**练习: 熟悉MARS**

## 编一个简短的MIPS程序

**练习 1**

假定你想编写一个MIPS程序foo，该程序使用5个字的数组，数组元素初始化为整数1, ..., 5.

.data

foo: .word 1,2,3,4,5

你用程序来把数组foo中的每个数加2再写回数组foo

.text

main:

la $t0, $0, foo

lw $t1,0($t0)

addiu $t1, $t1,2

sw $t1, 0($t0)

.....

你也可以使用循环来完成。

回答下列关于MARS的问题.

1. .data, .word, .text 指示器（directives）的含义是什么(即, 在每段中放入什么内容)?
2. 在MARS中如何设置断点breakpoint?
3. 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?
4. 如何知道某个寄存器register的值是多少? 如何修改寄存器的值.

将你的代码存储到文件lab\_ex.s 中，然后给老师检查.

**练习2**

**从键盘输入两个数，计算并输出这两个数的和**

.data

**str1: .asciiz "Enter 2 numbers:"**

**str2: .asciiz "The sum is "**

.text

main:

ori $v0, $0, 4 #System call code 4 for printing a string

la $a0, Str1 #address of Str1 is in $a0

syscall #print the string

ori $v0, $0, 5 #System call code 5 for read integer，$v0 contains integer read

**add $t0, $v0, $zero** #

….

ori $v0, $0, 4 #System call code 4 for printing a string

la $a0, Str2 #address of Str2 is in $a0

syscall #print the string

ori $v0, $0, 1 #System call code 4 for print integer，$a0 = integer to print

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ #calculate the sum

syscall #print the sum

exit: ori $v0, $0, 10 #System call code 10 for exit

syscall #print the sum

3. **计算12+22+…+1002 ,** 参照练习2输出结果

.data

**.align 2**

**Str: .asciiz "The sum of square from 1 to 100 is "**

4. 编写两个版本的first1pos (starting from [first1pos.s](http://xgxy.cug.edu.cn/rjgcx/lzw/COD/lab2/first1pos.txt)) 函数, 在$a0中给定一个数, 而在$v0 中返回$a0字中最左边的非零位的位置. 如果$a0 的值是0, 在$v0 中存-1 . 在查找此位置的过程中,允许你修改$a0 值.位置从0 (最右位)到31 (符号位). 其中一种解应该重复移位$a0, 每次移位后,检查符号位. 另一种方法是初始时使用0x80000000 作为掩码,并不断右移该掩码来检查$a0的每一位.

可以和你的同伴分别来做此工作, 其中一个同学做第一个版本,另一个同学做第二个版本. 工作完成后,向你的同伴解释你的代码, 然后把运行情况给老师检查.

[first1pos.s](http://xgxy.cug.edu.cn/rjgcx/lzw/COD/lab2/first1pos.txt)

main:

lui $a0,0x8000

jal first1pos

jal printv0

lui $a0,0x0001

jal first1pos

jal printv0

li $a0,1

jal first1pos

jal printv0

add $a0,$0,$0

jal first1pos

jal printv0

li $v0,10

syscall

first1pos: # your code goes here

Jr ra

printv0:

addi $sp,$sp,-4

sw $ra,0($sp)

add $a0,$v0,$0

li $v0,1

syscall

li $v0,11

li $a0,'\n'

syscall

lw $ra,0($sp)

addi $sp,$sp,4

jr $ra

**附:系统调用**

A number of system services, mainly for input and output, are available for use by your MIPS program. They are described in the table below.

MIPS register contents are not affected by a system call, except for result registers as specified in the table below.

### How to use SYSCALL system services

Step 1. Load the service number in register $v0.  
Step 2. Load argument values, if any, in $a0, $a1, $a2, or $f12 as specified.  
Step 3. Issue the SYSCALL instruction.  
Step 4. Retrieve return values, if any, from result registers as specified.

**Example: display the value stored in $t0 on the console**

li $v0, 1 # service 1 is print integer

add $a0, $t0, $zero # load desired value into argument register $a0, using pseudo-op

syscall

自己定义字符串并显示。

#---------------------------------

.data 0x0

startString: .asciiz "Converting pixels to grayscale:\n"

finishString: .asciiz "Finished."

newline: .asciiz "\n"

.text 0x3000

main:

ori $v0, $0, 4 #System call code 4 for printing a string

la $a0, startString #address of startString is in $a0

syscall #print the string

#------- INSERT YOUR CODE HERE -------

#

#(You may delete the comment here when you insert your code)

#

#------------ END CODE ---------------

exit:

ori $v0, $0, 4 #System call code 4 for printing a string

la $a0, finishString #address of finishString is in $a0; we computed this

#simply by counting the number of chars in startString,

#including the \n and the terminating \0

syscall #print the string

ori $v0, $0, 10 #System call code 10 for exit

syscall #exit the program

**# Read a string**

.data

m1: .asciiz "Read next line\n"

.text

.globl main

main:

li $v0 4 # syscall 4 (print\_str)

la $a0 m1

syscall

li $v0 8 # syscall 8 (read\_string)

syscall

li $v0 4 # syscall 4 (print\_str)

syscall

jr $ra

### Table of Available Services

|  |  |  |  |
| --- | --- | --- | --- |
| **Service** | **Code in $v0** | **Arguments** | **Result** |
| print integer | 1 | $a0 = integer to print |  |
| print float | 2 | $f12 = float to print |  |
| print double | 3 | $f12 = double to print |  |
| print string | 4 | $a0 = address of null-terminated string to print |  |
| read integer | 5 |  | $v0 contains integer read |
| read float | 6 |  | $f0 contains float read |
| read double | 7 |  | $f0 contains double read |
| read string | 8 | $a0 = address of input buffer $a1 = maximum number of characters to read | *See note below table* |
| sbrk (allocate heap memory) | 9 | $a0 = number of bytes to allocate | $v0 contains address of allocated memory |
| exit (terminate execution) | 10 |  |  |
| print character | 11 | $a0 = character to print | *See note below table* |
| read character | 12 |  | $v0 contains character read |
| open file | 13 | $a0 = address of null-terminated string containing filename $a1 = flags $a2 = mode | $v0 contains file descriptor (negative if error). *See note below table* |
| read from file | 14 | $a0 = file descriptor $a1 = address of input buffer $a2 = maximum number of characters to read | $v0 contains number of characters read (0 if end-of-file, negative if error). *See note below table* |
| write to file | 15 | $a0 = file descriptor $a1 = address of output buffer $a2 = number of characters to write | $v0 contains number of characters written (negative if error). *See note below table* |
| close file | 16 | $a0 = file descriptor |  |
| exit2 (terminate with value) | 17 | $a0 = termination result | *See note below table* |
| Services 1 through 17 are compatible with the SPIM simulator, other than Open File (13) as described in the Notes below the table. Services 30 and higher are exclusive to MARS. | | | |
| time (system time) | 30 |  | $a0 = low order 32 bits of system time $a1 = high order 32 bits of system time. *See note below table* |
| MIDI out | 31 | $a0 = pitch (0-127) $a1 = duration in milliseconds $a2 = instrument (0-127) $a3 = volume (0-127) | Generate tone and return immediately. *See note below table* |
| sleep | 32 | $a0 = the length of time to sleep in milliseconds. | Causes the MARS Java thread to sleep for (at least) the specified number of milliseconds. This timing will not be precise, as the Java implementation will add some overhead. |
| MIDI out synchronous | 33 | $a0 = pitch (0-127) $a1 = duration in milliseconds $a2 = instrument (0-127) $a3 = volume (0-127) | Generate tone and return upon tone completion. *See note below table* |
| print integer in hexadecimal | 34 | $a0 = integer to print | Displayed value is 8 hexadecimal digits, left-padding with zeroes if necessary. |
| print integer in binary | 35 | $a0 = integer to print | Displayed value is 32 bits, left-padding with zeroes if necessary. |
| print integer as unsigned | 36 | $a0 = integer to print | Displayed as unsigned decimal value. |
| (not used) | 37-39 |  |  |
| set seed | 40 | $a0 = i.d. of pseudorandom number generator (any int). $a1 = seed for corresponding pseudorandom number generator. | No values are returned. Sets the seed of the corresponding underlying Java pseudorandom number generator (java.util.Random). *See note below table* |
| random int | 41 | $a0 = i.d. of pseudorandom number generator (any int). | $a0 contains the next pseudorandom, uniformly distributed int value from this random number generator's sequence. *See note below table* |
| random int range | 42 | $a0 = i.d. of pseudorandom number generator (any int). $a1 = upper bound of range of returned values. | $a0 contains pseudorandom, uniformly distributed int value in the range 0 <= [int] < [upper bound], drawn from this random number generator's sequence. *See note below table* |
| random float | 43 | $a0 = i.d. of pseudorandom number generator (any int). | $f0 contains the next pseudorandom, uniformly distributed float value in the range 0.0 <= f < 1.0 from this random number generator's sequence. *See note below table* |
| random double | 44 | $a0 = i.d. of pseudorandom number generator (any int). | $f0 contains the next pseudorandom, uniformly distributed double value in the range 0.0 <= f < 1.0 from this random number generator's sequence. *See note below table* |
| (not used) | 45-49 |  |  |
| ConfirmDialog | 50 | $a0 = address of null-terminated string that is the message to user | $a0 contains value of user-chosen option 0: Yes 1: No 2: Cancel |
| InputDialogInt | 51 | $a0 = address of null-terminated string that is the message to user | $a0 contains int read $a1 contains status value 0: OK status -1: input data cannot be correctly parsed -2: Cancel was chosen -3: OK was chosen but no data had been input into field |
| InputDialogFloat | 52 | $a0 = address of null-terminated string that is the message to user | $f0 contains float read $a1 contains status value 0: OK status -1: input data cannot be correctly parsed -2: Cancel was chosen -3: OK was chosen but no data had been input into field |
| InputDialogDouble | 53 | $a0 = address of null-terminated string that is the message to user | $f0 contains double read $a1 contains status value 0: OK status -1: input data cannot be correctly parsed -2: Cancel was chosen -3: OK was chosen but no data had been input into field |
| InputDialogString | 54 | $a0 = address of null-terminated string that is the message to user $a1 = address of input buffer $a2 = maximum number of characters to read | *See Service 8 note below table* $a1 contains status value 0: OK status. Buffer contains the input string. -2: Cancel was chosen. No change to buffer.  -3: OK was chosen but no data had been input into field. No change to buffer. -4: length of the input string exceeded the specified maximum. Buffer contains the maximum allowable input string plus a terminating null. |
| MessageDialog | 55 | $a0 = address of null-terminated string that is the message to user $a1 = the type of message to be displayed: 0: error message, indicated by Error icon  1: information message, indicated by Information icon  2: warning message, indicated by Warning icon  3: question message, indicated by Question icon  other: plain message (no icon displayed) | N/A |
| MessageDialogInt | 56 | $a0 = address of null-terminated string that is an information-type message to user $a1 = int value to display in string form after the first string | N/A |
| MessageDialogFloat | 57 | $a0 = address of null-terminated string that is an information-type message to user $f12 = float value to display in string form after the first string | N/A |
| MessageDialogDouble | 58 | $a0 = address of null-terminated string that is an information-type message to user $f12 = double value to display in string form after the first string | N/A |
| MessageDialogString | 59 | $a0 = address of null-terminated string that is an information-type message to user $a1 = address of null-terminated string to display after the first string | N/A |

**实验报告要求：**

**设计思路说明**

**算法说明(必要时可以包含流程图)**

**程序清单(应当包含适当的注释)运行结果**