College of Computing at AlGunfudha *Computer Science Department* *Semester 2, 1441/1442*	كلية الحاسب الآلي بالقنفذة Computing College at AlQunfudah	*Course: Computer organization and Architecture*

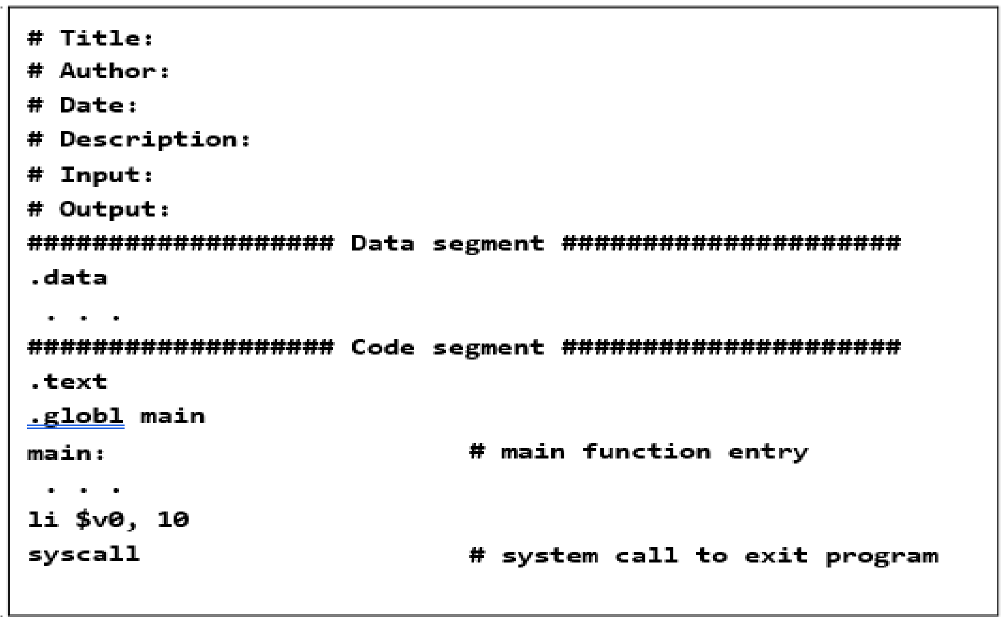
Lab 2: Introduction to MIPS Assembly Programming

Objectives :

After completing this lab, you will:

- Write simple MIPS programs
- Use system calls for simple input and output

MIPS Assembly Language Program Template



```
# Title:
# Author:
# Date:
# Description:
# Input:
# Output:
#################### Data segment ######################
.data
 . . .
#################### Code segment ######################
.text
.globl main
main:                           # main function entry
 . . .
li $v0, 10
syscall                         # system call to exit program
```

Figure 2.1: MIPS Assembly Language Program Template

There are three types of statements that can be used in assembly language, where each statement appears on a separate line:

1. *Assembler directives*: These provide information to the assembler while translating a program. Directives are used to define segments and allocate space for variables in memory. An assembler directive always starts with a dot. A typical MIPS assembly language program uses the following directives:

`.data`	Defines the data segment of the program, containing the program's variables.
`.text`	Defines the code segment of the program, containing the instructions.
`.globl`	Defines a symbol as global that can be referenced from other files.

2. *Executable Instructions*: These generate machine code for the processor to execute at runtime. Instructions tell the processor what to do.

3. *Pseudo-Instructions and Macros*: Translated by the assembler into real instructions. These simplify the programmer task.

In addition there are comments. Comments are very important for programmers, but ignored by the assembler. A comment begins with the **#** symbol and terminates at the end of the line. Comments can appear at the beginning of a line, or after an instruction. They explain the program purpose, when it was written, revised, and by whom. They explain the data and registers used in the program, input, output, the instruction sequence, and algorithms used.

MIPS Instructions

The general assembly language syntax of a MIPS instruction is:
`[label:] mnemonic [operands] [# comment]`

The **`label`** is optional. It marks the memory address of the instruction. It must have a colon. In addition, a **`label`** can be used for referring to the address of a variable in memory.
The **`mnemonic`** specifies the operation: **`add`**, **`sub`**, etc.
The **`operands`** specify the data required by the instruction. Different instructions have different number of operands. Operands can be registers, memory variables, or constants. Most arithmetic and logical instructions have three operands.

To be able to write programs, a basic set of instructions is needed. Only few instructions are described in the following tables. Table 2.1 lists the basic arithmetic instructions and Table 2.2 lists basic control instructions.

Instruction	Meaning
`add  Rd, Rs, Rt`	`Rd = Rs + Rt`. Overflow causes an exception.
`sub  Rd, Rs, Rt`	`Rd = Rs - Rt`. Overflow causes an exception.
`addi Rt, Rs, Imm`	`Rt = Rs + Imm` (16-bit constant). Overflow causes an exception.
`li   Rt, Imm`	`Rt = Imm` (pseudo-instruction).
`la   Rt, var`	`Rt = address of var` (pseudo-instruction).
`move Rd, Rs`	`Rd = Rs` (pseudo-instruction).

Table 2.1 the basic arithmetic instructions

Instruction	Meaning
`beq Rs, Rt, label`	if `(Rs == Rt)` branch to `label`.
`bne Rs, Rt, label`	if `(Rs != Rt)` branch to `label`.
`j   label`	Jump to `label`.

Table 2.2 basic control instructions.

System Calls

Programs do input and output using system calls. On a real-system, the operating system provides system call services to application programs. The MIPS architecture provides a special **`syscall`** instruction that generates a system call exception, which is handled by the operating system.

System calls are operating-system specific. Each operating system provides its own set of system calls. Because MARS is a simulator, there is no operating system involved. The MARS simulator handles the **`syscall`** exception and provides system services to programs. Table 2.3 shows a small set of services provided by MARS for doing basic I/O.

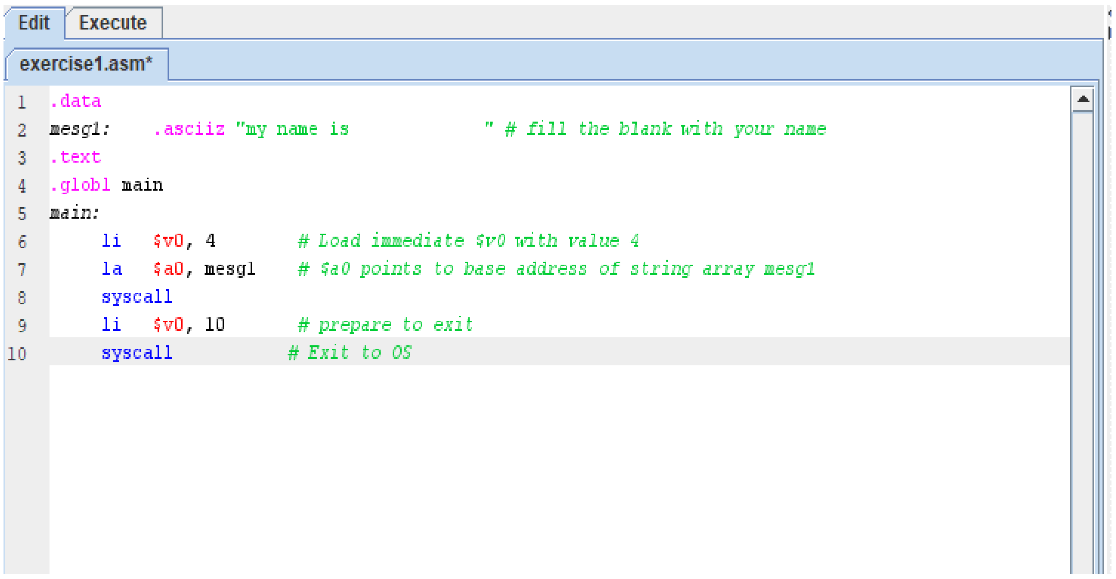
Before using the **`syscall`** instruction, you should load the service number into register **`$v0`**, and load the arguments, if any, into registers **`$a0`**, **`$a1`**, etc. After issuing the **`syscall`** instruction, you should retrieve return values, if any, from register **`$v0`**.

Service	$v0	Arguments / Result
Print Integer	1	$a0 = integer value to print
Print Float	2	$f12 = float value to print
Print Double	3	$f12 = double value to print
Print String	4	$a0 = address of null-terminated string
Read Integer	5	Return integer value in $v0
Read Float	6	Return float value in $f0
Read Double	7	Return double value in $f0
Read String	8	$a0 = address of input buffer $a1 = maximum number of characters to read
Allocate Heap memory	9	$a0 = number of bytes to allocate Return address of allocated memory in $v0
Exit Program	10	
Print Char	11	$a0 = character to print
Read Char	12	Return character read in $v0
Open File	13	$a0 = address of null-terminated filename string $a1 = flags (0=read, 1=write, 9=append) $a2 = mode (ignored) Return file descriptor in $v0 (negative if error)
Read from File	14	$a0 = File descriptor $a1 = address of input buffer $a2 = maximum number of characters to read Return number of characters read in $v0
Write to File	15	$a0 = File descriptor $a1 = address of buffer $a2 = number of characters to write Return number of characters written in $v0
Close File	16	$a0 = File descriptor

Table 2.4: Basic System Call Services Provided by MARS.

Exercise 1: Printing your Name using SYSCALL

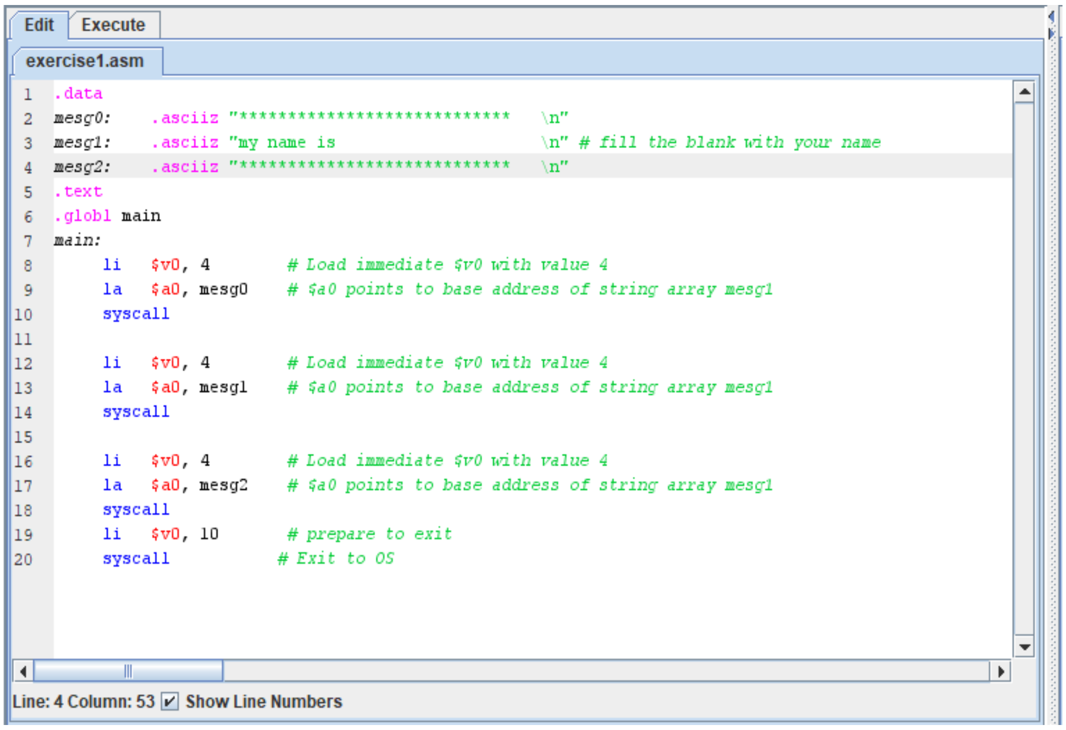
Use the following code to print your name on an output screen.



```
.data
mesg1:    .asciiz "my name is             " # fill the blank with your name
.text
.globl main
main:
     li   $v0, 4       # Load immediate $v0 with value 4
     la   $a0, mesg1   # $a0 points to base address of string array mesg1
     syscall
     li   $v0, 10      # prepare to exit
     syscall          # Exit to OS
```

Exercise 2: Modify the above program to print your name between 2 lines a follow:

```
*********************************************

   My name is xxxxxxxxxxxxxxxxxx

**********************************************
```

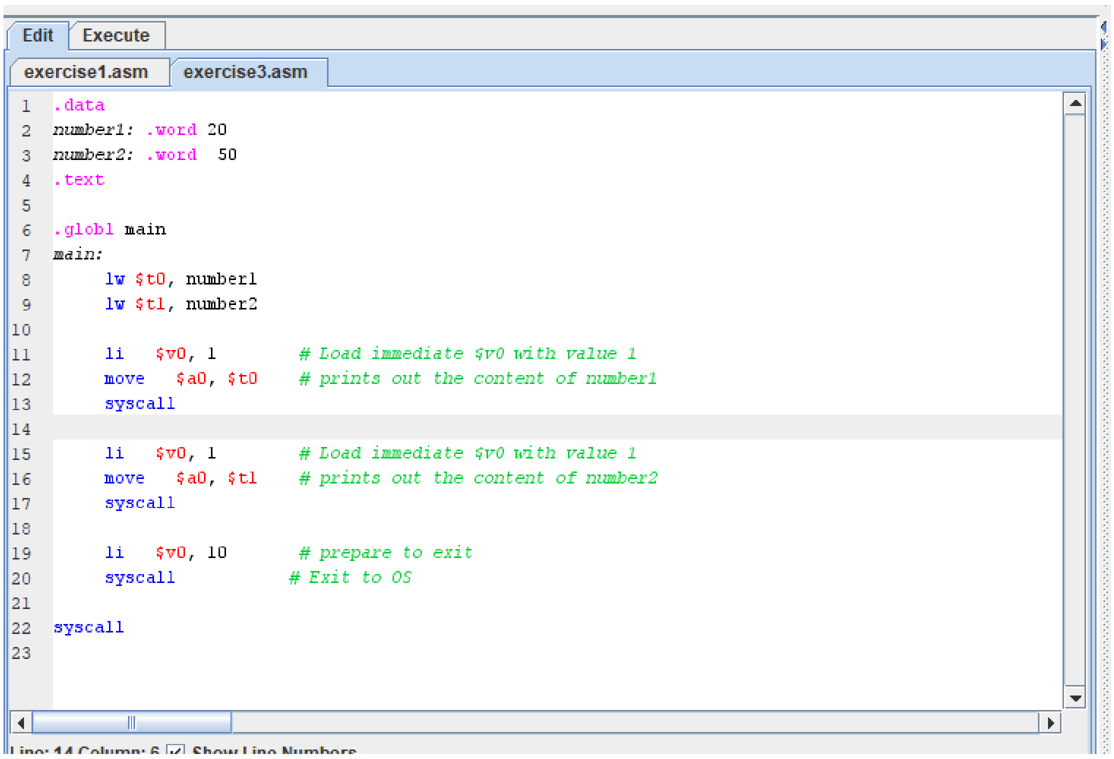


```
.data
mesg0:    .asciiz "****************************     \n"
mesg1:    .asciiz "my name is                       \n" # fill the blank with your name
mesg2:    .asciiz "****************************     \n"
.text
.globl main
main:
     li   $v0, 4       # Load immediate $v0 with value 4
     la   $a0, mesg0   # $a0 points to base address of string array mesg1
     syscall

     li   $v0, 4       # Load immediate $v0 with value 4
     la   $a0, mesg1   # $a0 points to base address of string array mesg1
     syscall

     li   $v0, 4       # Load immediate $v0 with value 4
     la   $a0, mesg2   # $a0 points to base address of string array mesg1
     syscall
     li   $v0, 10      # prepare to exit
     syscall          # Exit to OS
```

Exercise 3:

College of Computing at AlGunfudha *Computer Science Department* *Semester 2, 1441/1442*		*Course: Computer organization and Architecture*

Write a new program to print the content of 2 integers (number1, number2)



```
.data
number1: .word 20
number2: .word  50
.text

.globl main
main:
     lw $t0, number1
     lw $t1, number2

     li   $v0, 1        # Load immediate $v0 with value 1
     move   $a0, $t0    # prints out the content of number1
     syscall

     li   $v0, 1        # Load immediate $v0 with value 1
     move   $a0, $t1    # prints out the content of number2
     syscall

     li   $v0, 10       # prepare to exit
     syscall           # Exit to OS

syscall
```

Exercise 4:

Write a MIPS assembly language program contains 3 messages : (2 strings and 1 integer):

- Messg1 : "My name is : *************** "
- Messg2: "My age is :"
- Age : "   "