COMP 300

FORMAN CHRISTIAN COLLEGE

(A CHARTERED UNIVERSITY)

COMPUTER ORGANIZATION WITH ASSEMBLY LANGUAGE

Programming Assignment 2

Total Marks: 100

Hard Deadline: Sunday Jun 13, 11:59 pm

This assignment MUST be completed in isolation on individual basis.

It's an open books and open notes task. Use of Internet is allowed, but you must provide references to web sites and /or tutorials from where you got help for this assignment. You MUST NOT share your work with any of your class fellow. Any such attempt will result in a ZERO grade.

Important Note:

- **Start early.**
- **You need to go through the MIPS assembly video lectures uploaded in the last week as well as in this week to complete this assignment.**
- **A delayed submission will be capped with 20% reduction in marks per day. This means if your submission is five days late, you will get a ZERO grade in that assignment.**
- **You need to upload the file/s of working code on google classroom.**
- **If you have multiple parts in a home work, make sure to submit code file of each part separately.**
- **Each file should be saved following the naming format given below as an example:**

  `pa-1-part-1<your FCC roll number>.asm`

  Here pa stands for programming assignment 1, followed by part 1 (or 2, …) of the handout.
- **You should zip your files and name the file as <pa-1<your roll number>>, for example pa-1-25-10548**

Well formatted report should also accompany the code files. Report should be comprehensive and should carry step by step description of your logic.

You MUST add a data dictionary at the start of your program as follows:

```
#Program Name: a1_pb1.asm
#Programmer Name: Rauf Butt
#Programmer Roll Number: 99-99999
```

Problem-1 [50 Marks]

In this part you need to translate the following C program in MIPS assembly language. Do not worry about the `include` statements in the following C program. We won't use any such statement in our assembly program. The `syscall` instruction takes care of this.

Objective of this assignment is to learn about how functions are called and returned.

```
#include <stdio.h>
#include <stdlib.h>
int firstSum(int);
int secondSum(int);
int getNum();
int main()
{
     int n = getNum();
     int out1 = firstSum(n);
     printf("Result is: %d\n",out1);

     return 0;
}

int getNum()
{
     int n;
     printf("Enter a number: ");
     scanf("%d",&n);
     return n;
}
int firstSum(int n)
{
     int k = getNum();
     n = n + k;
     int j = secondSum(n);
     return j;

}
int secondSum(int n)
{
     int m = getNum();
     n = n + m;
     return n;
}
```

As you can see, this C program calls nested functions to add three user given numbers. The algorithm is listed below:

- In the main function, getNum() is called which simply prompts user to enter a number.
- Suppose user enters 5.
- The user given number, 5 in this case, is returned to the main function.

- Here 5 is provided as input argument to a function firstSum().
- This function calls getNum() function again.
- Suppose this time user enters 10.
- Now getNum() function returns to firstSum() and here 10 is added to 5.
- The result 15 is now passed as input argument to another function secondSum().
- The function secondSum() calls getNum() and gets another number from user.
- Suppose this time user enters 7.
- Now the function secondSum() adds 15 and 7, and the result 22 is returned to caller function firstSum().
- The function firstSum() returns the result 22 to its caller function, that is, the main() function.
- Inside the main() function, the result is printed on console.

A skeleton code of the assembly program is shown below. Make sure that you take care of the return addresses and values passed as input argument of the caller/callee functions. Note that conventionally, you only have one register $ra that stores the return address of the function. **Use temporary registers to save the return addresses of functions as well as input arguments to the functions.**

```
.data
prompt1:        .asciiz "Enter a number: "
result:         .asciiz "Result is: "
.text
main:
     jal  getNum
     jal  firstSum
     #now print the result
     #exit the program
     jal  Exit

getNum:
     #here you prompt the user to enter a number
     #the number entered by the user is then returned to the
     #calling function

firstSum:
     #some housekeeping stuff is required
     #call getNum to get a number from user
     #and then call secondSum
     #firstSum returns the result from secondSum to main function
secondSum:
     #call getNum to get third number from user
     #add this number to the result of previous two numbers
     #obtained from user
     #return the result to firstSum

Exit:
     li   $v0,10
     syscall
```

Problem-2 [50 Marks]

Repeat problem 1, but now you need to use stack to save return addresses as well as input arguments to functions. **DONOT USE TEMPORARY VARIABLES IN THIS PART AS YOU DID IN PART 1.**