



# 《计算机组成原理实验》 实验报告

(实验二)

学院名称：数据科学与计算机学院

专业（班级）：18 计教学 3 班

学生姓名：张洪宾

学号：18340208

时间：2019 年 10 月 19 日

成绩：

## 实验二：MIPS汇编语言程序设计实验

### 一. 实验目的

1. 初步认识和掌握MIPS汇编语言程序设计的基本方法；
2. 熟悉QEMU模拟器的使用

### 二. 实验内容

(必做)  $08 \% 4 + 1 = 1$ ，第一题如下：

1. 【排序】从键盘输入 10 个无符号数字并从大到小进行排序，排序结果在屏幕上显示出来。

例如对于输入

```
4 1 3 1 6 5 17 9 8 6
```

其输出为：

```
17 9 8 6 6 5 4 3 1 1
```

做完这个题目我感觉体会颇深，又看到第三题觉得很有感觉，并且有很多代码可以重复使用，又做了第三题：

3. 【跳一跳】近来，跳一跳这款游戏风靡全国微信圈，受到不少玩家的喜爱。

简化后的跳一跳规则如下：玩家每次从当前方块跳到下一个方块，如果没有跳到下一个方块上则游戏结束。如果跳到了方块上，但没有跳到方块的中心则获得 1 分；跳到方块中心时，若上一次的得分为 1 分或这是本局游戏的第一次跳跃则此次得分为 2 分，否则此次得分比上一次得分多两分（即连续跳到方块中心时，总得分将+2，+4，+6，+8...）。现在给出一个人跳一跳的全过程，请你求出他本局游戏的得分（按照题目描述的规则）。

输入包含多个数字，用空格分隔，每个数字都是 1，2，0 之一，1 表示此次跳跃跳到了方块上但是没有跳到中心，2 表示此次跳跃跳到了方块上并且跳到了方块中心，0 表示此次跳跃没有跳到方块上（此时游戏结束）。输出一个整数，为本局游戏的得分（在本题的规则下）。

例如对于输入

1 1 2 2 2 1 1 2 2 0

其输出为:

22

对于上述样例而言, 结果计算过程为:  $1+1+2+2+2+1+1+2+2+0=22$ .

### 三. 实验器材

装有Ubuntu Linux系统的PC机一台, QEMU模拟器软件一套, GNU跨平台交叉编译环境一套。

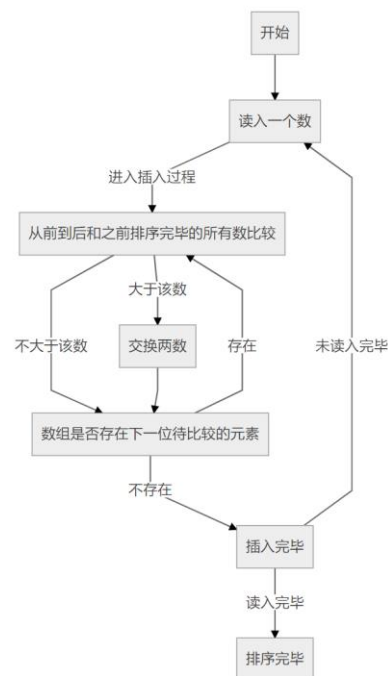
### 四. 实验过程与结果

#### 1.第一题:

对于第一个题目, 读入十个整数再排序, 我本来想要使用快排算法, 但是发现用MIPS实现快排有点超出我的能力范围, 那就用最简单的冒泡排序吧。

冒泡排序算法其实是我们在大一用的最基本的排序算法, 我们都非常熟悉。用C++先将问题的基本思路理一下, 并画出程序框图如右:

```
#include <iostream>
using namespace std;
int main() {
    int a[10];
    for (int i = 0; i < 10; i++) {
        cin >> a[i];
    }
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 9; j++) {
            if (a[j] < a[j + 1]) {
                int temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
    for (int i = 0; i < 10; i++) {
        cout << a[i] << endl;
    }
}
```



这一切看上去非常简单, 好像也没有什么特别困难的操作。但是看起来简单的一切到了MIPS汇编就不再简单。TA上课时说, 我们这次只用到三个系统调用, 然后我仔细一看傻眼

了。首先是exit，这个没什么问题，然后是read和write。看上去十分正常，但是仔细一看，问题很大：这玩意就类似于C++中二进制文件流，一次读取无格式若干位的二进制位数。但是没有任何不像C++一样可以用类型转换，必须自己手动分配内存，就很难受。于是我去网上查了一下系统调用表，看到了Linux内核的系统调用表和非Linux内核的系统调用表，才明白助教的良苦用心：Linux系统调用表是不能调用读取和输出int型变量。于是我们也就只能用把输入的整数以字符串的形式处理，输出的整数得先写成字符串的形式，十分刺激。以第一题为例，大部分的代码都是用于输入和输出的。现在就来具体的分析将输入和输出的整数与用字符串的方式处理的具体方式：

首先是输入部分：第一题要读十个整数进数组，首先我们将输入的所有整数当作应该字符串，首先定义一个临时变量作为当前数字，每次读一个字符，然后判断它是不是数字，是的话让它成为当前数字的最后一位，若是空格等其他非数字的字符我们就将已经获取的当前数字保存到数组中。当读取的数字的数量到达十个的时候，结束读取。用C语言代码模拟一下：

```
#include <stdio.h>
#define MAX_LENGTH 10000
int main() {
    char s[MAX_LENGTH];
    int a[10];
    int count = 0;
    int temp = 0; // 临时变量用于保存正在读取的整数的值
    while (true) {
        if (count == 10) {
            break;
        }
        char c = getchar();
        //读取的这一个字符代表整数的情况
        if (c >= '0' && c <= '9') {
            int last_bit = c - '0'; //将字符的ASCII码转换字符代表的整数值
            temp = temp * 10 + last_bit; //将字符添加到当前整数的最后一位
        }
        //读取的字符是空格、回车的其他情况
        else {
            a[count] = temp; //当前整数读取结束，将整数存到数组中
            count++; //计数加1，下次访问数组的下一个位置
            temp = 0; //将临时变量清零，用于接下来新的数字的读取
        }
    }
}
```

当然在MIPS中这段代码的表示相对更加复杂，但是思路大同小异，MIPS代码如下：

```

li $t0,0 #计数变量
li $t2,0 #存整数的变量
loop:
li $t1,0 #读字符的临时变量
li $t3,0 #存 48 和 57 的临时变量
la $s0, char_value #将 char 的地址存到 s0 寄存器
loop1:
    bge $t0,10,sort #如果计数变量大于等于 10 则跳转，对应 C 代码中的 break
    li $v0, 4003 #调用 read 函数
    li $a0, 0
    la $a1, char_value
    li $a2, 1
    syscall #读 char，对应 C 代码中的 getchar 函数

    lb $t1,0($s0) #将读取的字节存到 char 变量中，便于下一步操作
    #接下来就判断读进来的字符是否在'0'到'9'之间，若不是则跳转到存数操作 oper_1
    li $t3,47
    ble $t1,$t3,oper_1
    li $t3,58
    bge $t1,$t3,oper_1
    #将 ASCII 码减 48 得到字符对应的数字
    addi $t1,$t1,-48
    #然后更新当前读取的数字
    mul $t2,$t2,10
    add $t2,$t1,$t2
    #继续循环
    j loop1
    #若读取的是空格或回车等非数字字符，跳转到此处进行存数操作，然后再重新开始读数
oper_1:
    la $a1,int_array
    mul $t4,$t0,4 #偏移量
    add $a1,$a1,$t4
    sw $t2,0($a1)
    addi $t0,$t0,1
    li $t2,0 #存整数的变量
    j loop

```

在这里有一个细节，就是我们在 C 语言中数组下标加 1 的时候，因为一个 int 整数占 4 个字节，所以相应的 MIPS 汇编语言应该数组的地址加上四倍的偏移量取出，然后用 sw 存数。这样子之后读进来的字符串就保存在一段连续的内存空间中。

而冒泡排序的代码则是非常简单的，我根据 C++ 代码，将冒泡排序的部分用汇编代码加以替代，如下：

```

sort:
    li $t0, -1          #外层计数变量
    la $a1, int_array   #整数数组的首地址
loop4:                  #外层循环
    addi $t0, $t0, 1
    bge $t0, 10, end    #如果外层计数变量到 10 则结束循环跳转到后面的 end 部分
    li $t1, -1          #内层计数变量
loop5:                  #内层循环
    addi $t1, $t1, 1
    mul $t2, $t1, 4
    bge $t1, 9, loop4   #内层循环计数变量到达 9 则跳回外层循环
    add $t3, $a1, $t2    #获取数组中当前内存计数变量对应的地址
    addi $t4, $t3, 4     #获取数组中当前内存计数变量加 1 对应的地址
    lw $t5, ($t3)        #取值
    lw $t6, ($t4)        #取值
    bgt $t6, $t5, swap   #若出现 a[j] < a[j+1] 则交换两段内存中的值
    j loop5

swap:                   #交换语句
    sw $t6, ($t3)
    sw $t5, ($t4)
    j loop5

```

当这段代码运行完成后数组中的元素以降序排列。于是我们剩下的事情就是整个数组输出了。这也是折磨我最久的环节。

Linux内核不提供输出整数的系统调用，所以我们必须再将整数转换为字符串来输出。所以我们得继续纠结字符串转为整数得问题。如何将字符串转换为字符串并输出呢？

在程序设计的课程上我们学过用C语言对目标整数除10不断取模，将模以字符的形式输出的方法，在这里我们可以使用类似的方法。首先我阅读了MIPS指令集中的div函数，格式如下：

```

div $t0, $t1 #lo 赋值为$t0 / $t1, hi 赋值为 $t0 % $t1
mflo $t0 #将 lo 寄存器中内容赋值给$t0
mfhi $t1 #将 hi 寄存器中的值赋值给$t1

```

所以可以将数值中的值赋值给\$t0，然后用div语句除以10，获取的余数再加上48就是该数值最后一位的ASCII码。不断循环直至div语句除以10的商为0。

用C代码表示该算法：

```

char s[10]; //char 数组用来表示字符串
char* ptr = s; //指向数组初始位置的指针
for (int i = 0; i < 10; i++) {

```

```

int offset = 10;    //偏移量，最开始指向数组的尾部
int quotient = a[i]; //取出数组第 i 个变量并赋值给存商的变量
int remainder = 0;   //存余数的变量初始化为 0
while (quotient != 0) {
    offset--;        //偏移量减 1
    remainder = quotient % 10;
    remainder = remainder + 48; //获取整数的最后一个数的 ASCII 码
    quotient /= 10;        //除以 10 获取下一个整数
    *(ptr + offset) = remainder; //数字末位 ASCII 码赋值给字符串最后一位
}
for (int i = offset; i < 10; i++) {
    printf("%c", *(ptr + i)); //输出字符串
}
printf("\n");
}

```

就这样我们就可以获取用字符串表示的整数值并输出。

用MIPS汇编代码表示的算法如下：

```

end:
li $t0, -1 #第 i 个整数的索引
loop2:
    li $t7, 10 #偏移量初始化为 10
    la $t6, char_value
    la $a1, int_array #整数数组的首地址
    addi $t0, $t0, 1
    bge $t0, 10, bye #处理完数组则跳转到结束部分
    mul $t1, $t0, 4 #乘 4 得到偏移量
    add $a1, $a1, $t1 #第 i 个整数的地址
    lw $t2, ($a1) #取出第 i 个整数并存放在 t2
loop3:
    div $t2, 10 #将整数除 10
    mflo $t2 #商赋值给 t2 做接下来的循环
    mfhi $t3 #t3 储存余数
    addi $t3, $t3, 48 #将余数转换为 ascii 码并存入 t3
    add $t4, $t6, $t7 #用 t4 储存存储地址
    sb $t3, ($t4) #将 t3 存储到 t4 指向的地址
    addi $t7, $t7, -1 #偏移量减 1
    beqz $t2, print
    j loop3

print:
    li $v0, 4004
    move $a1, $t4
    li $t6, 10

```

```

sub $t6,$t6,$t7
move $a2,$t6
syscall      #输出字符串表示的整数
li $v0, 4004
li $a0, 1
la $a1, stringa1
li $a2, 1
syscall #输出换行符
j loop2

```

于是就这样子我们完成了第一题所有的所有的工作，运行一下代码：



```

zhb@zhb-Lenovo-YOGA-710-11IKB: ~/Desktop/lab
File Edit View Search Terminal Help
zhb@zhb-Lenovo-YOGA-710-11IKB:~$ cd Desktop/lab/
zhb@zhb-Lenovo-YOGA-710-11IKB:~/Desktop/lab$ mips-linux-gnu-as -g 1.5 -o 1.o
zhb@zhb-Lenovo-YOGA-710-11IKB:~/Desktop/lab$ mips-linux-gnu-ld -g 1.0 -e main -o
1
zhb@zhb-Lenovo-YOGA-710-11IKB:~/Desktop/lab$ qemu-mips 1
4 1 3 1 6 5 17 9 8 6
17
9
8
6
6
5
4
3
1
1
zhb@zhb-Lenovo-YOGA-710-11IKB:~/Desktop/lab$ 

```

如图：我们得到了输入的10个整数的降序排列，即题目所要求。

## 2.第三题：

第三题乍看好像很复杂，但是做完第一题后再看发现其实这道题目很简单。它的输入只有三种，所以我们不需要将它的输入转换为整数，直接比较输入的ASCII码并使用三种条件下的分支语句就行了。而经过计算的整数，可以用第一题的输出部分的代码将整数转换为字符串再输出，就不用再写一次，于是第三题在第一题的基础上就变得相对简单了很多。

所以我们现在最重要的事情是写出整个程序的算法。为了表示这个算法，我用C++实现了这个程序，代码如下：



```

#include <iostream>

using namespace std;

int main() {
    int n = 0; //定义总分并初始化为 0
    int last_try = 0; //用于存放上一次的得分的变量
    while (true) {
        char x; //临时变量用于获取当前得分的 ASCII 码
        cin >> x;
        if (x == 48) break; //如果 ASCII 码为 48，即输入的得分为 0，退出循环
        //如果输入的 ASCII 码为 49，即输入的得分为 1，总分加 1，上一次得分赋值为 1
        else if (x == 49) {
            last_try = 1;
            n += 1;
        }
        //如果输入的 ASCII 为 49，即输入的总分为 2
        else if (x == 50) {
            //上次得分为 1 的情况
            if (last_try == 1) {
                last_try = 2;
                n += 2;
            }
            //上次的分不为 1 的情况
            else {
                n += last_try;
                last_try += 2;
                n += 2;
            }
        }
    }
    cout << n << endl;
}

```

可以看到除去我们在问题一已经解决的输出问题外，我们唯一应该解决的是这个相对复杂的分支语句。相比将整数转换为字符串输入输出，这个显得尤为简单：

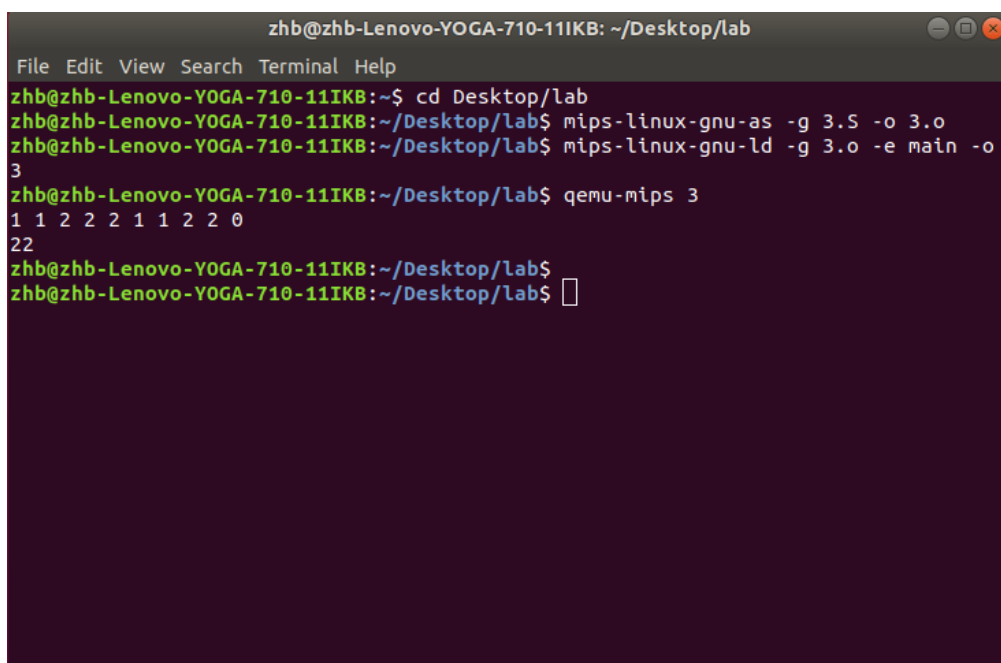
```

li $t0, 0 #记录总分
li $t1, 0 #记录上一波得分
loop:
    li $v0, 4003
    li $a0, 0
    la $a1, char_value #读字符串
    li $a2, 1
    syscall

```

```
li $s0, 48
li $s1, 49
li $s2, 50
lb $t2, ($a1)
#等于 0 则跳转到打印区
beq $t2, $s0, end
beq $t2, $s1, case1
beq $t2, $s2, case2
blt $t2, $s0, case4
bgt $t2, $s2, case4
case1:
    add $t0, $t0, 1
    li $t1, 1
    j loop
case2:
    li $t3, 1
    beq $t1, $t3, case3
    addi $t1, $t1, 2
    add $t0, $t0, $t1
    j loop
case3:
    addi $t0, $t0, 2
    li $t1, 2
    j loop
case4:
    j loop
```

然后将我们得到的结果输出就行了。运行程序后结果如下：



```
zhh@zhh-Lenovo-YOGA-710-11IKB: ~/Desktop/lab
File Edit View Search Terminal Help
zhh@zhh-Lenovo-YOGA-710-11IKB:~$ cd Desktop/lab
zhh@zhh-Lenovo-YOGA-710-11IKB:~/Desktop/lab$ mips-linux-gnu-as -g 3.S -o 3.o
zhh@zhh-Lenovo-YOGA-710-11IKB:~/Desktop/lab$ mips-linux-gnu-ld -g 3.o -e main -o 3
zhh@zhh-Lenovo-YOGA-710-11IKB:~/Desktop/lab$ qemu-mips 3
1 1 2 2 2 1 1 2 2 0
22
zhh@zhh-Lenovo-YOGA-710-11IKB:~/Desktop/lab$
zhh@zhh-Lenovo-YOGA-710-11IKB:~/Desktop/lab$
```

满足题目要求，第3题完成。

## 五. 实验心得

首先，我不太想要使用TA给的那个Debian虚拟机，因此我自己配置了我在自己的电脑装的Ubuntu系统的环境，但是在用命令行安装库的过程发现404，无可奈何我只能去换源，换上中科大的源后才解决这一问题。不过也通过这个过程更加了解了Linux系统的一些操作。

此外，通过这个实验，我终于明白了为什么C语言教材中反对用goto语句，在这个实验中各种分支跳转，如果是运行比较复杂的程序，那这个代码的复杂程度与可读性简直就是一场灾难。著名算法科学家迪杰斯特拉曾提出“goto有害论”，也是针对这个问题。MIPS语句中各种跳转，非常容易忘记跳转回去，直接执行紧随其后的代码，效果令人窒息。

然后，对寄存器的合理调用也是一个非常大的考验。在写第一题代码的过程中，我因为调用了很多寄存器，对寄存器的调用不是特别合理，如果这个程序再复杂一些的话，我可能会因为寄存器的调用与各种跳转导致的寄存器内容的变化而翻车。而因为我并没有写函数，而是将所有的东西写在main函数中，所以我并没有考虑过不同寄存器种类不同带来的影响，如果在复杂的代码中调用了函数，那对寄存器的调用会有更高的要求，就更加麻烦。

整体来说这个实验的难度是高于我们上课时讲MIPS时的难度，涉及了很多细节指令，都是要我们上网查询的。也通过这个实验，我更加深刻地理解了汇编。

### 【程序代码】

问题1的代码：

```
.data
char_value:
    .space 12
stringa1:
    .asciiz "\n"
int_array:
    .space 40
.text
.global main
main:
    li $t0,0 #计数变量
    li $t2,0 #存整数的变量
loop:
    li $t1,0 #读字符的临时变量
```

```

li $t3,0 #存 48 和 57 的临时变量
la $s0, char_value #将 char 的地址存到 s0 寄存器
loop1:
    bge $t0,10,sort #如果计数变量大于等于 10 则跳转，对应 C 代码中的 break
    li $v0, 4003 #调用 read 函数
    li $a0, 0
    la $a1, char_value
    li $a2, 1
    syscall #读 char，对应 C 代码中的 getchar 函数

    lb $t1,0($s0) #将读取的字节存到 char 变量中，便于下一步操作
    #接下来就判断读进来的字符是否在'0'到'9'之间，若不是则跳转到存数操作
    li $t3,47
    ble $t1,$t3,oper_1
    li $t3,58
    bge $t1,$t3,oper_1
    #将 ASCII 码减 48 得到字符对应的数字
    addi $t1,$t1,-48
    #然后更新当前读取的数字
    mul $t2,$t2,10
    add $t2,$t1,$t2
    #继续循环
    j loop1
#若读取的是空格或回车等非数字字符，跳转到此处进行存数操作
oper_1:
    la $a1,int_array
    mul $t4,$t0,4 #偏移量
    add $a1,$a1,$t4
    sw $t2,0($a1)
    addi $t0,$t0,1
    li $t2,0 #存整数的变量
    j loop

sort:
    li $t0,-1 #外层计数变量
    la $a1,int_array #整数数组的首地址
    loop4: #外层循环
        addi $t0,$t0,1
        bge $t0,10,end #如果外层计数变量到 10 则结束循环跳转到后面的 end 部分
        li $t1,-1 #内层计数变量
        loop5: #内层循环
            addi $t1,$t1,1
            mul $t2,$t1,4
            bge $t1,9,loop4 #内层循环计数变量到达 9 则跳回外层循环

```

```

add $t3,$a1,$t2 #获取数组中当前内存计数变量对应的地址
addi $t4,$t3,4 #获取数组中当前内存计数变量加 1 对应的的地址
lw $t5,($t3) #取值
lw $t6,($t4) #取值
bgt $t6,$t5,swap#若出现 a[j] < a[j+1] 则交换两段内存中的值
j loop5

```

```

swap:          #交换语句
    sw $t6,($t3)
    sw $t5,($t4)
    j loop5

```

end:

```

li $t0,-1 #第 i 个整数的索引
loop2:
    li $t7,10 #偏移量初始化为 10
    la $t6,char_value
    la $a1,int_array #整数数组的首地址
    addi $t0,$t0,1
    bge $t0,10,bye #处理完数组则跳转到结束部分
    mul $t1,$t0,4 #乘 4 得到偏移量
    add $a1,$a1,$t1#第 i 个整数的地址
    lw $t2,($a1) #取出第 i 个整数并存放在 t2
loop3:
    div $t2,10#将整数除 10
    mflo $t2 #商赋值给 t2 做接下来的循环
    mfhi $t3 #t3 储存余数
    addi $t3,$t3,48 #将余数转换为 ascii 码并存入 t3
    add $t4,$t6,$t7 #用 t4 储存存储地址
    sb $t3,($t4) #将 t3 存储到 t4 指向的地址
    addi $t7,$t7,-1 #偏移量减 1
    beqz $t2,print
    j loop3

```

```

print:
    li $v0, 4004
    move $a1, $t4
    li $t6,10
    sub $t6,$t6,$t7
    move $a2,$t6
    syscall #输出字符串表示的整数
    li $v0, 4004
    li $a0, 1
    la $a1, stringa1

```

```

        li $a2, 1
        syscall #输出换行符
        j loop2

```

```

bye:
    li $v0, 4001
    syscall

```

问题3的代码:

```

.data
int_value:
    .space 4
char_value:
    .space 1
stringa1:
    .asciiz "\n"
.text
.global main
main:
    li $t0, 0 #记录总分
    li $t1, 0 #记录上一波得分
loop:
    li $v0, 4003
    li $a0, 0
    la $a1, char_value #读字符串
    li $a2, 1
    syscall

    li $s0, 48
    li $s1, 49
    li $s2, 50
    lb $t2, ($a1)
    #等于 0 则跳转到打印区
    beq $t2, $s0, end
    beq $t2, $s1, case1
    beq $t2, $s2, case2
    blt $t2, $s0, case4
    bgt $t2, $s2, case4
case1:
    add $t0, $t0, 1
    li $t1, 1
    j loop

```

```
case2:
    li $t3,1
    beq $t1,$t3,case3
    addi $t1,$t1,2
    add $t0,$t0,$t1
    j loop
case3:
    addi $t0,$t0,2
    li $t1,2
    j loop
case4:
    j loop
end:
    move $t1,$t0
    li $t7,10
    la $t6,char_value
loop3:
    div $t1,10#将整数除 10
    mflo $t1 #商赋值给 t2 做接下来的循环
    mfhi $t3 #t3 储存余数
    addi $t3,$t3,48 #将余数转换为 ascii 码并存入 t3
    add $t4,$t6,$t7 #用 t4 储存存储地址
    sb $t3,($t4)
    addi $t7,$t7,-1
    beqz $t1,print
    j loop3

print:
    li $v0, 4004
    move $a1, $t4
    li $t6,10
    sub $t6,$t6,$t7
    move $a2,$t6
    syscall
    li $v0, 4004
    li $a0, 1
    la $a1, stringa1
    li $a2, 1
    syscall
    li $v0,4001
    syscall
```