# Laboratory Exercise 5

## Character string with SYSCALL function, and sorting

### *Goals*

After this laboratory exercise, you should understand the mechanism of storing ASCII and Unicode string. You will be able to program to process string and put string to console. In addition, you should know how to sort a list of elements.

### *Literature*

Patterson, Henessy (COD): section 2.8, 2.13

### *Preparation*

Before you start the exercise, you should review the textbook, section 6.1 and read this laboratory carefully. You should also read the Mips Lab Environment Reference to find the usage of printf, putchar procedures … and so on.

### *About SYSCALL*

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

1. Load the service number in register $v0.
2. Load argument values, if any, in $a0, $a1, $a2, or $f12 as specified.
3. Issue the SYSCALL instruction.
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
li  $a0, 0x307      # the interger to be printed is 0x307
syscall             # execute
```

# Table of Frequently Available Services

| Service | Code in $v0 | Arguments | Result |
|---|---|---|---|
| **print decimal integer** | 1 | $a0 = integer to print | |
| **print string** | 4 | $a0 = address of null-terminated string to print | |
| **read integer** | 5 | | $v0 contains integer read |
| **read string** | 8 | $a0 = address of input buffer<br>$a1 = maximum number of characters to read | *See note below table* |
| **exit** | 10 | (terminate execution) | |
| **print character** | 11 | $a0 = character to print | *See note below table* |
| **read character** | 12 | | $v0 contains character read |
| **open file** | 13 | $a0 = address of null-terminated string containing filename<br>$a1 = flags<br>$a2 = mode | $v0 contains file descriptor (negative if error). *See note below table* |
| **read from file** | 14 | $a0 = file descriptor<br>$a1 = address of input buffer<br>$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 | $a0 = file descriptor<br>$a1 = address of output buffer<br>$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* |
| **time (system time)** | 30 | | $a0 = low order 32 bits of system time<br>$a1 = high order 32 bits of system time. *See note below table* |
| **MIDI out** | 31 | $a0 = pitch (0-127)<br>$a1 = duration in milliseconds<br>$a2 = instrument (0-127)<br>$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 | $a0 = pitch (0-127)<br>$a1 = duration in milliseconds<br>$a2 = instrument (0-127)<br>$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 | $a0 = i.d. of pseudorandom number generator (any int).<br>$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 | $a0 = i.d. of pseudorandom number generator (any int).<br>$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* |
| **ConfirmDialog** | 50 | $a0 = address of null-terminated string that is the message to user | $a0 contains value of user-chosen option<br>0: Yes<br>1: No<br>2: Cancel |
| **InputDialogInt** | 51 | $a0 = address of null-terminated string that is the message to user | $a0 contains int read<br>$a1 contains status value<br>0: OK status<br>-1: input data cannot be correctly parsed<br>-2: Cancel was chosen<br>-3: OK was chosen but no data had been input into field |
| **InputDialogString** | 54 | $a0 = address of null-terminated string that is the message to user<br>$a1 = address of input buffer<br>$a2 = maximum number of characters to read | *See Service 8 note below table*<br>$a1 contains status value<br>0: OK status. Buffer contains the input string.<br>-2: Cancel was chosen. No change to buffer.<br>-3: OK was chosen but no data had been input into field. No change to buffer.<br>-4: length of the input string exceeded the specified maximum. Buffer contains the maximum allowable input string plus a terminating null. |
| **MessageDialog** | 55 | $a0 = address of null-terminated string that is the message to user<br>$a1 = the type of message to be displayed:<br>0: error message, indicated by Error icon<br>1: information message, indicated by Information icon<br>2: warning message, indicated by Warning icon | N/A |

| | | | |
|---|---|---|---|
| | | 3: question message, indicated by Question icon<br>other: plain message (no icon displayed) | |
| **MessageDialogInt** | 56 | $a0 = address of null-terminated string that is an information-type message to user<br>$a1 = int value to display in string form after the first string | N/A |
| **MessageDialogString** | 59 | $a0 = address of null-terminated string that is an information-type message to user<br>$a1 = address of null-terminated string to display after the first string | N/A |

## 1. print decimal integer

print an integer to standard output (the console).

*Argument(s):*

$v0 = 1
$a0 = number to be printed

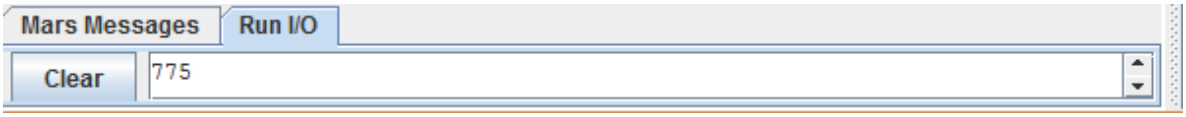
*Return value:*

none

*Example:*

```
li  $v0, 1          # service 1 is print integer
li  $a0, 0x307      # the interger to be printed is 0x307
syscall             # execute
```

*and result is*




## 2. MessageDialogInt

show an integer to a information-type message dialog.

*Argument(s):*

$v0 = 56
$a0 = address of the null-terminated message string
$a1 = int value to display in string form after the first string

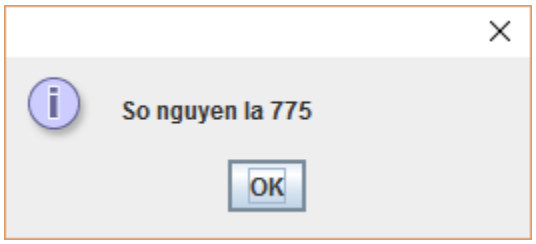
*Return value:*

none

*Example:*

```
.data
Message: .asciiz "So nguyen la "
.text
   li   $v0, 56
   la   $a0, Message
   li   $a1, 0x307        # the interger to be printed is 0x307
   syscall                # execute
```

*and result is*




## 3. print string

Formatted print to standard output (the console).

*Argument(s):*

$v0 = 1
$a0 = value to be printed

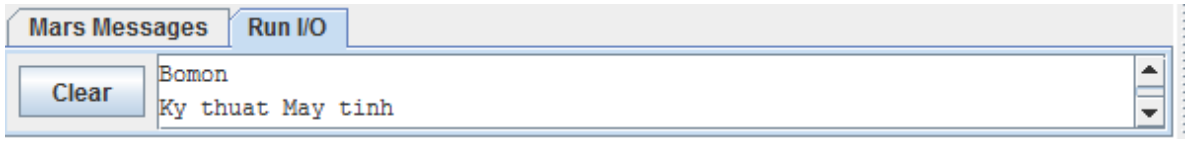
*Return value:*

none

*Example:*

```
.data
Message: .asciiz "Bomon \nKy thuat May tinh"
.text
   li   $v0, 4
   la   $a0, Message
   syscall
```

*and result is*




## 4. MessageDialogString

Show a string to a information-type message dialog

*Argument(s):*

$v0 = 59
$a0 = address of the null-terminated message string
$a1 = address of null-terminated string to display

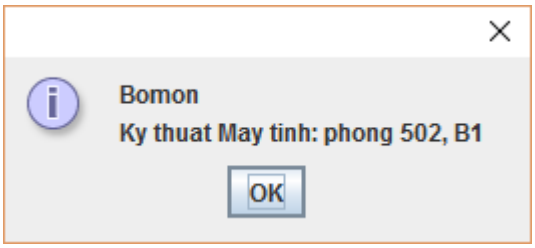
*Return value:*

none

*Example:*

```
.data
Message: .asciiz "Bomon \nKy thuat May tinh:"
Address: .asciiz " phong 502, B1"
.text
   li  $v0, 59
   la  $a0, Message
   la  $a1, Address
   syscall
```

*and result is*




## 5. read integer

Get a integer from standard input (the keyboard).

*Argument(s):*

$v0 = 5

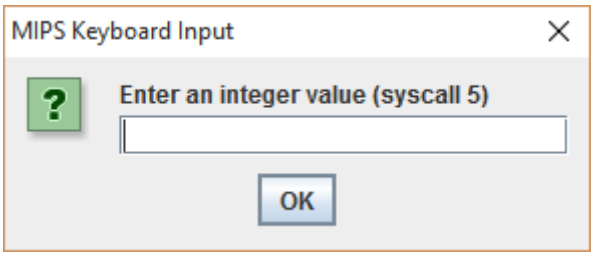
*Return value:*

$v0 = contains integer read

*Example:*

```
    li     $v0, 5
    syscall
```

*and result is*




## 6. InputDialogInt

Show a message dialog to read a integer with content parser

*Argument(s):*

$v0 = 51

$a0 = address of the null-terminated message string

*Return value:*

$a0 = contains int read

$a1 contains status value

0: OK status

-1: input data cannot be correctly parsed

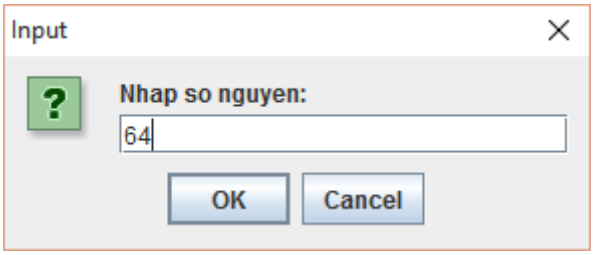
-2: Cancel was chosen

-3: OK was chosen but no data had been input into field

*Example:*

```
.data
Message: .asciiz "Nhap so nguyen:”
.text
   li   $v0, 51
   la   $a0, Message
   syscall
```

*and result is*




## 7. read string

Get a string from standard input (the keyboard).

*Argument(s):*

$v0 = 8
$a0 = address of input buffer
$a1 = maximum number of characters to read

*Return value:*

none

*Remarks:*

For specified length n, string can be no longer than n-1.

- If less than that, adds newline to end.
- In either case, then pads with null byte

Just in special cases:

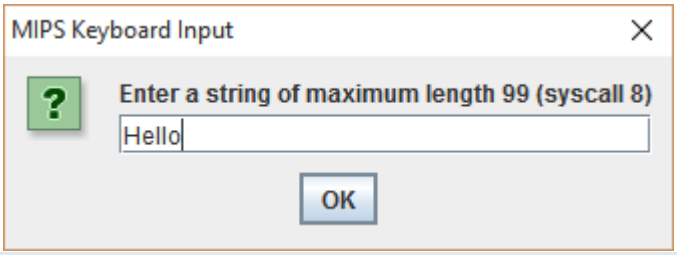
If n = 1, input is ignored and null byte placed at buffer address.

If n < 1, input is ignored and nothing is written to the buffer.

*Example:*

```
.data
Message: .space 100    # Buffer 100 byte chua chuoi ki tu can
.text
   li  $v0, 8
   la  $a0, Message
   li  $a1, 100
   syscall
```

*and result is*




Data Segment

| Address | Value (+0) | Value (+4) | Value (+8) | Value (+c) | Value (+10) |
|---|---|---|---|---|---|
| 0x10010000 | l l e H | \0 \0 \n o | \0 \0 \0 \0 | \0 \0 \0 \0 | \0 \0 \0 \0 |
| 0x10010020 | \0 \0 \0 \0 | \0 \0 \0 \0 | \0 \0 \0 \0 | \0 \0 \0 \0 | \0 \0 \0 \0 |
| 0x10010040 | \0 \0 \0 \0 | \0 \0 \0 \0 | \0 \0 \0 \0 | \0 \0 \0 \0 | \0 \0 \0 \0 |
| 0x10010060 | \0 \0 \0 \0 | \0 \0 \0 \0 | \0 \0 \0 \0 | \0 \0 \0 \0 | \0 \0 \0 \0 |

## 8. InputDialogString

Show a message dialog to read a string with content parser

*Argument(s):*

$v0 = 54
$a0 = address of the null-terminated message string
$a1 = address of input buffer
$a2 = maximum number of characters to read

*Return value:*

$a1 contains status value

0: OK status

-2: OK was chosen but no data had been input into field. No change to buffer.

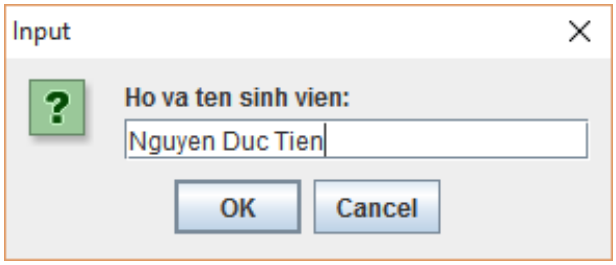
-3: OK was chosen but no data had been input into field

-4: length of the input string exceeded the specified maximum. Buffer contains the maximum allowable input string plus a terminating null.

*Example:*

```
.data
Message: .asciiz "Nhap so nguyen:”
string:  .space   100
.text
   li    $v0, 54
   la    $a0, Message
   la    $a1, string
   la    $a2, 100
   syscall
```

*and result is*



## 9. print character

Print a character to standard output (the console).

*Argument(s):*

$v0 = 11

$a0 = character to print (at the lowest significant byte)

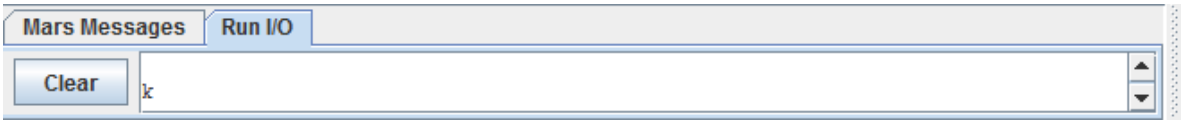
*Return value:*

none

*Example:*

```
li  $v0, 11
li  $a0, 'k'
syscall
```

*and result is*




## 10. read character

Get a character from standard output (the keyboard).

*Argument(s):*

$v0 = 12

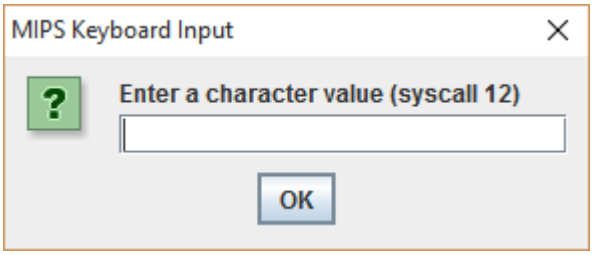
*Return value:*

$v0 contains character read

*Example:*

```
li  $v0, 12
syscall
```

*and result is*




## 11. ConfirmDialog

Show a message bog with 3 button: Yes | No | Cancel

*Argument(s):*

$v0 = 50

$a0 = address of the null-terminated message string

*Return value:*

$a0 = contains value of user-chosen option

0: Yes

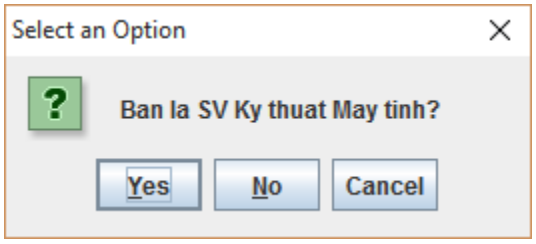
1: No

2: Cancel

*Example:*

```
.data
Message: .asciiz "Ban la SV Ky thuat May tinh?"
.text
   li   $v0, 50
   la   $a0, Message
   syscall
```

*and result is*




## 12. MessageDialog

Show a message bog with icon and button OK only

*Argument(s):*

$v0 = 55
$a0 = address of the null-terminated message string
$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)

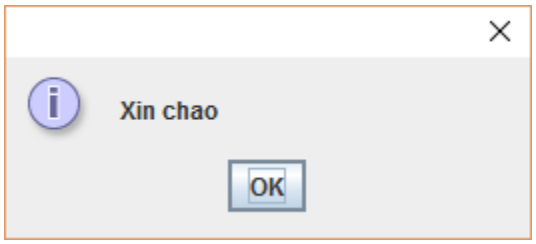
*Return value:*

none

*Example:*

```
.data
Message: .asciiz "Xin chao"
.text
   li   $v0, 55
   la   $a0, Message
   syscall
```

*and result is*



## 13. MIDI out

Make a sound
*Argument(s):*

$v0 = 31
$a0 = pitch (0-127)
$a1 = duration in milliseconds
$a2 = instrument (0-127)
$a3 = volume (0-127)

*Return value:*

Generate tone and return immediately

*Example:*

```
li $v0, 33
li $a0, 42    #pitch
li $a1, 2000  #time
li $a2, 0     #musical instrusment
li $a3, 212   #volume
```

## 14. MIDI out synchronous

Make a sound
*Argument(s):*

$v0 = 33
$a0 = pitch (0-127)
$a1 = duration in milliseconds
$a2 = instrument (0-127)
$a3 = volume (0-127)

*Return value:*

Generate tone and return upon tone completion

*Example:*

```
li $v0, 33
li $a0, 42    #pitch
li $a1, 2000  #time
li $a2, 0     #musical instrusment
li $a3, 212   #volume
syscall
```

## 15. Exit

Terminated the software. Make sense that there is no EXIT instruction in the Instruction Set of any processors. Exit is a service belongs to Operating System.
*Argument(s):*

$v0 = 10

*Return value:*

none

*Example:*

```
li        $v0, 10    #exit
syscall
```

### 16. Exit with code

Terminated the software. Make sense that there is no EXIT instruction in the Instruction Set of any processors. Exit is a service belongs to Operating System.

*Argument(s):*

$v0 = 17
$a0 = termination result

*Return value:*

none

*Example:*

```
   li      $v0, 17      # exit
   li      $a0, 3       # with error code = 3
   syscall
```

## *Assignments at Home and at Lab*

### Home Assignment 1

The following simple assembly program will display a welcome string. We use printf function for this purpose. Read this example carefully, pay attention to the way to pass parameters for printf function. Read Mips Lab Enviroment Reference for details.

```
#Laboratory Exercise 5, Home Assignment 1
.data
test: .asciiz "Hello World"
.text
   li  $v0, 4
   la  $a0, test
   syscall
```

### Home Assignment 2

Procedure strcpy copies string y to string x using the null byte termination convention of C. Read this example carefully, try to understand all of this code section.

```
#Laboratory Exercise 5, Home Assignment 2
.data
x: .space 1000                    # destination string x, empty
y: .asciiz "Hello"                # source string y

.text
strcpy:
      add   $s0,$zero,$zero        #s0 = i=0
L1:
      add   $t1,$s0,$a1            #t1 = s0 + a1 = i + y[0]
                                   #   = address of y[i]
      lb    $t2,0($t1)             #t2 = value at t1 = y[i]
      add   $t3,$s0,$a0            #t3 = s0 + a0 = i + x[0]
                                   #   = address of x[i]
      sb    $t2,0($t3)             #x[i]= t2 = y[i]
      beq   $t2,$zero,end_of_strcpy #if y[i]==0, exit
      nop
```

```
      addi   $s0,$s0,1                #s0=s0 + 1 <-> i=i+1
      j      L1                       #next character
      nop
end_of_strcpy:
```

## Home Assignment 3

The following program count the length of a null-terminated string. Read this example carefully, analyse each line of code.

```
#Laboratory Exercise 5, Home Assignment 3
.data
string:       .space   50
Message1:     .asciiz "Nhap xau:"
Message2:     .asciiz "Do dai la "
.text
main:
get_string:    # TODO

get_length:    la   $a0, string           # a0 = Address(string[0])
               xor  $v0, $zero, $zero     # v0 = length = 0
               xor  $t0, $zero, $zero     # t0 = i = 0
check_char:    add  $t1, $a0, $t0         # t1 = a0 + t0
                                          #= Address(string[0]+i)
               lb   $t2, 0($t1)           # t2 = string[i]
               beq  $t2,$zero,end_of_str  # Is null char?
               addi $v0, $v0, 1           # v0=v0+1->length=length+1
               addi $t0, $t0, 1           # t0=t0+1->i = i + 1
               j    check_char
end_of_str:
end_of_get_length:
print_length: # TODO
```

$t0 = 0 = i

## Assignment 1

Create a new project to implement the program in Home Assignment 1. Compile and upload to simulator. Run and observe the result. Go to data memory section, check how test string are stored and packed in memory.

## Assignment 2

Create a new project to print the sum of two register $s0 and $s1 according to this format:

"The sum of (s0) and (s1) is (result)"

## Assignment 3

Create a new project to implement the program in Home Assignment 2. Add more instructions to assign a test string for y variable, and implement *strcpy* function. Compile and upload to simulator. Run and observe the result.

## Assignment 4

Accomplish the Home Assignment 3 with syscall function to get a string from dialog, and show the length to message dialog.

## Assignment 5

Write a program that let user input a string. Input process will be terminated when user press Enter or then length of the string exceed 20 characters. Print the reverse string.

## *Conclusions*

Before you pass the laboratory exercise, think about the questions below:

- What the difference between the string in C and Java?
- In C, with 8 bytes, how many characters that we can store?
- In Java, with 8 bytes, how many characters that we can store?