

计算机组成原理实验一 实验报告

**学 院： 计 算 机 工 程 学 院**

**班 级： 计算2114**

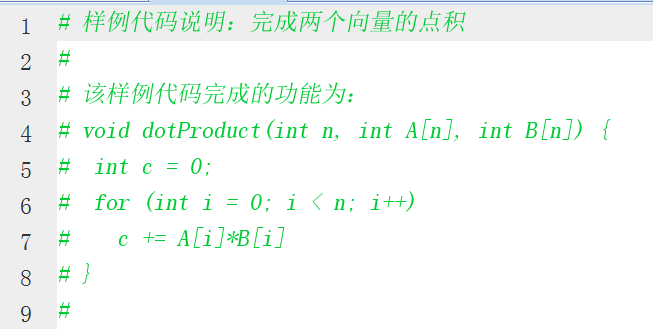
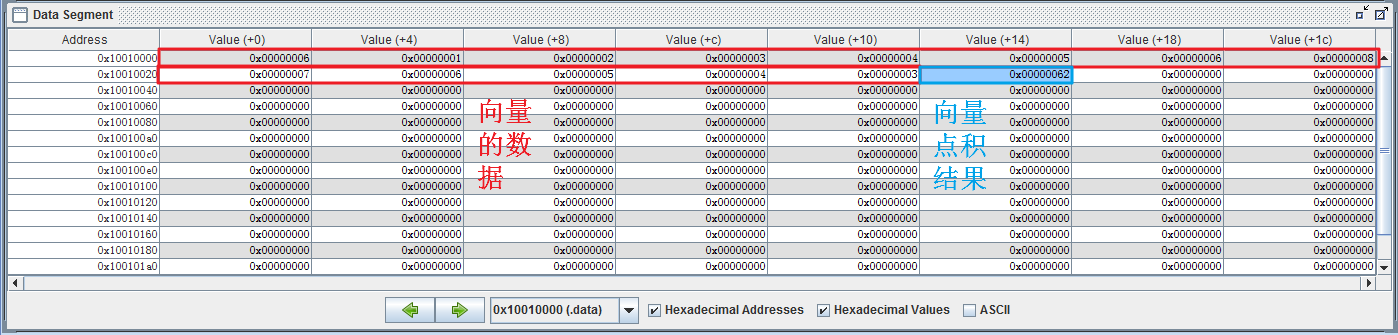
**姓 名: 庄佳强 章立早**

**学 号： 202121331104 202121331118**

二〇二二 年 9 月21 日

# 实验一.C

# 一 题目要求

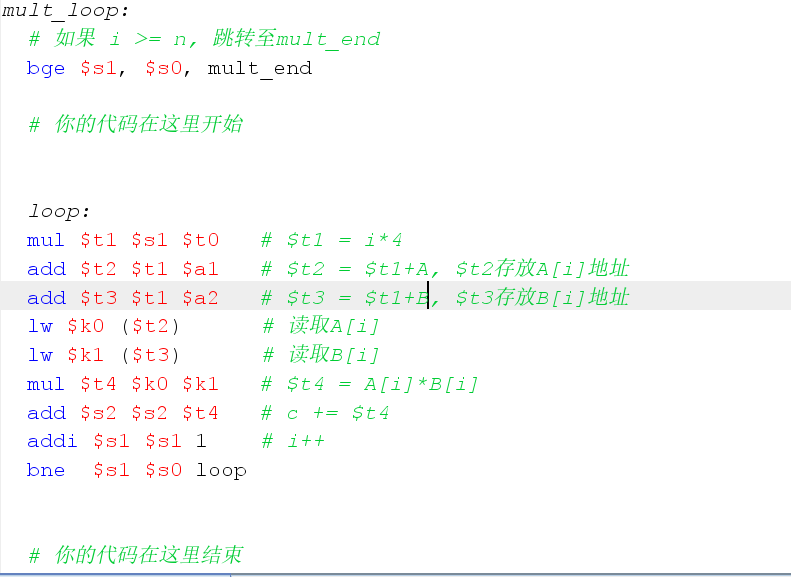
向量点积的实现  
 矩阵乘法将是本学期课程的一个贯穿始终的案例，因此我们将利用这个实验一边熟悉这个经典的数学问题，一边加深对MIPS指令集体系结构的认识。  
 首先，我们用一个简单的“向量点积”运算作为热身。你将拿到一个不完整的MIPS汇编代码“task2-向量点积.asm”，我们的目标是按照C语言描述的功能：  
  
并在相应的地方补充完整汇编代码。  
  
当你补充完整汇编代码并运行之后，你将可以在观察内存的窗口看到两个向量和点积结果：  


# 二 设计思路

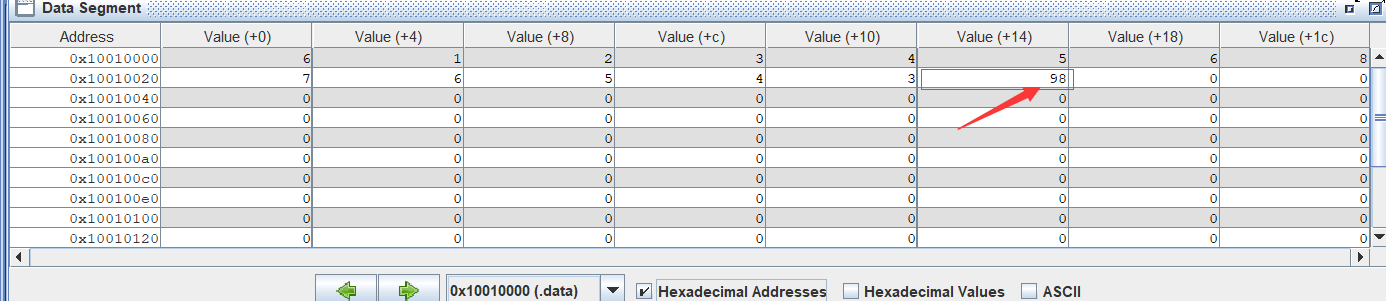
1.根据题目要求，把内存中存储的数据一个个提出到寄存器中，再用一个寄存器，把他们一个个加起来，最后输出到0x10010034的内存中。

# 三 成果展现

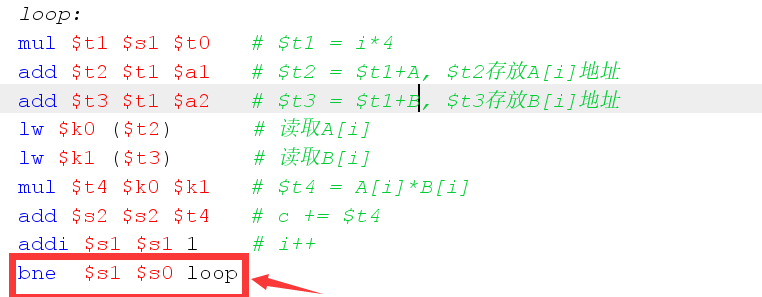
**代码图：**



**输出展示图：**



# 四 实验细节



注意循环的次数，要记得跳出循环，防止出错。 在读取前，要先进4位内存地址 到下一位内存中，再读取数据。

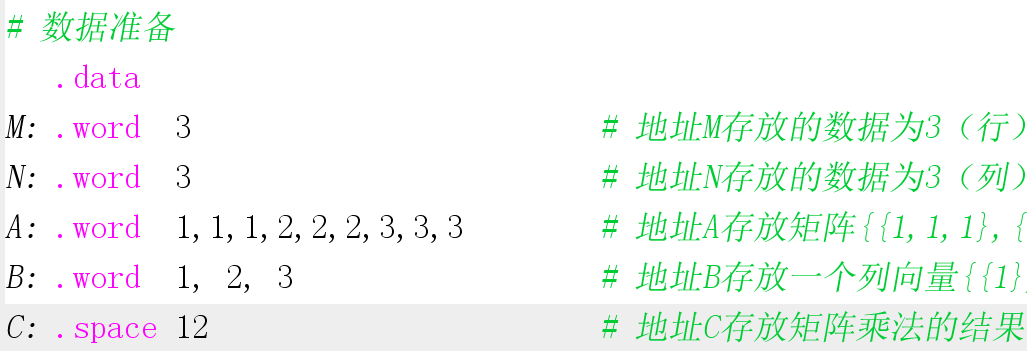
Ps:但好像已经有循环了。

# 五 实验总结

通过这个实验，我学习到了循环的和连续读取内存中数据的方法，下一个矩阵乘向量打下基础。

# 实验一.D

# 一 题目要求

矩阵乘法的实现  
 接下来，你将拿到一份汇编代码的框架“task3-矩阵乘向量.asm”，你需要在这份代码当中相应地完成一个矩阵乘上一个向量的运算。测试数据已经事先为你准备好了：  
  
分别是一个矩阵A（3×3大小）和一个列向量B（3×1大小）。 根据线性代数的知识，我们可以计算出结果：  

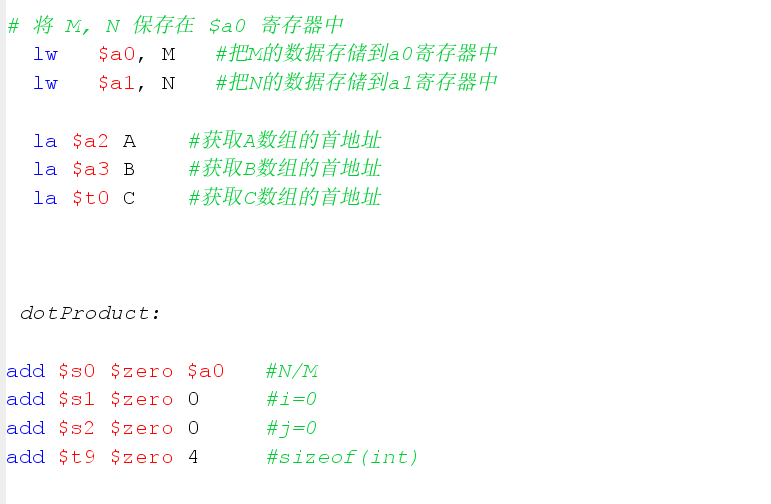

# 二 设计思路

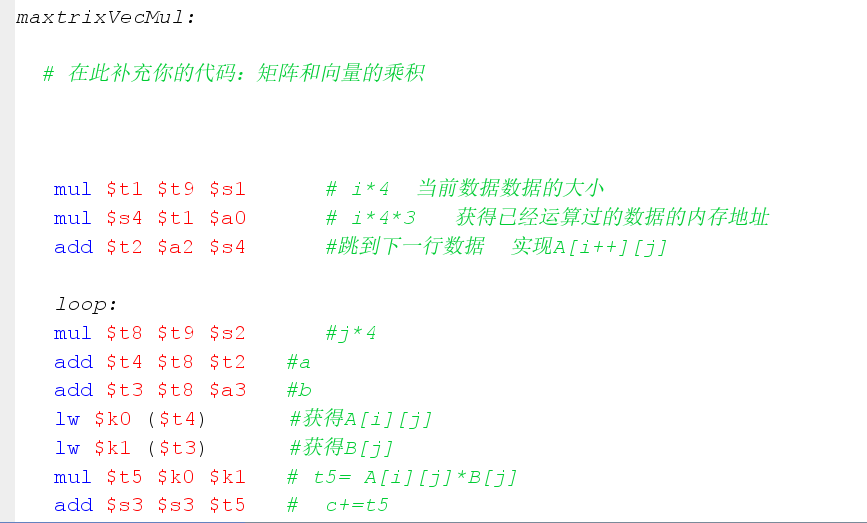
通过上一个实验，已经实现了单矩阵数组的乘法相加和存储，在这个基础上只要实现二次循环，再依次存储到内存中，就可以实现矩阵的乘法。

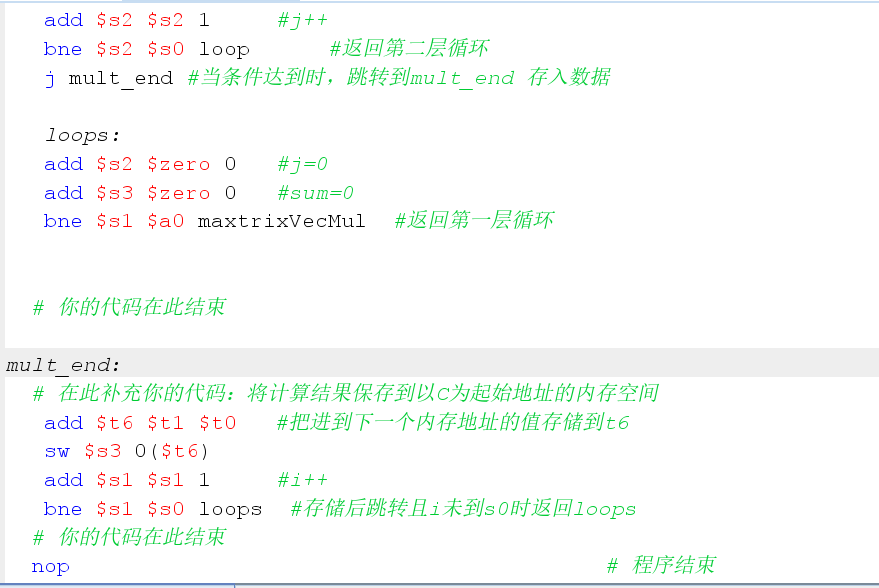
用一个寄存器来存储i 一个寄存器来存储j ,在j循环中存储到s3中，用bne判断循环是否结束，在用j 跳转到mult\_end函数中 在获得内存后根据地址变化增加后,存储到内存中。

# 三 成果展现

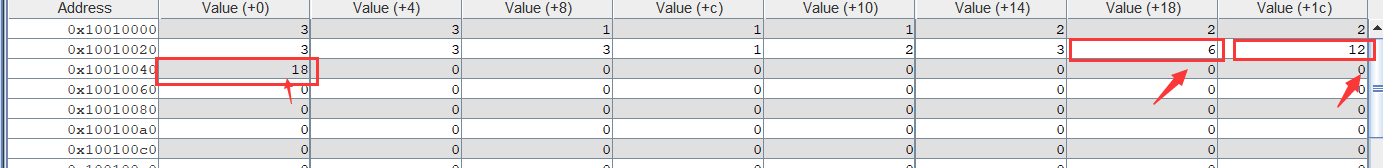
代码图：



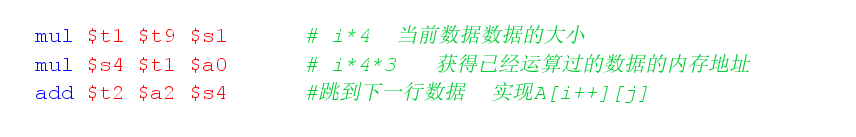




结果图：



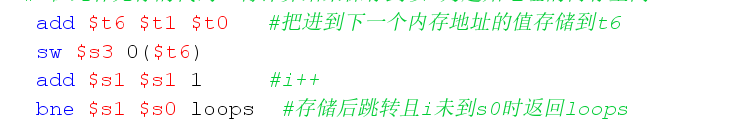
# 四 实验细节



t1存储当前数据长度，每一次循环，i++，在i\*4 ，再加初始地址得出下一个内存的地址

s4 存储A是跳到下一行的地址需要添加的量。

t2存储A是跳到下一行的地址。



当s1 ==s0时结束。

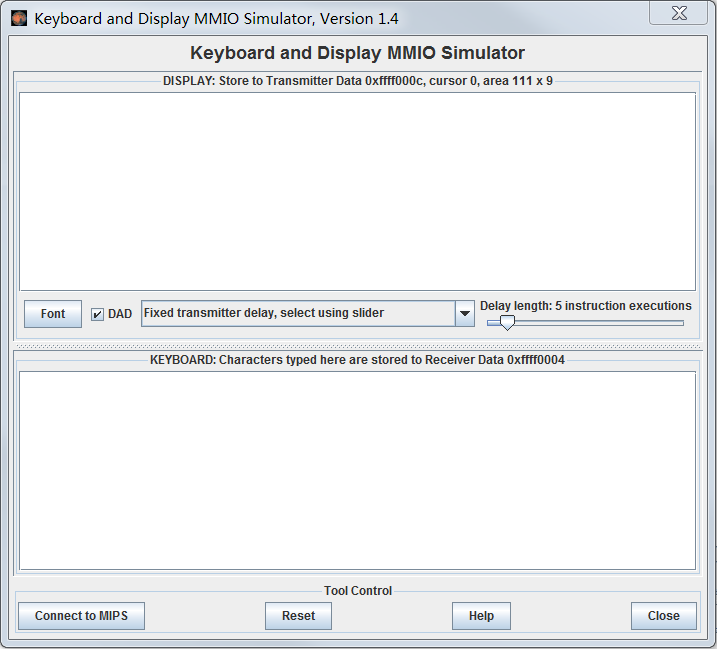
I++;

# 五 实验总结

通过实验c和实验d 认识学习到了多重循环和内存存储的调用方式。

# 实验一.E

# 一 题目要求

围绕虚拟键盘和虚拟终端开发一个小应用程序  
Mars仿真器提供一个虚拟键盘和虚拟终端（如下图所示）：  
  
在这个题目当中，你的任务是围绕虚拟键盘和虚拟终端开发一个小应用程序。

# 二 设计思路

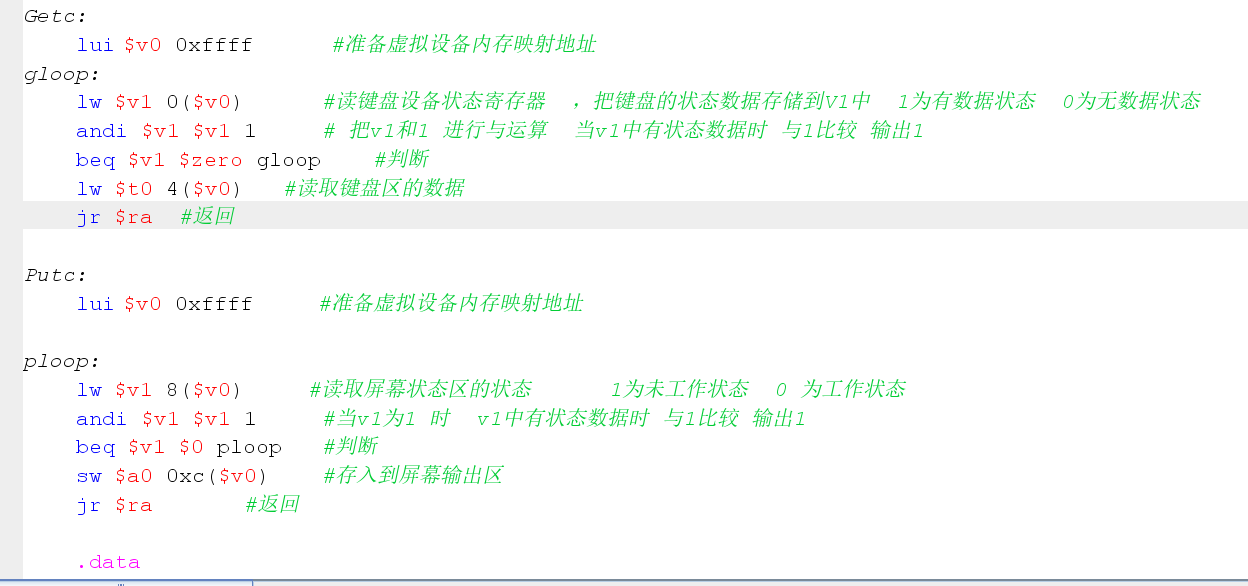
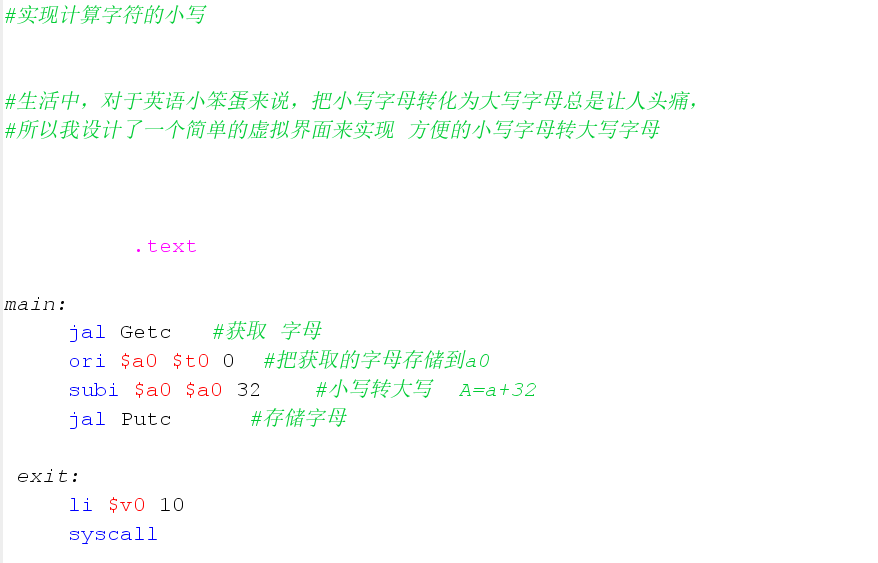
#生活中，对于英语小笨蛋来说，把小写字母转化为大写字母总是让人头痛，

#所以我设计了一个简单的虚拟界面来实现 方便的小写字母转大写字母

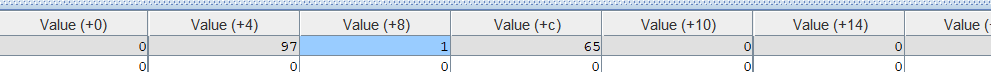
小写-32就会变成大写模式，当键盘设备状态寄存器为1时读入数据 再转化成大写字母，当屏幕状态寄存器为1 时 输出出去

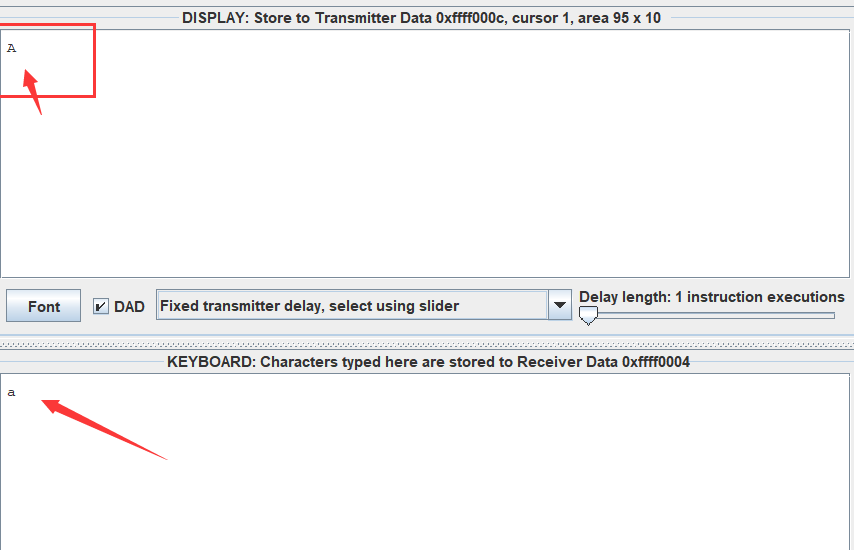
# 三 成果展现

代码区：

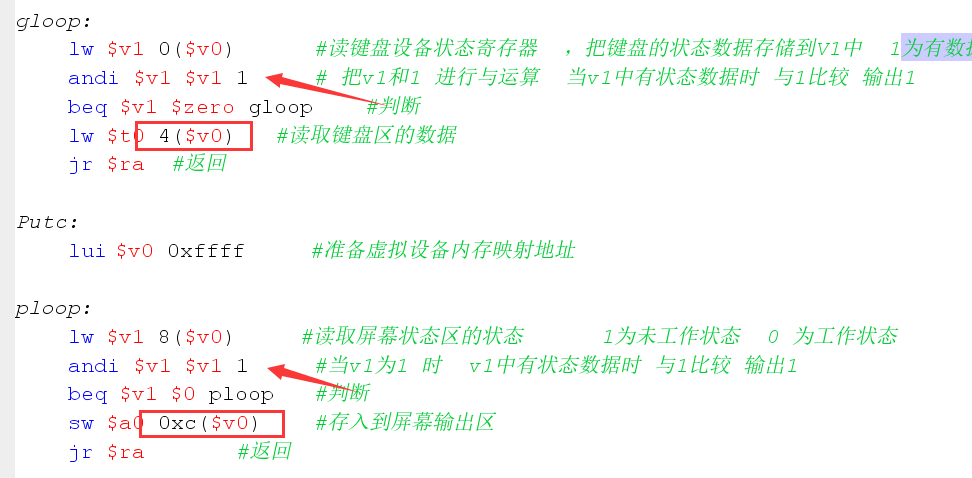


结果：





# 四 实验细节



键盘 1为工作 0为未工作

屏幕 1为未工作 0为工作

# 五 实验总结

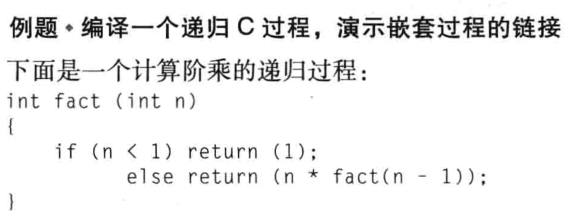
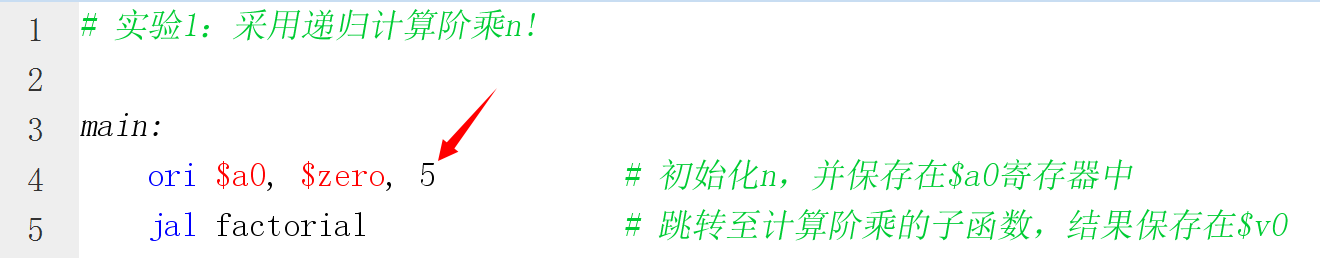
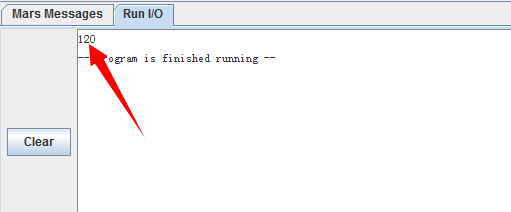
学会了通过虚拟键盘 虚拟内存 实现用户的输入—> 运算 —> 输出 的简单操作

**键盘 1为工作 0为未工作 状态寄存器0xffff 输入 0xffff0004**

**屏幕 1为未工作 0为工作 状态寄存器0xffff0008 输出 0xffff000c**

# 实验一.F

# 一 题目要求

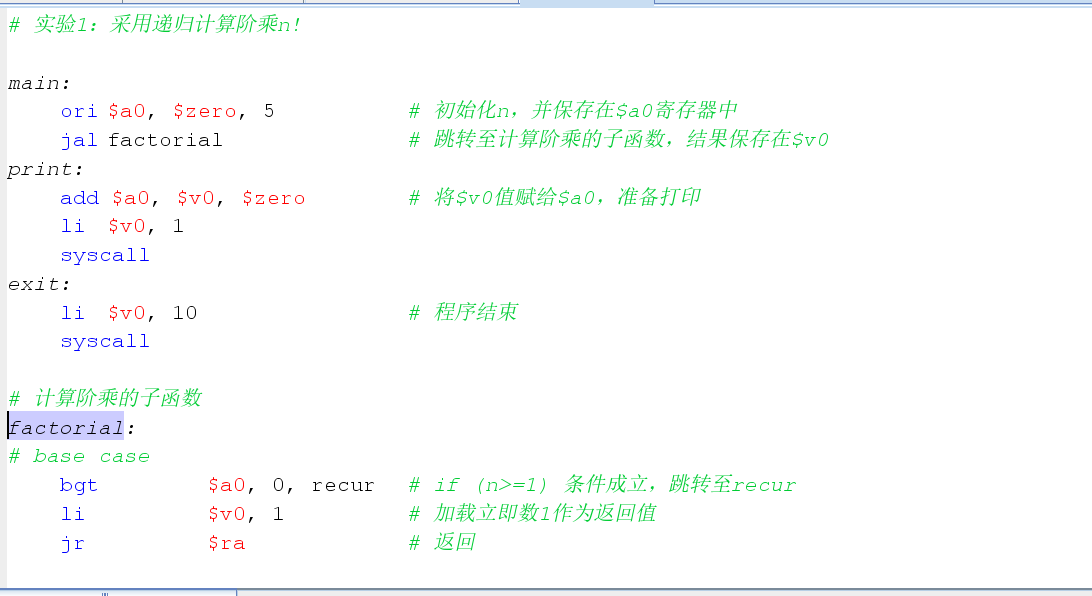
使用递归的方式计算阶乘  
 一个典型的计算阶乘的递归过程如下图所示：  
  
在这个任务中，一份汇编代码的框架“task4-阶乘.asm”你需要使用MIPS汇编程序以递归的形式解决这个问题。你可能会遇到的挑战包括：（1）如何使用栈保护某些重要的寄存器；（2）如何结合MIPS的jal分支跳转指令实现递归。当你进入测试阶段时，你可以在第4行代码修改n的值：  
  
  
如果你的代码实现正确，你可以在Mars仿真器的对话框中看到正确的计算结果：  


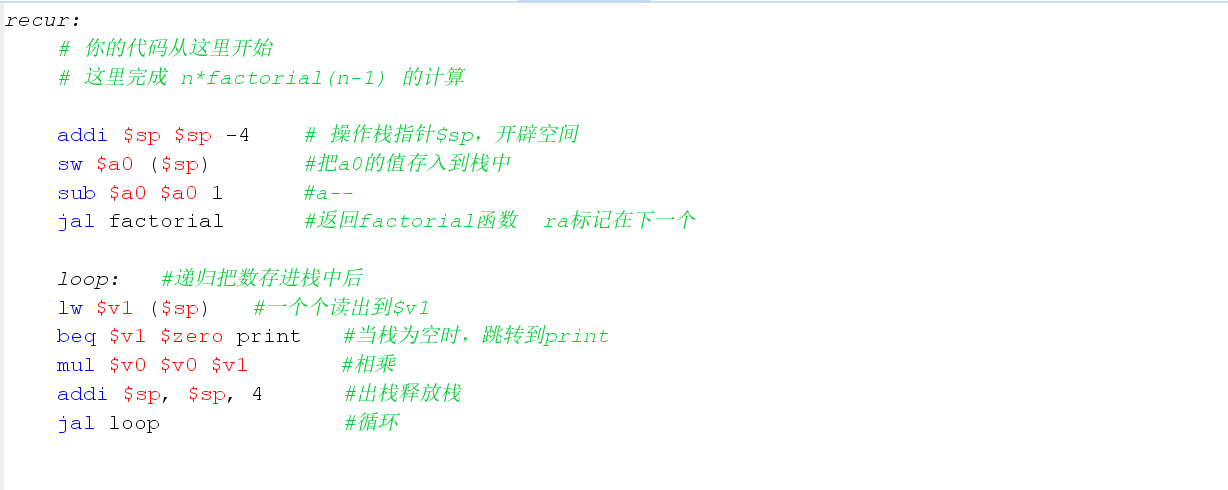
# 二 设计思路

1. 解法一：先开辟一个栈空间 ，把a0存储进去，再a0--循环jar返回factorial，，ra为下一步当a0=0时，ra到接下来，v0=1;出栈，把栈的数乘以v0 再循环出栈。实现阶乘。

# 三 成果展现

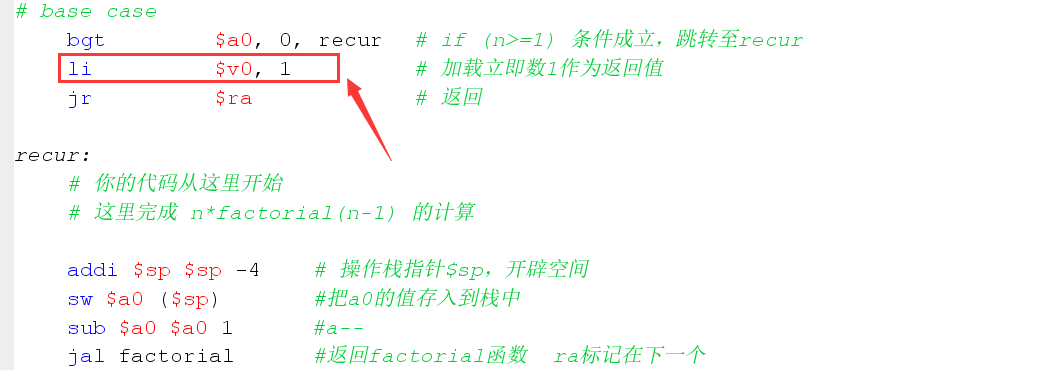
代码图：







# 四 实验细节



因为有这个，所以可以先实现a0的进栈，而不用再判断v0是否为0 ，当a0为0时表示递归到尽头，再给v0赋值，实现递归。不然还要给v0赋值。

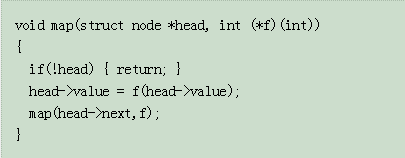
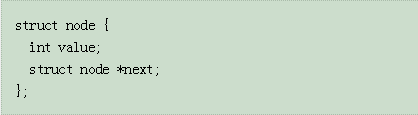
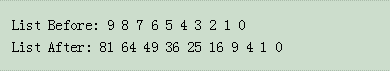
# 五 实验总结

通过这个实验，学到了压栈和出栈的操作，通过对栈的操作从而可以实现类似递归的操作。

学到了对内存指针方面进行操作。

# 实验一.附加

# 一 题目要求

* 1. 介绍  
      在这部分实验当中，你将拿到一个不完整的汇编源文件（文件名为lab1\_fujiati.s），然后根据一段C语言代码和相应的注释，编写出相应的MIPS汇编代码，并最终实现C语言函数的功能。
  2. map（）函数  
      你要实现的C语言函数如下：  
       
      这个函数有两个参数。第一个参数是一个单向链表的头节点，这个单向链表的节点采用如下结构体进行定义：  
       
     第二个参数是一个函数的地址，这个函数使用一个整形变量作为参数，并返回一个整形变量。如果你对*int (\*f)(int)* 的语法并不熟悉，不必太担心，它的基本功能是利用函数指针*f* 进行函数调用，你可以在网络上找到很多关于函数指针的使用方法。
     1. 测试数据说明  
        当你完成了lab1\_fujiati.s的编写，你可以在MARS仿真器当中运行你的代码，如果一切正确的话，你应该能看到如下输出：  
        
     2. 如何在MIPS汇编当中实现C语言的函数指针功能  
         在课堂上，你已经学过MIPS指令集当中的*j, jal, jr*等指令。为了实现C语言当中的函数指针的功能，我们可以利用MIPS指令集当中的*jalr* 指令。这条指令的使用样例如下：  
          
        如果*garply*是一个函数名的话，你可以利用MIPS指令集当中的 *la* 指令将函数*garply*的地址加载到寄存器$t0当中，然后使用 *jalr $t0* 完成函数调用。你可以通过查询教材附录的A.10.9小节获得更多关于jalr指令的信息。
     3. 思考题
        1. 源码第25行问题“思考：哪一个函数才能完成实验目标？”
        2. 源码第55行问题“思考：你如何访问value? 你如何访问指向下一个节点的指针？”
        3. 源码第58行问题“思考: 为什么是$a0?”
        4. 源码第62行问题“思考：$s1存放的是什么的值？”
        5. 源码第66行问题“思考：返回的value”
        6. 源码第70行问题“怎么获得指向下一个节点的指针？”

源码第74行问题“为什么是$a1? 为什么不是 $a0?”

# 二 设计思路

根据实验要求，完成对于代码。

# 三 成果展现

源码第25行问题“思考：哪一个函数才能完成实验目标？”

square可以完成实验目标。

源码第55行问题“思考：你如何访问value? 你如何访问指向下一个节点的指针？”

通过访问$s0寄存器存储节点地址所对应的值得到当前的Value。

源码第58行问题“思考: 为什么是$a0?”

因为$a0的值因为有寄存器$s0暂时复制，可以用于存储的是当前节点的值。

源码第62行问题“思考：$s1存放的是什么的值？”

$s1 保存的是square函数的地址

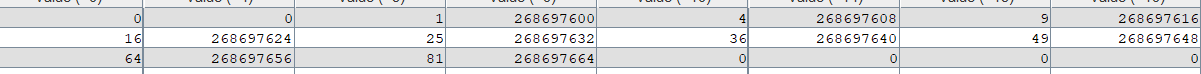
源码第66行问题“思考：返回的value”

假设 value在$v0, 存放回当前节点里。

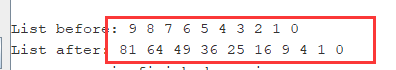
源码第70行问题“怎么获得指向下一个节点的指针？”

把当前地址偏移4，获得下一个节点的地址

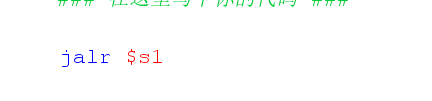
堆图：



结果图：



# 四 实验细节



当指令中的指令码为SPECIAL，功能码为6b001001时，表示jalr指令。

将地址为rs的通用寄存器的值赋给寄存器PC，作为新的指令地址，同时将跳转指令后面第2条指令的地址作为返回地址保存到地址为rd的通用寄存器，如果没有在指令中指明rd，那么默认将返回地址保存到寄存器$31.



Syscall的用法：当$v0==9时 ，开启堆，并把堆的地址存储到$V0中。

# 五 实验总结

通过附加题：深入掌握了递归和栈的处理，认识到了堆的处理。

实现了链表的地址存储，和遍历到下一个节点的方法。