

**UNIVERSIDAD DE SANTIAGO DE CHILE**  
**FACULTAD DE INGENIERÍA**  
**Departamento de Ingeniería Informática**



**LABORATORIO 3: ACERCÁNDOSE AL HARDWARE - PROGRAMACIÓN  
EN LENGUAJE ENSAMBLADOR**

**Organización de Computadores**  
**Isaac Alejandro Espinoza Barría 21.278.828-7**

**Sección:**

13275-0-C-3 Diurno

**Profesor:**

Viktor Tapia

**Ayudante:**

Nicolás Henríquez

**Fecha:**

1 de diciembre de 2024

# TABLA DE CONTENIDO

TABLA DE CONTENIDO.....	2
1. INTRODUCCIÓN.....	3
2. MARCO TEÓRICO .....	4
2.1. ARQUITECTURA MIPS Y SUS REGISTROS.....	4
2.2. INSTRUCCIONES MIPS .....	5
2.3. INSTRUCCIONES DE DIRECCIONAMIENTO.....	5
2.4. RUTINA Y SUBROUTINA .....	5
2.5. LLAMADAS DE SISTEMA O SYSCALL.....	6
2.6. MANEJO DE STACK O PILA .....	7
2.7. SERIE DE TAYLOR .....	8
3. DESARROLLO Y RESULTADOS .....	9
3.1. FUNCIÓN EXPONENCIAL.....	9
3.2. FUNCIÓN COSENO.....	10
3.3. FUNCIÓN LOGARITMO NATURAL.....	11
4. CONCLUSIONES.....	12
REFERENCIAS .....	13

# 1. INTRODUCCIÓN

La invención del computador marcó un hito importante en la historia de la humanidad. En el mundo actual, la gran mayoría de procedimientos funcionan con computadores; estos pueden automatizar procesos que hace 40 años resultaban muy tardíos y difíciles de desarrollar. Su impacto ha llegado a tal punto que la gran mayoría de la población posee un teléfono celular, el que desde cierto punto de vista podría considerarse un computador móvil.

A pesar del uso mundial de estos aparatos, no entendemos cómo funcionan. Los computadores ejecutan instrucciones, donde éstas son un conjunto de valores binarios, también llamado lenguaje máquina. Sin embargo, son difíciles de interpretar para el ser humano, por lo que se emplea el lenguaje ensamblador. Este consiste en ser un intermediario entre el lenguaje que entienden los computadores y el programador, posee elementos que facilitan la programación a nivel de hardware y la interpretación de las instrucciones empleadas en lenguaje máquina.

La programación en lenguaje ensamblador resulta más difícil que programar en lenguajes de alto nivel como Python o Java. Esto debido a errores comunes que se cometen al programar sin considerar las limitaciones del lenguaje. Por ello, se requiere tener en cuenta aspectos como el uso del stack, subrutinas y llamadas de sistema (syscall).

El presente informe aborda un tercer acercamiento a la programación en lenguaje ensamblador para un laboratorio del curso Organización de Computadores, enfocado en la arquitectura MIPS de 32 bits.

Los objetivos de la experiencia son: usar MARS (un IDE para MIPS) para escribir, ensamblar y depurar programas MIPS; escribir programas MIPS que usan 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; e implementar algoritmos en MIPS para resolver problemas matemáticos.

Con esto, se busca desarrollar un mayor conocimiento del lenguaje ensamblador, al considerar el uso de syscall, el stack y las subrutinas.

## 2. MARCO TEÓRICO

Un computador se define como una “máquina capaz de aceptar unos datos de entrada, efectuar con ellos operaciones lógicas y aritméticas, y proporcionar los datos resultantes a través de medios de salida; todo ello ... bajo el control de un programa de instrucciones” (Prieto, Lloris, & Torres, 1989). Donde estas instrucciones se pueden entender de dos maneras: en lenguaje máquina, que consiste en un conjunto de valores binarios de difícil lectura que indican al computador qué acciones debe realizar; y en lenguaje ensamblador, el que también especifica funcionalidades al computador pero es una representación más legible para el programador y la escritura de programas (Castillo Catalán & Claver Iborra, 2001).

### 2.1. ARQUITECTURA MIPS Y SUS REGISTROS

Dentro de un computador, el procesador se entiende como el “cerebro” que ejecuta las instrucciones según el tipo de arquitectura implementada. “Una arquitectura son los atributos del sistema computacional que son visibles a un programador, ya sean, set de instrucciones, número de bits usados para representar un dato, mecanismos de entrada y salida, etc.” (Barriga, 2005).

MIPS es una arquitectura de procesadores que implementa la filosofía RISC (Reduced Instruction Set Computer), por lo que busca ser simple y regular, y en consecuencia, fácil de aprender y entender (Castillo Catalán & Claver Iborra, 2001). Por esta razón, sus instrucciones son de únicamente 32 bits e implementa 4 principios clave: lo simple favorece la regularidad, hacer que el caso común sea rápido, pequeño es rápido, y un buen diseño demanda buenos compromisos (Money Harris & Harris, 2007).

Esta arquitectura implementa el uso de registros y de la memoria para almacenar valores durante la ejecución de sus programas. Cuenta con 32 registros de propósito general, numerados desde el 0 al 31 (Ortega, 2002). Dado que esta cantidad es reducida, se permite la transferencia de datos entre la memoria y los registros, a través de la escritura y lectura de valores. Instrucciones como “load word” (lw) y “store word” (sw) facilitan la interacción de la memoria con los registros. “La memoria puede considerarse como un enorme arreglo unidimensional y las direcciones (empezando en cero) son simplemente los índices a este arreglo” (Ortega, 2002).

## 2.2. INSTRUCCIONES MIPS

En la búsqueda de ser una arquitectura sencilla, MIPS implementa únicamente tres tipos de instrucciones: “Las instrucciones de tipo R operan en tres registros. Las instrucciones de tipo I operan en dos registros y un registro inmediato de 16 bits. Las instrucciones de tipo J (jump/salto) operan en un registro inmediato de 26 bits” (Money Harris & Harris, 2007). Cada instrucción se codifica en lenguaje máquina de modo diferente, pero siguiendo una estructura similar para cada tipo específico. MIPS posee instrucciones de operaciones aritméticas como “add” o “addi”; operaciones lógicas como “and” o “xor”; condicionales como “beq” o “bne”; etc.

## 2.3. INSTRUCCIONES DE DIRECCIONAMIENTO

Una de las principales diferencias entre los computadores y las calculadoras simples, es la capacidad de tomar decisiones. Según parámetros determinados, los computadores pueden decidir y ejecutar diferentes instrucciones.

En lenguajes de alto nivel, las instrucciones de direccionamiento de flujo se representan mediante cláusulas condicionales como “if” y “else” (Ortega, 2002). En MIPS, existen dos tipos de instrucciones condicionales de tipo I: “beq” (Branch if equal) y “bne” (Branch if not equal). Estas instrucciones comparan los valores de los registros y, dependiendo si se cumple la condición, redirigen la ejecución a una instrucción específica; de lo contrario, la ejecución sigue de manera secuencial (Money Harris & Harris, 2007).

Otro modo de direccionamiento es utilizar instrucciones de tipo J. Estas incluyen “j”, que salta a una instrucción etiquetada; “jr”, que dirige el flujo hacia la instrucción cuya dirección está almacenada en un registro particular; y “jal”, que no solo salta hacia una instrucción etiquetada, sino que también guarda la dirección de la siguiente instrucción previa al salto en el registro \$ra. Esto último resulta útil para el uso de subrutinas.

## 2.4. RUTINA Y SUBROUTINA

Entendemos por rutina (o subrutina) a la implementación a bajo nivel de lo que en alto nivel se conoce como procedimientos o funciones. La diferencia entre un procedimiento y una función radica en que las funciones retornan valores (Villena, Asenjo, & Corbera, 2015).

En MIPS, podemos hacer uso de las subrutinas al etiquetarlas y direccionar el flujo de ejecución hacia ellas. Una subrutina puede trabajar con valores de entrada (\$a0-\$a3) y de salida (\$v0-\$v1), similar a las funciones. O bien, solo realizar procedimientos.

## 2.5. LLAMADAS DE SISTEMA O SYSCALL

MIPS implementa un pequeño set de servicios de instrucciones para el sistema operativo (syscall).

Estos servicios permiten introducir datos y visualizar resultados, de forma cómoda, en los programas que se desarrollan. Para pedir un servicio, el programa debe cargar el número identificador del tipo de llamada al sistema en el registro \$v0 y el/los argumento/s en los registros \$a0-\$a3 (o f12 para valores en coma flotante). Las llamadas al sistema que retornan valores ponen sus resultados en el registro \$v0 (\$f0 en coma flotante).

A continuación, se presenta una tabla con los datos necesarios para ejecutar algunas llamadas de sistema:

Service	Code in Sv0	Arguments	Result
print integer	1	\$a0 = integer to print	
print float	2	\$f12 = float to print	
print double	3	\$f12 = double to print	
print string	4	\$a0 = address of null-terminated string to print	
read integer	5		\$v0 contains integer read
read float	6		\$f0 contains float read
read double	7		\$f0 contains double read
read string	8	\$a0 = address of input buffer \$a1 = maximum number of characters to read	<i>See note below table</i>
sbrk (allocate heap memory)	9	\$a0 = number of bytes to allocate	\$v0 contains address of allocated memory
exit (terminate execution)	10		
print character	11	\$a0 = character to print	<i>See note below table</i>
read character	12		\$v0 contains character read

Imagen 1: Sección de tabla de instrucciones syscall (MARS, s.f.).

En la siguiente imagen se observa un ejemplo del uso de syscall para imprimir en la consola el valor almacenado en el registro \$s0:

```
li $v0, 1
add $a0, $zero, $s0
syscall
```

Imagen 2: Ejemplo de syscall para imprimir valor (Fuente propia).

## 2.6. MANEJO DE STACK O PILA

Se denomina stack o pila de programa a la zona de memoria organizada de forma LIFO (Last In, First Out), que el programa utiliza principalmente para el almacenamiento temporal de datos (Villena, Asenjo, & Corbera, 2015). Las operaciones que se realizan incluyen apilar (push) y desapilar (pop) elementos.

Se define como:

estructura de datos caracterizada por que el último dato que se almacena es el primero que se obtiene después. Para gestionar la pila se necesita un puntero a la última posición ocupada de la misma, con el fin de conocer dónde se tiene que dejar el siguiente dato a almacenar, o para saber dónde están situados los últimos datos almacenados en ella. Para evitar problemas, el puntero de pila siempre debe estar apuntando a una palabra de memoria. (Castillo Catalán & Claver Iborra, 2001)

MIPS posee un registro específico para almacenar el puntero hacia el tope de la pila, llamado “stack pointer” (\$sp). Para trabajar correctamente el uso del stack, es necesario actualizar el puntero antes de apilar elementos. Esto implica, utilizar la instrucción “addi \$sp, \$sp, -X”, donde X es la cantidad de elementos a apilar multiplicada por 4. Luego mediante instrucciones “sw”, se escriben los valores de los registros en el stack y se realizan los procedimientos del programa. Al finalizar el uso, se deben desapilar los elementos mediante la instrucción “lw” para guardarlos en los registros en los que estaban originalmente. Se actualiza el valor de \$sp, de modo “addi \$sp, \$sp, X”, y se concluye el uso de la pila.

La siguiente imagen ejemplifica un uso de stack para apilar valores:

El siguiente fragmento de código muestra cómo se puede realizar el apilado de los registros \$t0 y \$t1 en la pila:

```
.text
main:  li $t0,10
       li $t1, 13    #inicializar reg. t0,$t1
       addi $sp, $sp, -4 #actualizar el sp
       sw  $t0, 0($sp) #apilar t0
       addi $sp, $sp, -4 #actualizar el sp
       sw  $t1, 0($sp) #apilar t1
```

Imagen 3: Ejemplo gestión de pila (Castillo Catalán & Claver Iborra, 2001).

## 2.7. SERIE DE TAYLOR

Expresión matemática que logra representar una función mediante una sumatoria infinita de términos de las derivadas de la función determinada. A continuación se observa su expresión formal, donde  $f^{(n)}(x)$  es la  $n$ -ésima derivada de la función  $f(x)$  en torno al punto  $c$ , y  $n!$  es el factorial de  $n$  (Larson & Edwards, 2016).

### TEOREMA 9.23 Convergencia de la serie de Taylor

Si  $\lim_{n \rightarrow \infty} R_n = 0$  para todo  $x$  en el intervalo  $I$ , entonces la serie de Taylor de  $f$  converge y es igual a  $f(x)$ ,

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(c)}{n!} (x - c)^n.$$

Imagen 4: Convergencia de la serie de Taylor (Larson & Edwards, 2016).

Adicionalmente, ciertas funciones pueden expresarse de un modo específico obtenido según la convergencia de la serie de Taylor. A continuación, se indican algunas de estas funciones.

Función	Intervalo de convergencia
$\frac{1}{x} = 1 - (x - 1) + (x - 1)^2 - (x - 1)^3 + (x - 1)^4 - \dots + (-1)^n (x - 1)^n + \dots$	$0 < x < 2$
$\frac{1}{1+x} = 1 - x + x^2 - x^3 + x^4 - x^5 + \dots + (-1)^n x^n + \dots$	$-1 < x < 1$
$\ln x = (x - 1) - \frac{(x - 1)^2}{2} + \frac{(x - 1)^3}{3} - \frac{(x - 1)^4}{4} + \dots + \frac{(-1)^{n-1} (x - 1)^n}{n} + \dots$	$0 < x \leq 2$
$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \dots + \frac{x^n}{n!} + \dots$	$-\infty < x < \infty$
$\text{sen } x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \dots + \frac{(-1)^n x^{2n+1}}{(2n+1)!} + \dots$	$-\infty < x < \infty$
$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots + \frac{(-1)^n x^{2n}}{(2n)!} + \dots$	$-\infty < x < \infty$

Imagen 5: Series de potencias para funciones elementales (Larson & Edwards, 2016).



### 3. DESARROLLO Y RESULTADOS

Como requisito previo para desarrollar el presente laboratorio, fue necesario el uso de ciertos códigos creados en la experiencia anterior. Se utilizó la función multiplicación, factorial y división, donde en esta última se realizó una pequeña modificación al código. Anteriormente, la función sólo permitía obtener el cociente de números positivos y mediante un algoritmo externo a la subrutina, se mostraba en pantalla el resultado con el signo correspondiente. Sin embargo, fue indispensable agregar dentro de la propia función un algoritmo que permita obtener la división considerando el signo dentro del mismo retorno. Adicionalmente, se creó la función elevado, la que obtiene el resultado de un número natural elevado a otro número natural.

Como resultado se obtiene un único código que retorna las aproximaciones de 3 funciones según un parámetro de entrada como un número natural.

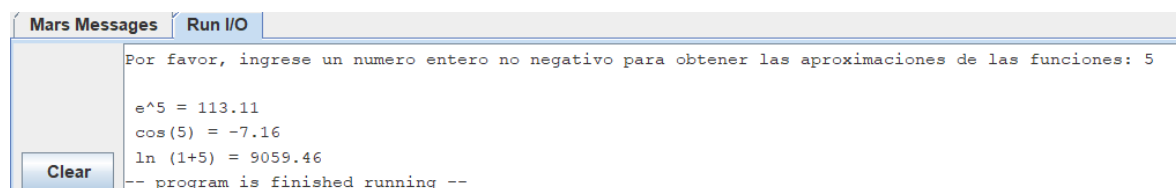


Imagen 6: Resultados del programa en MIPS (Fuente propia).

Se destaca que las 3 funciones tienen un orden distinto de la sumatoria para calcularlas. En particular, la función exponencial y logaritmo natural son de orden 7, mientras que la función coseno se implementó de orden 4. Al aumentar el orden de la sumatoria se produce una mayor precisión para obtener la aproximación de las funciones. Eventualmente la función coseno al utilizar solo 4 iteraciones, es muy poco precisa al compararla con una función de orden 10 o superior.

#### 3.1. FUNCIÓN EXPONENCIAL

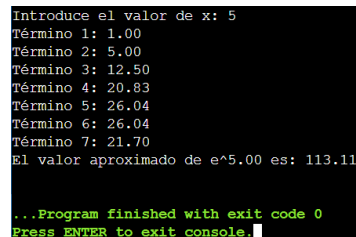
El programa utilizado para la función exponencial consiste en calcular cada término de la siguiente sumatoria:  $\sum_{n=0}^{\infty} \frac{x^n}{n!}$ , hasta llegar a 7 términos calculados (orden 7). Donde en cada iteración se acumule a una variable resultado cada uno de los términos.

Considerando que MIPS posee instrucciones de máximo 3 operandos, para realizar el cálculo de cada término fueron necesarias las siguientes operaciones: elevar el valor de entrada  $x$  a la  $n$ ; obtener el factorial de  $n$ ; dividir el valor de  $x^n$  en  $n!$ . En este punto, ya se tienen el valor de la división, pero en registros separados, los que serán acumulados en variables distintas para la parte entera y otra para la parte decimal.

El mayor desafío del programa fue la existencia de las divisiones, debido a que se debió considerar la operación de números racionales donde la parte decimal y la entera se encuentran en registros distintos. Para trabajar esto, se realizó el cálculo para la parte entera y la decimal separadamente. Donde al finalizar el bucle principal, ambos valores deben sumarse considerando el siguiente algoritmo: primero se deben obtener los primeros 2 valores de la parte decimal, esto será considerado como la parte decimal real de la expresión; luego a la parte decimal anterior obtenida de las iteraciones, se debe dividir por 100 para obtener la división entera (no considera la parte decimal); finalmente, para obtener la parte entera real de la expresión, se debe sumar la división entera con la parte entera obtenida de las iteraciones. La palabra “real” no se refiere al conjunto de los números reales, sino a que el valor es el verdadero resultado de la división. De este modo, la función retorna en \$v0 la parte entera y en \$v1 la parte decimal.

Como resultado, el programa opera correctamente considerando el orden 7, sin embargo, si se requiere obtener la aproximación de la función utilizando más términos, el programa funciona pero demora mucho.

A continuación se presentan los resultados un programa en el lenguaje C al aproximar  $e^5$ . Notamos que al comparar este resultado con el obtenido del código MIPS (ver Imagen 6), ambos programas obtienen el mismo valor.



```

Introduce el valor de x: 5
Término 1: 1.00
Término 2: 5.00
Término 3: 12.50
Término 4: 20.83
Término 5: 26.04
Término 6: 26.04
Término 7: 21.70
El valor aproximado de e^5.00 es: 113.11
...Program finished with exit code 0
Press ENTER to exit console

```

Imagen 7: Resultados de programa en C para aproximar  $e^x$  (Fuente propia).

## 3.2. FUNCIÓN COSENO

Esta función fue desarrollada de un modo similar a la anterior, se debe calcular cada término según la sumatoria:  $\sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n}}{2n!}$ , donde en cada iteración, el valor calculado se debe ir acumulando en dos variables de resultado, una para la parte entera y otra para la decimal, hasta terminar el bucle.

El problema existente en la implementación del código es que no se obtienen resultados al intentar realizar más de 4 iteraciones, debido a que el proceso de cálculo demora mucho. Por lo que el programa sólo considera 4 términos para realizar la aproximación.

Esta función es la que más operaciones requirió para calcular cada término, las que mencionan a continuación: primero se debe elevar el  $-1$  a  $n$ ; posteriormente multiplicar  $2n$  para calcular  $x^{2n}$ ; luego obtener  $(-1)^n \cdot x^{2n}$ ; calcular  $2n!$ ; para finalmente determinar el valor  $((-1)^n \cdot x^{2n})/2n!$ . Al igual

que la función anterior, consideran el mismo trabajo de los valores para la parte entera y la decimal, además de considerar el signo del resultado.

A continuación, los resultados de un código en lenguaje C para aproximar  $\cos(5)$ . Notamos el mismo valor que el obtenido del programa en MIPS (ver Imagen 6).

```
Introduce el valor de x (en radianes): 5
Término 1: 1.00
Término 2: -12.50
Término 3: 26.04
Término 4: -21.70
El valor aproximado de cos(6.00) es: -7.16

...Program finished with exit code 0
Press ENTER to exit console.
```

Imagen 8: Resultados de programa en C para aproximar  $\cos(5)$  (Fuente propia).

### 3.3. FUNCIÓN LOGARITMO NATURAL

Similarmente, se debe calcular cada término de la sumatoria:  $\sum_{n=1}^{\infty} \frac{(-1)^{n+1} \cdot x^n}{n}$ ; de modo de ir acumularlo el valor en dos variables, una para la parte entera y otra para la parte decimal del resultado final, y así hasta llegar a las 7 iteraciones. Una vez terminado el bucle, mediante un algoritmo se analiza el signo del valor para retornar la parte entera en  $\$v0$ , y la parte decimal en  $\$v1$ .

Notamos que esta sumatoria, a diferencia de las anteriores, comienza en 1, esto es debido a que no se podría aproximar la función en torno a 0, ya que en ese punto converge a  $-\infty$ .

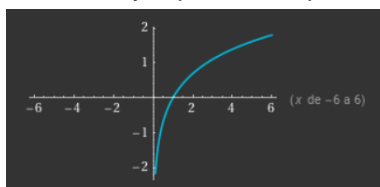


Imagen 9: Gráfica de función  $\ln(x)$  (WolframAlpha.com).

La secuencia de operaciones para calcular cada término es la siguiente: sumar  $n + 1$ ; obtener el valor de  $(-1)^{n+1}$ ; luego  $x^n$  para multiplicar  $(-1)^{n+1} \cdot x^n$ ; y finalmente se calcula  $((-1)^{n+1} \cdot x^n)/n$ . Donde este resultado se acumula en el registro correspondiente a la parte entera o decimal. A continuación, se observa el resultado de un código en C que aproxima  $\ln(5 + 1)$ . Notamos que es el mismo resultado que el obtenido en MIPS (ver Imagen 6).

```
Introduce el valor de x (para calcular ln(1+x)): 5
Término 1: 5.00
Término 2: -12.50
Término 3: 41.67
Término 4: -156.25
Término 5: 625.00
Término 6: -2604.17
Término 7: 11160.71
El valor aproximado de ln(1+5.00) es: 9059.46

...Program finished with exit code 0
Press ENTER to exit console.
```

Imagen 10: Resultados de programa en C para aproximar  $\ln(5 + 1)$  (Fuente propia).

## 4. CONCLUSIONES

En esta tercera experiencia de laboratorio, se alcanzó un relativo éxito para los 3 programas propuestos, dos de ellos consideran la condición de llegar hasta el orden 7 de la sumatoria, pero uno sólo logra aproximar la función hasta el orden 4. Esto no me satisface completamente, ya que a pesar de solventar varios errores de funcionamiento de los códigos de creados en el laboratorio anterior, no pude lograr que la función  $\cos(x)$  que funcionara hasta el orden 7.

Se cumplieron todos los objetivos: se usó el software MARS para escribir, ensamblar y depurar programas MIPS; se logró escribir programas MIPS usando instrucciones aritméticas, de salto y memoria al crear las funciones de aproximación; se comprendió el uso de subrutinas al emplear la ejecución de funciones en MIPS; se realizaron llamadas de sistema mediante syscall para solicitar e imprimir datos en la consola; y finalmente, se implementaron algoritmos para resolver problemas matemáticos.

Sinceramente, de algún modo pensé que me costaría menos realizar esta experiencia de laboratorio, sin embargo, no fue tan fácil pensar los algoritmos. En comparación con los laboratorios anteriores no invertí mucho más tiempo, y creo que esto fue debido a la experiencia que he adquirido. Va creciendo mi conocimiento y manejo del lenguaje; no me considero un experto, pero me alegra poder comprender en un nivel básico el funcionamiento del lenguaje ensamblador.

Es interesante descubrir que la programación en ensamblador considera elementos que no estamos acostumbrados a manejar en los lenguajes de programación populares. Me encanta la idea de que los lenguajes de alto nivel compilan los programas a lenguaje ensamblador y luego los convierten a lenguaje máquina para su ejecución. Me parece increíble que puedan realizar un proceso tan difícil de forma rápida y sin errores.

## REFERENCIAS

- Barriga, J. (2005). *Historia y evolución del computador y microprocesador*. Universidad Peruana de Ciencias Aplicadas - UPC.
- Castillo Catalán, M., & Claver Iborra, J. M. (2001). *Prácticas guiadas para el Ensamblador del MIPS R2000*.
- Larson, R., & Edwards, B. (2016). *Cálculo, Tomo I. Décima edición*. CENGAGE Learning.
- MARS. (s.f.). Document MARS 4.5 Help.
- Money Harris, D., & Harris, S. L. (2007). *Digital Design and Computer Architecture*. Morgan Kaufmann.
- Ortega, J. (Noviembre de 2002). *Manual de Laboratorio para Arquitectura y Organización de Computadoras Usando XSPIM*. Obtenido de <http://hp.fciencias.unam.mx/~jloa/AC/index.html>
- Prieto, A., Lloris, A., & Torres, J. C. (1989). *Introducción a la Informática*. McGraw-Hill.
- Villena, A., Asenjo, R., & Corbera, F. (2015). *Prácticas de ensamblador basadas en raspberry Pi*. Málaga, España: Universidad de Málaga, Departamento de Arquitectura de Computadores.