COMPUTER ORGANIZATION & Assembly Lang.
Course Instructor: Dr. Shafina

DATE: April 5, 2023

Title: Getting integer Input from User and displaying Strings in MIPS Assembly

Getting Input from User

'MIPS' allows programmers to define their own inputs at run-time and subsequent operations may be performed on this data. Following code demonstrates how system call may be used in order to do this job.

Let's implement the following algorithm;

- Read the numbers from the user, use $t0 & $t1 to hold these numbers
- Compute their sum and store the result in $t2
- Print the result

add2.s; A program that computes sum of two numbers defined by the user at run-time and stores the result in another register.
registers used: $t0 & $t1 – hold two numbers given by user; $t2 – holds the result of their sum

NOTE: You are required to run this code and remove all the errors, once removed find out the missing statements in this code and discuss in LAB

```
        .data
        .text
        .globl main
Main:
        Li      $v0, 5              # get input1 from user
        Syscall
        Move    $t0, $v0            # move this input into temporary register

        Li      $v0, 5              # get input2 from user
        Syscall
        Move    $t1, $v0            # move this input into temporary register

        Add     $t2, $t1, $t0       # compute the sum

        Move    $a0, $t2            # move result into the accumulator
        Li      $v0, 1              # print the integer
        Syscall

Exit:
        Move    $v0, 10             # exit code
        Syscall

# end of add2.asm
```

Printing Strings

Following assembly program shows how strings can be displayed along with integers to make the output rather interactive with users. For example, result of our previous program would be '5' if inputs entered by the user were '2' & '3'. Output '5' is not self explanatory, since it does not tell the viewer what the program does! A better way to display output would be to write 'Sum of 2 & 3 is 5'. This way, viewer knows what the program is meant to do the moment he looks at the output.

So, strings help you present your program in a much better way. Let's get started with just displaying a 'Hello World' message.

```
# hello.s – this program displays 'Hello World'

        .data
hello_msg:      .asciiz "Hello World"

        .text
        .globl main
main:
        la      $a0, Hello_msg          # Load the address of string into accumulator
        li      $v0, 4                  # To display string, '4' should be loaded into $v0
        syscall

Exit:
        li      $v0, 10                 # Exit code
        syscall
```

Program to Prompt the user for an integer and display string

```
.text
main:
    # Prompt for the integer to enter
    li $v0, 4
    la $a0, prompt
    syscall

    # Read the integer and save it in $s0
    li $v0, 5
    syscall
    move $s0, $v0

    # Output the text
    li $v0, 4
    la $a0, output
    syscall

    # Output the number
    li $v0, 1
    move $a0, $s0
    syscall

    # Exit the program
    li $v0, 10
    syscall
.data
prompt: .asciiz "Please enter an integer: "
output: .asciiz "\nYou typed the number "
```

Few Useful Instructions

Instruction	Description	Syntax	Remarks
Sgt	Set on Greater Than	sgt $a0, $t0, $t1	if [$t0] > [$t1] then '1' is written into $a0
J	Unconditional Jump	j target	useful in loops
Beq	Branch if equal	beq $t0, $t1, target	if [$t0] = = [$t1] then jump to target; useful in breaking loops
Move	Copy contents	move $t0, $a0	move contents of $a0 into $t0