Tutorial on MIPS Programming using MARS

It is expected that students should go through the code segments provided in this tutorial before proceeding with the asignments. This tutorial is meant for beginners of MIPS programming and assumes use of the MARS simulator for execution and debugging.

Execute the codes given below (in steps) and observe the values of registers and memory addresses as shown in the simulator during each step. It would help you understand how the code works.

1. Printing a character :

```
.data
 character : .byte 'a'

.text


li    $v0, 11                  #11=system code for printing a character, $v0=register that gets the system
                              code for  printing as value

la    $a0, character    #'a'=our example character, $a0=register that accepts the character for
                              printing
syscall                       #Call to the System to execute the instructions and print the character at the
                                a0
```

2. Printing a number :

```
.data
 age : .word  21

.text


li    $v0, 1                   #1= system code for printing a word ( 32 bit integer), $v0=register that gets
                              the system code for  printing as value

la    $a0, age              # age is the variable that contains the word to be printed, $a0=register that
                               accepts the word for  printing
syscall                       #Call to the System to execute the instructions and print the word at a0
```

3. Printing a floating point number :

```
.data
 PI : .float  3.14          # PI is the variable that contains the floating point nmumber 3.14 to be
                                printed ( loaded in the data memory)
.text

li    $v0, 2                   # 2= system code for printing a floating point number (32-bit IEEE 754
                              format), $v0=register that gets  the system code for  printing as value

lcw1    $f12, PI              # $f12 register is not available with MIPS but with the co-processor 1; lwfc1
                                means load the $f12 register of coprocessor 1
syscall                       #Call to the System to execute our instructions
```

4. Printing a double- precision floating point number :

```
.data
 test : .double  7.202        # test is the variable that contains the double precision floating point
                                  number 7.202 (64-bit IEEE 754 format), $v0=register that gets  the
                                  system code for  printing as value to be    printed ( loaded in the data
                                  memory)
.text

ldc1 $f2, test                # the 64-bit value in test variable is stored in $f2 (32-bit LSB) and $f3 ( 32-
                                bit USB)


li    $v0, 3                  #3= system code for printing a double precisionfloating point number
                              ( IEEE 754 format),$v0=register that gets  the system code for  printing as
                              value

move $f12,$f2               # move is a pseudo-instruction that transfers contents of $f2 to $f12

syscall                       #Call to the System to execute our instructions
```

5. Adding two numbers :

```
.data
 num1 : .word  2              # first number to be added stored in data memory
 num2: .word  3              # second number to be added stored in data memory

.text

lw $t0, num1                  # num 1 is stored in temporary register $t0
lw $t1, num2                  # num 2 is stored in temporary register $t1

add $t2 , $t0, $t1           # t2 <- t0 + t1

li    $v0, 1                  #1= system code for printing a word,
                               $v0=register that gets  the system code for  printing as value

move $a0, $t2               # move is a pseudo-instruction that transfers contents of $t2 to $a0
                              #a0 is the register that needs to hold the value that needs  to be printed

syscall                       #Call to the System to execute our instructions
```

6. Multiply two numbers :

```
.data

.text

addi $t0,$zero,10      # t0 <- 0+10
addi $t1, $zero,4      # t1 <- 0+4

mult $t0,$t1           # The result is in hi and low registers

li    $v0, 1                  #1= system code for printing a word,
                               $v0=register that gets  the system code for  printing as value

add $a0, $zer0, $s0     # a0 <- 0+t0
                              #a0 is the register that needs to hold the value that needs  to be printed

syscall                       #Call to the System to execute our instructions
```

7. To get the user input :

```
.data
 prompt : .aciiz "Enter your age"
 message : .asciiz " \n Your age is"
.text
li    $v0, 4                  #4= system code for printing a string,
                               $v0=register that gets  the system code for  printing as value

la $a0,prompt             # load address of prompt in $a0
syscall                       # prints the string " Enter your age"

# Get the users age
li $v0,5                       #5= system code for  user input
syscall                       #Call to the System to execute the instruction

# Store the result in $t0
move $t0, $v0             # move is a pseudo-instruction that transfers contents of $t0 to $v0
                                 t0 now contains the user input
# Display the user input
li    $v0, 4                  #4= system code for printing a string,
                               $v0=register that gets  the system code for  printing as value
la $a0, message          # load address of prompt in $a0
syscall                       # prints the string " Your age is"

# Show the age
li    $v0, 1                  #1= system code for printing a word,
                               $v0=register that gets  the system code for  printing as value
move $a0, $t0           # move is a pseudo-instruction that transfers contents of $t0 to $a0
                              #a0 is the register that needs to hold the value that needs  to be printed

syscall                       #Call to the System to execute our instructions
```

8. Passing Arguments to Functions :

```
.data              #data section

.text               #code section

main:

addi $a1, #zero, 50         # a1 <- 50
addi $a2, #zero, 100        # a2 <- 100

jal addnumbers          # Call the subroutine addnumbers and pass on values of a1 and a2 as
                          arguments of  addnumbers; Save the return address in $ra

li    $v0, 1                 #1= system code for printing a word,
                              $v0=register that gets  the system code for  printing as value
move $a0, $v1          # move is a pseudo-instruction that transfers contents of $v1 to $a0
                        #a0 is the register that needs to hold the value that needs  to be printed

syscall                   #Call to the System to execute our instructions


li      $v0, 10             # system call for terminating the execution
syscall


addnumbers :

add $v1, $a1, $a2      # v1 <- a1 + a2

jr $ra              # return to the address pointed to by the address held in return address register
```

9. Branch Instructions (If Statements) :

```
.data                #data section
  message : .asciiz “ The numbers are different”

.text                #code section

main:

addi $t0, #zero, 5            # t0 <- 5
addi $t1, #zero, 20            # t1 <- 20

# Conditional jump to label numbersdifferent if numbers $t0 and $t1 are different

bne $t0, $t1, numbersdifferent
li    $v0, 10                  #10= system code for exit
                                $v0=register that gets  the system code for  printing as value
syscall
numbersdifferent :

 # Display the user input

li    $v0, 4                   #4= system code for printing a string,
                                $v0=register that gets  the system code for  printing as value
la $a0, message          # load address of prompt in $a0
syscall                        # prints the string

li    $v0, 10                  #10= system code for exit
                                $v0=register that gets  the system code for  printing as value
syscall
```

There are alternate ways to compute the same problem :

1. use of slt instruction - it compares two registers and returns the value as 1 if true and 0 for false

2. Use of Pseudo branch instructions such as bgt/blt

----

10. Using While Loops :

```
.data              #data section
  message  : .asciiz " After the while loop is done"
  message 2 : .asciiz "\n"

.text                #code section

main:

addi $t0, #zero, 50        # to hold the index of the array

while :

             bgt $t0,10, exit          # if(i>10

            jal printnumbers

            addi $t0,$t0,1

            j while

exit :
            li    $v0, 4
            la $a0, message
            syscall

# End of program

li       $v0, 10
syscall


printnumbers :

      # Print the number

            li       $v0, 1
            move $a0, $t0
            syscall

       # Move to the next line

             li    $v0, 4
            la $a0, message2
            syscall

    # Return to main

            jr $ra
```