UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Organización de Computadores

# Laboratorio 2: Acercándose al Hardware: Programación en Lenguaje Ensamblador

Nombre: Agustín Vera Rojas

Sección: L-1

Profesor: Viktor Tapia

Ayudante: Nicolas Henríquez

# Introducción

En el presente informe se documentará las distintas actividades realizadas para el segundo laboratorio correspondiente a la asignatura Organización de Computadores. Este informe presenta la siguiente estructura: introducción, marco teórico, desarrollo y resultados, conclusiones del laboratorio y se finalizará con las referencias utilizadas para realizar este trabajo. Para la realización del laboratorio se utilizará el software MARS, para realizar las distintas actividades que requieren programación en lenguaje ensamblador.

Para la realización de la actividad se declaran los siguientes objetivos:

- Usar MARS (un simulador para MIPS) para escribir, ensamblar y depurar programas MIPS.
- Escribir programas MIPS incluyendo instrucciones aritméticas, de salto y memoria.
- Comprender el uso de subrutinas en MIPS, incluyendo el manejo del *stack.*
- Realizar llamadas de sistema en MIPS mediante *“syscall”.*
- Implementar algoritmos en MIPS para resolver problemas matemáticos.

Principalmente se tiene como problema de este laboratorio resolver las diferentes actividades planteadas en el enunciado del laboratorio 1. Las actividades por resolver se separan en tres partes, la primera es hacer uso de *syscall* (llamado a sistema) en un programa MIPS. La segunda parte corresponde a escribir programas MIPS para resolver problemas matemáticos. Finalmente, la tercera parte corresponde a realizar programas que realicen aproximaciones a diferentes funciones matemáticas.

# Marco Teórico

Con el fin de brindar un contexto sobre los diferentes temas en las distintas actividades que se abordaran en este laboratorio, se propone un marco teórico para que el lector se familiarice con los diferentes conceptos que se abordaran en este laboratorio.

**MARS:** “MARS, Mips Assembly and Runtime Simulator, ensamblará y simulará la ejecución de programas en lenguaje ensamblador MIPS” (MissouriState, 2014).

**Lenguaje Ensamblador:** El lenguaje ensamblador, o ASM, es un lenguaje de programación de bajo nivel que permite escribir programas que interactúan con el hardware del ordenador (Romero, 2022).

**Instrucciones aritméticas:** Son todas aquellas instrucciones que permiten realizar operaciones matemáticas, tales como: sumar, multiplicar, restar, etc. (Tapia, 2023).

**Instrucciones de salto:** Son todas aquellas instrucciones que permiten realizar distintos saltos en la ejecución de instrucciones de un programa, alteran el flujo en el cual se ejecuta el programa (Tapia, 2023).

**Instrucciones de memoria:** Son todas aquellas instrucciones que permiten acceder a la memoria principal, con el fin de poder leer información de la memoria o para escribir información en un sector especifico dentro de la memoria (Tapia, 2023).

**División:** Operación aritmética que consiste en separar en partes iguales un total, si se tiene una cantidad y se quiere dividir en partes iguales, se puede usar esta operación aritmética para determinar en cuántas partes iguales se pueden obtener y qué cantidad corresponde a cada parte de esta (Significados.com., 2023).

**Syscall:** “Una llamada al sistema o system call es un método utilizado por los programas de aplicación para comunicarse con el núcleo del sistema. En los sistemas operativos modernos, esto es necesario cuando una aplicación o proceso de usuario necesita transmitir a o leer información del hardware, de otros procesos o del propio núcleo” (IONOS, 2022).

**Subrutinas:** “Una subrutina es un conjunto de instrucciones que realizan una tarea concreta. Es un pequeño programa que puede incorporarse en un programa y accederse a él con una sentencia GOSUB, o puede ser externo al programa y accederse a él con una sentencia CALL. Los procesos comunes a menudo se guardan como subrutinas externas. Este método permite al programador acceder a ellos desde muchos programas distintos sin tener que volver a escribirlos” (IBM, 2021).

**Números Pares e Impares:** Los números pares son todos aquellos números que pueden dividirse por dos y que el resto de esta división sea cero, en cambio, los números impares son los que frente a la división anterior poseen un resto distinto a cero.

**Stack:** Contiene información acerca de la configuración inicial de la pila del programa, crece hacia abajo desde la dirección más alto, a medida que se le ingresen datos. Estos datos son temporales y generalmente son direcciones de retorno al invocar un procedimiento o subrutina (Depto. CS. e Ing de la Comp. Universidad Nacional del Sur, 2019).

**Factorial:** La función factorial se representa con un signo de exclamación “!” detrás de un número. Esta exclamación quiere decir que hay que multiplicar todos los números enteros positivos que hay entre ese número y el 1. Por ejemplo, el factorial de seis:
6! = 1 x 2 x 3 x 4 x 5 x 6 = 720 (Blanco, 2022).

# Desarrollo de la Solución

Inicialmente, para realizar una correcta solución para el laboratorio, se aprendió a utilizar el software MARS para solucionar las tareas que requerían soluciones con lenguaje ensamblador, esto fue aprendido en la experiencia anterior de laboratorio. Este aprendizaje se complementó con el aprendizaje adquirido sobre los diferentes conceptos abarcados en el marco teórico, los cuales fueron vistos en clases de teoría como también autónomamente.

Con la finalidad de explicar el desarrollo de la solución, se explicarán las soluciones planteadas para las actividades “Uso de Syscall”, “Subrutinas para Factorial y División” y las dificultades con el desarrollo de la parte: “Aproximación de Funciones Matemáticas”.

## 1. Uso de Syscall

Para el desarrollo de esta actividad, primeramente, se investigó que era syscall, en conjunto a los diferentes usos que se le puede dar. Posteriormente, se realizó un programa que haciendo uso de syscall le muestra al usuario por consola diferentes mensajes pidiendo que le ingrese números enteros, el usuario puede ingresar números enteros por consola en tiempo de ejecución, está interacción finaliza cuando el programa entrega el resultado de la paridad del resultado de la resta de los dos números ingresados por el usuario. Para el cálculo de la paridad se crea una subrutina que calcula el máximo entre los dos números y realizando esa diferencia se calcula la paridad de la resta entre el número máximo y el mínimo. Se presenta un fragmento del cómo se llega a la solución de este ejercicio.

```
calcularMaximo:
        bge $a0, $a1, entregarMaximo
        move $v0, $a1    # Retorna el
        move $v1, $a0    # Retorna el
        jr $ra    # Vuelve al main
entregarMaximo:
        move $v0, $a0    # Retorna el
        move $v1, $a1    # Retorna el
        jr $ra    # Vuelve al main
```

```
calculoDeParidad:
        sub $t0, $a0, $a1  # Se calcula
        # Se imprime el mensaje de difer
        li $v0, 4
        la $a0, output
        syscall
        # Impresion del numero respectiv
        li $v0, 1     # 1 indica a Sysca
        move $a0, $t0
        syscall

        # Calculo de paridad
        addi $t0, $0, 2 # $t0 = 2 --> Se
        div $a0, $t0    # div --> divide
        mfhi $t1        # Obtiene el res
        beq $t1, $0, esPar  # Si $t1 = 0
        # Se imprime el mensaje de Impar
        li $v0, 4
        la $a0, impar
        syscall
        jr $ra   # Vuelve al main
esPar:
        # Se imprime el mensaje de Par
        li $v0, 4
        la $a0, par
        syscall
        jr $ra   # Vuelve al main
```

*Figura 1: Subrutinas para ejercicio 1.*

## 2. Subrutinas para Factorial y División

Para el desarrollo de esta actividad, se realizó el primer ejercicio en torno a aplicación de la función factorial en un programa MIPS.

Primeramente, debido a que la fórmula de factorial implica realizar una serie de multiplicaciones para llegar a la solución, se utilizó el código del programa realizado para multiplicar dos números enteros, programa respectivo al laboratorio 1. Cabe recalcar que a dicho programa se le realizaron cambios menores con el fin de generalizar dicho programa.

Para poder hacer uso de la subrutina multiplicar en la nueva subrutina a crear "Factorial", se debió hacer uso de una forma nueva de realizar saltos de memoria, la cual es realizar saltos de memoria variados, utilizando la función *jal label* la cual permite saltar a un label almacenando el PC+4 en el registro $ra, al realizar esto en más de una ocasión, se sobrescribe el registro $ra y se pierden las direcciones guardadas, para resolver dicha problemática y dar solución a este ejercicio, se hizo uso del almacenamiento de direcciones en el *stack*.

Para la utilización del programa generado como solución a este ejercicio, el usuario deberá ingresar el número al cual se le calculara su factorial dentro del mismo programa, es decir, el usuario debe modificar "a mano" el registro correspondiente a $a0, el cual es el registro de entrada para la subrutina factorial. El usuario podrá ver el resultado del número que ingreso por la consola del IDE. A continuación, se presenta un fragmento del cómo se llega a la solución propuesta.

```
Factorial:
        addi $sp, $sp, -4   # $sp = $sp - 4 --> Decrementar el puntero de pila
        sw $ra, 0($sp)      # Guardar $ra en la pila
        move $s2, $a0       # $s2 = $a0 --> La entrada (factorial a ser calculado) se g
        addi $t3, $0, 1     # $t3 = 1 --> Este registro ayudara en los calculos del fac
        addi $t4, $0, 1     # $t4 = 1 --> Este registro ayudara en los calculos del fac
        j LoopFactorial

LoopFactorial:
        beq $s2, $0, FactorialDeCero    # ¿ $s2 = $0 ? --> Si, salta  a FactorialDeCero
        bgt $t4, $s2, FinFactorial      # ¿ $t4 > $s2 ? --> Si, salta a FinFactorial
        move $a0, $t3       # $a0 = $t3 --> Operando 1 contiene la acumulacion de multi
        move $a1, $t4       # $a1 = $t4 --> Operando 2 contiene el siguiente numero a m
        jal Multiplicar     # Salta a la subrutina Multiplicar
        move $t3, $v0       # $t3 = $v0 --> Almacena en $t3 el resultado de la multipli
        addi $t4, $t4, 1    # $t4 = $t4 + 1 --> Aumenta el operando 2
        j LoopFactorial     # Salta a LoopFactorial

FactorialDeCero:
        addi $v0, $0, 1     # $v0 = 1 --> El factorial de 0 es igual a 1
        jr $ra              # Salta a la instruccion posterior al uso de jal Factorial

FinFactorial:
        # Recuperar $ra de la pila (stack)
        lw $ra, 0($sp)      # Cargar $ra de la pila
        addi $sp, $sp, 4    # Incrementar el puntero de pila
        jr $ra              # Salta a la instruccion posterior al uso de jal Factorial
```

*Figura 2: Subrutina Factorial para ejercicio 2A.*

Para la realización del ejercicio dos, se debía tener en cuenta que no se podía hacer uso de instrucciones multiplicación, división y desplazamiento. Para calcular la división de dos enteros mediante la implementación de subrutinas, se realizaron distintas subrutinas alrededor del código con el fin de abarcar todos los casos que presenta la división, incluyendo cuando el cociente de la división es un número decimal, para ello se pide calcular dos decimales. La idea general para resolver esta problemática fue ir restando el dividendo menos el divisor, esto se realiza hasta que el dividendo se vuelve menor que el divisor, con ello se debe obtener el primer decimal, para ello se multiplica lo que queda del dividendo por diez, luego, se realiza lo mismo mencionado anteriormente, se disminuye el dividendo restándole el divisor, en caso de que nuevamente el dividendo llegue a ser menor que el divisor, se calcula el segundo decimal, se multiplica lo que queda del dividendo por diez y se realizan los mismos pasos mencionados anteriormente, para realizar las multiplicaciones se hace uso del programa creado para la multiplicación de dos números, respectivo al laboratorio 1.

Para el programa planteado para dar solución a este ejercicio, se creó la subrutina división como fue mencionado anteriormente, pero, al tener variados saltos de memoria, se hizo uso de almacenamiento de memorias en el stack. Se presenta un fragmento de la solución de este ejercicio.

```
Division:
        beq $a0, $0, FinDivision
        bgt $a1, $a0, MultiplicacionPrimerDecimal
        addi $v0, $v0, 1        # Se le agrega 1 a la parte entera por cada vez que
        sub $a0, $a0, $a1       # Dividendo - Divisor
        j Division

# Se multiplica por 10 el primer decimal y se continua la division
MultiplicacionPrimerDecimal:
        move $s3, $a0
        move $s4, $a1
        move $s5, $v0         # En $s5 se mantiene el resultado de la parte entera
        addi $a1, $0, 10
        jal Multiplicar    # DividendoRestante * 10
        move $a0, $v0
        move $a1, $s4
        move $v0, $s5
        j DivisionPrimerDecimal

DivisionPrimerDecimal:
        beq $a0, $0, FinDivision
        bgt $a1, $a0, MultiplicacionSegundoDecimal
        addi $v1, $v1, 1        # Se le agrega 1 al primer decimal por cada vez que
        sub $a0, $a0, $a1       # Dividendo - Divisor
        j DivisionPrimerDecimal
```

*Figura 3: Fragmento de subrutina división para el ejercicio 2B.*

# Resultados

Luego de realizar el desarrollo a las diferentes actividades planteadas, se procede a evaluar las soluciones propuestas en el apartado anterior. Para ello, se procederá a mostrar las diferentes actividades por separado.

## 1. Uso de Syscall

Para este apartado se mostrarán imágenes de variadas entradas ingresadas por el usuario cuando el programa lo indique, y las diferentes salidas que este provee para cada caso generado por el usuario.

```
Por favor ingrese el primer entero: 31
Por favor ingrese el segundo entero: 10
La diferencia es: 21 (Impar)
-- program is finished running --
```

*Figura 4: Evaluación de los enteros 31 y 10 en ejercicio 1.*

```
Por favor ingrese el primer entero: 20
Por favor ingrese el segundo entero: -20
La diferencia es: 40 (Par)
-- program is finished running --
```

*Figura 5: Evaluación de los enteros 20 y -20 en ejercicio 1.*

```
Por favor ingrese el primer entero: -50
Por favor ingrese el segundo entero: -25
La diferencia es: 25 (Impar)
-- program is finished running --
```

*Figura 6: Evaluación de los enteros -50 y -25 en ejercicio 1.*

## 2. Subrutina para Factorial y División

Para este apartado se mostrarán distintas tablas con los distintos resultados de las actividades, con diferentes evaluaciones a estas.

| Registro de Entrada | Número | Resultado = Número! |
|---|---|---|
| $a0 | 0 | 1 |
| $a0 | 1 | 1 |
| $a0 | 2 | 2 |
| $a0 | 3 | 6 |
| $a0 | 4 | 24 |

| | | |
|---|---|---|
| $a0 | 5 | 120 |
| $a0 | 6 | 720 |

*Tabla 1: Distintos valores obtenidos en evaluación de ejercicio 2A.*

| Variable | Registro | Valor | Resultado |
|---|---|---|---|
| Dividendo | $a0 | 6 | 2.00 |
| Divisor | $a1 | 3 | |
| Dividendo | $a0 | 3 | 0.50 |
| Divisor | $a1 | 6 | |
| Dividendo | $a0 | 10 | -3.33 |
| Divisor | $a1 | -3 | |
| Dividendo | $a0 | -9 | 1.28 |
| Divisor | $a1 | -7 | |

*Tabla 2: Distintos valores obtenidos en evaluación de ejercicio 2B.*

# Conclusiones

En base a los objetivos planteados, se puede comentar que se logró cumplir con la gran mayoría de los objetivos planteados en el laboratorio: se logra en entender los diferentes usos de *syscall* y con ello, la implementación de diferentes programas que hacen uso de este, además se comprende y se aplica la utilización del *stack* para el almacenamiento de variadas direcciones de memoria. Asimismo, se realizan distintos algoritmos para resolver diferentes problemas matemáticos.

Se tuvieron complicaciones con el desarrollo de la actividad “Parte 3 : Aproximación de funciones matemáticas”, debido a que no se distribuyo de manera adecuada el tiempo que requería la realización de este laboratorio, sin embargo, se espera en un futuro visualizar la posibilidad de completar la actividad debido que con lo desarrollado en el laboratorio 1 y lo realizado en este laboratorio, específicamente en la actividad 2A y 2B, la parte 3 se podría haber realizado, debido a que estas son las bases de la actividad.

Con respecto a la entrega de laboratorio 1 y realizando una comparación con las actividades de el presente informe de laboratorio, se diferencian en la complejidad del nivel de programación que se presenta en el laboratorio 2, en el cual se implementó una gran cantidad de subrutinas a diferencia del laboratorio anterior, además se implementó el acceso al stack para manejar los saltos de memoria, en conjunto de la realización de ejercicios matemáticos de mayor complejidad.

Finalmente, se puede concluir que se realiza un trabajo en su mayoría, adecuado, se realizan casi todas las actividades planteadas y se muestra un desglose de estas a través de presente informe en cada apartado que este posee, agregando que estas actividades proveen una serie de conocimientos al realizarlas, conocimiento que posiblemente será empleado en las próximas entregas del laboratorio.

# Referencias

Blanco, I. D. (22 de 12 de 2022). *Factoriales. ¿Para qué los utilizamos?* Obtenido de https://www.smartick.es/blog/matematicas/numeros-enteros/factoriales/

Depto. CS. e Ing de la Comp. Universidad Nacional del Sur. (2019). Lenguaje Ensamblador. *Organización de Computadoras Módulo 10*, 11, 43.

IBM. (28 de 2 de 2021). *Introducción a InfoSphere DataStage BASIC*. Obtenido de https://www.ibm.com/docs/es/iis/11.5?topic=basic-subroutines

IONOS, D. G. (6 de 5 de 2022). *System calls: ¿qué son y para qué se emplean?* Obtenido de https://www.ionos.es/digitalguide/servidores/know-how/que-son-las-system-calls-de-linux/

MissouriState. (8 de 2014). *MARS - Mips Assembly and Runtime Simulator*. Obtenido de http://courses.missouristate.edu/kenvollmar/mars/Help/MarsHelpIntro.html

Romero, I. (3 de 12 de 2022). *Profesional review*. Obtenido de https://www.profesionalreview.com/2022/12/03/ensamblador-vs-codigo-maquina/

Significados.com. (13 de 3 de 2023). *Qué es la División (matemáticas)*. Obtenido de https://www.significados.com/division/

Tapia, V. (2023). Lenguaje Ensamblador y Máquina. *Organización de Computadores – Segundo Semestre 2023*.