FORMAN CHRISTIAN COLLEGE
(A CHARTERED UNIVERSITY)

Computer Organization and Assembly Language – COMP 300 B

Spring 21

Programming Assignment 2

Muhammad Sameed Gilani - 231488347

You should attach the lab / assignment handout as second page of this report.

From third page onwards following headings should be included:

- **Introduction**
  - **Should carry information of all major library functions.**
- **Your logic / algorithm in simple English. Bullet points are appreciated.**
- **Your code**
- **Screen shots of at least three outputs of your code with appropriate inputs.**
- **References**

INTRODUCTION

- li – Load immediate. → It is used to set the register to the immediate value we enter.

Ex:

```
li    $v0,1
```
This sets the register $v0, to 1

- la – Load address → It is used to set the register to the contents of another register or to an immediate value we enter.

Ex:

```
la    $a0,$t0
```
This loads the contents of $t0 onto $a0

- lw - Load Word → Set a register to contents of effective memory word address,

Ex:

```
lw   $a0,input
```

This loads the address of the .word input, we created in the data segment.

- .asciiz → Store the string in the Data segment and add null terminator. Used in the program to store strings.

Ex:

```
x:   .asciiz " Enter a value for x: "
```

In the data segment, this string is stored in x.

- move → Move the contents of one register to another.

Ex:

move $t0,$t1

Contents of $t1 are moved to $t0

- jal (Jump and link) → Set $ra to Program Counter (return address) then jump to statement at target address.

  Used to jump and link to a function. $ra can be used to return to the position we jumped from.

- add → Used to add the values in 2 registers and store it in a register

Ex:

add $t0, $t1, $t2

$t1 and $t2 are added and answer is stored in $to

- addi → Used to add an immediate value to a register and store the value in another.

Ex:

add $t0, $t1, 5

$t1 and 5 are added and answer is stored in $to

- sw → Used to store a word into the mentioned memory address.

Ex:

sw $t0, ($t1)

- jr → Jump register unconditionally : Jump to statement whose address is in the following register.

Ex:

jr $ra

- Service numbers used are,
  - 1 → print integer | $a0 = integer to print
  - 4 → print string | $a0 = address of null-terminated string to print
  - 5 → read integer | $v0 contains integer read
  - 10 → exit (terminate execution)

LOGIC

Problem 1:

- Stored relevant asciiz text in the data section.
- In main, we only call getNum. FirstNum. Print the final sum and call Exit.
- getNum prompts the user to enter a number and it is read and stored in $v1. To act as a return value.
- We initially call getNum. It stores the users number and returns back to main. After which, firstSum is called.
- In firstSum, we first store $ra (return address to main) in $t8. As $ra will be overwritten.
  Then we move the return value from the initial getNum call to a temp register.
  Then we call getNum again. $ra is overwritten here.
  After recieving the second input we add it to the first input.
  Finally we call secondSum.
- In secondSum, we first store $ra (return address to firstSum) in $t9. As $ra will be overwritten.
  Then we move the return value from firstSum (Sum of first two inputs) to a temp register.
  Then we call getNum again. $ra is overwritten here.
  After recieving the third input, we add it to the initial sum.
- Now we use $t9 to return to firstSum.
- Here we use $t8 to retrun to main.
- After reaching main, we print a result statement and the sum of the three inputs.
- Finally we call Exit. Which gracefully terminates the program.

Sample outputs – Problem 1:

```
Enter a number: 5
Enter a number: 10
Enter a number: 7
Result is: 22
-- program is finished running --

Enter a number: 25
Enter a number: 35
Enter a number: 45
Result is: 105
-- program is finished running --

Enter a number: 6
Enter a number: 6
Enter a number: 6
Result is: 18
-- program is finished running --

Enter a number: 9
Enter a number: 9
Enter a number: 9
Result is: 27
-- program is finished running --
```

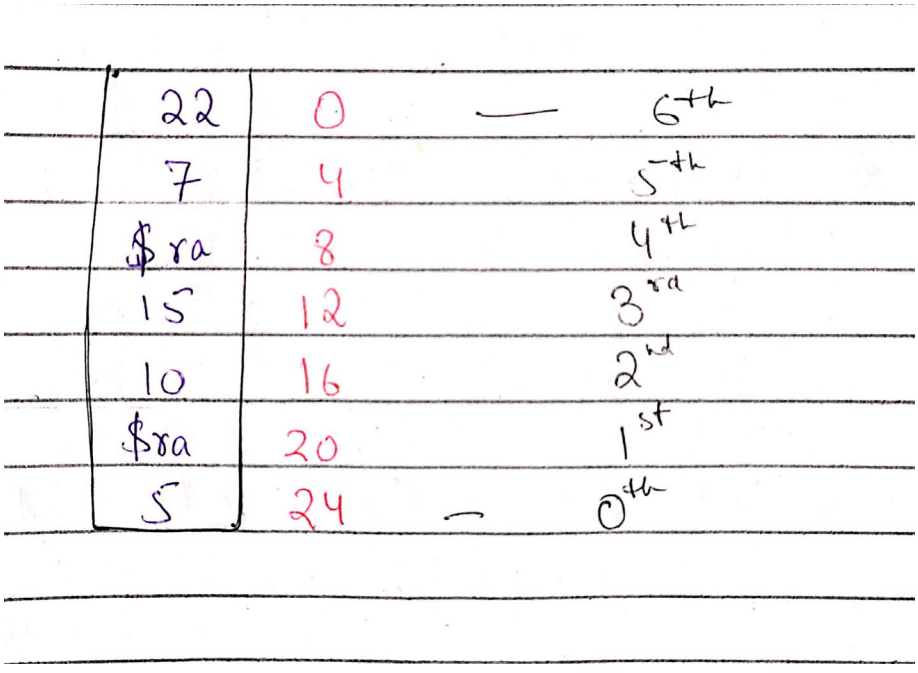
Problem 2:

- Stored relevant asciiz text in the data section.
- In main, we only call getNum. FirstNum. Print the final sum and call Exit.
- It operates exactly like problem 1, except that we use a stack to store all the values and return addresss.
- We initially call getNum. Where we take an input number from the user. And we store it in the stack. This first input, is stored at the 0th position of the stack. We then return to main.
- After we return to main, we go to firstSum. Here we first store the return address to main in the stack at the 1st position.
  We then call getNum. Which stores the second input at the 2nd position.
  Then we load the first and second input from the stack and add both of them. After which we store the sum back into the stack. At the 3rd position.
  We then go to SecondSum.
- Here we first store the return address to firstSum in the stack. At the 4th position.
  We then call getNum. Which stores the 3rd input at the 5th position.
  We then load the initial sum and the 3rd input. Add them together and store it back into the stack at the 6th position.
- Now we load the return address to firstSum from the stack and jump back to firstSum.
- Here we load the return address to main from the stack and jump back to main.
- Now in main we print a result statement and load the final sum from the stack and print it.
- Finally we go to Exit and the program terminates gracefully.

What the stack should be looking like, if the inputs are. 5, 10 & 7.

$ra at the 1st position is the return address to main.

$ra at the 4th position is the return address to firstSum.



Sample outputs – Problem 2:

```
Enter a number: 5
Enter a number: 10
Enter a number: 7
5 + 10 + 7 = 22
-- program is finished running --

Enter a number: 12
Enter a number: 34
Enter a number: 56
12 + 34 + 56 = 102
-- program is finished running --

Enter a number: 78
Enter a number: 910
Enter a number: 1112
78 + 910 + 1112 = 2100
-- program is finished running --

Enter a number: 1
Enter a number: 1
Enter a number: 1
1 + 1 + 1 = 3
-- program is finished running --
```

CODE

Problem 1:

```
.data
        #Program Name: a2_pb1.asm
        #Programmer Name: Muhammad Sameed Gilani
        #Programmer Roll Number: 231488347

prompt: .asciiz "Enter a number: "
result: .asciiz "Result is: "

.text
main:
        jal getNum
        jal firstSum

        #Print fresult statement
        li      $v0,4
        la      $a0,result
        syscall
        #Print final sum
        li      $v0,1
        move    $a0,$v1
        syscall

        jal Exit

getNum:
        #Prints prompt asking user to enter a number
        li      $v0,4
```

```
        la      $a0,prompt
        syscall

        #Reads user input and stores in $v0
        li      $v0,5
        syscall
        move    $v1,$v0         #Store number as return value, in $v1

        jr      $ra             #Go back to caller

firstSum:
        move    $t8,$ra  # RETURN ADDRESS TO MAIN
        move    $t0,$v1 # get initial value from getnum

        jal     getNum
        move    $t1,$v1 #getnum returned value

        add $v1,$t0,$t1 #initial sum of the first 2 inputs



        jal secondSum
        jr      $t8 #Going back to main

secondSum:
        move    $t9,$ra # RETURN address TO firstSum
        move    $t0,$v1 # Sum of first 2 inputs value from firstSum

        jal getNum
        move    $t1,$v1 #return value from getnum
```

```
        add     $v1,$t0,$t1 #add sum of first 2 num and new num from getNum

        jr      $t9     #return to firstsum


Exit:
        #End program gracefully
        li      $v0,10
        syscall
```

Problem 2:

```
.data
        #Program Name: a2_pb2.asm
        #Programmer Name: Muhammad Sameed Gilani
        #Programmer Roll Number: 231488347

prompt: .asciiz "Enter a number: "
plus: .asciiz " + "
eq: .asciiz " = "

.text
main:
        jal getNum
        jal firstSum


        #Print final sum
        lw      $a0,24($sp)     # Accessing first input from stack
```

```
        li      $v0,1
        syscall
        jal     print_plus

        lw      $a0,16($sp)     # Accessing second input from stack
        li      $v0,1
        syscall
        jal     print_plus

        lw      $a0,4($sp)      # Accessing third input from stack
        li      $v0,1
        syscall
        jal     print_eq

        lw      $a0,0($sp)      # Accessing Final Sum from stack
        li      $v0,1
        syscall

        jal Exit


getNum:
        #Prints prompt asking user to enter a number
        li      $v0,4
        la      $a0,prompt
        syscall
        #Reads user input and stores in $v0
        li      $v0,5
        syscall
```

```
        addi    $sp,$sp,-4      # Moves the stack pointer, $sp 4 positions, to make space for an integer
to be stored
        sw      $v0, 0($sp)     # the users input is stored at that position



        jr      $ra             #Go back to caller

firstSum:
        addi    $sp,$sp,-4
        sw      $ra, 0($sp)     # the address to return to main is stored in the 1st position of the stack

        jal     getNum

        lw      $t0,0($sp) #    loading the first input. which is at the 0th position of the stack
        lw      $t1,8($sp) #    loading the second input. which is at the 2nd position of the stack

        add     $t2,$t0,$t1     # adding the 2 inputs together



        addi    $sp,$sp,-4
        sw      $t2, 0($sp)     # storing the sum into the stack at the 3rd position

        jal     SecondSum

        lw      $ra,20($sp)     # loading the address to return to main from the stack. which is at the 1st
position
        jr      $ra     # going back to main

SecondSum:
        addi    $sp,$sp,-4
```

```
        sw      $ra, 0($sp)     # the address to return to firstSum is stored in the 4th position of the
stack

        jal     getNum

        lw      $t0,0($sp) #    loading the third input. which is at the 5th position of the stack
        lw      $t1,8($sp) #    loading the sum of input 1 & 2. which is at the 3rd position of the stack

        add     $t2,$t0,$t1     # adding the inout 3 and the previous sum together

        addi    $sp,$sp,-4
        sw      $t2, 0($sp)     # storing the sum into the stack at the 6th position

        lw      $ra,8($sp)      # loading the address to return to firstSum from the stack. which is at the
4th position
        jr      $ra     #       going back to firstSum

print_plus:
        # Print a plus sign
        li      $v0,4
        la      $a0,plus
        syscall

        jr      $ra

print_eq:
        # print an equal sign
        li      $v0,4
        la      $a0,eq
```

```
        syscall

        jr      $ra

Exit:
        #End program gracefully
        li      $v0,10
        syscall
```