2
27. lw $ra, ($sp) # 恢复返回地址
28. daddi $sp, $sp, 4 # sp += 4
29. jr $ra # return
30. # @function printStr :
31. # @param a0 : 字符串地址
32. # @return : ----
33. printStr:
34. daddi $sp, $sp, -4 # sp -= 4
35. sw $ra, ($sp) # 保存返回地址
36. daddi $t0, $zero, 4 # t0 = 4
37. dadd $t1, $zero, $a0 # t1 = arg0
38. lw $t2, DATA($zero) # t2 = M[DATA] = 0x10008
39. lw $t3, CONTROL($zero) # t3 = M[CONTROL] = 0x10000
40. sw $t1, ($t2) # M[0x10008] = t1 = arg0
41. sw $t0, ($t3) # M[0x10000] = t0 = 4
42. lw $ra, ($sp) # 恢复返回地址
43. daddi $sp, $sp, 4 # sp += 4
44. jr $ra # return

然后开始写初始化的代码，即.text 节中的代码。

首先打印提示字符串，然后从键盘 读取两个整数到 data 区：

1. .text
2. main:
3. daddi $sp, $zero, STACKTOP  # assign stack
5. daddi $a0, $zero, STR       # print str
6. jal printStr
8. daddi $a0, $zero, NUM1      # read num1
9. jal readInt
11. daddi $a0, $zero, NUM2      # read num2
12. jal readInt
14. daddi $a0, $zero, RES       # print res
15. jal printStr
17. daddi $s0, $zero, 32    # s0 = i
18. lw $s1, NUM1($zero)     # s1 = num1
19. lw $s2, NUM2($zero)     # s2 = num2

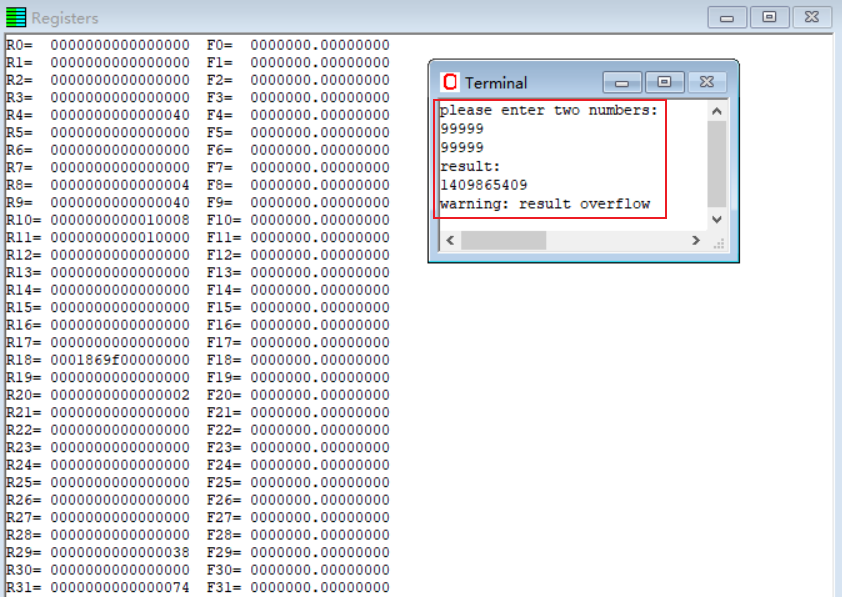
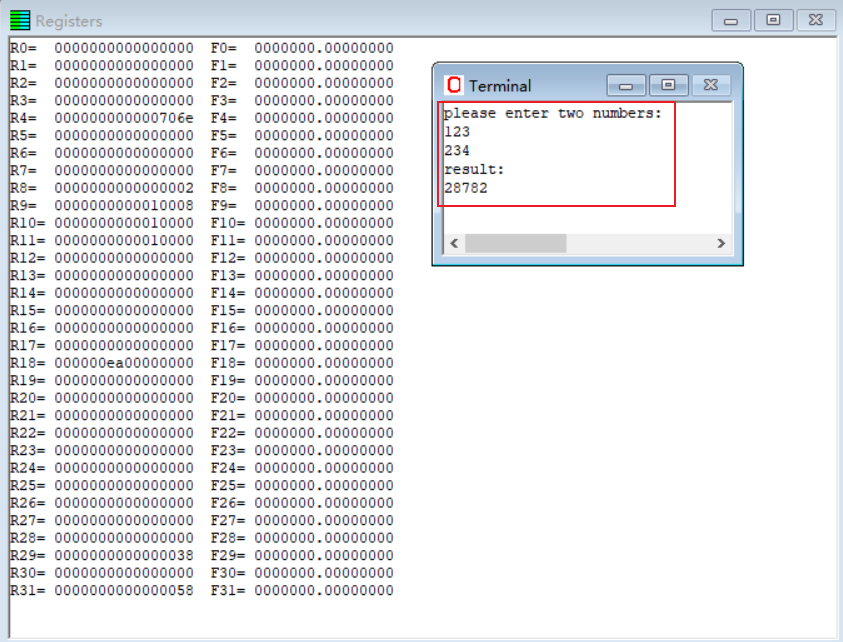
编写循环逻辑。

1. .text
2. main:
3. daddi $sp, $zero, STACKTOP # 创建栈
5. daddi $a0, $zero, STR # 打印输入提示
6. jal printStr
7. daddi $a0, $zero, NUM1 # 读入num1
8. jal readInt
9. daddi $a0, $zero, NUM2 # 读入num2
10. jal readInt
11. daddi $a0, $zero, RES # 打印结果提示
12. jal printStr
13. daddi $s0, $zero, 32 # s0 = i
14. lw $s1, NUM1($zero) # s1 = num1
15. lw $s2, NUM2($zero) # s2 = num2
16. loop:
17. beq $s0, $zero, end
18. daddi $s0, $s0, -1
19. andi $s3, $s1, 1 # s3 = 被乘数的最后一位
20. beq $s3, $zero, notadd # 最后一位 = 0 --> not add
21. dadd $s4, $s4, $s2 # ans += s2
22. notadd:
23. dsll $s2, $s2, 1 # s3<<1
24. dsrl $s1, $s1, 1 # s1>>1
26. j loop
27. end:
28. daddi $a0, $s4, 0 # 输出s4
29. jal printInt
30. halt:
31. halt

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。完善上述的乘法器，使得该乘法器会在结果溢出时候提示。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。在得出结果的时候，额外判断其高32位是否不为零。将s4右移32位，然后判断其是否为0即可。在 halt 结束之前，加上：

1. # 溢出检测
2. dsrl $s4, $s4, 8        # 高32位
3. dsrl $s4, $s4, 8
4. dsrl $s4, $s4, 8
5. dsrl $s4, $s4, 8
6. beq $s4, $zero, halt    # 高32位 = 0 --> halt
7. daddi $a0, $zero, OFSTR # 输出溢出警告
8. jal printStr
9. 结果展示



**五、实验总结与体会**

通过这一次实验，我对如何用mips的输入输出的控制，写法有更深刻的理解，对循环的写法有了更深的认识。对循环的认识和写法，掌握更深。乘法器的本质原理也有了认识。

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