中山大学数据科学与计算机学院本科生实验报告

（2019 学年秋季学期）

课程名称：**计算机组成原理实验**　　任课教师：　　助教：

年级&班级	19 计科超算	专业(方向)	
学号	18324034	姓名	林天皓
电话		Email	linth5@mail2.sysu.edu.cn
开始日期	2020.9.25	完成日期	2020.10.2

一、 实验题目

实验 4 计算机结构与组成 MIPS 汇编程序

二、实验目的

本实验的目的是让你熟悉 MARS 仿真器的使用，学习用它来运行和调试程序。如果你需要在其他地方使用 MARS，可以直接下载 mars.jar。MARS 程序由密苏里州立大学开发。UCB 的教师消除了一些 BUG，但还没被主要开发者接受. MARS 程序是用 JAVA 写的，因此使用它之前，需要在你的机器上安装 Java J2SE 1.5.0 SDK 或者更高版本,该程序可以从 Sun 公司下载。

三、实验内容与

1 实验原理

通过 MARS4.5 程序解析与执行 mips 汇编程序，掌握基本汇编程序语句的作用，了解程序中如何设置断点与单步运行的调试方法，了解 mars 中系统调用 syscall 指令的对不同输出语句类型的调用，输出程序运行的结果。了解内存中对于不同类型的数据的存储方式，了解通过代码块的名字调用汇编代码的方法。

2. 实验步骤

第一部分：

使用 Help(问号图标)回答下列关于 MARS 的问题.

a. .data, .word, .text 指示器（directives）的含义是什么(即, 在每段中放入什么内容)?
b. 在 MARS 中如何设置断点 breakpoint? 请在第 15 行设置断点，并在所有问题解答完后，将此结果给老师检查。
c. 在程序运行到断点处停止时，如何继续执行? 如何单步调试代码?
d. 如何知道某个寄存器 register 的值是多少? 如何修改寄存器的值.
e. n 存储在内存中的哪个地址? 通过修改此内存处的值来计算第 13 个 fib 数.
f. 16 和 18 行使用了 syscall 指令. 其功能是什么，如何使用它? (提示: syscall 在 Help 中有说明!如何英文不是太好，可以一边运行，一边看效果，来体会其用途)

第二部分：

编写 MIPS 代码完成:在给定$s0 和 $s1 的值的前提下,将下列值放到 $t? 寄存器中(其中? 表示任意 0-7 之间的数）:

$t0 = $s0
$t1 = $s1
$t2 = $t0 + $t1
$t3 = $t1 + $t2
...
$t7 = $t5 + $t6

换言之, 对$t2 到 $t7 的每个寄存器, 都存储其前两个$t? 寄存器的值. 寄存器$s0 和 $s1 中包含初始值.

不要在代码中设置$s0 和 $s1 的值. 取而代之, 学会如何在 MARS 中手动设置它们的值.

将你的代码存储到文件 lab4_ex2.s 中, 然后给老师检查.

第三部分：

调试程序 lab4_ex3.s 中的循环. 该程序将从$a0 所指示的内存地址中复制一个整数到$a1 所指示的内存地址, 起到读入一个 zero 值时结束. 复制的整数的个数(不含 zero 值)应存储在

中$v0.

请在文件 lab4_ex3.txt 中描述代码的错误 bug(s). 新建一个文件 lab4_ex3_ok.s ，其中放的是没有 bug 的代码 lab4_ex3.s.把程序给老师看。

四、实验结果

第一部分：

回答：

a. 1.data subsequent items stored in data segment at next available asddress

在下一个可用地址存储在数据段中的后续项’

2..word Store the listed value(s) as 32 bit words on word boundary

将列出的值存储为32位的单词边界

3..text subsequent items(instructions) stored in Text segment at next available address

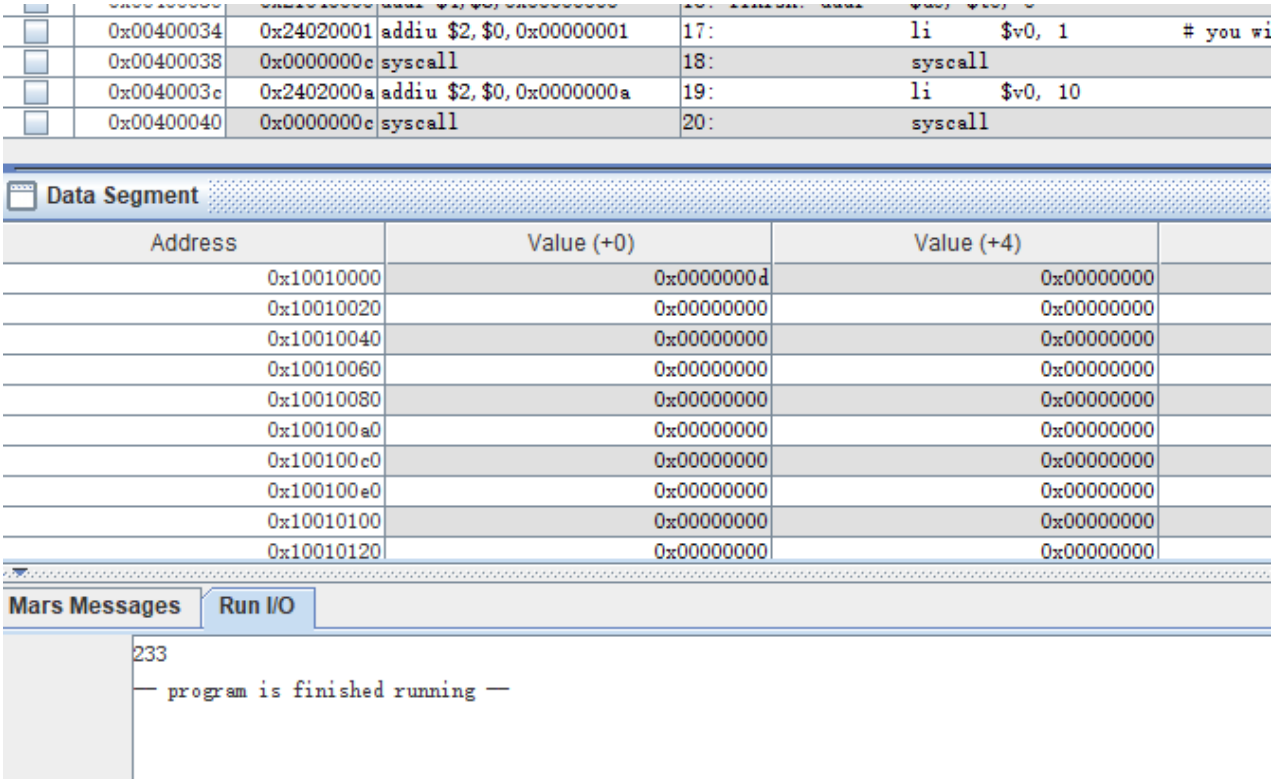
后续项目(指令)存储在下一个可用地址的文本段中

b.在Excute选项卡中勾选指令前方的Bkpt选框，即可设置断点。

c.运行到断点后，可以继续选择运行按钮继续运行，直到遇到下一个断点或者程序结束才停止。点击单步执行按钮即可单步执行，仅仅会执行一条指令。

d.可以在右侧Register选项卡中查看某个寄存器的值。双击寄存器对应的值可以编辑对应寄存器的值。

e.通过下方Data segment选项卡中的数据，n储存的地址为0x10010000.根据代码n存储的为求第n位fib数，通过在内存中将该位数改为0x00000000d即可使程序求第13个fib数的值。运行结果如 图



0x00400034	0x24020001	addiu $2, $0, 0x00000001	17:	li $v0, 1	# you wi
0x00400038	0x0000000c	syscall	18:	syscall	
0x0040003c	0x2402000a	addiu $2, $0, 0x0000000a	19:	li $v0, 10	
0x00400040	0x0000000c	syscall	20:	syscall	

Data Segment

Address	Value (+0)	Value (+4)
0x10010000	0x0000000d	0x00000000
0x10010020	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000

Mars Messages | Run I/O

233

— program is finished running —

图1-第13个fib数字运行结果

f.第 16行的syscall指令的作用为以整数的形式输出$t0，第18行syscall指令的作用为结束程序。

通过将需要操作的系统调用的数字放入$v0寄存器，再使用syscall调用。

第二部分：

原来给出的程序如下，

```
        .data
        .text
main:add $t0,$s0,$zero
   add $t1,$s1,$zero
   add $t2,$t0,$t1
   add $t3,$t1,$t2
   add $t4,$t2,$t3
   add $t5,$t3 , $t4
   add $t6,$t4 , $t5
   add $t7,$t5 , $t6
   j main
syscall
```

通过手动设置$s0,$s1寄存器的值，发现$t0-$t7寄存器能够正常计算与储存前两个寄存器的值，

只是j main指令导致程序无限循环无法结束。

修正方法：这里将倒数第二行的 j main指令替换为 addi $v0,$zero,10 后面的运行会调用syscall 正常结束程序。

修正后的代码：

```
        .data
        .text
main:add $t0,$s0,$zero
   add $t1,$s1,$zero
   add $t2,$t0,$t1
   add $t3,$t1,$t2
   add $t4,$t2,$t3
   add $t5,$t3 , $t4
   add $t6,$t4 , $t5
   add $t7,$t5 , $t6
   addi $v0,$zero,10
syscall
```

经运行检查，各寄存器的值符合题目所给出的要求。

第三部分：

原本代码如下

```
          .data
source:   .word   3, 1, 4, 1, 5, 9, 0
dest:     .word   0, 0, 0, 0, 0, 0, 0
countmsg: .asciiz " values copied. "

          .text

main:   la      $a0,source
        la      $a1,dest

loop:   lw      $v1, 0($a0)     # read next word from source
        addiu   $v0, $v0, 1     # increment count words copied
        sw      $v1, 0($a1)     # write to destination
        addiu   $a0, $a0, 1     # advance pointer to next source
        addiu   $a1, $a1, 1     # advance pointer to next dest
        bne     $v1, $zero, loop# loop if word copied not zero

loopend:
        move    $a0,$v0         # $a0 <- count
        jal     puti            # print it
```

```
        la      $a0,countmsg    # $a0 <- countmsg
        jal     puts            # print it

        li      $a0,0x0A        # $a0 <- '\n'
        jal     putc            # print it

finish:
        li      $v0, 10         # Exit the program
        syscall
```

该代码中存在两个bug，一是在loop中第4行和第5行（黄色标注），复制源地址与目标地址的指针加的时候仅仅加1，而不是一个word的长度4.，二是给出了跳转的puti，puts，putc却并没有实现。所以在这里需要我们按照syscall的作用补充puti，puts，putc的实现。

经过改正后的代码如下：

```
        .data
source:   .word   3, 1, 4, 1, 5, 9, 0
dest:     .word   0, 0, 0, 0, 0, 0, 0
countmsg: .asciiz " values copied. "

        .text

main:   la      $a0,source
        la      $a1,dest

loop:   lw      $v1, 0($a0)     # read next word from source
        addiu   $v0, $v0, 1     # increment count words copied
        sw      $v1, 0($a1)     # write to destination
        addiu   $a0, $a0, 4     # advance pointer to next source
        addiu   $a1, $a1, 4     # advance pointer to next dest
        bne     $v1, $zero, loop# loop if word copied not zero

loopend:
        move    $a0,$v0         # $a0 <- count
        jal     puti            # print it
puti:
        li      $v0, 1
        syscall

        la      $a0,countmsg    # $a0 <- countmsg
        jal     puts            # print it
puts:
        li      $v0, 4
```

```
        syscall

        li      $a0,0x0A        # $a0 <- '\n'
        jal     putc            # print it
putc:
        li      $v0, 11
        syscall
finish:
        li      $v0, 10         # Exit the program
        syscall
```

运行后输出结果如下图：

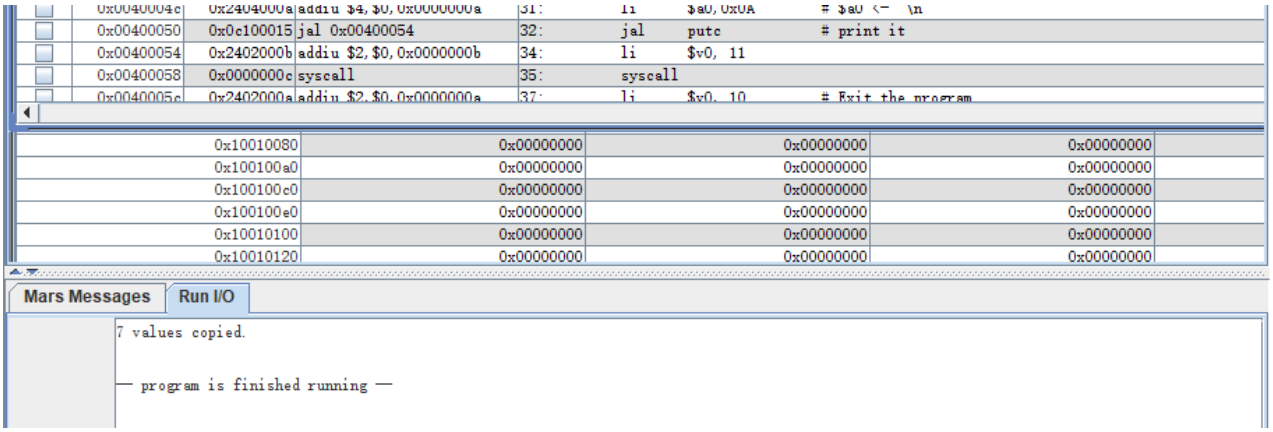


图2-debug后的代码运行结果

整数，字符串，回车字符均正常输出。完成了debug的预定任务

五、实验感想

本次实验是一次初步学习mips汇编代码的实验，通过mars程序执行mips汇编代码，可以直观的看到各个内存，寄存器中的值，直观的理解了汇编代码指令的执行过程。理解了寄存器和内存是如何通过汇编代码的调用进行数据的交换。经过翻译的指令可以看出，有的一句汇编代码经过翻译后可以生成多条指令，通过这种方式，在没有增加底层的硬件实现电路的情况下扩充了汇编指令。由于本次汇编的代码较为简单，没有涉及递归函数调用中的入栈出栈操作，今后将会继续学习mips汇编代码。

附录（流程图，注释过的代码）：

本次实验涉及到自己修改过的代码均在上文出现，此处不再重复。