Diseño e implementación de un emulador de instrucciones MIPS en una arquitectura x86 de 64 bits.

Proyecto 1 de Laboratorio de Estructura de Microprocesadores

Danny Gabriel Mejías Anchía *(2014159999)*Javier Alonso Cordero Quirós*(2014115782)*Keylor Andrés Mena Venegas *(2014108164)*Luis Gerardo Leon Vega *(2014069639)*Luis Orlando Merayo Gatica *(2014049811)*

EL4313 – Lab. Estructura Microprocesadores

1er Semestre 2017

Tecnológico de Costa Rica

*Abstract*—In this paper details the design, development and implementation process of a MIPS architecture (32 bits) emulator that runs in a x86 (64 bits) computer with Ubuntu 16.04 LTS as operative system (OS). Likewise, this paper has all the analysis about the techniques used during the project development process, discussing and contrasting some of these techniques to finish successfully and with an acceptable throughput the emulator.

*Resumen*—En este documento, se detallan aspectos del proceso de diseño, desarrollo e implementación de un emulador de una arquitectura MIPS de 32 bits sobre un computador de arquitectura x86 de 64 bits con sistema operativo Ubuntu 16.04 LTS. Asimismo, se esbozará el análisis de las técnicas empleadas para el proceso de desarrollo del proyecto, discutiendo y discriminando algunas de estas para llegar de forma adecuada a una solución final.

# Introducción

Introduce –en 1 solo párrafo- el objetivo del proyecto, el principal resultado obtenido y los alcances de este documento.

El objetivo de este proyecto es desarrollar un emulador de una arquitectura MIPS capaz de ejecutar instrucciones en este set de instrucciones, ensamblado en un ordenador con una arquitectura x86 de 64 bits con un sistema operativo Ubuntu 16.04. El alcance de este documento se limitará a esbozar y realizar análisis del proceso de diseño, destacar el proceso de desarrollo y analizar los resultados obtenidos en pro del cumplimiento de los requisitos del sistema. Este emulador fue programado en ensamblador x86\_64 para cumplir con los requisitos mínimos del proyecto y, además, es capaz leer el programa mediante un archivo con instrucciones decodificadas en MIPS en formado de texto plano ASCII, satisfaciendo los requisitos del sistema.

# Marco Teórico

## Ambiente computacional x86\_64 y Ubuntu 16.04

El ambiente con arquitectura x86\_64 es una de las más comunes actualmente, ya que predominan en el mercado de los ordenadores personales portátiles y de escritorios, siendo muy utilizados para aplicaciones en el hogar, la oficina, la educación y la empresa.

Este ambiente requiere de un sistema operativo para poder extraer todo su potencial y facilitar su uso al usuario, facilitando comandos amigables y un entorno de interfaz gráfica. Uno de esos sistemas operativos es Ubuntu 16.04, proyecto GNU-Linux soportado la empresa Canonical [1]. Esta empresa se encarga de darle continuación al proyecto sacando versiones de este sistema operativo cada seis meses, sin embargo, solo los años pares y versiones de abril (XX.04) son las que tienen servicio de soporte especial (LTS – Long Term Support). Esto sugiere que el sistema operativo está en constante renovación, lo que permite que se mantenga a la vanguardia de la seguridad de los sistemas e innovación del software para optimizarlo a los sistemas modernos.

Otra de las ventajas de este sistema operativo es que es de código abierto y gratuito, es decir, puede copiarse, modificarse y derivar productos con base a este sistema operativo sin que haya problemas de derechos de autor. Asimismo, sigue la ideología GNU [2].

Por otro lado, algo que hace atractivo a los sistemas operativos Linux como Ubuntu, es su flexibilidad, su estabilidad y su manejo óptimo de recursos. Existen una gran variedad de programas y aplicaciones que corren sobre este sistema operativo y existen grandes comunidades que brindan soporte para el desarrollo y manejo de estos sistemas operativos, destacando los más populares como StackOverFlow y Ubuntu Forums. Eso facilita a los que se inicien en este sistema operativo o que quieran desarrollar para esta plataforma.

Asimismo, por ser Open Source, no requiere licencia para instalación, que a diferencia de sistemas operativos comerciales como Windows, se requiere el pago de una licencia para poder instalarlos y usarlos en equipos de arquitectura x86\_64.

## Ambiente de trabajo

.

* No posee una interfaz gráfica preinstalada.
* Se accede a él mediante consola SSH.
* Tiene un núcleo más ligero que el sistema operativo dirigido a escritorio, debido a que no posee aplicaciones preinstaladas e interfaz gráfica.

Ante esta situación, para preparar el sistema operativo se deben realizar las siguientes tareas:

1. *Instalar un entorno gráfico:*

Para el entorno gráfico, se ha seleccionado MATE [3]. Esta elección fue llevada a cabo debido a que es una de las más ligeras comparado a GNOME, KDE o Unity (default de Ubuntu Desktop). Eso facilitará la transferencia de interfaz gráfica mediante escritorio remoto. Para instalarlo, se emplearon los siguientes comandos:

Instalación de mate

| sudo apt-get update  sudo apt-get install mate-desktop-environment-core |
| --- |

Esta instalación garantiza instalar únicamente el núcleo, dejando por fuera aplicaciones adicionales, como OpenOffice.

1. *Instalar un servicio para escritorio remoto:*

Para poder acceder al servidor de forma gráfica y disfrutar del GUI (Graphical User Interface), es necesario instalar un servidor de compartición de escritorio. Se ha seleccionado el XRDP por su facilidad de instalación e implementación. Además, es posible conectarse usando el cliente de Escritorio Remoto de Windows [4].

Instalación de XRDP

| sudo apt-get update  sudo apt-get upgrade  sudo apt-get install xrdp |
| --- |

1. *Crear un usuario para el Proyecto*

Agregar un usuario

| adduser tec |
| --- |

Después, se seguirán las instrucciones que implica ese comando, como configuración de la contraseña y credenciales.

1. *Instalar Github*

No trae el Github preinstalado, entonces se instala mediante el gestor de paquetes APT:

Instalar Github

| sudo apt-get install git |
| --- |

Luego, se hace el clonado del repositorio:

Clonar repositorio

| cd cd Desktop/ mkdir Github & cd Github  git clone https://github.com/lleon95/proyecto\_1\_LabMicros\_SEM1\_2017\_Grupo3 |
| --- |

1. *Instalar el NASM, el GCC y el SASM:*

Para desarrollar el emulador, es requerido un ensamblador (NASM), un debugger (GCC) y el entorno gráfico de desarrollo (SASM). Se ha elegido usar SASM[[1]](#footnote-1) como entorno IDE para poder usar el debugger de forma más amigable y tener mayor control sobre los “breakpoints”. Asimismo, se puede crear el archivo objeto muy fácilmente:

Instalación NASM y GCC

| sudo apt-get update  sudo apt-get upgrade  sudo apt-get install nasm  sudo apt-get install gcc |
| --- |

Para instalar el SASM

Instalación NASM y GCC

| wget download.opensuse.org/repositories/home:/Dman95/xUbuntu\_14.04/amd64/sasm\_3.8.0\_amd64.deb  sudo dpkg -i sasm\_3.8.0\_amd64.deb |
| --- |

Ya con esto, queda preparado el ambiente de desarrollo. A partir de esto, ya se comenzó a desarrollar la solución.

## Otras consideraciones

.

*1. Nomenclatura y compatibilidad de los registros*

Para los registros, ambas arquitecturas tienen nomenclaturas diferentes para referirse a sus registros. Sin embargo, los registros de arquitecturas inferiores son compatibles con las nuevas arquitecturas, es decir, se pueden usar los registros de 32 bits en una arquitectura de 64 bits [5].

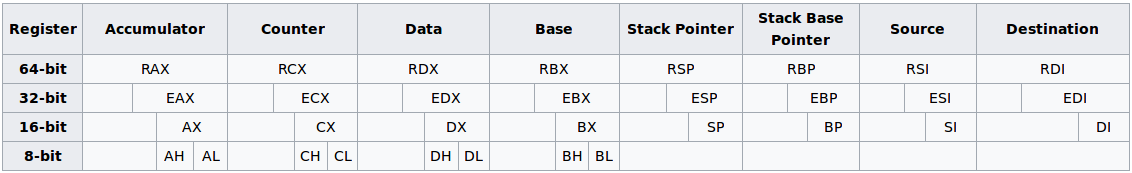


Figura 1 – Registros en las diversas arquitecturas x86 [6]

Como es posible observar mediante la figura 1, cuando se invoca un registro EAX en una arquitectura de 64 bits, el contenido se cargará en la parte baja del RAX (registro en 64 bits). Por otro lado, los registros de nombre “r8” - “r15” en formato de 64 bits, su parte baja tendrá como nomenclatura “r8d” - “r15d”.

*2. Sintaxis de las instrucciones*

Las instrucciones en el set x86 por lo general tienen dos operandos. El primer operando es, adicionalmente, el registro o campo de memoria destino del resultado de la operación. El segundo operando, por su parte, solo cumple un rol de operando, siendo posible usar registro, memoria o un valor inmediato [7]. Asimismo, este puede variar de acuerdo con la sintaxis empleada, siendo la sintaxis Intel la preferida para el desarrollo de este proyecto. Por ejemplo, para la instrucción:

| mov rax, rbx; mov dest, src |
| --- |

La instrucción dicta que el contenido del registro rbx se copie a rax.

# Solución Implementada

## Descripción de la solución

Para el proceso de desarrollo de la solución, se ha decidido usar un modelo comportamental de una arquitectura MIPS de 32 bits. Este modelo involucra el siguiente diagrama de bloques:

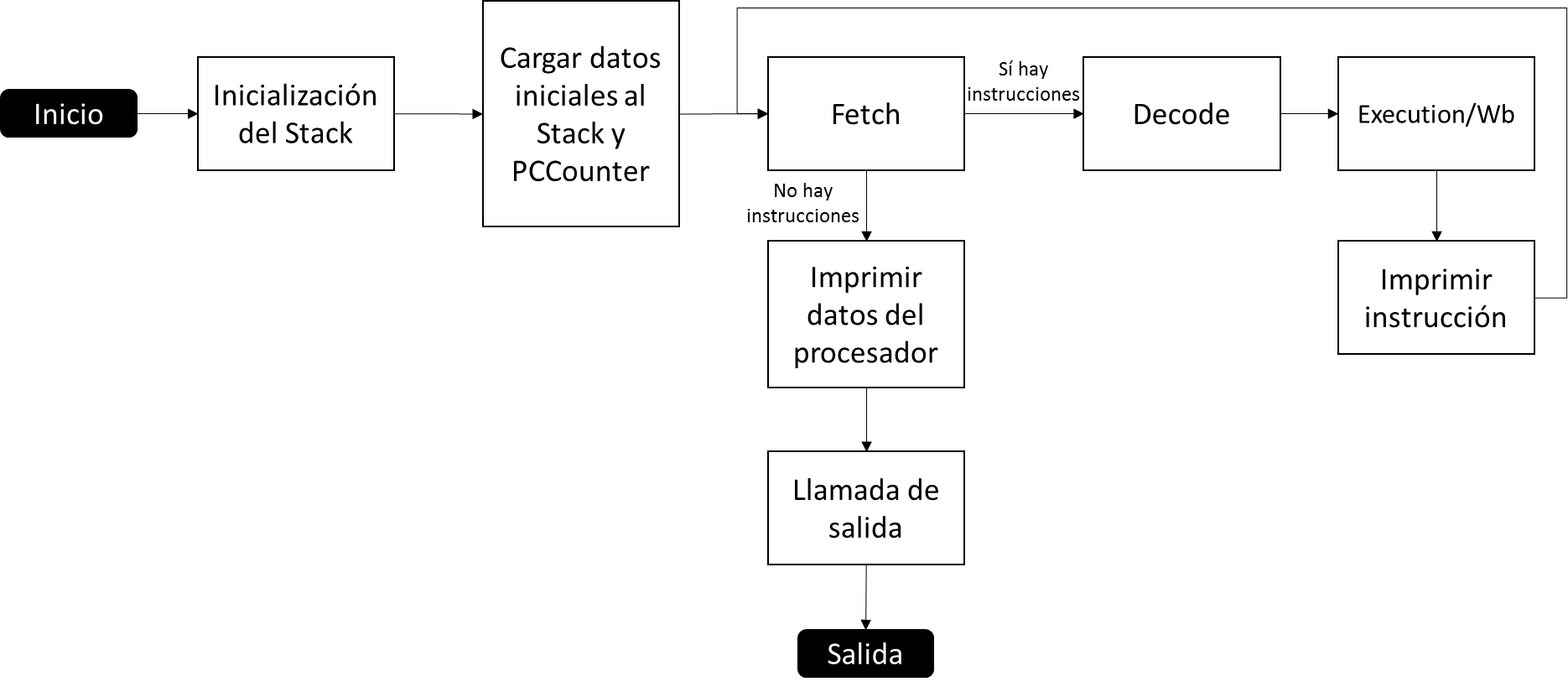


Figura 2 – Diagrama de bloques descriptivo de la solución

En el inicio de la solución, se inicializa el stack y el banco de registros, para poder cargar los argumentos del programa. Asimismo, se procede a cargar el archivo ROM.txt, que contiene toda la información de las instrucciones que deben ser ejecutadas y los datos que se almacenan en la memoria.

Realizado el proceso de preparación, se procede a inicializar el PC Counter en 00400000H para apuntar a la primera instrucción y se inicia el proceso de la arquitectura brincando al Fetch (proceso de adquisición de instrucción próxima). Posteriormente, al adquirir una instrucción, se inicia la decodificación de la misma, donde se determina el tipo de instrucción (sean R, J e I posibles) y que la instrucción sea válida. Luego se determina la localización de los registros que se requieren para ejecutar dicha instrucción y se llama a una subrutina que ejecuta la operación correspondiente al OpCode (Operation Code) obtenido.

La ejecución de dicha operación se realiza a cabo en el proceso de Execution/Wb, que se encarga de realizar la operación de forma comportamental en código x86\_64 y carga los datos en un stack de registros ya preconfigurado en el programa. Cuando se ejecuta alguna de las subrutinas que pertenecen a este proceso, realizan el “Write Back” al final de cada subrutina, siendo posible escribir en el registro contador de instrucciones (PC Counter), operaciones sobre memoria o, bien, escribir valores en los registros del procesador MIPS.

Asimismo, durante la ejecución, se llama al procedimiento para imprimir la instrucción que se está ejecutando en ese momento, los registros involucrados y la impresión de todos los registros empleables en la arquitectura emulada.

Al terminar la ejecución de la operación detectada, se procede a volver a la subrutina del “Fetch” para adquirir la próxima instrucción que se ejecutará. En caso de que haya instrucciones, el proceso de decodificación y ejecución se repiten de acuerdo con lo mencionado con anterioridad.

En caso contrario, se finalizan las estructuras del emulador de arquitectura y se procede a imprimir los datos de la arquitectura servidor (Host) y se realiza un llamado al sistema para finalizar la ejecución. Esto se resume en lo siguiente:

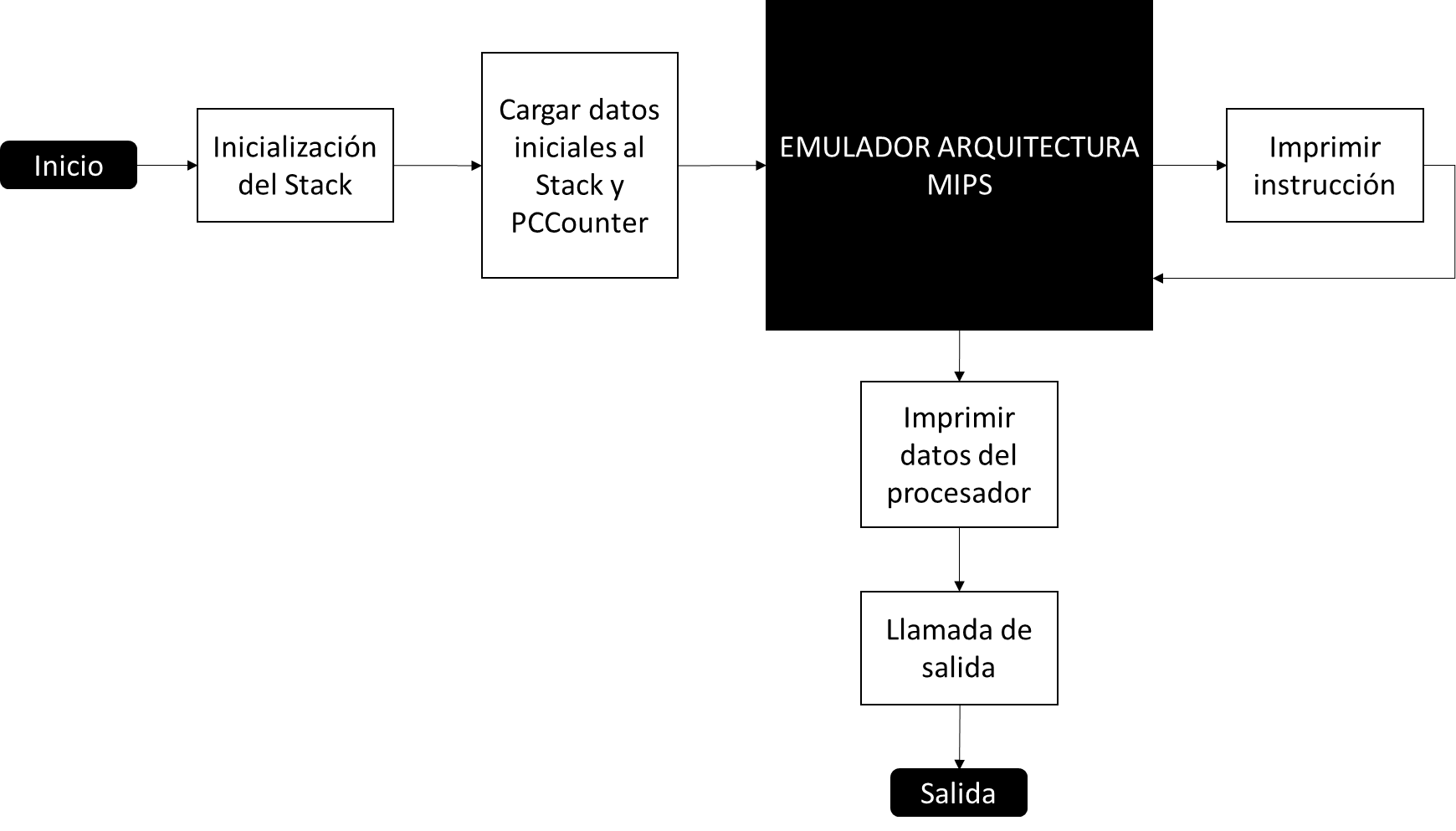


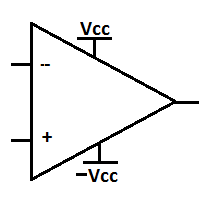
Figura 3 – Diagrama de bloques general de la solución

FALTA DESCOMPONER MÁS

Describir de forma general la solución propuesta. Es decir, la estructura del programa que se escribió en lenguaje ensamblador.

Se recomienda usar diagramas de flujo y/o pseudocódigo para dar más claridad.

Utilice el siguiente formato para insertar figuras:



1. Amplificador operacional

Si necesita describir un proceso, se recomienda enumerarlo con viñetas para que se vea ordenado. Por ejemplo:

1. Paso1
2. Paso2
3. Paso3
4. Paso “n”

El pseudocódigo se recomienda insertarlo como parte de una tabla (de una sola celda) y usar un tipo de letra distinto para resaltarlo. Por ejemplo:

Pseudocódigo

| Cargar 0xFF en registro RAX  Cargar 0xFF en registro RBX  Limpiar registro banderas  Llamar sistema |
| --- |

## Limitaciones y recomendaciones

.

# Resultados

Se muestran resultados de uso del programa. Se recomienda usar imágenes que muestren la interacción con el programa. Recuerde mantener el formato adecuado para las imágenes.

##### Conclusiones

Describa, en enunciados cortos y concisos: ¿Cuáles son las principales conclusiones, aprendizajes y recomendaciones luego de completar el proyecto?

##### Referencias

Las referencias bibliográficas utilizan el siguiente formato:

1. B. Sánchez. “Cuaderno Práctico de Linux. Sistemas Operativos Monopuesto”, Segunda Edición, pp. 21, Septiembre 2015.
2. M. Overley. “The Open Source Handbook”, Pike & Fisher, USA, pp. 1-11, 2003.
3. "Install [wiki.mate-desktop.org]", Wiki.mate-desktop.org, 2017. [Online]. Available: http://wiki.mate-desktop.org/download. [Accessed: 13- Mar- 2017].
4. "xrdp", *Xrdp.org*, 2017. [Online]. Available: http://www.xrdp.org/. [Accessed: 13- Mar- 2017].
5. Lawlor, "x86\_64 NASM Assembly Quick Reference ("Cheat Sheet")", UAF Computer Science Department, 2007. [Online]. Available: https://www.cs.uaf.edu/2007/fall/cs301/support/x86\_64/index.html. [Accessed: 19- Feb- 2017].
6. "X86 Assembly/X86 Architecture - Wikibooks, open books for an open world", En.wikibooks.org, 2017. [Online]. Available: https://en.wikibooks.org/wiki/X86\_Assembly/X86\_Architecture. [Accessed: 19- Feb- 2017].
7. "X86-64 Architecture Guide", Cons.mit.edu. [Online]. Available: http://cons.mit.edu/sp14/x86-64-architecture-guide.html. [Accessed: 19- Feb- 2017].
8. N. Apellido1, N. Apellido2, and N. Apellido3, “Título” Casa Editora. Ciudad, vol. X, pp. 529–551, Mes Año.

1. Se deben instalar el NASM y GCC de primero antes del SASM, debido a que esto es únicamente un IDE. [↑](#footnote-ref-1)