
# Laboratorio de Organización de Computadores n° 2

## "Acercándose al Hardware:

## Programación en Lenguaje Ensamblador"

Profesor:Viktor Tapia Vásquez

Nombre: Cristopher René Alexander Angulo Ahumada

Fecha de Entrega: 27-06-2022

# Tabla de Contenidos



# Introducción

El presente informe tiene como objetivo dar a conocer los resultados obtenidos del Laboratorio n°2. Se hará uso de la interfaz de desarrollo **Mars**, y del lenguaje de programación de bajo nivel **Mips Assembly**.

El objetivo de la experiencia de este laboratorio es comprender el uso de las instrucciones aritméticas, de salto y memoria, así como también entender cómo escribir subrutinas o funciones en **Mips Assembly.**

Se plantean 5 ejercicios para poner en práctica los conocimientos:

1. Calcular la diferencia entre 2 números y determinar si es par o impar.
2. Calcular la diferencia entre 2 números y determinar si es par o impar. Luego, sumar la diferencia dependiendo de si es par o impar, a uno de los números enteros.
3. Programar la multiplicación.
4. Cálculo de Factorial.
5. Programar la división.

Respecto al marco teórico tenemos lo siguiente:

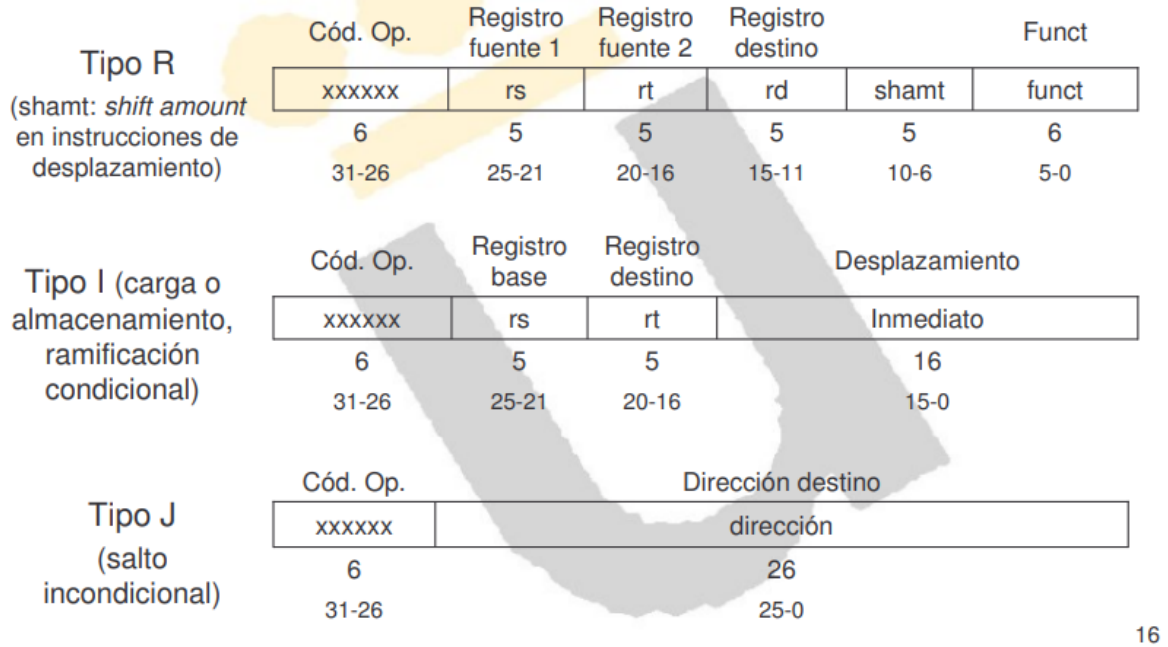
1. Registros: Usaremos los registros para operar. Cada registro tiene un espacio de 32 bits para almacenar nuestras variables.
2. Memoria Principal: Usaremos la memoria para guardar referencias o recuperar valores por medio de ella.
3. Subrutinas: Las subrutinas son las funciones
4. Operadores: Tenemos operadores con los cuáles vamos y existen 3 tipos. Operaciones de tipo R, I y J.

| Registro | Número | Uso | ¿Preservado? |
|---|---|---|---|
| $zero | 0 | Constante con valor 0 | No aplicable |
| $at | 1 | Temporal para el ensamblador | No |
| $v0 – $v1 | 2 – 3 | Valores devueltos en funciones | No |
| $a0 – $a3 | 4 – 7 | Argumentos en funciones | No |
| $t0 – $t7 | 8 – 15 | Temporales | No |
| $s0 – $s7 | 16 – 23 | Temporales salvados | Sí |
| $t8 – $t9 | 24 – 25 | Temporales | No |
| $k0 – $k1 | 26 – 27 | Reservados para kernel | No |
| $gp | 28 | Puntero global | Sí |
| $sp | 29 | Puntero de pila | Sí |
| $fp | 30 | Puntero de marco | Sí |
| $ra | 31 | Dirección de retorno | Sí |

**Descripción**: Registros, contienen un número que los identifica, el uso que se les da y si es preservado.

# 5. Lenguaje máquina MIPS

• Recordemos los tres **formatos de instrucciones máquina** en MIPS:




16

**Descripción**: Tipos de operaciones (R,I,J)

Se deja en evidencia los resultados obtenidos y el aprendizaje adquirido de esta actividad haciendo uso de las herramientas vistas en clases y lo investigado.

## Desarrollo y Resultados

Como se mencionó anteriormente, tenemos 5 preguntas que resolver, por lo tanto, en esta sección dejaremos en evidencia el desarrollo de este laboratorio.

a) Escribe un programa que lea dos enteros ingresados por el usuario, determine si su diferencia es par o impar y luego imprima este resultado. El cálculo del máximo debe ser realizado en una subrutina, utilizando los registros apropiados para argumentos y salida de un procedimiento.

**Solución:**

La solución se puede encontrar en el archivo “parte1a.asm”.

Para esta solución, lo más es importante es lo siguiente:

1. **andi**: Con esta instrucción aislamos el bit menos significativo y lo comparamos con 1, si el resultado es 1 significa que es impar y si es 0 significa que es par. El punto es que al restar o sumar 1 número par con un par siempre obtenemos un impar, y si sumamos 2 valores pares siempre obtenemos par, ahí está el fundamento de esta solución.
2. **beqz**: Comparamos el resultado que obtenemos en andi con 0 y saltamos a la instrucción correspondiente (Even, Odd).

```
andi $s1, $s0, 1
beqz $s1, Even    # branch equal to zero

j Odd
```

Un detalle importante es que siempre transformamos los números a positivos para evitar posibles errores. Además, el resultado siempre será un valor positivo.

**Resultados**:

```
Por favor ingrese el primer entero: 4
Por favor ingrese el segundo entero: 8
El numero es par: 4
-- program is finished running --

Por favor ingrese el primer entero: 3
Por favor ingrese el segundo entero: 4
El numero es Impar: 1
-- program is finished running --
```

b) A partir del código anterior: en otra subrutina, si el resultado de la diferencia obtenida entre el primer y segundo entero es par, sume el valor de la diferencia al primer entero y luego imprima el resultado obtenido por pantalla. Si la diferencia es impar, sume el valor al segundo entero y muestre el resultado por pantalla.

**Solución:**

La solución se puede encontrar en el archivo “parte1b.asm”.

En este ejercicio, además de calcular la diferencia entre 2 números, teníamos que sumar su resultado a 1 de los números ingresados por el usuario. Aquí podríamos tomar 1 de los 2 supuestos que se describen a continuación:

1. El usuario puede ingresar números negativos por error.
2. El usuario deliberadamente quiere saber la diferencia entre un número negativo y otro positivo.

Se escogió la opción 2 para resolver el ejercicio y aprovechamos el código del ejercicio uno para resolverlo.

**Resultados**:

```
Por favor ingrese el primer entero: 5
Por favor ingrese el segundo entero: 4
El numero es Impar: 1
El nuevo segundo entero es: 5
-- program is finished running --

Por favor ingrese el primer entero: 5
Por favor ingrese el segundo entero:7
El numero es par: 2
El nuevo segundo entero es: 7
-- program is finished running --
```

```
Por favor ingrese el primer entero: -5
Por favor ingrese el segundo entero: 4
El numero es Impar: 9
El nuevo segundo entero es: 13
-- program is finished running --

Por favor ingrese el primer entero: -4
Por favor ingrese el segundo entero: 5
El numero es Impar: 9
El nuevo segundo entero es: 14
-- program is finished running --
```

Para clarificar, la diferencia entre entre -4 y 5 son 9 posiciones. La diferencia entre -5 y 4, es 9 también, pero los resultados son diferentes puesto que en el caso se suma 9+5, siendo 5 un número positivo. En el segundo caso, se suma 9+4.

```
Por favor ingrese el primer entero: 5
Por favor ingrese el segundo entero: -4
El numero es Impar: 9
El nuevo segundo entero es: 5
-- program is finished running --

Por favor ingrese el primer entero: 4
Por favor ingrese el segundo entero:-5
El numero es Impar: 9
El nuevo segundo entero es: 4
-- program is finished running --
```

Aquí los números son los mismos, pero sus posiciones están cambiadas. Nos damos cuenta que el 9 se suma con el -4 o con el -5.

A) Escribe un programa en MIPS que calcule la multiplicación de dos enteros mediante la implementación de subrutinas. **No** se pueden utilizar instrucciones de multiplicación, división y desplazamiento: mul, mul.d, mul.s, mulo, mulou, mult, multu, mulu, div, divu, rem, sll, sllv, sra, srav, srl, srlv; sino que se debe implementar una técnica de multiplicación basada en otras operaciones matemáticas y el uso de subrutinas. Para este programa, los operandos deben estar escritos "en duro" en el mismo código (*i.e.*, no es necesario pedirlos al usuario vía consola) y deben estar claramente identificados para poder probar con otros valores al evaluar tu trabajo.

**Solución:**

La solución se puede encontrar en el archivo “parte2a.asm”.

El fragmento de código que se muestra a continuación es el más importante, puesto que simula la operación de multiplicación usando la suma.

A través de un loop vamos sumando el registro $t1 y su acumulado en $t4, hasta que el registro $t3 llegue a cero.

Luego, recuperamos el signo con el operador xor para poder agregarle el signo negativo si es necesario al resultado final, porque como se aprecia en el código, transformamos los números a valores positivos para simular multiplicación.

```
abs $t1, $s1
abs $t3, $s3

Loop:
  beqz $t3, End_Loop

  add $t4, $t4, $t1
  addi $t3, $t3, -1

  j Loop

End_Loop:

   xor $t5, $s1, $s3
   bltz $t5, Negate

   j End

Negate:
   neg $t4, $t4
   j End
```

**Resultados**:

A continuación se muestran los resultados de todas las combinaciones posibles de la multiplicación de enteros positivos y negativos. También, se puede probar la multiplicación por cero.

```
.data
  operand1:  .word 2
  operand2:  .word -10
```

```
-20
-- program is finished running --
```

```
.data
  operand1:  .word -2
  operand2:  .word -10
```

```
20
-- program is finished running --
```

```
.data
  operand1:  .word -2
  operand2:  .word 10
```

```
-20
-- program is finished running --
```

```
.data
  operand1:  .word 2
  operand2:  .word 10
```

```
20
-- program is finished running --
```

B) Utilizando tu programa de la parte A), escribe un programa que calcule el factorial de un número entero.

**Solución:**

La solución se puede encontrar en el archivo “parte2b.asm”.

El siguiente código es un poco más complejo así que lo explicaremos por pasos.

1. Si el valor del operando es menor que 1 el factorial es 1 y lo enviamos a la subrutina Less_Than_1
2. En Init_Loop guardamos en 2 registros el valor del operando menos 1
3. En Loop sumamos en cada iteración el valor del operando. Cuando llegué a cero el registro $t5 saltamos a Loop_Setting
4. En Loop_Setting, preguntamos si el registro $t6 es igual o menor a 1. Asignamos llas variables para multiplicar el siguiente número. El ciclo se repite hasta que ya no quedé ningún número por multiplicar

```
la $t0, operand1
lw $t1, 0($t0)
ble $t1,1 Less_Than_1
move $t4, $zero
Init_Loop:
    subi $t5, $t1, 1
    subi $t6, $t1, 1
    j Loop
Loop_Setting:
    ble  $t6,1 End
    subi $t6, $t6,  1
    move $t5, $t6
    move $t1, $t4
    move $t4, $zero
    j Loop
Loop:
   beqz $t5,  Loop_Setting
   add $t4, $t4, $t1
   addi $t5, $t5, -1
   j Loop
Less_Than_1:
   addi $t4, $t4, 1
```

**Resultados**:

```
.data
    operand1:  .word  1
```

```
1
-- program is finished running --
```

```
.data
    operand1:  .word  4
```

```
24
-- program is finished running --
```

C) Escribe un programa en MIPS similar al de A) que calcule la división de dos enteros mediante la implementación de subrutinas. Al igual que en A), **no** se pueden utilizar instrucciones de multiplicación, división y desplazamiento: mul, mul.d, mul.s, mulo, mulou, mult, multu, mulu, div, divu, rem, sll, sllv, sra, srav, srl, srlv; sino que se debe implementar una técnica de división basada en otras operaciones matemáticas y el uso de subrutinas. Para el caso de divisiones no exactas (*i.e.*, resto no nulo), el programa debe ser capaz de calcular hasta 2 decimales del cociente, sin atender a errores de precisión más allá del segundo decimal.

**Solución:**

La solución se puede encontrar en el archivo “parte2c.asm”.

Este problema por lejos es el más complejo de resolver, puesto que debemos encontrar alguna manera de almacenar los decimales de la división.

El proceso general del algoritmo se divide en 2:

1. Cálculo del Cociente: Para el cálculo se tomó en cuenta varios factores. Transformar el dividendo y divisor a positivos, saltar el cálculo del cociente si el dividendo es menor al divisor e ir calcular directamente los números decimales. La división básicamente es una resta sucesiva que se hace entre el divisor y el dividendo en un loop.
2. Cálculo de los decimales: Es prácticamente una división pero vamos guardando los valores decimales en un arreglo, luego es cuestión de mostrarlos por pantalla en algún formato. Para realizar este cálculo el dividendo se va multiplicando por 10 ( sin usar el uso del mul), y se efectúa la división. Se va contando y almacenando ese valor en el arreglo.

Nota. El código es demasiado extenso como para adjuntar una imagen.

Se probaron todos los casos posibles, pero a continuación mostraremos solo un par (también se guarda el signo de la división).

**Resultados**:

```
.data
dividend:       .word 20
divisor:        .word -10
```

```
Remainder: 0 Quotient: -2.0000
-- program is finished running --
```

```
.data
dividend:       .word -20
divisor:        .word -10
```

```
Remainder: 0 Quotient: 2.0000
-- program is finished running --
```

```
.data
dividend:       .word 10
divisor:        .word -20
```

```
Remainder: 10 Quotient: -0.5000
-- program is finished running --
```

```
.data
dividend:       .word 10
divisor:        .word -22
```

```
Remainder: 10 Quotient: -0.4545
-- program is finished running --
```

# Conclusiones

En síntesis, logramos cumplir los objetivos. Algunas soluciones como la multiplicación o el factorial fueron resueltas de una manera elegante, a diferencia de la división en la cuál tuvimos que aplicar varios trucos para lograr el objetivo. Lo más complejo del laboratorio sin duda fue programar la división con decimales. Además, hubo varios topes en el camino, como por ejemplo, mantener el signo, multiplicar por 10, donde guardar los decimales, pero al final se logró hacer.

También, programamos las soluciones  hasta cumplir con todos los casos bordes posibles.

Finalmente, aprendimos a desarrollar programas en el lenguaje de programación Mips Assembly, los cuáles estarán adjuntos para su revisión.

# Referencias Bibliográficas

[1] Amell Peralta. (2014, December 26). MIPS Tutorial 1 Intro and Mars [Video]. YouTube. https://www.youtube.com/watch?v=u5Foo6mmW0I

[2] Wikipedia contributors. (2023). MIPS architecture. Wikipedia. https://en.wikipedia.org/wiki/MIPS_architecture