Arquitectura de Computadores.

Pablo Montoya
Ingeniería de Ejecución en Computación e Informática
Universidad del Bio-Bio
01 de Abril de 2020

Tarea 1.

Al encontrarme con una forma distinta de resolver problemas de programación decidí realizar un código en el **Lenguaje de Programación C** para poder guiarme en el algoritmo que debía implementar en el **Lenguaje Assembler Mips** y también para guiarme en lo que en la parte 2 del enunciado sería el llamar a una función encapsulando el cálculo de la suma de los primeros 10 múltiplos del número ingresado por el usuario en una función. El código y algoritmo implementado es el siguiente:

```c
/* Programa que calcula la suma de los primeros 10 multiplos
    de un numero entero menor a 20. */

#include <stdio.h>

//Función que se usará.
int sumaGaussMultiplo(int numero);

int main(){

    //Variable que almacena el numero ingresado.
    int numero;
    //Variable que almacenará el resultado de la suma.
    int suma;

    //Se imprime mensaje y se pide dato al usuario.
    printf("Ingrese numero positivo menor que 20: ");
    scanf("%d",&numero);

    //Si el numero ingresado es mayor que 20 o si es negativo.
    if(numero>=20 || numero<=0){

        //Se imprime "-1" en pantalla
        printf("-1\n");
    }

    //Se llama a la función si el numero es positivo y menor que 20.
    if(numero<20 && numero>0){

        //"suma" guardará el valor de retorno de la función.
        suma = sumaGaussMultiplo(numero);
```

```
32
33          //Se imprime el resultado en pantalla.
34          printf("La suma de los 10 primeros multiplos es: %d",suma);
35
36      }
37
38      //Se finaliza el programa.
39      return 0;
40  }
41
```

```
42  //Desarrollo de la función.
43  int sumaGaussMultiplo(int numero){
44
45      //Variable que nos ayudará en el ciclo.
46      int i;
47      //Variable almacena la suma.
48      int suma = 0;
49      //Variable que nos dice los multiplos.
50      int multiplo = numero;
51
52      //Ciclo For que nos ayudará con los 10 primeros multiplos.
53      for(i=0;i<10;i++){
54
55          //Se suman los multiplos.
56          suma = suma + multiplo;
57
58          //Se aumenta el multiplo.
59          multiplo = multiplo + numero;
60
61      }
62
63      return suma;
64  }
```

Basándome en el algoritmo implementado entre las líneas 43 y 64 donde nos encontramos con la función **“sumaGaussMultiplo”**, escribí el código en **Lenguaje Assembler Mips** el cual quedó de la siguiente forma:

- Utilizamos el **addi $s1, $zero, 20** ($s1 = 0 + 20) para preguntar luego si el número que se ingrese cumple con la condición de ser menor que 20.

- En la línea 17 le decimos al programa que vamos a imprimir un texto el cual se encuentra en la variable *“pedirNumero”*.

- En la línea 22 le decimos al programa que se pedirá el ingreso de un número entero por teclado, el programa entiende que se pide un número entero gracias al 5 que se encuentra en la instrucción.

- Con **syscall** le decimos al programa que ejecute lo que se pide, es necesario escribirlo una vez que se pide algo.

```
1  # ------------------------------------ Variables necesarias. ------------------------------------ #
2  .data
3          pedirNumero: .asciiz "Ingrese numero positivo menor que 20: "
4          mensajeResultado: .asciiz "La suma de los 10 primeros multiplos es: "
5          mensajeError: .asciiz "-1\n"
6
7  # ------------------------------------ Programa Principal. ------------------------------------ #
8  .text
9
10 .globl main
11
12         main:
13                 # Instrucción agregada en este codigo para la condición if.
14                 addi $s1, $zero, 20
15
16                 # printf("Ingrese numero positivo menor que 20: ");
17                 li $v0, 4
18                 la $a0, pedirNumero # Se imprime el mensaje.
19                 syscall
20
21                 # scanf("%d",&numero);
22                 li $v0, 5  # Se le pide al usuario el ingreso de un dato.
23                 syscall
```

- Validamos el número ingresado utilizando la instrucción **slt** (Set Less Than) dónde obtenemos un 0 o un 1 dependiendo de si se cumple la condición (True: 1, False: 0), al no cumplir la condición de número < 20 nos trasladamos a las instrucciones de error, de la misma forma ocurre si el número ingresado es negativo. Si el número cumple con los requisitos entramos en lo que sería nuestro ciclo donde declaramos las variables que nos ayudarán a que funcione de forma correcta cómo sería la variable *"i"* del código hecho en **Lenguaje C** y también la condición de que se recorran los 10 múltiplos.

- El registro **$s4** tomará el valor 0 y pasará a ser la variable *"suma"* que se encuentra en código hecho en **Lenguaje C**.

```
24
25              # ---- Validación del numero. ---- #
26
27              # if (numero > 20)
28              slt $s5, $v0, $s1   # Si $v0[numero] < $s1[20] -> $s5 = 1
29              beq $s5, $zero, noCumpleCondicion  # Si $s5[True/False] = 0 -> noCumpleCondicion (Porque $s5 = 0 = False)
30
31              # if (numero < 0)
32              slt $s5, $v0, $zero # Si $v0[numero] < 0 -> 1
33              bne $s5, $zero, noCumpleCondicion # Si $s5[True/False] != 0 -> $s5 = 1 (Significa que el numero es negativo)
34
35              # ---- Ciclo for. ---- #
36              move $s0, $v0   # $s0 = numero.
37
38              # Variables necesarias para implementar ciclo For.
39                      # ** for (...
40              add $s2, $zero, $zero   # i=0;
41              addi $t0, $0,10         # i<10;
42
43              # int suma = 0;
44              addi $s4 ,$zero, 0      # Registro que almacenará el resultado de la suma el cual será retornado al main.
45
```

- El registro **$s3** tomará el valor del número y pasará a ser la variable *"múltiplo"* que se encuentra en código hecho en **Lenguaje C**.

- El valor de **$s4** irá aumentando al igual que la variable *"suma"* guardando el resultado de la suma de los múltiplos.

- Antes de repetir el ciclo el registro **$s3** aumenta para que cuando se vuelva a entrar al ciclo se le sume al registro **$s4** el siguiente múltiplo.

```
46              # int multiplo = numero;
47              add $s3, $zero, $v0     # $s3 = numero.
48
49              for: # {
50                      # Esta intrucción tendrá 2 funciones:
51                              # 1) Evaluar la condicion de salida del ciclo For (i<10).
52                              # 2) Salir de la función siendo la instrucción de retorno en el codigo en C (return suma;)
53                      beq $s2,$t0, resultadoFinal
54
55                      # suma = suma + multiplo;
56                      add  $s4,$s4,$s3    # Sumar los multiplos.
57
58                      # multiplo = multiplo + numero;
59                      add  $s3,$s3,$s0        # Se aumenta el multiplo.
60
61                      addi $s2, $s2, 1  # i++) **
62
63                      j for # } Volvemos a repetir el ciclo For
64
```

- Una vez que se hayan sumado los 10 múltiplos el ciclo nos enviará a *"resultado Final"* el cual nos mostrará el resultado de la suma de los múltiplos. Podemos ver las instrucciones para imprimir el mensaje guardado en la variable "*mensajeResultado"* y luego la instrucción **li** con un 1 lo que nos indica que se imprimirá un entero.

- Vemos la instrucción **li** con un 10 lo cual le dice al programa que debe finalizar por lo que nuestra ejecución terminaría luego de imprimir el mensaje de resultado.

```
65              # ---- Mensajes de Resultados. ---- #
66                 # y finalizacion del programa.
67
68              # Resultado de la suma.
69              resultadoFinal:
70
71                      # printf("La suma de los 10 primeros multiplos es:
72                       li $v0, 4
73                       la $a0, mensajeResultado
74                       syscall
75
76                      # %d", suma);
77                       li $v0, 1
78                       move $a0, $s4
79                       syscall
80
81                      #return 0;
82                      li $v0, 10
83                      syscall
84
```

- Si el número ingresado no cumple la condición de ser positivo y menor que 20 el programa nos envía a *"noCumpleCondicion"* donde se imprime el mensaje de error ("-1") y luego finalizamos el programa con la instrucción **li $v0, 10**.

```
85              # Si el numero ingresado no cumple las condiciones.
86              noCumpleCondicion:
87
88                      # printf("1\n");
89                      li $v0, 4
90                      la $a0, mensajeError
91                      syscall
92
93                      #return 0;
94                      li $v0, 10
95                      syscall
```

Para la parte 2 de la **Tarea 1** intenté implementar en **Lenguaje Assembler Mips** lo que está escrito en la línea 31 del programa escrito en **Lenguaje C**. El programa escrito en **Lenguaje Assembler Mips** con la función **"sumaGaussMultiplo"** implementada se encuentra con el nombre **Tarea1.s** ubicado en la carpeta *"Ejercicios"*.

Tarea 2.

Preguntas.

a) **Explique que es el stack o pila de una función:** Una pila nos sirve para poder almacenar los datos de las instrucciones que se van ejecutando. Algunos de sus propósitos son:

- Almacenar dirección de retorno.
- Almacenar datos locales.
- Paso de parámetros.

b) **¿Qué registros de MIPS se emplean para el manejo del stack?**

- **$fp**
- **$gp**
- **$sp**

c) **¿Por qué se señala que la recursividad requiere mucho uso de la memoria?**

Porque al realizar nuevas llamadas se almacenan nuevos datos en la pila lo cual va aumentando la memoria de ésta haciendo cada vez más uso de la memoria.

Código.

Al igual que en el código que se realizó en la tarea 1 escribí el código recursivo en el **Lenguaje de Programación C** para guiarme en **Assembler Mips**. En el procedimiento se agregaron dos variables globales las cuales nos ayudarán para obtener el resultado de la operación. A continuación se muestran las líneas de código de la llamada a la función y la función recursiva con los nuevos parámetros y el algoritmo utilizado:

Variables Globales y Función.

Se agregaron 2 variables globales donde la variable *"numero"* nos servirá como una constante ya que una vez ingresado el número por teclado se sabrá en todo el programa con qué valor trabajamos. Se cambió el nombre del parámetro de la función para poder entender qué valor se está sumando en cada llamada a la función.

Llamado a la función.

Como lo indica el comentario de la línea 34, enviamos el argumento *"número*10"* ya que al tener una función recursiva debemos tener el último múltiplo del valor ingresado para que en las siguientes llamadas se vaya restando y acercándose al caso base.

```
33
34      //(numero*10): Nos servirá para guardar el ultimo multiplo que se sumará.
35          suma = sumaGaussMultiploRecursivo(numero*10);
36
```

Función Recursiva.

En el desarrollo de la función nos encontramos con el caso base y el caso recursivo. En el caso recursivo vemos que se retorna la suma del múltiplo actual con la función llamándose a sí misma pero enviando el múltiplo anterior (*“multiplo-numero”*), estas llamadas harán que el múltiplo vaya disminuyendo hasta llegar al caso base *“multiplo==0”* donde se retornará 0 como neutro aditivo terminando y retornando las llamadas volviendo al **main** con el resultado de la operación.

```
46  //Desarrollo de la función.
47  int sumaGaussMultiploRecursivo(int multiplo){
48
49      //Caso base, cuando se comiencen a retornar los valores.
50      if (multiplo==0)
51          return 0;   //Se retorna 0 ya que es el neutro aditivo.
52
53      else
54          /* Caso recursivo, realizamos esta resta para saber cual es el
55              siguiente numero en la suma. */
56          return multiplo + sumaGaussMultiploRecursivo (multiplo-numero);
57
58  }
```

El código escrito en el simulador **MARS** se encuentra en la carpeta *“Ejercicios”* con el nombre **Tarea2.s** donde se implementó una idea similar a la que se escribió en el programa realizado en el **Lenguaje C**.

Detalles Generales.

Al realizar las tareas 1 y 2 me encontré con problemas para poder implementar el algoritmo en el **Lenguaje Assembler Mips** por lo que fue necesario guiarse con el **Lenguaje de Programación C**.

Los videos tutoriales enviados por el profesor de la asignatura ayudaron para familiarizarse con el programa **MARS**.

Tutorial visitado para aprender sobre las instrucciones:
https://www.youtube.com/playlist?list=PL5b07qlmA3P6zUdDf-o97ddfpvPFuNa5A

Tutorial visitado para guiarse en el desarrollo de la Tarea 2:
https://www.youtube.com/watch?v=VWTfowHHlnU&list=PL5b07qlmA3P6zUdDf-o97ddfpvPFuNa5A&index=34&t=93s