SYSTEM CALLS

MIPS Assembly

Systems Calls

INPUT/OUTPUT

MIPS:Input/Output

`Syscall`: Functions available in MARS

- A number of system services, mainly for input and output, are available for use by the MIPS Assembly.

To read about system services:
https://courses.missouristate.edu/KenVollmar/mars/Help/SyscallHelp.html

System Call for exit of a program

```
li $v0, 10
```

SystemCall: li $v0,10

```
        .text
        .globl main
main:
        li   $t0,1
        li   $t1,3

        li   $t2,4
        add  $t6,$t1,$t0
        add  $t7,$t6,$t2
        li   $v0,10
        syscall
```

$t7=?

“More …”

System **Call**s

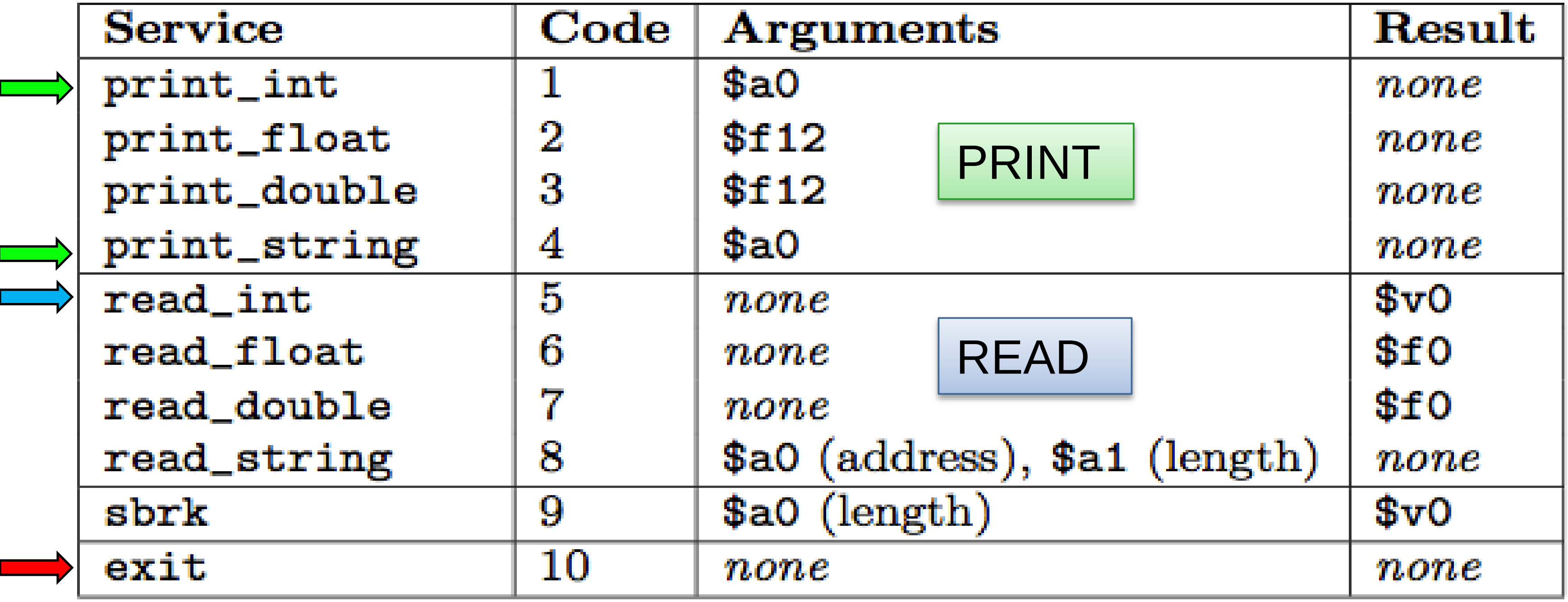
Sys Calls

PRINT

READ

EXIT

Sys Calls (Syscall)



Service	Code	Arguments	Result
print_int	1	$a0	*none*
print_float	2	$f12	*none*
print_double	3	$f12	*none*
print_string	4	$a0	*none*
read_int	5	*none*	$v0
read_float	6	*none*	$f0
read_double	7	*none*	$f0
read_string	8	$a0 (address), $a1 (length)	*none*
sbrk	9	$a0 (length)	$v0
exit	10	*none*	*none*

PRINT

READ

- «**Service**» explains the function of the `syscall` code
- «**Code**» is the number to be loaded
- «**Arguments**» states the arguments used and where specifically they'd be located
- «**Result**» explains the output

How to use `syscall` services

1. Load the service number in register: `$v0`
2. Load argument values, if any, in registers: `[$a0], [$a1], [$a2]`
3. Issue the `syscall` instruction
4. Retrieve return values, if any, from the used registers.

`la` (**`l`**oad **`a`**ddress)

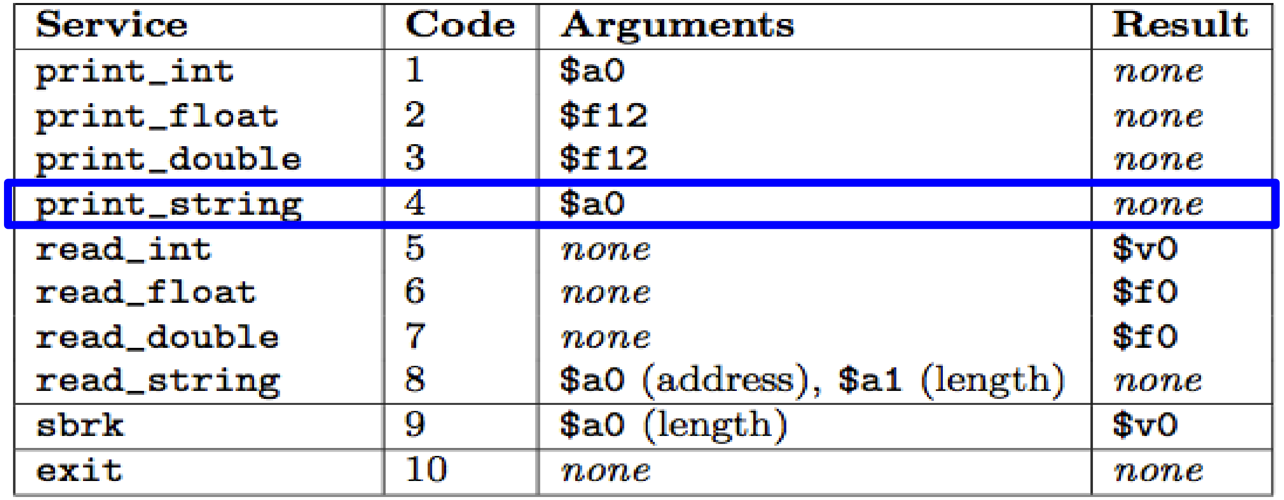
New MIPS Assembly instruction

la instruction (load address)

```
la $a0, msg          # Load the address (msg) of a string of text
```

```
.msg: .asciiz “hello world!"
```

Print out a string of text



Service	Code	Arguments	Result
print_int	1	$a0	*none*
print_float	2	$f12	*none*
print_double	3	$f12	*none*
print_string	4	$a0	*none*
read_int	5	*none*	$v0
read_float	6	*none*	$f0
read_double	7	*none*	$f0
read_string	8	$a0 (address), $a1 (length)	*none*
sbrk	9	$a0 (length)	$v0
exit	10	*none*	*none*

`Syscall` → `4` (argument = `$a0`)

Print a string of text

System Calls:

```
4
10
```

Example-1

Print “out” a string of text

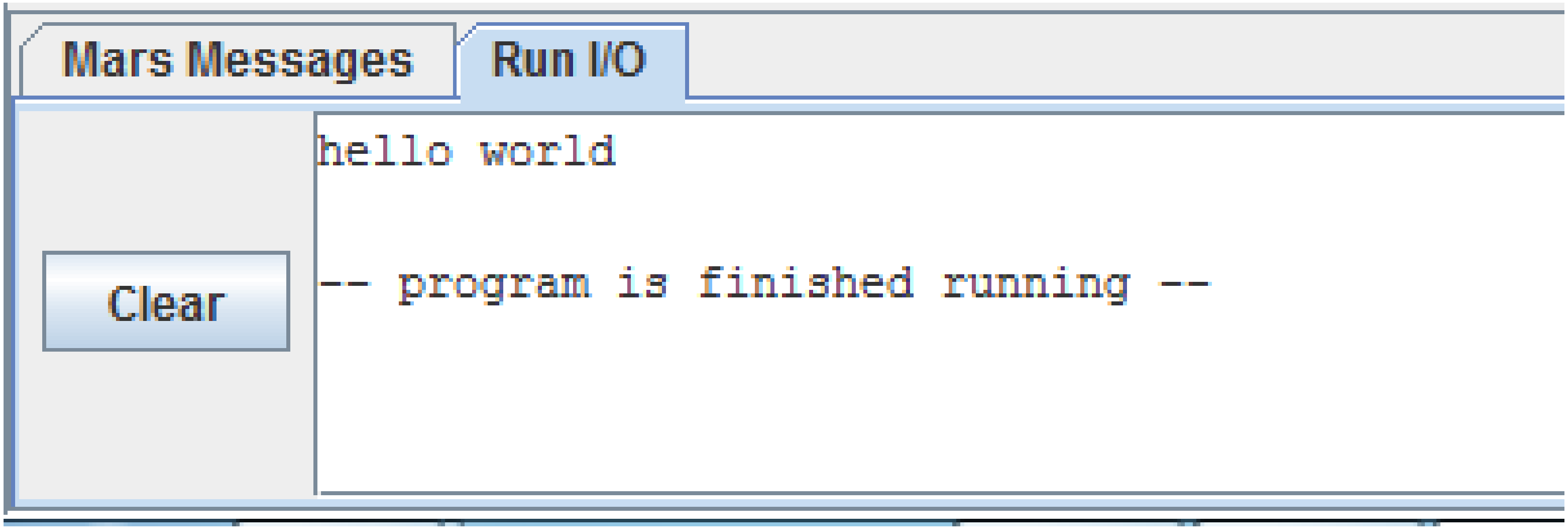
```
        .text
        .globl main

main:
        la      $a0, msg
        li      $v0, 4
        syscall

        li      $v0, 10
        syscall

        .data
msg:    .asciiz "hello world!"
```

Print “out” to console string of text

```
#  code-4.asm

       .text
       .globl main
                          # The following are to be assembled in to text segment
main:
       la     $a0, msg    # Load the address of the message text
       li     $v0, 4      # Load the syscall (4) code for printing the string of text
       syscall
       li     $v0, 10     # Load the syscall (10) code for exiting
       syscall

       .data              # Informs the assembler that data needed within instructions follows
msg:   .asciiz            “hello world! "
```

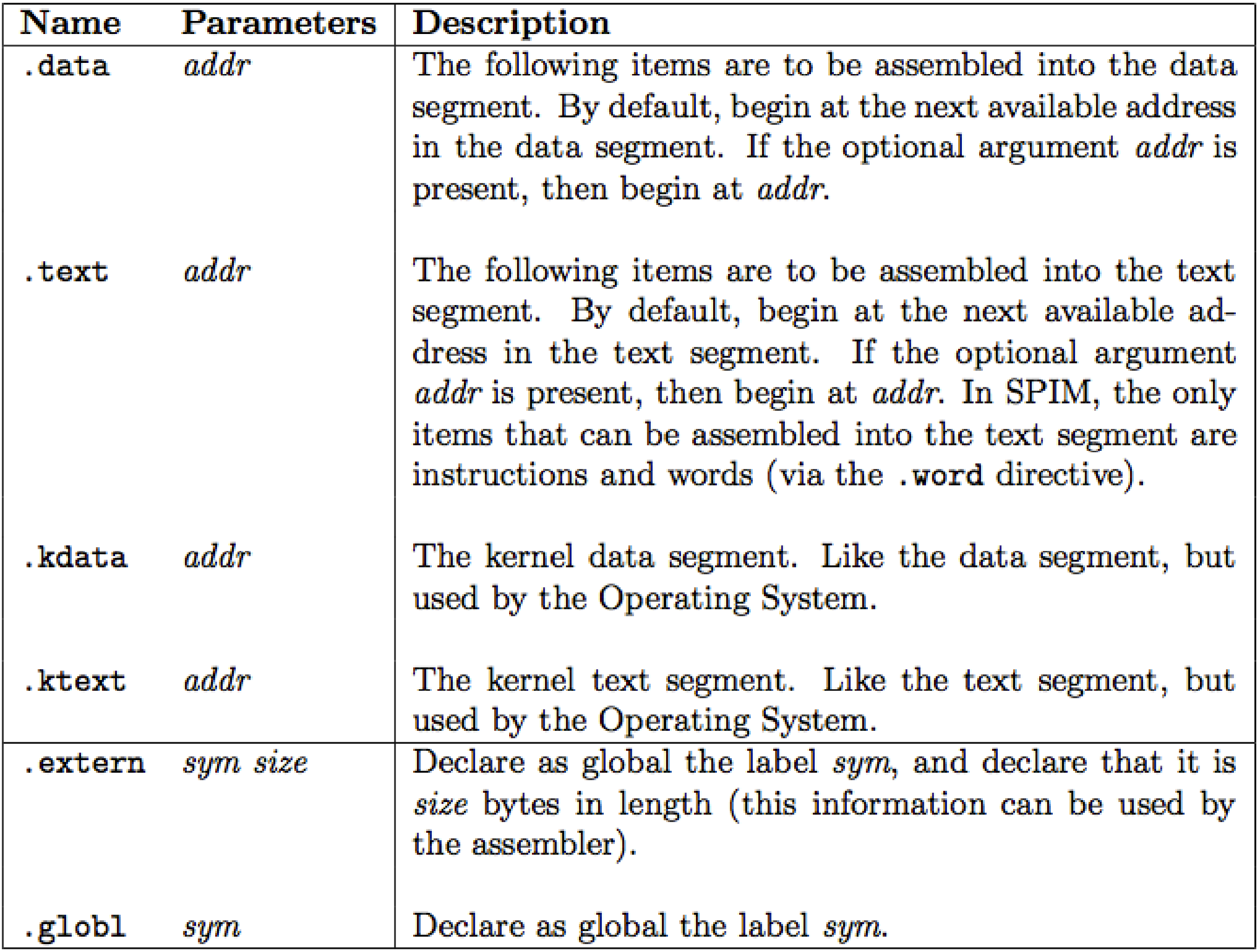
la is a pseudo-instruction; (will talk about it in the next lecture)

Assemble … GO



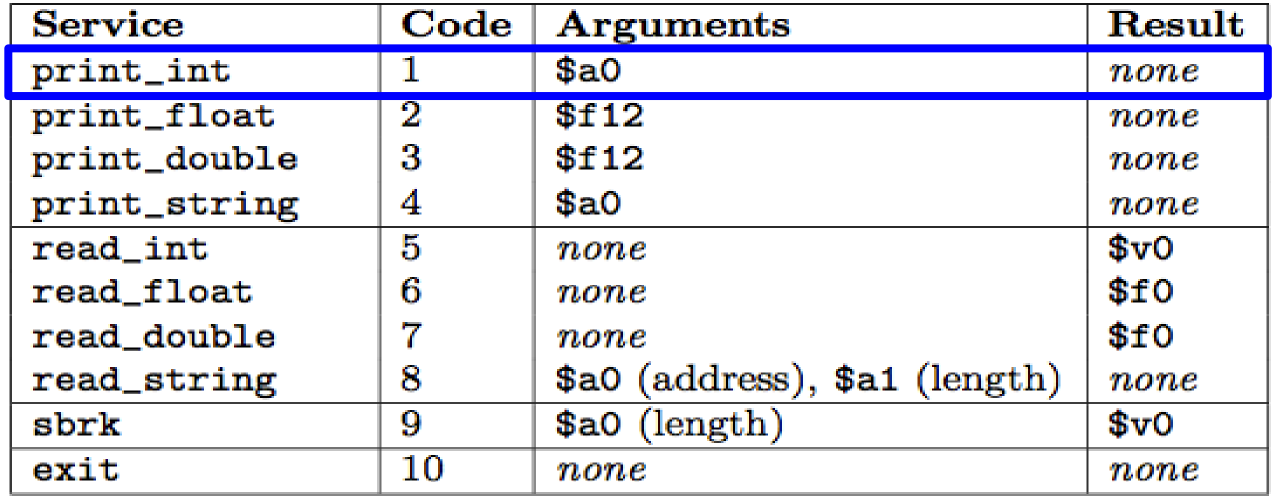
```
hello world

-- program is finished running --
```

Data Directives



Name	Parameters	Description
`.data`	*addr*	The following items are to be assembled into the data segment. By default, begin at the next available address in the data segment. If the optional argument *addr* is present, then begin at *addr*.
`.text`	*addr*	The following items are to be assembled into the text segment. By default, begin at the next available address in the text segment. If the optional argument *addr* is present, then begin at *addr*. In SPIM, the only items that can be assembled into the text segment are instructions and words (via the `.word` directive).
`.kdata`	*addr*	The kernel data segment. Like the data segment, but used by the Operating System.
`.ktext`	*addr*	The kernel text segment. Like the text segment, but used by the Operating System.
`.extern`	*sym size*	Declare as global the label *sym*, and declare that it is *size* bytes in length (this information can be used by the assembler).
`.globl`	*sym*	Declare as global the label *sym*.

Print “out” an Integer



Service	Code	Arguments	Result
print_int	1	$a0	*none*
print_float	2	$f12	*none*
print_double	3	$f12	*none*
print_string	4	$a0	*none*
read_int	5	*none*	$v0
read_float	6	*none*	$f0
read_double	7	*none*	$f0
read_string	8	$a0 (address), $a1 (length)	*none*
sbrk	9	$a0 (length)	$v0
exit	10	*none*	*none*

`Syscall` → `1` (argument = `$a0)`

Print an integer to console

System Calls:

```
4
1
10
```

Example-2

Prints “ out ” an integer

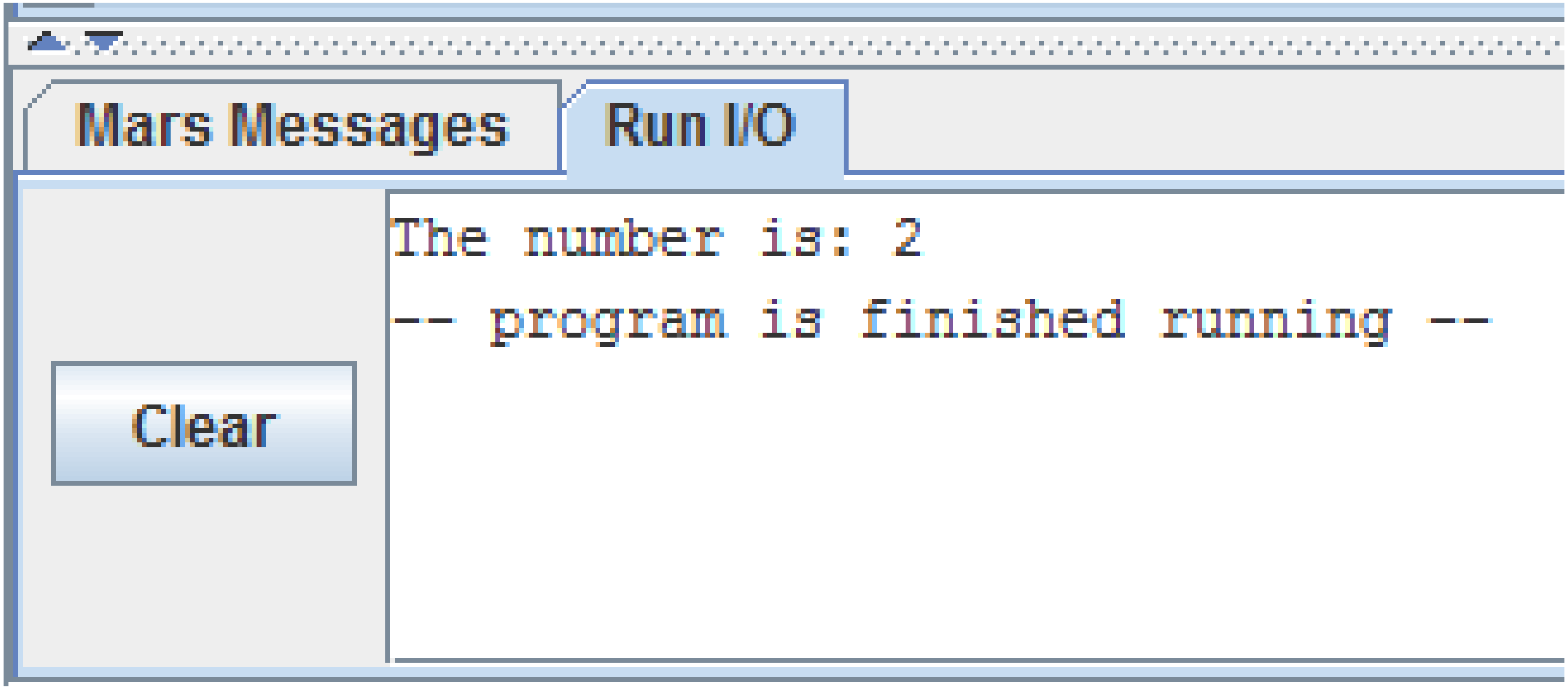
```
         .text
         .globl main
main:
         la       $a0, output
         li       $v0,4
         syscall
         li       $t0,2
         move     $a0,$t0
         li       $v0,1
         syscall
         li       $v0,10
         syscall

         .data
output:  .asciiz "The number is:  "
```

(Copy the contents of register `$t0` to register `$a0`)

How to use **`syscall`** system services

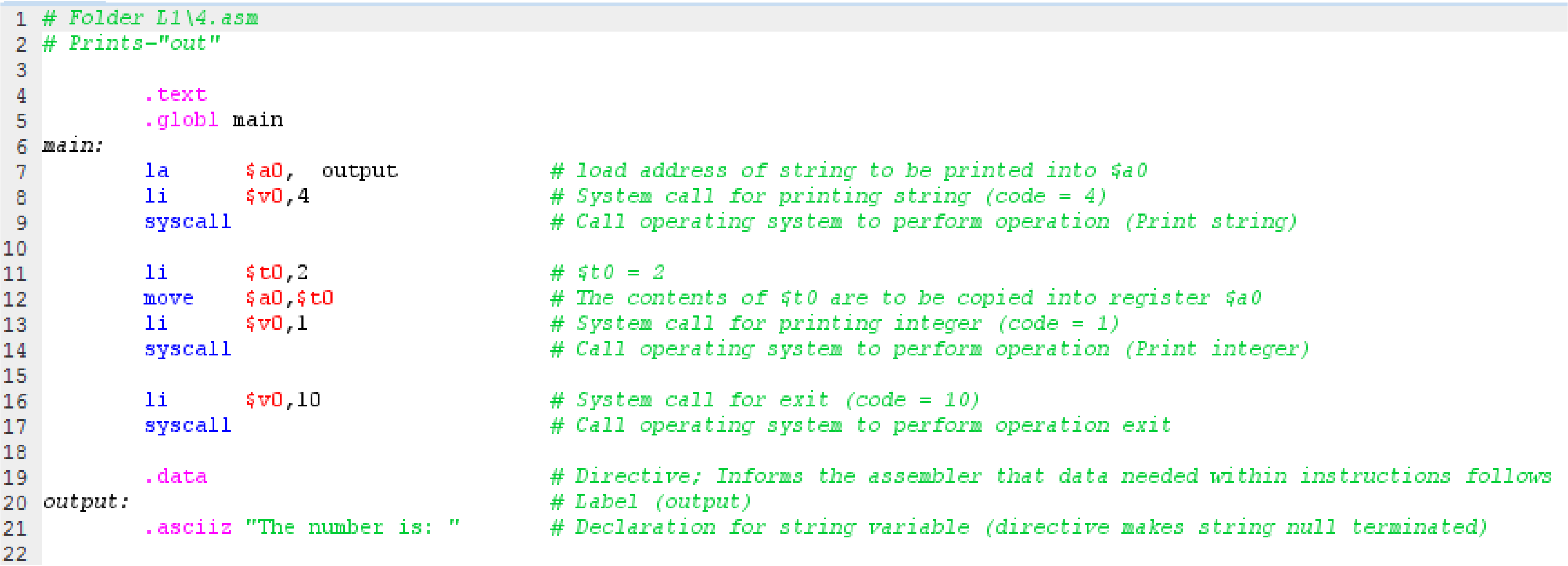
1. Load the service number to register: **`$v0`**
2. Load argument values, if any, to registers: **`[$a0]`**`,` `[$a1], [$a2]`
3. Issue the **`syscall`** instruction
4. Retrieve return values, if any, from the used registers.

Assemble … GO; `The number is: 2`



```
The number is: 2
-- program is finished running --
```

Prints “out” the result to the console



```
# Folder L1\4.asm
# Prints-"out"

        .text
        .globl main
main:
        la      $a0,  output        # load address of string to be printed into $a0
        li      $v0,4               # System call for printing string (code = 4)
        syscall                     # Call operating system to perform operation (Print string)

        li      $t0,2               # $t0 = 2
        move    $a0,$t0             # The contents of $t0 are to be copied into register $a0
        li      $v0,1               # System call for printing integer (code = 1)
        syscall                     # Call operating system to perform operation (Print integer)

        li      $v0,10              # System call for exit (code = 10)
        syscall                     # Call operating system to perform operation exit

        .data                       # Directive; Informs the assembler that data needed within instructions follows
output:                             # Label (output)
        .asciiz "The number is: "   # Declaration for string variable (directive makes string null terminated)

```

move is a pseudo-instruction; (will talk about it in the next lecture)

System Calls:

4

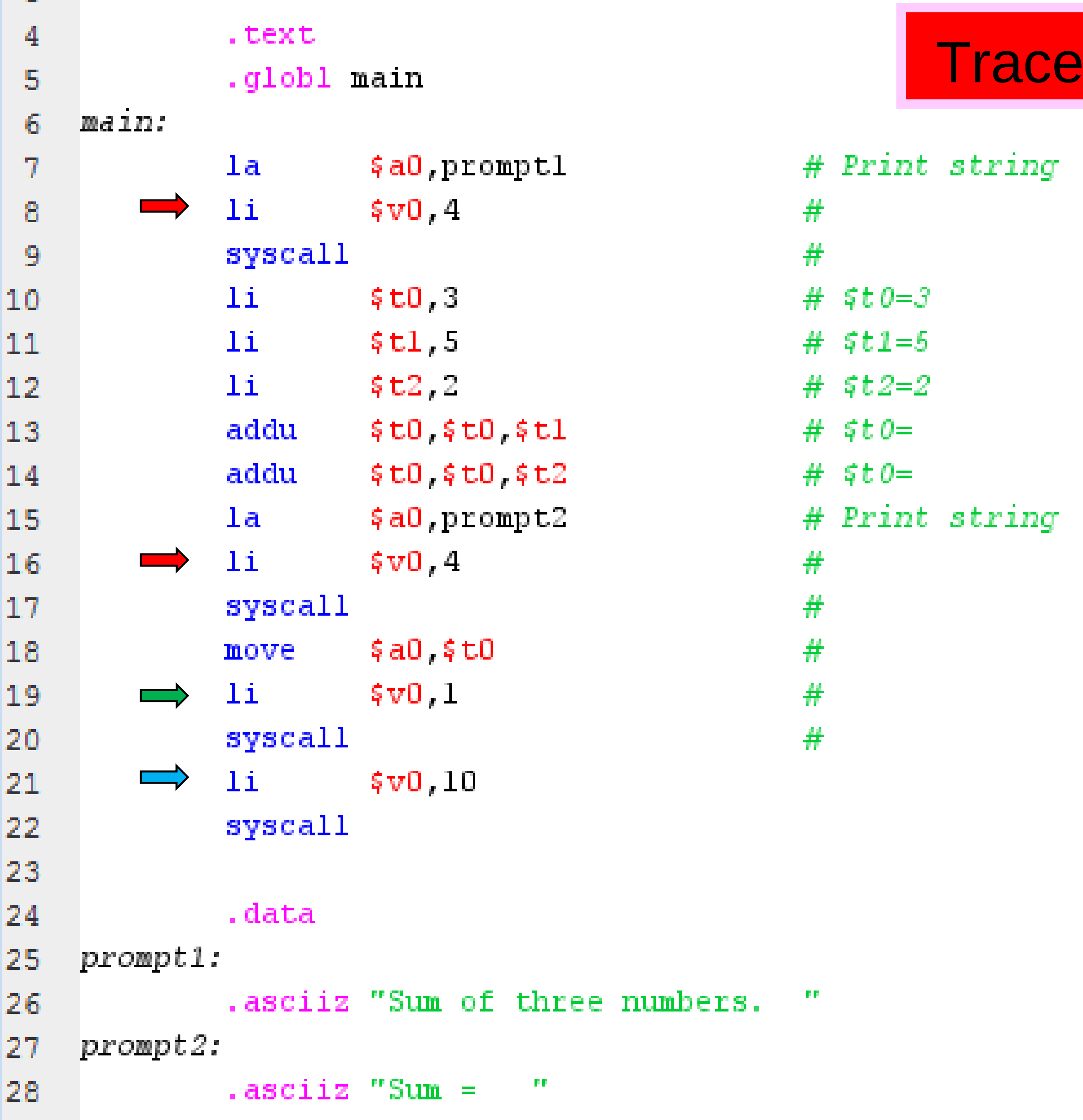
4

1

10

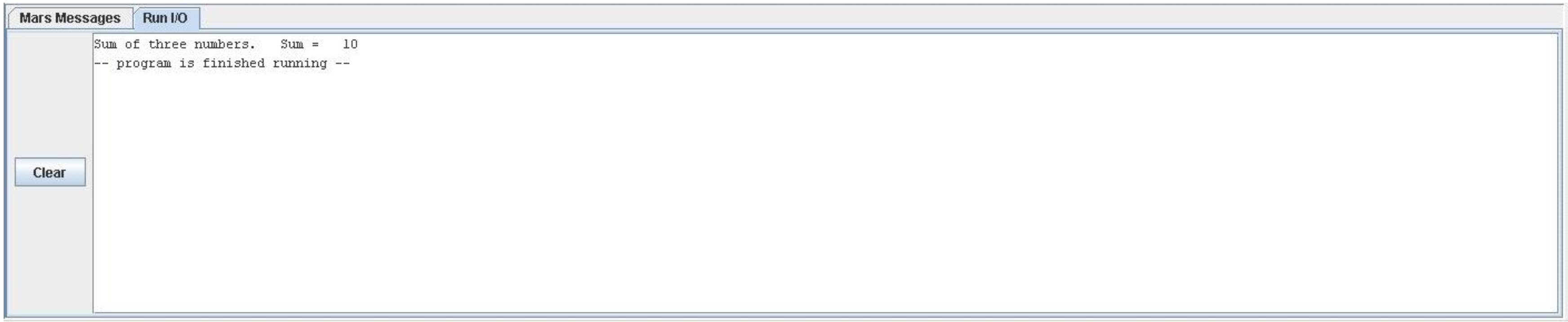
Example-3

Trace the program

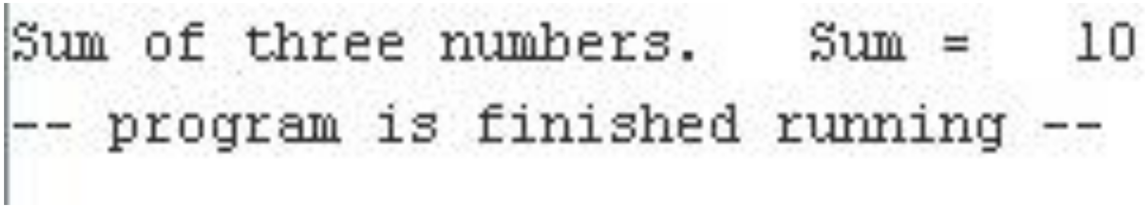


```
        .text
        .globl main
main:
        la      $a0,prompt1             # Print string
        li      $v0,4                   #
        syscall                         #
        li      $t0,3                   # $t0=3
        li      $t1,5                   # $t1=5
        li      $t2,2                   # $t2=2
        addu    $t0,$t0,$t1             # $t0=
        addu    $t0,$t0,$t2             # $t0=
        la      $a0,prompt2             # Print string
        li      $v0,4                   #
        syscall                         #
        move    $a0,$t0                 #
        li      $v0,1                   #
        syscall                         #
        li      $v0,10
        syscall

        .data
prompt1:
        .asciiz "Sum of three numbers.  "
prompt2:
        .asciiz "Sum =   "
```

GO …. (result)

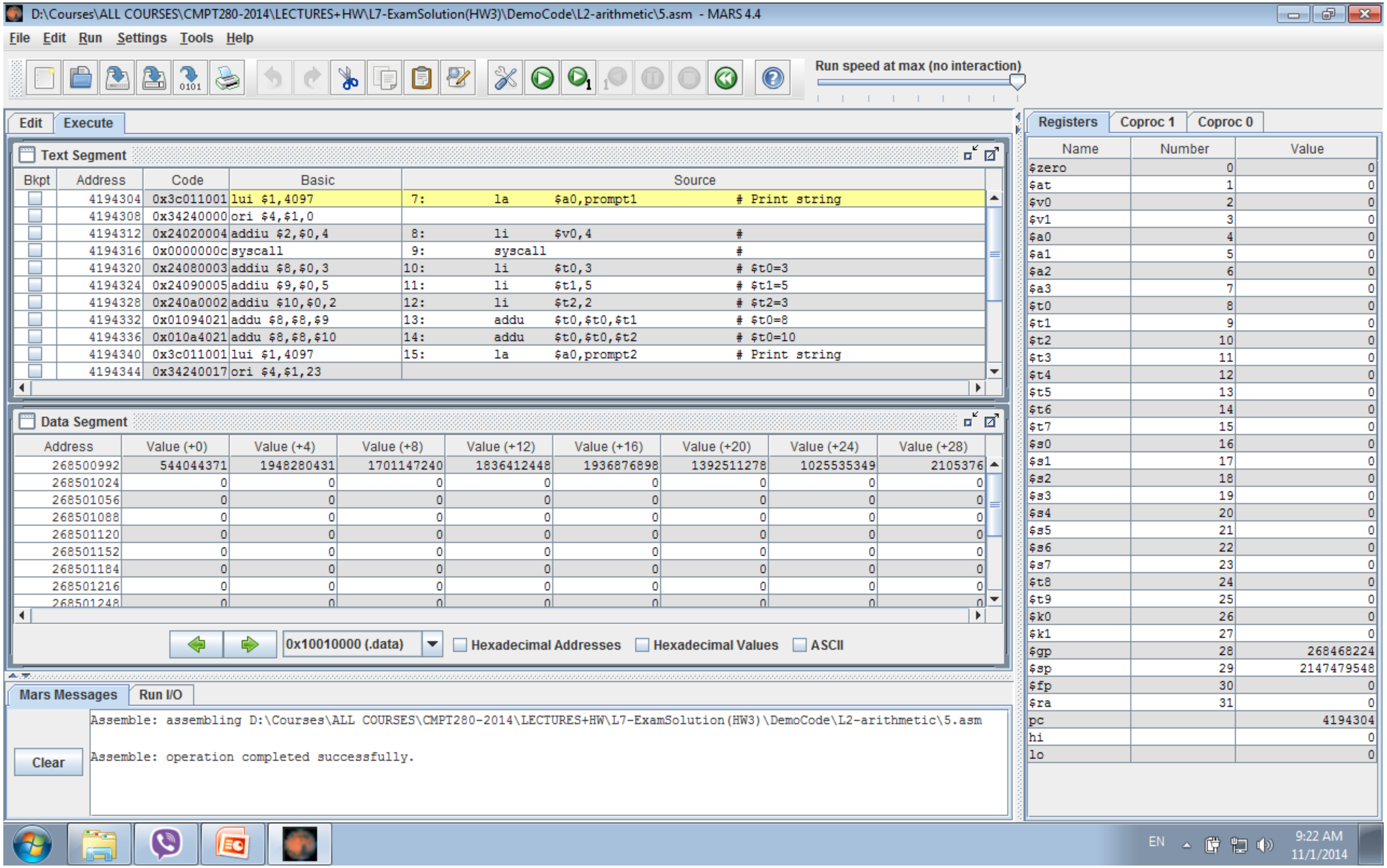


```
Sum of three numbers.   Sum =   10
-- program is finished running --
```



```
Sum of three numbers.   Sum =   10
-- program is finished running --
```

Assemble …


D:\Courses\ALL COURSES\CMPT280-2014\LECTURES+HW\L7-ExamSolution(HW3)\DemoCode\L2-arithmetic\5.asm - MARS 4.4
File
Edit
Run
Settings
Tools
Help
0101
Run speed at max (no interaction)
Edit
Execute
Text Segment
Data Segment
Registers
Coproc 1
Coproc 0
Mars Messages
Run I/O
Clear
EN
9:22 AM
11/1/2014

Bkpt	Address	Code	Basic	Source
□	4194304	0x3c011001	lui $1,4097	7: la $a0,prompt1 # Print string
□	4194308	0x34240000	ori $4,$1,0	
□	4194312	0x24020004	addiu $2,$0,4	8: li $v0,4 #
□	4194316	0x0000000c	syscall	9: syscall #
□	4194320	0x24080003	addiu $8,$0,3	10: li $t0,3 # $t0=3
□	4194324	0x24090005	addiu $9,$0,5	11: li $t1,5 # $t1=5
□	4194328	0x240a0002	addiu $10,$0,2	12: li $t2,2 # $t2=3
□	4194332	0x01094021	addu $8,$8,$9	13: addu $t0,$t0,$t1 # $t0=8
□	4194336	0x010a4021	addu $8,$8,$10	14: addu $t0,$t0,$t2 # $t0=10
□	4194340	0x3c011001	lui $1,4097	15: la $a0,prompt2 # Print string
□	4194344	0x34240017	ori $4,$1,23	

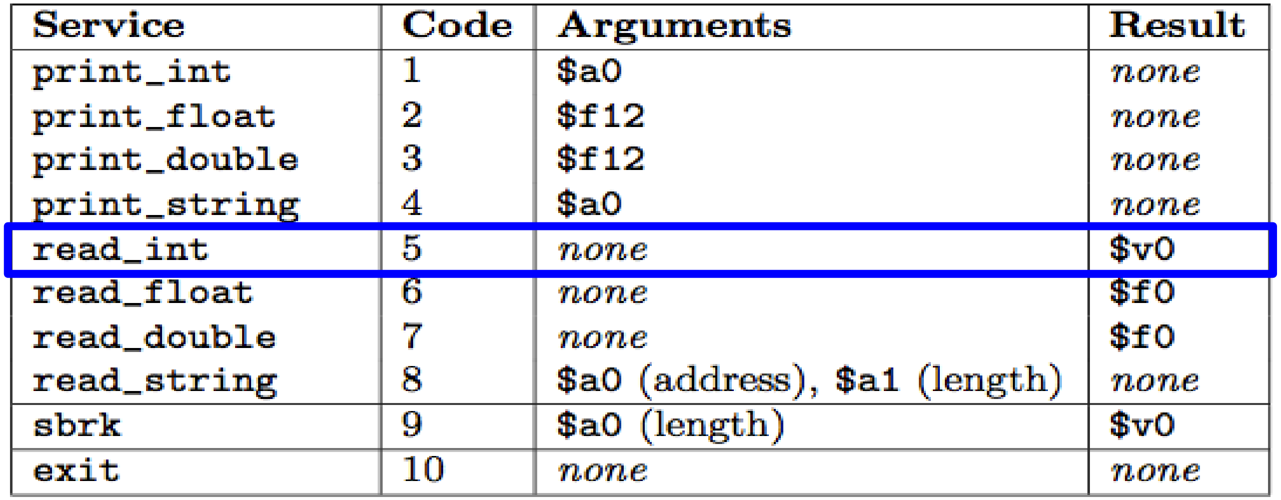
Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	544044371	1948280431	1701147240	1836412448	1936876898	1392511278	1025535349	2105376
268501024	0	0	0	0	0	0	0	0
268501056	0	0	0	0	0	0	0	0
268501088	0	0	0	0	0	0	0	0
268501120	0	0	0	0	0	0	0	0
268501152	0	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0
268501248	0	0	0	0	0	0	0	0

0x10010000 (.data) □ Hexadecimal Addresses □ Hexadecimal Values □ ASCII

Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	0
$t1	9	0
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194304
hi		0
lo		0

```
Assemble: assembling D:\Courses\ALL COURSES\CMPT280-2014\LECTURES+HW\L7-ExamSolution(HW3)\DemoCode\L2-arithmetic\5.asm

Assemble: operation completed successfully.
```

Read integer from the command line (console)



Service	Code	Arguments	Result
print_int	1	$a0	*none*
print_float	2	$f12	*none*
print_double	3	$f12	*none*
print_string	4	$a0	*none*
read_int	5	*none*	$v0
read_float	6	*none*	$f0
read_double	7	*none*	$f0
read_string	8	$a0 (address), $a1 (length)	*none*
sbrk	9	$a0 (length)	$v0
exit	10	*none*	*none*

Syscall → 5

Read integer from the command line

System Calls:

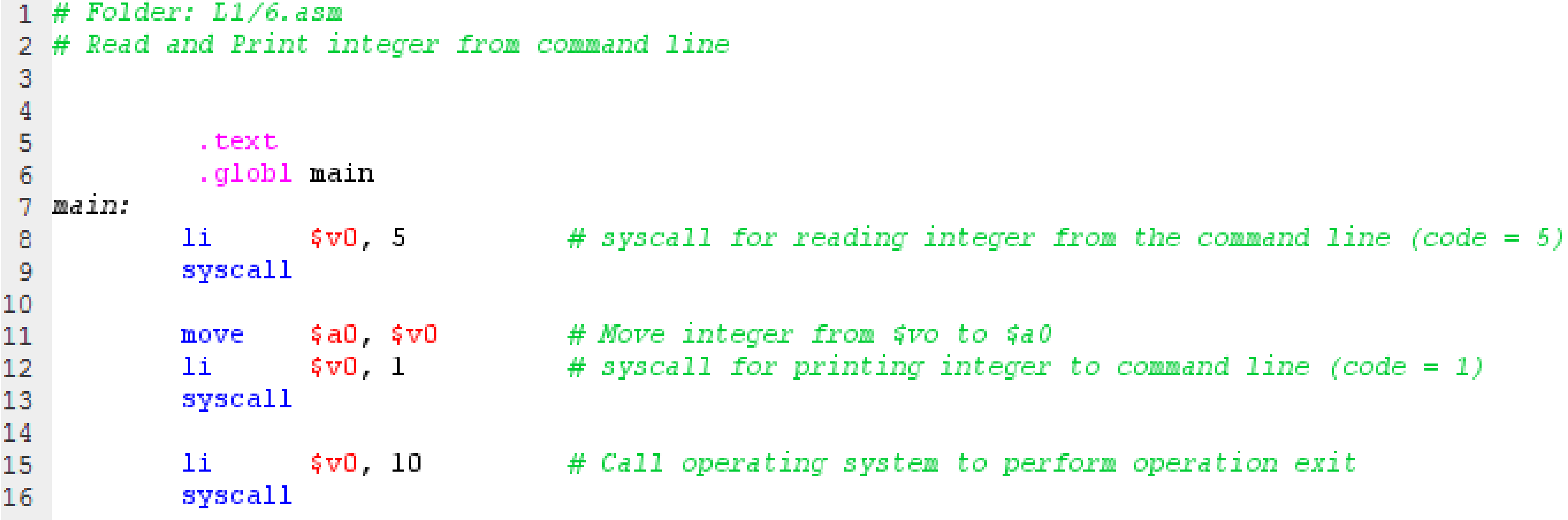
```
5
1
10
```

Example-4

Read/Print integer from/to command line

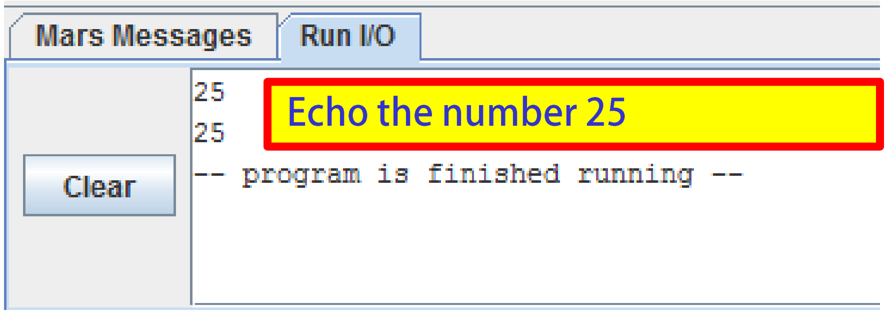
```
        .text
        .globl main
main:
        li          $v0, 5
        syscall
        move        $a0, $v0

        li           $v0, 1
        syscall

        li            $v0,10
        syscall
```

Read/Print integer from/to command line



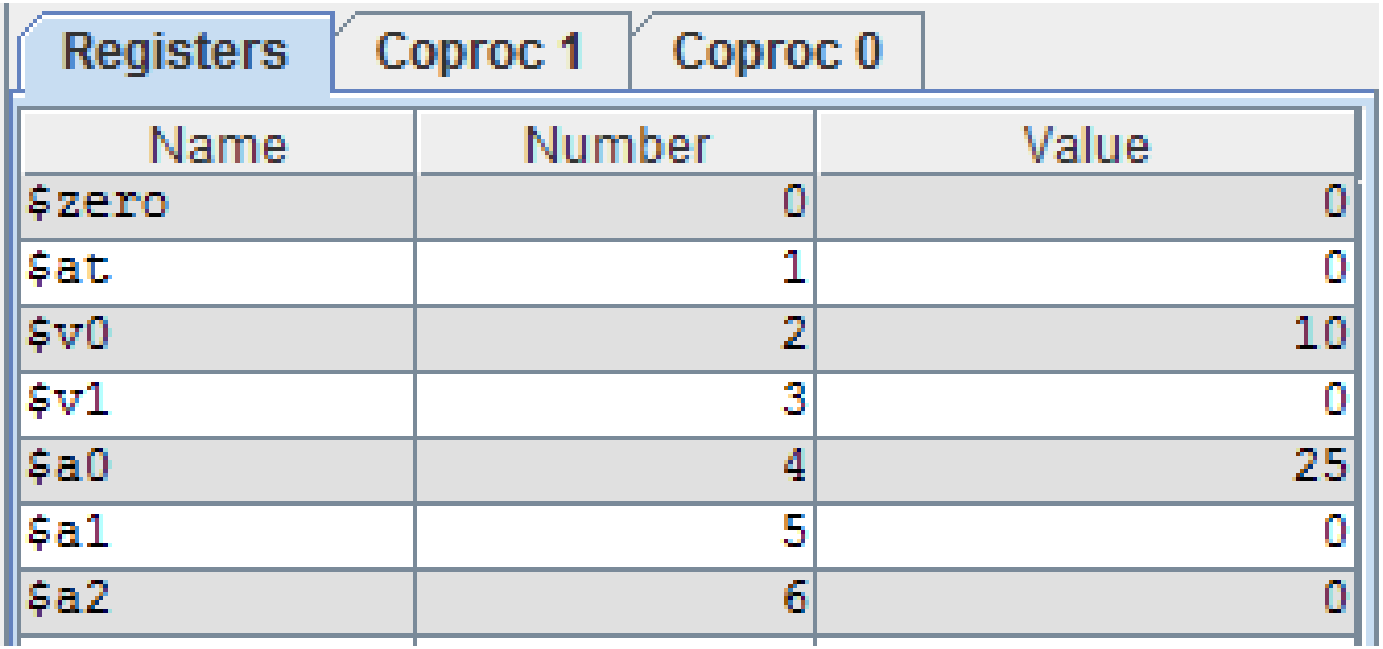
```
# Folder: L1/6.asm
# Read and Print integer from command line


         .text
         .globl main
main:
        li      $v0, 5          # syscall for reading integer from the command line (code = 5)
        syscall

        move    $a0, $v0        # Move integer from $vo to $a0
        li      $v0, 1          # syscall for printing integer to command line (code = 1)
        syscall

        li      $v0, 10         # Call operating system to perform operation exit
        syscall
```

Assemble … GO;



```
25
25
-- program is finished running --
```

Echo the number 25



Name	Number	Value
$zero	0	0
$at	1	0
$v0	2	10
$v1	3	0
$a0	4	25
$a1	5	0
$a2	6	0

Input/Output

Example-5

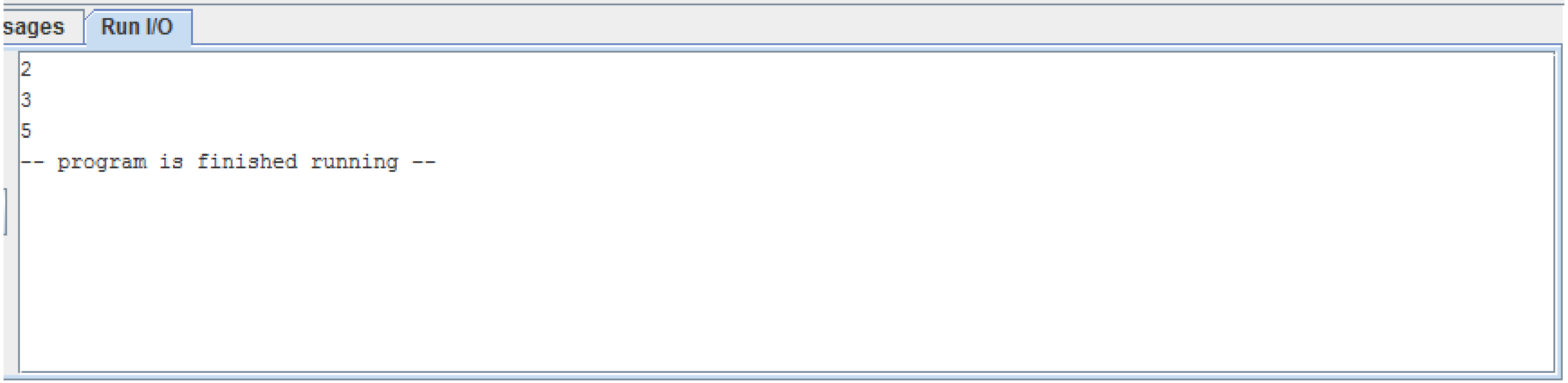
What does it do?

Trace the program

```
        .text
        .globl main
main:
        li          $v0, 5
        syscall
        move        $t0, $v0
        li          $v0, 5
        syscall
        move        $t1, $v0
        add         $a0, $t0, $t1
        li          $v0, 1
        syscall
        li          $v0, 10
        syscall
```

2 minutes

Assemble-GO



```
2
3
5
-- program is finished running --
```

Reads two integers/prints integer from/to command line

System Services (`syscall`) in MIPS

To print an integer to the screen (console):

Set `$v0` to `1`

```
syscall
```

System Services (syscall) in MIPS

To print an integer to the screen (console): Set $v0 to 1 syscall
To print a string to the screen (console): Set $v0 to 4 syscall

System Services (`syscall`) in MIPS

To print an integer to the screen (console): Set `$v0` to `1` `syscall`
To print a string to the screen (console): Set `$v0 to 4` `syscall`
To read an integer from the keyboard (console): Set `$v0 to 5` `syscall`

System Services (syscall) in MIPS

To print an integer to the screen (console): Set $v0 to 1 syscall
To print a string to the screen (console): Set $v0 to 4 syscall
To read an integer from the keyboard (console): Set $v0 to 5 syscall
To exit: Set $v0 to 10 syscall

Class problem

Input/Output (using **`addi`**)

Write a MIPS Assembly program

- Input a (any) number from the keyboard
- Add 3 to that number **(**use **`addi`**)
- Output the result **(**console**)**

Class problem… 5 min

Solution

```
# addi-input-output
        .text
        .globl main
main:
        li      $v0,5
        syscall

        addi    $a0, $v0, 3

        li      $v0,1
        syscall

        li      $v0,10
        syscall
```

END