《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 ：	王若琪
学　　号	18340166
时　　间 ：	2019　年　10　月　17　日

成 绩 ：

实 验 二 ： MIPS汇编语言程序设计实验

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模拟器和GNU/Linux下x86-MIPS交叉编译环境的使用。

实验内容

用MIPS汇编语言设计程序，并调试运行。

我的必做题是第3题：

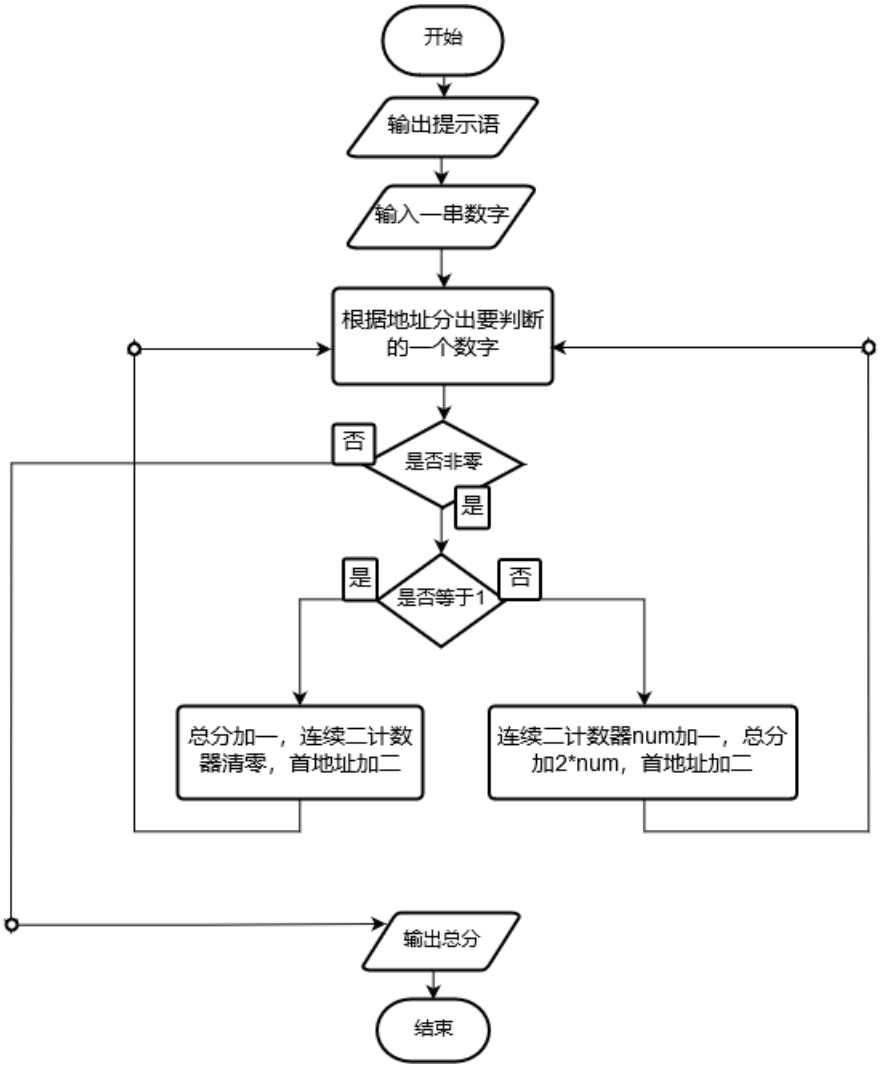
【跳一跳】近来，跳一跳这款小游戏风靡全国微信圈，受到不少玩家的喜爱。简化后的跳一跳规则如下：玩家每次从当前方块跳到下一个方块，如果没有跳到下一个方块上则游戏结束。如果跳到了方块上，但没有跳到方块的中心则获得 1 分；跳到方块中心时，若上一次的得分为 1 分或这是本局游戏的第一次跳跃则此次得分为 2 分，否则此次得分比上一次得分多两分（即连续跳到方块中心时，总得分将+2，+4，+6，+8...）。现在给出一个人跳一跳的全过程，请你求出他本局游戏的得分（按照题目描述的规则）。输入包含多个数字，用空格分隔，每个数字都是 1，2，0 之一，1 表示此次跳跃跳到了方块上但是没有跳到中心，2 表示此次跳跃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）。

实验器材

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

实验过程与结果

1．程序流程图



2. 设计思想和方法

第三题的小程序要求比较简单，为求方便，将数字一个个从字符串中分离出来，再逐个判断操作即可。用到了循环、分支和跳转等方法。

3. 分析

第三题本身的程序过程很清晰，对于输入1，将总分加一，并且将连续2的计数器归零，表示连跳被打断；对于输入2，计数器加一，总分加连跳次数的二倍；对于输入0，直接跳出循环即可。对我而言，因为没有给出整型数的输入输出方式，所以这一部分的操作要比较复杂一些，需要转化为字符串输出，输出时也用到了一个循环过程，是从后往前进行的。具体代码分析如下：

首先定义数据段，在这里我定义了sentence1表示提示句的地址，endl指换行符，便于输出换行，又给字符串str 60个字节的空间，string指输出的字符串的地址：

```
1       .data     # 定义数据段，数据将存放于此
2   sentence1:    # 定义变量名，实际是一个地址
3       .asciiz "Please enter the integers with zero to finish:\n"
4   endl:
5       .asciiz "\n"            # 伪指令
6   str:                        # 存输入的字符串
7       .space 60
8   string:
9       .space 4                # 存输出的字符串
10      .set noreorder          # 编译用伪指令，不关心
```

接着写代码段，先准备了一些之后会用到的，比如将$t0, $t1, $t2赋值为48,49,50，代表字符 ‘0’ ‘1’ ‘2’，便于和之后的字符形式的输入作比较：

```
12      .text                   # 代码段，可执行的代码存放于此
13    .global main              # 指出main为全局作用域符号
14  main:
15      li $v0, 4004            # 调用号4004 (write) 放入$v0
16      li $a0, 1               # 目标1号文件（标准输出）放入$a0
17      la $a1, sentence1       # 待输出的字符串首地址放入$a1
18      li $a2, 47              # 要输出字节数为47，放入$a2
19      syscall                 # 准备就绪，交付内核完成调用
20      li $t0, 48              # '0'
21      li $t1, 49              # '1'
22      li $t2, 50              # '2'
23      li $t3, 0               # 存放answer
24      li $t4, 0               # $t4存放连续的2的个数
```

接下来输入一个有数字0,1,2的字符串，一共不能超过60字节，也就是限制最多能输入30个整数。

```
26      li $v0, 4003            # 调用号4003 (read) 放入$v0
27      li $a0, 0               # 目标0号文件（标准输入）放入$a0
28      la $a1, str             # $a1存放输入字符串str的首地址
29      li $a2, 60              # 读取60个字节
30      syscall                 # 准备就绪，交付内核完成调用
31      la $t5, str             # $t5存放输入字符串str的首地址
```

接下来进入循环环节，先取出一个字节，判断它是0、1、2中的哪一个，分别采取不同的跳

转指令。如果取出的是0，直接输出，程序结束。

```
loop:
    lb $t6, 0($t5)          # 将一个字节从内存取到寄存器$t6中
    beq $t6, $t0, print     # 如果是0，直接输出结果
    nop                     # important to make program happy
    beq $t6, $t1, addone    # 如果是1，跳转至addone
    nop
    beq $t6, $t2, addtwo    # 如果是2，跳转至addtwo
    nop
```

如果是1，就让那个储存连续2的寄存器清零，并且总分加一，$t5寄存器里是str的地址，加二表示越过空格，来到下一个数字的地址。最后跳回loop的开始。

```
addone:
    addi $t3, $t3, 1        # $t3++
    li $t4, 0               # $t4清零
    addi $t5, $t5, 2        # $t5=$t5+2(跳过空格找下一个数)
    j loop                  # 无条件跳转
    nop
```

如果是2，总分加2*连续二的次数，$t5寄存器加二表示越过空格，来到下一个数字的地址。最后跳回loop的开始。

```
addtwo:
    addi $t4, $t4, 1        # $t4++
    add $t7, $t4, $t4       # $t7=2*$t4
    add $t3, $t3, $t7       # $t3=$t7+$t3
    addi $t5, $t5, 2        # 待读取的地址加两字节，跳过空格
    j loop
    nop
```

循环结束后，调用printfunc函数，输出最终得分：

```
print:
    addi $a0, $t3, 0        # 传参
    jal printfunc           # 过程调用，jump and link
    nop
```

我们来看看printfunc函数的内容：

这个函数的灵魂在于将数字转化为字符串输出。因为我设定了读取60个字节，也就是最多30个数，这样得分不会超过3位数，于是将需要的空间设为3，从首地址+2的地方开始写入

末位字符。

```
62    .global printfunc
63  printfunc:
64      addi $t0, $a0, 0        # $t0=$a0
65      li $t2, 2               # 需要的空间-1
66      li $t5, 2               # 最后用来计算输出字符个数
67      la $t3, string          # 最后一个字符的地址是$t3+$t2中
```

进入循环loop2，通过对10求余得到每位的数字，并且不断从后往前存入字符串中：

```
68  loop2:
69      rem $t1, $t0, 10        # $t1存余数
70      div $t0, $t0, 10        # $t0=$t0/10
71      addi $t1, $t1, 48       # 转化为字符
72      add $t4, $t3, $t2       # 存储地址
73      sb $t1, 0($t4)          # 存字节
74      addi $t2, $t2, -1       # 向前挪一个字节
75      beq $t0, $zero, printstring    # 如果是0，跳出循环
76      nop
77      j loop2                 # 如果不为0，继续循环
78      nop
```

当被除数变为零时跳出循环，输出结果字符串：

```
79  printstring:
80      sub $a2, $t5, $t2           # 要输出这么多字符
81      addi $t2, $t2, 1            # 开始的数的顺序
82      li $v0, 4004
83      add $a1, $t3, $t2           # 开始地址
84      li $a0, 1                   # 标准输出
85      syscall
```

输出完数字之后，再输出换行，这个printfunc过程就结束啦：

```
86  endline:                        # 换行
87      li $v0, 4004
88      li $a0, 1
89      la $a1, endl
90      li $a2, 1
91      syscall
```

函数执行完毕，跳回寄存器里存放的地址：

```
93      jr $ra                          # 跳回寄存器中的地址
94      nop
```

以上是我对最终版代码的分析。具体实验经历在下部分——实验步骤中详述。

4. 实验步骤

（1）首先初步按照流水线过程，用MIPS汇编语言写出程序代码，图为第一版debug之前的部分原始代码:

```
1  .data
2  sentence1:
3      .asciiz "Please enter the integers with zero to finish:\n"
4  sentence2:
5      .asciiz "The score is:"
6  endl:
7      .asciiz "\n"
8  str:
9      .space 60
10     .set noreorder
11 .global main
12     .text
13 main:
14     li $v0, 4004
15     li $a0, 1
16     la $a1, sentence1
17     li $a2, 46
18     syscall
19     li $t0, 48 # '0'
20     li $t1, 49 # '1'
21     li $t2, 50 # '2'
22     li $t3, 0 #answer
23     li $t4, 0 #count of 2
24     li $t5, 0 #count of py
25     j input
26 input:
27     li $v0, 4003
28     li &a0, 0
29     la $a1, str    # $a1=address of input str
30     li $a2, 60
31     syscall    #input a string
32 loop:
33     lb $t6, $t5($a1)  #the $t5 th
34     beq $t6, $t0, print
35     beq $t6, $t1, addone
36     beq $t6, $t2, addtwo
37 addone:
38     addi $t3, $t3, 1
39     li $t4, 0
40     addi $t5, $t5, 2
41     j loop
42 addtwo:
```

（2）接着交叉编译和运行程序

将写好的.S文件汇编为可重定位目标文件时，输出了一堆错误:

```
sysu@debian:~$ mips-linux-gnu-as -g tyt.S -o tyt.o
tyt.S: Assembler messages:
tyt.S:28: Error: bad expression
tyt.S:28: Error: invalid operands `li &a0,0'
tyt.S:33: Error: operand 2 must be an immediate expression `lb $t6,$t5($a1)'
tyt.S:50: Error: invalid operands `li $t12,10'
tyt.S:56: Error: invalid operands `li $t11,0'
tyt.S:57: Error: operand 3 must be constant `rem $t8,$t3,$t12'
tyt.S:58: Error: operand 3 must be constant `div $t3,$t3,$t12'
tyt.S:59: Error: bad expression
tyt.S:59: Error: invalid operands `rem $t9,&t3,$t12'
tyt.S:60: Error: operand 3 must be constant `div $t3,$t3,$t12'
tyt.S:61: Error: invalid operands `rem $t10,$t3,$t12'
tyt.S:62: Error: operand 3 must be constant `div $t3,$t3,$t12'
tyt.S:63: Error: invalid operands `beq $t10,$0,print9'
tyt.S:64: Error: invalid operands `li $t11,1'
tyt.S:67: Error: operand 2 must be constant `addi $a0,$t10,48'
tyt.S:71: Error: unrecognized opcode `jne $t11,$1,judge9'
tyt.S:86: Error: unrecognized opcode `jne $t9,$0,print9'
sysu@debian:~$ _
```

这些是语法错误，很容易就改完了，于是继续汇编：

```
sysu@debian:~$ mips-linux-gnu-as -g tyt.S -o tyt.o
tyt.S: Assembler messages:
tyt.S:72: Warning: used $at without ".set noat"
sysu@debian:~$ _
```

发现有一个warning，这个问题先放着，继续将生成的.o文件链接成可执行文件，并指定入口点为main过程，结果又出现了一堆问题：

```
sysu@debian:~$ mips-linux-gnu-ld -g tyt.o -e main -o ./tyt
mips-linux-gnu-ld: tyt.o: in function `print':
(.text+0x98): undefined reference to `centense2'
mips-linux-gnu-ld: (.text+0x9c): undefined reference to `centense2'
mips-linux-gnu-ld: (.text+0x110): undefined reference to `judge9'
mips-linux-gnu-ld: tyt.o: in function `judge5':
(.text+0x144): undefined reference to `print9'
sysu@debian:~$
```

这些问题也比较容易解决，于是经过debug，很快得到了能运行的程序。

（4）调试程序

然而这个程序虽然开始运行，但得不到想要的结果，于是开始用调试工具GDB对所写代码进行调试，按照实验指导上的方法设定好架构，链接端口后，开始对程序进行调试：

```
(gdb) disas loop
Dump of assembler code for function loop:
=> 0x0040013c <+0>:     lb      t6,0(t5)
   0x00400140 <+4>:     beq     t6,t0,0x40017c <print>
   0x00400144 <+8>:     nop
   0x00400148 <+12>:    beq     t6,t1,0x400158 <addone>
   0x0040014c <+16>:    nop
   0x00400150 <+20>:    beq     t6,t2,0x400168 <addtwo>
   0x00400154 <+24>:    nop
End of assembler dump.
```

```
(gdb) info r
          zero       at       v0       v1       a0       a1       a2       a3
 R0   00000000 00000000 00000fa4 00000000 00000000 004102a0 0000003c 00000000
           t0       t1       t2       t3       t4       t5       t6       t7
 R8   00000030 00000031 00000032 00000005 00000002 004102a4 00000030 00000004
           s0       s1       s2       s3       s4       s5       s6       s7
 R16  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
           t8       t9       k0       k1       gp       sp       s8       ra
 R24  00000000 00000000 00000000 00000000 00000000 40800750 00000000 00000000
           sr       lo       hi      bad    cause       pc
      20000010 00000000 00000000 00000000 00000000 0040017c
          fsr      fir
      00000000 00739300
```

像这样一步步进行，每一步都查看一下各个寄存器中的值，又发现了一些bug，其中有一处错误是我在汇编代码中误将$0写成了$1，这就是造成汇编时有warning产生的原因。

在调试的过程中，还发现了某些寄存器的值会莫名其妙地变化，这使我百思不得其解，后来突然意识到，程序有些地方并没有按照我的预期去跳转，又想起助教的提醒：每个跳转语句后面都要加nop，于是返回修改，完美解决了这个问题！

5. 实验结果及分析

最终得到了要求的程序结果，对不同数据的测试结果如下：

```
sysu@debian:~$ qemu-mips ./tyt1
Please enter the integers with zero to finish:
1 2 0
3
sysu@debian:~$ qemu-mips ./tyt1
Please enter the integers with zero to finish:
1 1 2 2 2 1 1 2 2 0
22
sysu@debian:~$ qemu-mips ./tyt1
Please enter the integers with zero to finish:
1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 0
100
sysu@debian:~$ qemu-mips ./tyt1
Please enter the integers with zero to finish:
1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 0
101
```

经过计算和分析，以上这些实验结果都是正确的，并且输出不同位数的结果也准确无误。除此之外，编译连接运行过程都符合要求和预期。

实验心得

在本次实验中，我遇到过许多问题：

一开始发现自己的虚拟机显示不了IP，PUTTY和Filezilla都连接不上，在当晚我毫无办法，心情烦躁，卸载重装了很多次，折腾到两点都没解决，于是第二天一早寻求助教的帮助，助教很快就发来了解决方案，完美处理了问题，多谢助教！这让我明白了遇到问题要善于表达和求助，不能自己死磕，否则事倍功半。

接下来写代码调试，一开始遇到语法错误都很轻松地解决了，但后来遇到了warning，自己开始看不懂，在网络上查找资料也无济于事，同时运行程序也没有得到想要的结果。我便一步步调试，每执行一行指令之后都查看一遍寄存器的内容，找到了出错的3个原因：一是我将$0误写成$1，正是这个原因导致了那条warning；而是有部分jump语句之后没有写nop，导致运行出错；三是输出时的逻辑问题，也在一步步调试中被发现。其中那个每个跳转指令之后都加nop是助教讲课时专门强调过的，但我写代码时却没有太注意，导致debug很久才反应过来。以后做实验时我一定要把助教讲过的内容先回忆一遍，防止出现类似错误。

【程序代码】

```
    .data    # 定义数据段，数据将存放于此
sentence1:   # 定义变量名，实际是一个地址
    .asciiz "Please enter the integers with zero to finish:\n"
endl:
    .asciiz "\n"          # 伪指令
str:                      # 存输入的字符串
    .space 60
string:
    .space 4              # 存输出的字符串
    .set noreorder        # 编译用伪指令，不关心

    .text                 # 代码段，可执行的代码存放于此
.global main              # 指出main为全局作用域符号
main:
    li $v0, 4004          # 调用号4004（write）放入$v0
    li $a0, 1             # 目标1号文件（标准输出）放入$a0
    la $a1, sentence1     # 待输出的字符串首地址放入$a1
    li $a2, 47            # 要输出字节数为47，放入$a2
    syscall               # 准备就绪，交付内核完成调用
    li $t0, 48            # '0'
    li $t1, 49            # '1'
```

```
    li $t2, 50              # '2'
    li $t3, 0               # 存放answer
    li $t4, 0               # $t4存放连续的2的个数

    li $v0, 4003            # 调用号4003（read）放入$v0
    li $a0, 0                 # 目标0号文件（标准输入）放入$a0
    la $a1, str             # $a1存放输入字符串str的首地址
    li $a2, 60              # 读取60个字节
    syscall                 # 准备就绪，交付内核完成调用
    la $t5, str             # $t5存放输入字符串str的首地址
loop:
    lb $t6, 0($t5)          # 将一个字节从内存取到寄存器$t6中
    beq $t6, $t0, print     # 如果是0，直接输出结果
    nop                     # important to make program happy
    beq $t6, $t1, addone    # 如果是1，跳转至addone
    nop
    beq $t6, $t2, addtwo    # 如果是2，跳转至addtwo
    nop
addone:
    addi $t3, $t3, 1        # $t3++
    li $t4, 0               # $t4清零
    addi $t5, $t5, 2        # $t5=$t5+2(跳过空格找下一个数)
    j loop                  # 无条件跳转
    nop
addtwo:
    addi $t4, $t4, 1        # $t4++
    add $t7, $t4, $t4       # $t7=2*$t4
    add $t3, $t3, $t7       # $t3=$t7+$t3
    addi $t5, $t5, 2        # 待读取的地址加两字节，跳过空格
    j loop
    nop
print:
    addi $a0, $t3, 0        # 传参
    jal printfunc           # 过程调用，jump and link
    nop

    li $v0, 4001            # 调用号4001(exit)放入$v0
    syscall                 # 准备就绪，交付内核完成调用


.global printfunc
printfunc:
    addi $t0, $a0, 0        # $t0=$a0
    li $t2, 2               # 需要的空间-1
```

```
    li $t5, 2               # 最后用来计算输出字符个数
    la $t3, string          # 最后一个字符的地址是$t3+$t2中
loop2:
    rem $t1, $t0, 10        # $t1存余数
    div $t0, $t0, 10        # $t0=$t0/10
    addi $t1, $t1, 48       # 转化为字符
    add $t4, $t3, $t2       # 存储地址
    sb $t1, 0($t4)          # 存字节
    addi $t2, $t2, -1       # 向前挪一个字节
    beq $t0, $zero, printstring   # 如果是0，跳出循环
    nop
    j loop2                 # 如果不为0，继续循环
    nop
printstring:
    sub $a2, $t5, $t2         # 要输出这么多字符
    addi $t2, $t2, 1          # 开始的数的顺序
    li $v0, 4004
    add $a1, $t3, $t2         # 开始地址
    li $a0, 1                 # 标准输出
    syscall
endline:                      # 换行
    li $v0, 4004
    li $a0, 1
    la $a1, endl
    li $a2, 1
    syscall

    jr $ra                    # 跳回寄存器中的地址
    nop
```