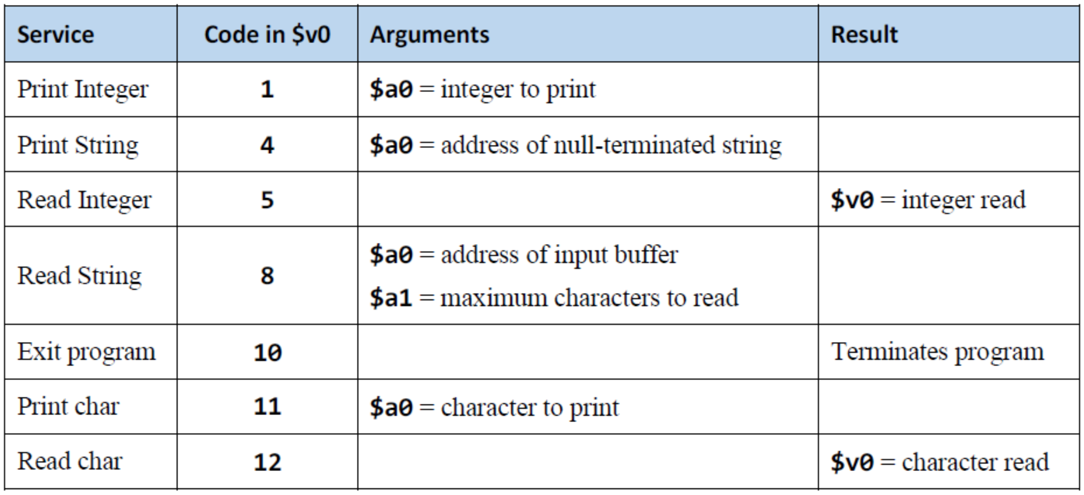
LAB #02 System Calls and Integer Arithmetic

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. The following table shows a small set of services provided by MARS for doing basic I/O.

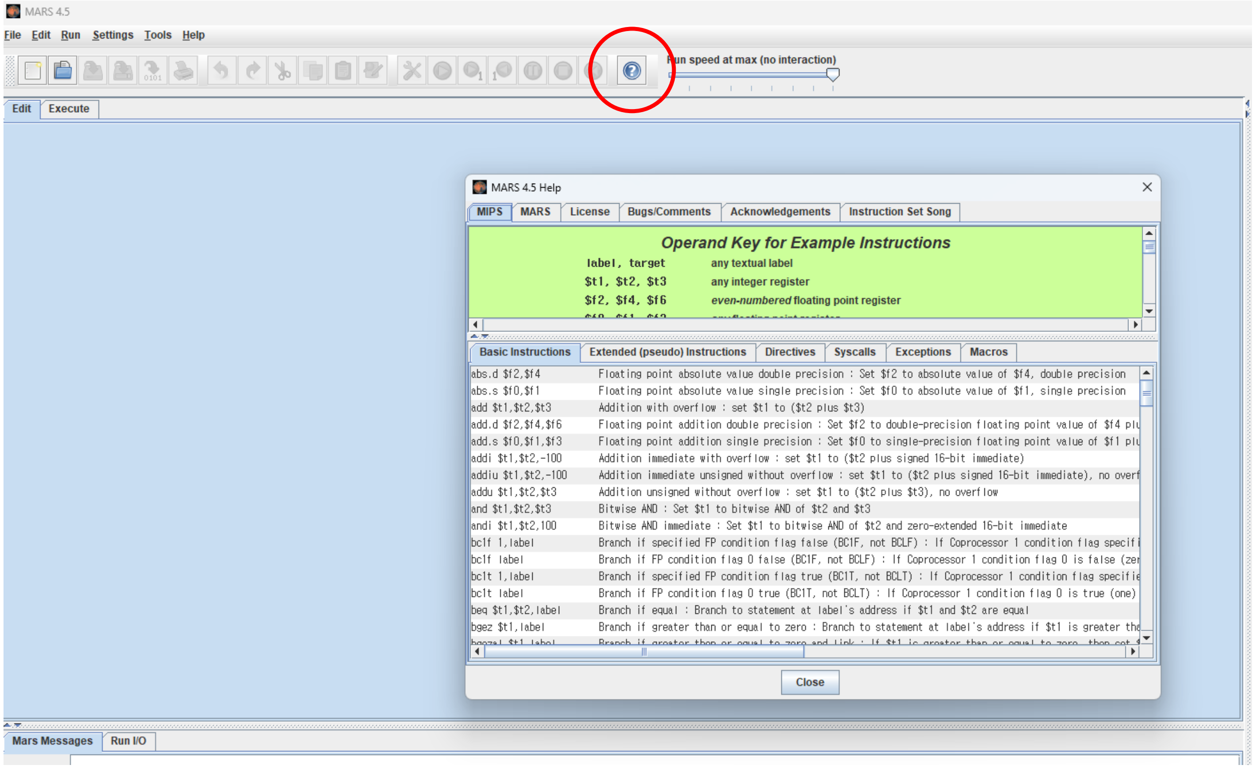
Before using the **syscall** instruction, you should load the service number into register $v0, and load the arguments, if any, into registers $a0, $a1, etc. After issuing the **syscall** instruction, you should retrieve return values, if any, from register $v0.



Service	Code in $v0	Arguments	Result
Print Integer	**1**	**$a0** = integer to print	
Print String	**4**	**$a0** = address of null-terminated string	
Read Integer	**5**		**$v0** = integer read
Read String	**8**	**$a0** = address of input buffer **$a1** = maximum characters to read	
Exit program	**10**		Terminates program
Print char	**11**	**$a0** = character to print	
Read char	**12**		**$v0** = character read

Fig. 1 syscall summary.

NOTE: All instructions, pseudo-instructions, directives, etc. can be found in MARS help.


MARS 4.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
MARS 4.5 Help
MIPS MARS License Bugs/Comments Acknowledgements Instruction Set Song
Basic Instructions Extended (pseudo) Instructions Directives Syscalls Exceptions Macros
Close
Mars Messages Run I/O

Operand Key for Example Instructions

label, target	any textual label
$t1, $t2, $t3	any integer register
$f2, $f4, $f6	*even-numbered* floating point register

abs.d $f2,$f4	Floating point absolute value double precision : Set $f2 to absolute value of $f4, double precision
abs.s $f0,$f1	Floating point absolute value single precision : Set $f0 to absolute value of $f1, single precision
add $t1,$t2,$t3	Addition with overflow : set $t1 to ($t2 plus $t3)
add.d $f2,$f4,$f6	Floating point addition double precision : Set $f2 to double-precision floating point value of $f4 plu
add.s $f0,$f1,$f3	Floating point addition single precision : Set $f0 to single-precision floating point value of $f1 plu
addi $t1,$t2,-100	Addition immediate with overflow : set $t1 to ($t2 plus signed 16-bit immediate)
addiu $t1,$t2,-100	Addition immediate unsigned without overflow : set $t1 to ($t2 plus signed 16-bit immediate), no overf
addu $t1,$t2,$t3	Addition unsigned without overflow : set $t1 to ($t2 plus $t3), no overflow
and $t1,$t2,$t3	Bitwise AND : Set $t1 to bitwise AND of $t2 and $t3
andi $t1,$t2,100	Bitwise AND immediate : Set $t1 to bitwise AND of $t2 and zero-extended 16-bit immediate
bc1f 1,label	Branch if specified FP condition flag false (BC1F, not BCLF) : If Coprocessor 1 condition flag specifi
bc1f label	Branch if FP condition flag 0 false (BC1F, not BCLF) : If Coprocessor 1 condition flag 0 is false (zer
bc1t 1,label	Branch if specified FP condition flag true (BC1T, not BCLT) : If Coprocessor 1 condition flag specifie
bc1t label	Branch if FP condition flag 0 true (BC1T, not BCLT) : If Coprocessor 1 condition flag 0 is true (one)
beq $t1,$t2,label	Branch if equal : Branch to statement at label's address if $t1 and $t2 are equal
bgez $t1,label	Branch if greater than or equal to zero : Branch to statement at label's address if $t1 is greater tha

Fig. 2 MARS help

Add/Subtract Instructions

The difference between **add/sub** and **addu/subu** instructions is that in case of overflow occurrence, the **add/sub** instructions will cause an arithmetic exception and the result will not be written to the destination register. However, for the instructions **addu/subu**, overflow occurrence is ignored.

Operation	Meaning
add $s1, $s2, $s3	$s1 = $s2 + $s3
addu $s1, $s2, $s3	$s1 = $s2 + $s3
sub $s1, $s2,	$s3 $s1 = $s2 – $s3
subu $s1, $s2, $s3	$s1 = $s2 – $s3
addi $s1, $s2, 10	$s1 = $s2 + 10
addiu $s1, $s2, 10	$s1 = $s2 + 10

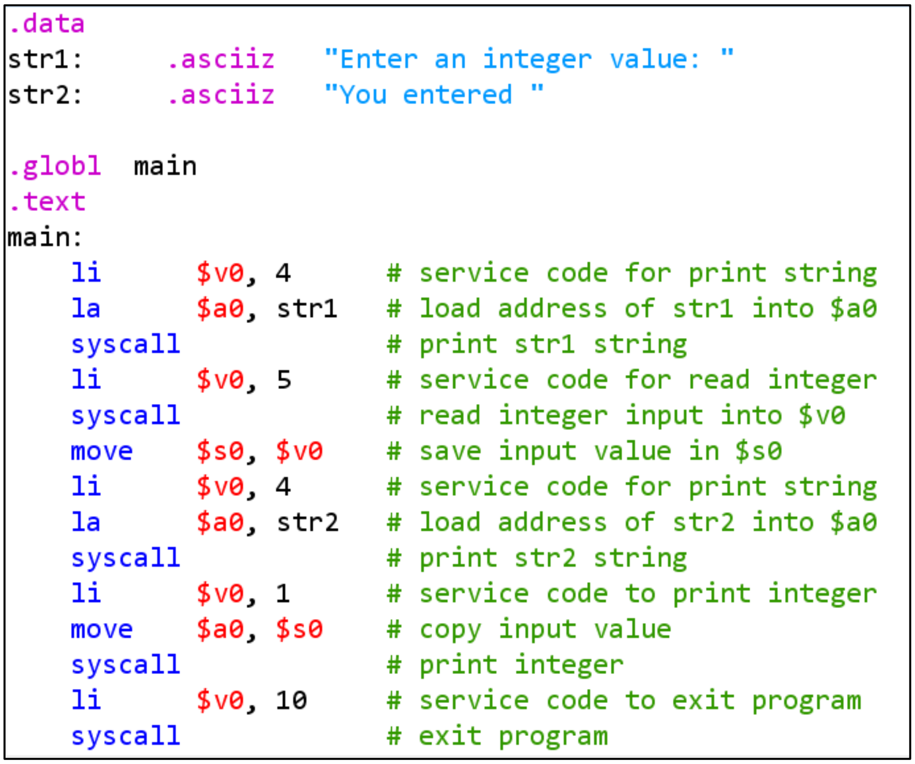
Fig. 3 Arithmetic instructions

Task to do

1. Modify the following program in Fig.4.

Condition:

- Ask the user “enter an integer value: ”
- Print the result of twice of that number (e.g., input x becomes 2x)
- Use the add instruction.



```
.data
str1:     .asciiz   "Enter an integer value: "
str2:     .asciiz   "You entered "

.globl  main
.text
main:
    li      $v0, 4      # service code for print string
    la      $a0, str1   # load address of str1 into $a0
    syscall             # print str1 string
    li      $v0, 5      # service code for read integer
    syscall             # read integer input into $v0
    move    $s0, $v0    # save input value in $s0
    li      $v0, 4      # service code for print string
    la      $a0, str2   # load address of str2 into $a0
    syscall             # print str2 string
    li      $v0, 1      # service code to print integer
    move    $a0, $s0    # copy input value
    syscall             # print integer
    li      $v0, 10     # service code to exit program
    syscall             # exit program
```

Fig. 4 An example of a program using syscall.

2. Modify your program of question 1 as follows:

a) At the end, ask the user to repeat the program as "\nRepeat [y/n]? ".

b) Use service code 12 to read a character and repeat the main function if the answer is 'y'.

3. Write a MIPS program that executes the statement:

s = (a + b) – (c + 101), where a, b, and c are user provided integer inputs, and s is computed and printed as an output. Answer the following:

a) Suppose the user enters a = 5, b = 10, and c = -30, what is the expected value of s?

b) Which instruction in your program computed the value of s and which register is used?

c) What is the address of this instruction in memory?

d) Put a breakpoint at this instruction and write the value of the register used for computing s in decimal and hexadecimal.

4. Write a MIPS program as follows:

Condition:

- I/O input: two integer values
- I/O output: **equal** if those two integers are equal. **not equal** otherwise
- Use the branch instruction to check for equality.