Herramienta de simulación para dar soporte a la enseñanza de arquitectura de computadoras

Maestría en Sistemas de Información

Ruiz Jose Maria

Director: Colombani Marcelo Alberto

2024

Índice

Resumen 1

Agradecimientos 3

1 Introducción 5
- 1.1 Justificación . . . 7
- 1.2 Objetivos . . . 8
- 1.3 Metodología de Desarrollo . . . 8
- 1.4 Organización del Documento . . . 9

2 Estado del arte 11
- 2.1 Arquitectura de computadoras . . . 11
- 2.2 Lenguaje ensamblador . . . 16
- 2.3 Simulación . . . 17

3 Comparativa de simuladores 21
- 3.1 Estudios similares . . . 21
- 3.2 Simuladores bajo análisis . . . 22
- 3.3 Criterios de evaluación . . . 22
- 3.4 Selección de simuladores . . . 24
- 3.5 Participantes en la evaluación . . . 24
- 3.6 Análisis comparativo . . . 25
- 3.7 Resultados . . . 26

4 Diseño y construcción del simulador 29
- 4.1 Requisitos de la herramienta . . . 30
- 4.2 Desarrollo y Pruebas . . . 32
- 4.3 Portabilidad . . . 32
- 4.4 Simplicidad . . . 32
- 4.5 Mantenibilidad . . . 32
- 4.6 Escalabilidad . . . 32
- 4.7 Memoria . . . 33
- 4.8 Repertorio de instrucciones propuesto . . . 33
- 4.9 Repertorio de instrucciones . . . 33
- 4.10 Procesador `VonSim8` . . . 44

4.11 Instrucciones 44
4.12 Assembler 45
4.13 Registros 46
4.14 Componentes 46
4.15 Ejecución de Instrucciones 47

Bibliografía **49**

Índice de cuadros

2.1 Hitos procesadores x86 13

2.2 Línea de Tiempo de la Evolución de la Arquitectura x86 14

2.3 Comparación de Arquitecturas 14

3.1 Comparativa según criterios de evaluación preestablecidos 26

Índice de figuras

Resumen

Existe un consenso creciente en el uso de herramientas de simulación en la enseñanza para procesos dinámicos complejos, como las operaciones intrínsecas de la computadora, que permiten representar de forma visual e interactiva la organización y arquitectura interna de la computadora, facilitando así la comprensión de su funcionamiento por parte de los alumnos y el desarrollo de los temas por parte del docente. En este contexto, los simuladores juegan una pieza clave en el campo de la Arquitectura de Computadoras, permitiendo conectar fundamentos teóricos con la experiencia práctica, simplificando abstracciones y haciendo más rica la labor docente. La arquitectura x86 es ampliamente utilizada en computadoras de escritorio y servidores. Este documento pretende realizar una comparación de los simuladores x86 que más se adecuan en el dictado de la asignatura Arquitectura de Computadoras de la carrera Licenciatura en Sistemas, establecer los criterios de evaluación y evaluar los simuladores seleccionados de acuerdo con estos criterios.

La presente investigación surge en el marco del proyecto de investigación I/D novel PID-UNER 7065: “Enseñanza/aprendizaje de asignatura Arquitectura de Computadoras con herramientas de simulación de sistemas de cómputos”. El Proyecto es llevado a cabo en la Facultad de Ciencias de la Administración de la Universidad Nacional de Entre Ríos, se vincula directamente con la asignatura Arquitectura en Computadoras que se dicta en segundo año de la carrera Licenciatura en Sistemas perteneciente a la Facultad de Ciencias de la Administración de la Universidad Nacional de Entre Ríos.

Palabras clave: x86, simulador, aprendizaje, enseñanza, arquitectura de computadoras.

Agradecimientos

Agradecimientos aquí.

Capítulo 1

Introducción

En nuestra vida cotidiana, utilizamos dispositivos como computadoras de escritorio, teléfonos y relojes inteligentes, todos ellos basados en arquitecturas específicas. Comprender su funcionamiento e interacciones nos permite diseñar y desarrollar aplicaciones más eficientes.

Es esencial que los estudiantes de Arquitectura de Computadoras comprendan tanto la estructura como el funcionamiento interno de una computadora, y adquieran experiencia práctica con ellas. Para lograrlo, es fundamental disponer de un laboratorio bien equipado con el hardware adecuado y suficiente tiempo para que los estudiantes se familiaricen con las herramientas prácticas. En este contexto, se han desarrollado numerosos simuladores que facilitan la comprensión de la estructura y el funcionamiento de las computadoras, proporcionando valiosas experiencias de aprendizaje.

Esta tesis, inscrita en la Maestría en Sistemas de Información de la Facultad de Ciencias de la Administración, está directamente vinculada con el proyecto de investigación I/D novel PID-UNER 7065, titulado "Enseñanza/aprendizaje de Arquitectura de Computadoras con herramientas de simulación de sistemas de cómputo", desarrollado en la Facultad de Ciencias de la Administración de la Universidad Nacional de Entre Ríos [1].

La asignatura Arquitectura de Computadoras forma parte del plan de estudios de la carrera de Licenciatura en Sistemas, Universidad Nacional de Entre Ríos. Su objetivo es que los estudiantes comprendan la estructura y funcionamiento de las computadoras, y la ejecución lógica de un programa a nivel de instrucciones de máquina.

El primer paso para comprender las computadoras es entender que son máquinas que toman datos del exterior, los procesan y almacenan los resultados en la memoria o los envían a dispositivos de entrada y salida.

El procesamiento se realiza a través del procesador o CPU, y es en este componente donde los estudiantes encuentran mayor complejidad y dificultades para comprender su funcionamiento.

A pesar de que es posible explicar las partes del procesador, su funcionamiento, la interacción de sus componentes y enseñar lenguaje ensamblador mediante prácticas, los estudiantes suelen tener dificultades para lograr una comprensión completa del funcionamiento.

Sin embargo, la utilización de simuladores permite afianzar los conocimientos de los temas vistos en las clases teóricas, a fin de evitar que los estudiantes desvíen su atención hacia el aprendizaje del simulador propiamente dicho, es necesario que estos sean de manejo simple, intuitivo y visualmente atractivo, simplificando su aprendizaje de su uso.

La simulación es un término de uso diario en muchos contextos: medicina, militar, entretenimiento, educación, etc., debido a que permite ayudar a comprender cómo funciona un sistema, responder preguntas como "qué pasaría si", con el fin de brindar hipótesis sobre cómo o por qué ocurren ciertos fenómenos.

Para continuar, se define simulación como el proceso de imitar el funcionamiento de un sistema a medida que avanza en el tiempo. Entonces para llevar a cabo una simulación, es necesario desarrollar previamente un modelo conceptual que representa las características o comportamientos del sistema, mientras que la simulación representa la evolución del modelo a medida que avanza en el tiempo [2][3][4].

Con los avances en el mundo digital, la simulación se ha convertido en una metodología de solución de problemas indispensable para ingenieros, docentes, diseñadores y gerentes. La complejidad intrínseca de los sistemas informáticos los hace difícil comprender y costosos de desarrollar sin utilizar simulación [3].

Muchas veces en el ámbito educativo, resulta difícil transmitir fundamentos teóricos de la organización y arquitectura interna de las computadoras debido a la complejidad de los procesos involucrados. Si sólo incorporamos los medios de enseñanza tradicionales, como ser una pizarra, un libro de texto o diapositivas, los mismos tienen una capacidad limitada para representar estos fundamentos. En consecuencia, es imprescindible un alto nivel de abstracción por parte del estudiante para desarrollar un modelo mental adecuado para capturar la organización y arquitectura interna de las computadoras [5][6][7].

Es evidente la necesidad de utilizar nuevas tecnologías como recursos didácticos y medios de transferencia de conocimiento, ya que ayudan a los estudiantes a relacionar conceptos abstractos con realidades concretas. Estas tecnologías permiten situar al estudiante en un contexto que imita aspectos de la realidad, facilitando la detección de problemáticas similares a las que podrían ocurrir en situaciones reales. Este enfoque promueve un mejor entendimiento a través del trabajo exploratorio, la inferencia, el aprendizaje por descubrimiento y el desarrollo de habilidades [8][9].

Un simulador de arquitectura es una herramienta que imita el hardware de un sistema, representando sus aspectos arquitectónicos y funciones. Permiten realizar cambios,

pruebas y ejecutar programas sin riesgo de dañar componentes ni depender de equipos físicos disponibles [10].

Algunas herramientas ofrecen una representación en forma visual e interactiva de la organización y arquitectura interna de la computadora, facilitando así la comprensión de su funcionamiento. En este sentido, los simuladores juegan una pieza clave en el campo de la Arquitectura de Computadores, permitiendo conectar fundamentos teóricos con la experiencia práctica, simplificando abstracciones y facilitando la labor docente [11][12][13][14].

El repertorio de instrucciones de la arquitectura x86, ampliamente utilizado en computadoras de escritorio y servidores, comenzó con el procesador Intel 8086 en 1978 como una arquitectura de 16 bits. Evolucionó a una arquitectura de 32 bits con el procesador Intel 80386 en 1985 (i386 o x86-32) y posteriormente a 64 bits con las extensiones de AMD (AMD64) y su adopción por Intel (Intel 64) [15][16].

Un procesador x86-64 mantiene la compatibilidad con los modos x86 existentes de 16 y 32 bits, y permite ejecutar aplicaciones de 16 y 32 bits, como así también de 64 bits. Esta compatibilidad hacia atrás protege las principales inversiones en aplicaciones y sistemas operativos desarrollados para la arquitectura x86 [15][16][17].

Por ello, la enseñanza de la arquitectura x86 es de gran relevancia en la asignatura Arquitecturas de Computadoras debido a los diferentes temas que aborda.

Para brindar esta experiencia, es necesario un laboratorio equipado con el hardware adecuado y tiempo suficiente para que los estudiantes se familiaricen con las herramientas. Por este motivo, se han desarrollado muchos simuladores que facilitan la comprensión del funcionamiento y la estructura del computador, ofreciendo valiosas experiencias de aprendizaje [18].

1.1 Justificación

Aunque ya existen simuladores de la arquitectura x86 que apoyan la enseñanza en los cursos de Arquitectura de Computadoras [10][11], estos suelen presentar una gran cantidad de contenidos preestablecidos. Si bien estos contenidos son relevantes, ofrecer toda la especificación de la arquitectura x86 desde el principio puede ser abrumador para los estudiantes y dificultar su comprensión. Sin embargo, desde esta tesis se propone un enfoque diferente: desarrollar una herramienta de simulación de la arquitectura x86 para apoyar la enseñanza de los contenidos específicos de la currícula de Arquitectura de Computadoras. Partiendo de una visión global de la estructura y funcionamiento de la computadora (CPU, memoria, módulo de E/S y buses), mostrando los micropasos necesarios para la realización del ciclo básico de una instrucción, ofreciendo un repertorio reducido de instrucciones que se habiliten las instrucciones a medida que se dictan en la asignatura, permitiendo la generación y ejecución de

programas escritos en lenguaje ensamblador, ya sea paso a paso por instrucción o completa, gestión básica de interrupciones permitiendo la interacción con el teclado y la pantalla, comunicación con los módulos de entrada y salida e interacciones con los periféricos, y por último, medidas de rendimiento sobre la ejecución de un programa.

Con el objeto de ofrecer al estudiante un simulador bien diseñado, robusto, modular y por tanto flexible y sencillo de modificar o ampliar, se explorará la utilización de técnicas formales de modelización y simulación como las redes de Petri o DEVS (Discrete Event System Specification). Estas técnicas permiten separar conceptualmente las capas de modelización y simulación y ofrecen por ello una separación ortogonal de ambas, facilitando la comprensión y modificación del software. Además, permiten el escalado transparente de las simulaciones, pudiéndose ejecutar en entornos de cómputo paralelo o distribuido sin necesidad de modificar el modelo en sí, lo que aporta grandes ventajas de escalado [19][20][21].

1.2 Objetivos

El objetivo principal de esta tesis es construir una herramienta de simulación de la arquitectura x86 para apoyar la enseñanza de arquitectura de computadoras, enfocándose específicamente en los contenidos de la currícula de Arquitectura de Computadoras. Para lograr este objetivo, se plantean los siguientes objetivos específicos:

1. Estudiar y evaluar diferentes herramientas actuales de simulación destinadas a dar apoyo a la enseñanza de la arquitectura x86.
2. Construir una herramienta de apoyo para impartir los contenidos de la asignatura Arquitectura de Computadoras, para ello debe cumplir:

    - Una visión global de la estructura y funcionamiento de la computadora.
    - Generación y ejecución de programas en ensamblador.
    - Repertorio de instrucciones x86 reducido y habilitado progresivamente.
    - Simulación visual e interactiva de micropasos de instrucciones.
    - Gestión de interrupciones y comunicación con periféricos.
    - Medidas de rendimiento de ejecución de programas.

1.3 Metodología de Desarrollo

Teniendo en cuenta los objetivos propuestos en la sección anterior, se pretende alcanzar los mismos a través de los pasos que se describen en esta sección.

1.3.1 Etapas de la Investigación

a. Análisis bibliográfico. Se realizó una revisión continua de las publicaciones científicas y tecnológicas, libros e informes técnicos relacionados con el objeto de estudio.
b. Recopilación de simuladores. Se realizó un relevamiento del estado actual y las actualizaciones de los simuladores aplicados a la enseñanza de arquitectura de computadoras.
c. Estudio de los simuladores. En base a la documentación relevada de los simuladores se estudió en profundidad al menos 5 simuladores y se elaboró una comparativa de los simuladores seleccionados en cuanto a los contenidos que se imparten en la asignatura.
d. Construir el simulador. A través de métodos y técnicas de ingeniería de software se construyó un simulador de la arquitectura x86 donde abarque los aspectos más relevantes de la asignatura Arquitectura de Computadoras, permitiendo desarrollar los contenidos en una plataforma unificada, evitando así la pérdida de tiempo y dificultad que supone para el estudiante habituarse a diferentes entornos. Se utilizarán para ello técnicas formales de modelización y simulación, que facilitan un desarrollo modular y enfocan el esfuerzo en la definición del modelo de la arquitectura x86 más que en el protocolo de simulación, permitiendo además el escalado a entornos de ejecución paralelos o distribuidos sin necesidad de modificar el modelo de la arquitectura simulada.

1.4 Organización del Documento

El resto de este documento se organiza de la siguiente manera: el capítulo 2 define formalmente las características y el set de instrucciones de la arquitectura. Luego, el capítulo 3 repasa y motiva el interesante rol que la simulación desde un punto de vista didáctico puede desempeñar para el dictado de la asignatura donde se abordan estos tópicos. El capítulo 4 comparativo de los simuladores estudiados según criterios preestablecidos. Finalmente, el capítulo 5 construcción de un simulador como soporte para el uso de la enseñanza y aprendizaje de arquitectura de computadora.

Capítulo 2

Estado del arte

En este capítulo se examina el estado del arte en la arquitectura de computadoras, enfocándose en la arquitectura x86 y el uso de herramientas de simulación en el ámbito educativo. Este análisis es clave para comprender el contexto y la relevancia de los temas abordados en esta tesis.

A continuación, se abordará el papel fundamental que desempeñan las herramientas de simulación en la enseñanza de la arquitectura de computadoras. Estas herramientas facilitan la comprensión de conceptos complejos y permiten a los estudiantes interactuar con sistemas simulados, proporcionando una experiencia práctica valiosa que de otro modo sería difícil de obtener. Se examinarán las herramientas de simulación más destacadas y su aplicación en entornos educativos, evaluando sus beneficios y limitaciones.

Finalmente, se explorarán las tendencias actuales y futuras en la enseñanza de la arquitectura de computadoras, destacando las innovaciones tecnológicas y metodológicas que están revolucionando las prácticas educativas en este campo. Se analizarán enfoques pedagógicos innovadores y estrategias de enseñanza que aprovechan las herramientas digitales y la simulación para mejorar la comprensión y el rendimiento de los estudiantes.

2.1 Arquitectura de computadoras

La arquitectura de computadoras se ocupa del estudio y diseño de los componentes de hardware y software de una computadora y su interacción. Este campo abarca desde el diseño de circuitos hasta la integración de sistemas completos, siendo crucial para el desarrollo de tecnologías eficientes y avanzadas. Estudiar arquitectura de computadoras permite comprender los sistemas informáticos desde su nivel más básico, ofreciendo una visión holística de su funcionamiento y estructura. Además,

está estrechamente relacionada con la programación y el desarrollo de software, lo que permite optimizar el rendimiento y la eficiencia energética [14][13].

La arquitectura de computadoras se ocupa del estudio y diseño de los componentes de hardware y software de una computadora y su interacción. Este campo abarca desde el diseño de circuitos hasta la integración de sistemas completos, siendo crucial para el desarrollo de tecnologías eficientes y avanzadas. Estudiar arquitectura de computadoras permite comprender los sistemas informáticos desde su nivel más básico, ofreciendo una visión holística de su funcionamiento y estructura. Además, está estrechamente relacionada con la programación y el desarrollo de software, lo que permite optimizar el rendimiento y la eficiencia energética. También brinda una base sólida para comprender y aprovechar nuevas tendencias tecnológicas, como la computación en la nube, la inteligencia artificial y el internet de las cosas, preparándonos para contribuir a la innovación tecnológica [14][13].

En resumen, estudiar arquitectura de computadoras nos proporciona un conocimiento profundo de los sistemas informáticos, nos capacita para diseñar sistemas eficientes y escalables, y nos prepara para aprovechar las últimas tendencias tecnológicas. Es una disciplina esencial para aquellos que desean seguir una carrera en el campo de la informática y la tecnología, y contribuir al avance de la sociedad digital en la que vivimos.

Otro motivo relevante para estudiar arquitectura de computadoras es la capacidad de contribuir al desarrollo de nuevas tecnologías. La innovación en esta área ha sido constante, desde la miniaturización de componentes hasta el diseño de arquitecturas avanzadas como las basadas en computación cuántica. Al obtener conocimientos en arquitectura, podemos ser parte de esta evolución tecnológica y contribuir a la creación de sistemas más poderosos y eficientes.

Por último, el estudio de la arquitectura de computadoras nos proporciona una base sólida para comprender otros campos relacionados, como la seguridad informática y los sistemas embebidos. Estos conocimientos son cada vez más demandados en la industria, lo que abre oportunidades laborales en empresas de desarrollo de software, fabricantes de hardware, centros de investigación y muchas otras áreas relacionadas con la tecnología.

En conclusión, estudiar arquitectura de computadoras es fundamental para comprender el funcionamiento de los sistemas informáticos, resolver problemas de rendimiento y eficiencia, contribuir a la innovación tecnológica y acceder a una amplia gama de oportunidades laborales. Es una disciplina emocionante que impulsa la evolución de la tecnología y nos permite ser parte activa de este cambio.

2.1.1 Arquitectura x86

La arquitectura x86, una de las más influyentes y ampliamente utilizadas en el ámbito de las computadoras de escritorio y servidores, comenzó su desarrollo en 1978 con el lanzamiento del procesador Intel 8086, que introdujo una arquitectura de 16 bits. La arquitectura x86 evolucionó significativamente con el Intel 80386 en 1985, marcando el inicio de la era de 32 bits. En 2003, AMD lanzó la arquitectura AMD64, extendiendo x86 a 64 bits, lo que permitió un mayor acceso a la memoria y un mejor rendimiento en aplicaciones intensivas. Intel adoptó estas innovaciones, consolidando la arquitectura x86 como una de las más versátiles y potentes del mercado [15][16][17].

Contextualización Histórica y Evolución de la Arquitectura x86

La retrocompatibilidad de la arquitectura x86 ha sido un factor determinante en su éxito, permitiendo que aplicaciones de 16, 32 y 64 bits se ejecuten en el mismo sistema. Esta característica ha asegurado la continuidad y protección de las inversiones en software y sistemas operativos desarrollados para x86.

Cuadro 2.1: Hitos procesadores x86

Procesador	Año.de.Lanzamiento	Número.de.Bits	Nuevas.Características
Intel 8086	1978	16	Arquitectura inicial
Intel 80386	1985	32	Memoria virtual
AMD64	2003	64	Extensiones de 64 bits

La evolución de la arquitectura x86 ha estado marcada por hitos importantes que han impulsado la informática hacia nuevas alturas. Tras el Intel 8086, el lanzamiento del Intel 80286 en 1982 introdujo modos de operación adicionales que mejoraron la eficiencia y el manejo de memoria. En 1989, el Intel 80486 incorporó una unidad de punto flotante integrada y una mejor caché, aumentando significativamente el rendimiento.

La serie Pentium, iniciada en 1993, llevó la arquitectura x86 a nuevos niveles de rendimiento y eficiencia, con características avanzadas como la ejecución superescalar y la predicción de saltos. El Pentium Pro en 1995 mejoró la arquitectura con ejecución fuera de orden y una caché L2 integrada.

En la década de 2000, la arquitectura x86 se adaptó a las demandas de la computación moderna con la introducción del Intel Core, optimizando el rendimiento y la eficiencia energética. AMD también fue crucial con su serie Athlon y la introducción de AMD64, llevando la arquitectura x86 a 64 bits, permitiendo un mayor acceso a la memoria y mejorando el rendimiento en aplicaciones intensivas.

La arquitectura x86 ha tenido un impacto profundo en el desarrollo de software. Los sistemas operativos populares como Windows y Linux están optimizados para x86, lo que ha influido en el desarrollo y la optimización de aplicaciones para esta arquitectura. - Influencia en el Desarrollo de Software: Optimización del Rendimiento: Los desarrolladores de software han trabajado estrechamente con las características de la arquitectura x86 para optimizar el rendimiento de sus aplicaciones. Esto incluye el uso de instrucciones específicas de x86 y la optimización para cachés y pipelines. - Compatibilidad y Soporte: La compatibilidad hacia atrás de x86 ha permitido la continuidad de aplicaciones y sistemas operativos, protegiendo las inversiones en software y facilitando las actualizaciones. - Ecosistema de Desarrollo: Un amplio ecosistema de herramientas de desarrollo, bibliotecas y frameworks ha sido construido alrededor de la arquitectura x86, facilitando el desarrollo de aplicaciones de alto rendimiento y su depuración.

Cuadro 2.2: Línea de Tiempo de la Evolución de la Arquitectura x86

Año	Procesador	Innovación_Principal
1978	Intel 8086	Introducción de la arquitectura x86, 16 bits
1982	Intel 80286	Modos de operación adicionales
1985	Intel 80386	Arquitectura de 32 bits, memoria virtual
1989	Intel 80486	Unidad de punto flotante integrada, mejor caché
1993	Intel Pentium	Ejecución superescalar, predicción de saltos
1995	Intel Pentium Pro	Ejecución fuera de orden, caché L2 integrada
2003	AMD64	Extensiones a 64 bits, mayor acceso a memoria
2006	Intel Core	Optimización de rendimiento y eficiencia energética

Comparación con Otras Arquitecturas

La arquitectura x86 ha dominado el mercado de las computadoras de escritorio y servidores durante décadas, pero existen otras arquitecturas importantes que también han tenido un impacto significativo en la informática. Comparar x86 con arquitecturas como ARM [22], MIPS [13] y RISC-V [23] nos permite entender mejor sus ventajas y desventajas.

Cuadro 2.3: Comparación de Arquitecturas

Característica	x86	ARM	MIPS	RISC.V
Eficiencia Energética	Moderada	Alta	Moderada	Alta
Complejidad ISA	Alta	Baja	Moderada	Baja
Rendimiento	Alto	Moderado	Moderado	Variable

Característica	x86	ARM	MIPS	RISC.V
Compatibilidad	Alta (hacia atrás)	Moderada	Moderada	Alta
Áreas de Aplicación	Escritorio, servidores	Dispositivos móviles	Sistemas embebidos	Investigación, embebidos

La arquitectura ARM, conocida por su eficiencia energética, ha ganado popularidad en dispositivos móviles y sistemas embebidos. MIPS, aunque menos prominente en la actualidad, se ha utilizado en sistemas embebidos y en educación. RISC-V, una arquitectura abierta y libre, ha surgido como una alternativa flexible y eficiente, especialmente en investigación y aplicaciones embebidas [24].

2.1.2 Repertorio de instrucciones

El repertorio de instrucciones, conocido como ISA (Instruction Set Architecture), es fundamental para la interacción entre el software y el hardware de una computadora [14].

Filosofías CISC y RISC

La arquitectura x86 sigue la filosofía CISC (Complex Instruction Set Computer), la cual se caracteriza por tener un repertorio de instrucciones amplio y complejo. Esta filosofía busca reducir la cantidad de instrucciones por programa, a costa de aumentar la complejidad de cada instrucción individual. Esto contrasta con la filosofía RISC (Reduced Instruction Set Computer), que utiliza un repertorio de instrucciones más simple y optimiza el uso de registros y operaciones rápidas, aumentando el número de instrucciones necesarias por programa pero simplificando el diseño del hardware.

Instrucciones x86

El repertorio de instrucciones x86 incluye una variedad de instrucciones para realizar operaciones aritméticas, lógicas, de control y de manejo de memoria. Algunos ejemplos son:

- `MOV`: Mueve datos de una ubicación a otra.
- `ADD`: Suma dos valores.
- `SUB`: Resta un valor de otro.
- `JMP`: Salta a una dirección específica.

Además de estas, existen muchas otras instrucciones que permiten manipular registros, gestionar interrupciones y ejecutar operaciones en punto flotante, entre otras. La versatilidad y profundidad del ISA x86 la han convertido en una de las arquitecturas más utilizadas y estudiadas en el campo de la informática.

2.2 Lenguaje ensamblador

Un procesador puede interpretar y ejecutar solo instrucciones máquina. Estas instrucciones son simplemente secuencias de números binarios almacenados en la computadora y son leídas por el procesador e interpretadas por él. Si un programador quisiera programar directamente en el lenguaje máquina, necesita introducir los programas como datos binarios. Esto requeriría escribir las sentencias que necesita realizar el procesador directamente en binarios, por lo tanto, conocer la secuencia de ceros y unos para cada operación escribiéndose ordenadamente, además de respetar la estructura de memoria y direccionamientos del procesador. Esto sin duda es un trabajo complejo, difícil, pesado y muy susceptible a errores. Adicionalmente, una vez que se requiera realizar modificaciones, su lectura implica ir traduciendo estas secuencias de ceros y unos a su correspondiente instrucción, lo que es doblemente dificultoso [25].

Para facilitar la programación de bajo nivel, se creó el lenguaje ensamblador, que es un lenguaje de programación de bajo nivel que permite a los programadores escribir instrucciones comprensibles por el procesador. A diferencia del lenguaje máquina, que utiliza secuencias de números binarios, el ensamblador emplea mnemónicos simbólicos que hacen que el código sea más legible y manejable para los humanos. Cada arquitectura de procesador tiene su propio lenguaje ensamblador que usualmente es definida por el fabricante de hardware, por lo tanto es específica de cierta arquitectura de la computadora física. Un programa llamado ensamblador es usado para traducir sentencias del lenguaje ensamblador al código de máquina de la computadora. El ensamblador realiza una traducción casi directa uno a uno desde las sentencias mnemónicas a las instrucciones y datos de máquina. Esto está en contraste con los lenguajes de alto nivel, en los cuales una sola declaración generalmente da lugar a muchas instrucciones de máquina [14].

El código fuente generado en mnemotécnicos debe ser traducido a lenguaje máquina para poder ser ejecutado por la computadora. Este proceso lo realizan programas denominados ensambladores. Dentro del contexto de la arquitectura x86, existen diversos lenguajes ensambladores como TASM (Turbo Assembler) [26], MASM (Microsoft Macro Assembler) [27] y NASM (Netwide Assembler) [28], cada uno con características y beneficios únicos que se adaptan a distintas necesidades y preferencias de los programadores. Estos ensambladores convierten el código simbólico en código máquina, permitiendo su ejecución en el hardware específico de la arquitectura x86 [29].

2.3 Simulación

La simulación es una herramienta esencial en diversos campos, incluyendo la medicina, el ámbito militar, el entretenimiento y la educación.Facilita la comprensión del funcionamiento de sistemas, la generación de hipótesis, la realización de análisis predictivos y la respuesta a preguntas del tipo "¿qué pasaría si?".

Según Banks [2], la simulación se define como el proceso de imitar el comportamiento de un sistema a lo largo del tiempo, lo cual requiere desarrollar primero un modelo conceptual que capture las características y comportamientos del sistema real. La simulación, entonces, representa la evolución de este modelo conforme el tiempo avanza [2][4].

La capacidad de replicar y analizar sistemas complejos sin intervenir directamente en ellos convierte la simulación en una metodología indispensable para ingenieros, diseñadores y gerentes en el mundo digital actual. La simulación permite evaluar el rendimiento de sistemas, predecir su comportamiento en diferentes escenarios y optimizar su diseño antes de implementarlos en la realidad. Con los avances en el mundo digital, la simulación se ha convertido en una metodología indispensable para ingenieros, docentes, diseñadores y gerentes. La complejidad intrínseca de los sistemas informáticos los hace difíciles de comprender y costosos de desarrollar sin utilizar simulación. [3].

2.3.1 Aplicaciones de la Simulación en la Industria

En la industria automotriz, la simulación es fundamental para el diseño y prueba de sistemas de seguridad, como airbags y frenos. Mediante un modelo virtual del automóvil y sus componentes, es posible realizar pruebas de colisión y evaluar el rendimiento de los sistemas de seguridad sin recurrir a costosas pruebas físicas. Además, la simulación permite optimizar el diseño de motores, analizar el flujo aerodinámico y prever el comportamiento de los materiales en condiciones extremas. En el campo de la aviación, la simulación se utiliza para entrenar pilotos en simuladores de vuelo que replican condiciones reales sin riesgos. También se emplea en el diseño de aeronaves para evaluar la aerodinámica y el rendimiento de nuevos diseños bajo diversas condiciones de vuelo. Estos ejemplos ilustran cómo la simulación puede reducir costos, mejorar la seguridad y acelerar el desarrollo de productos complejos.

2.3.2 Simulación en la Educación

En el ámbito educativo, la simulación es una herramienta poderosa para enseñar conceptos complejos y fomentar el aprendizaje activo. Los simuladores permiten a los estudiantes interactuar con sistemas virtuales y experimentar con escenarios realistas,

facilitando la comprensión de conceptos abstractos y la aplicación de conocimientos teóricos en situaciones prácticas.

En la enseñanza de la arquitectura de computadoras, los simuladores son especialmente útiles para ilustrar el funcionamiento interno de los procesadores, la ejecución de instrucciones y el manejo de la memoria. Los estudiantes pueden experimentar con diferentes configuraciones y parámetros, observar el impacto en el rendimiento y comprender cómo se aplican los conceptos teóricos en la práctica. La simulación también permite a los estudiantes explorar escenarios hipotéticos y evaluar el comportamiento de sistemas complejos sin necesidad de hardware físico. En resumen, la simulación en la educación es una herramienta valiosa para mejorar la comprensión de los estudiantes, fomentar la experimentación y promover el aprendizaje activo [18][7][8].

Para superar estas limitaciones, se han desarrollado numerosos simuladores que ofrecen experiencias de aprendizaje valiosas al replicar el funcionamiento y la estructura de las computadoras. Simuladores como SimpleScalar, SPIM y GEM5 permiten a los estudiantes experimentar con arquitecturas complejas y técnicas avanzadas como el pipelining y la ejecución fuera de orden. Estas herramientas facilitan la comprensión de los conceptos teóricos a través de la interacción práctica, proporcionando un entorno seguro y accesible para la exploración y el aprendizaje [18].

Utilizando simuladores como SimpleScalar, los estudiantes pueden visualizar cómo las instrucciones se ejecutan en diferentes etapas del pipeline, y cómo se manejan las dependencias de datos y los riesgos de control. El pipelining es una técnica utilizada en las CPUs modernas para mejorar el rendimiento. Al simular el pipelining, los estudiantes pueden comprender cómo se dividen las instrucciones en etapas y cómo se gestionan los conflictos para evitar cuellos de botella. Además, los simuladores permiten a los estudiantes experimentar con diferentes configuraciones y parámetros para evaluar su impacto en el rendimiento y la eficiencia de la CPU.

Aporte pedagógico

En el ámbito educativo, transmitir los fundamentos teóricos de la organización y arquitectura interna de las computadoras puede ser un desafío debido a la complejidad de los procesos involucrados. Los métodos tradicionales de enseñanza, como el uso de pizarras, libros de texto y diapositivas, a menudo tienen una capacidad limitada para representar estos conceptos de manera efectiva. Esto requiere que los estudiantes tengan un alto nivel de abstracción para desarrollar un modelo mental adecuado para capturar la organización y arquitectura interna de las computadoras [5][6].

La integración de nuevas tecnologías como recurso didáctico es crucial para facilitar el aprendizaje. Las herramientas de simulación permiten a los estudiantes relacionar conceptos abstractos con situaciones reales, situándose en un contexto que imita aspectos de la realidad. Este enfoque pedagógico facilita la detección de problemáticas y

el desarrollo de habilidades a través del trabajo exploratorio, la inferencia y el aprendizaje por descubrimiento. Simuladores como Simulink, Proteus y Logisim juegan un papel esencial en la enseñanza de la arquitectura de computadoras, proporcionando una representación visual e interactiva que enriquece la comprensión teórica y práctica de los estudiantes [7][8].

El uso de herramientas de simulación en la enseñanza para procesos dinámicos complejos, como las operaciones intrínsecas de la computadora, que permiten representar de forma visual e interactiva la organización y arquitectura interna de la computadora, facilitando así la comprensión de su funcionamiento por parte de los alumnos y el desarrollo de los temas por parte del docente. En este contexto, los simuladores juegan una pieza clave en el campo de la Arquitectura de Computadoras, permitiendo conectar fundamentos teóricos con la experiencia práctica, implicando abstracciones y haciendo más rica la labor docente.

2.3.3 El Formalismo DEVS (Discrete Event System Specification)

DEVS, la abreviación de Discrete Event System Specification, es un formalismo modular y jerárquico para el modelado y análisis de sistemas que pueden ser representados como sistemas de eventos discretos, sistemas de estado continuo o sistemas híbridos. Este formalismo, desarrollado por Bernard P. Zeigler en los años 70, extiende el concepto de las máquinas de Moore al proporcionar una estructura para modelar sistemas complejos mediante la utilización de eventos cronometrados [20].

Descripción del Formalismo DEVS

El formalismo DEVS define el comportamiento de un sistema real utilizando eventos de entrada y salida, y transiciones entre estados concretos. Un sistema en DEVS está compuesto por modelos atómicos y acoplados. Los modelos atómicos representan las unidades básicas de comportamiento, mientras que los modelos acoplados consisten en combinaciones de modelos atómicos y/o otros modelos acoplados. Esta estructura jerárquica facilita la gestión y análisis de sistemas complejos, permitiendo la prueba de subsistemas de manera aislada antes de integrarlos en el sistema completo. Bajo un punto de vista general, un modelo DEVS está caracterizado por generar eventos de salida Y , en relación con el estado en el que se encuentre S y las entradas recibidas X, cada cierto tiempo.

Aplicaciones del Formalismo DEVS

El formalismo DEVS es aplicable a una amplia gama de sistemas, desde redes de comunicación hasta procesos de manufactura. Por ejemplo, en una red de comunicación,

un modelo DEVS puede simular el enrutamiento de paquetes de datos y la gestión de congestiones. En la manufactura, un modelo DEVS puede representar el flujo de materiales y el control de calidad en una línea de producción, ayudando a identificar cuellos de botella y optimizar procesos.

Ejemplo de Modelo DEVS

Consideremos un sistema de colas en un banco, donde los clientes llegan, esperan en la fila y son atendidos por cajeros. Un modelo DEVS puede representar los eventos de llegada de clientes, la asignación a los cajeros y el tiempo de servicio, permitiendo evaluar el tiempo de espera y la utilización de los recursos. Este ejemplo ilustra cómo DEVS puede ser utilizado para mejorar la eficiencia operativa y la satisfacción del cliente mediante la optimización del flujo de trabajo y la asignación de recursos.

Capítulo 3

Comparativa de simuladores

En este capítulo, se analizan y comparan varios simuladores x86 que podrían integrarse en la asignatura Arquitectura de Computadoras de la Licenciatura en Sistemas.

La selección y evaluación de estos simuladores se ha basado en criterios específicos diseñados para medir su efectividad en un entorno educativo, con el objetivo de identificar las herramientas que mejor apoyen el proceso de enseñanza y aprendizaje. Los criterios de evaluación incluyen aspectos como usabilidad, editor de código, documentación, ejecución de simulación, nivel de especificación de la arquitectura x86, características del producto y cobertura de los contenidos curriculares.

Los resultados de esta comparativa se publicaron en el XVII Congreso de Tecnología en Educación y Educación en Tecnología en 2022, bajo el título 'Herramientas de software para dar soporte en la enseñanza y aprendizaje de la arquitectura x86' [30].

3.1 Estudios similares

Cabe destacar que existen antecedentes de estudios comparativos que evalúan diferentes simuladores aplicados a la enseñanza de los cursos de arquitectura de computadoras: - "A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization", 2009 [11]: Otros estudios evalúan diferentes simuladores para abordar diferentes temas en el dictado de los cursos de Arquitectura de Computadoras, en general estos estudios evalúan simuladores en términos de dos categorías predefinidas: una referida a las características de la simulación, como ser granularidad, usabilidad, disponibilidad, presentación visual, flujo de simulación, etc., y otra sobre la cobertura de los contenidos preestablecidos en las currículas. - "Survey and evaluation of simulators suitable for teaching for computer architecture and organization Supporting undergraduate students at Sir Syed University of Engineering & Technology", 2012 [12]: Evalúa aspectos como usabilidad, disponibilidad, funda-

mentos de arquitectura informática, jerarquía de sistemas de memoria, comunicación e interfaz y diseño de sistemas de procesadores.

Este trabajo se diferencia al proponer un enfoque diferente: evaluar los simuladores x86 bajo criterios de evaluación basados en características de simulación y en contenidos específicos de la asignatura Arquitectura de Computadoras de la carrera de Licenciatura en Sistemas de la Universidad Nacional de Entre Ríos

3.2 Simuladores bajo análisis

Un simulador de arquitectura es un software que imita una situación del mundo real y, en este contexto, puede imitar el hardware de un sistema de cómputo. Estos simuladores se enfocan principalmente en representar los aspectos arquitectónicos y funcionales del hardware. El uso de simuladores permite realizar cambios, pruebas y ejecución de programas sin temor a dañar ningún componente o por falta de la computadora. [10].

Algunos programas ofrecen una representación visual e interactiva de la organización y arquitectura interna de la computadora, facilitando así la comprensión de su funcionamiento. Entre estos se encuentran los simuladores Assembly debugger (x86), Simple 8-bit Assembler Simulator, Microprocessor Simulator, Simulador de ensamblador de 16 bits y Emu8086. En este sentido, los simuladores juegan un papel clave en el campo de la Arquitectura de Computadoras, permitiendo conectar fundamentos teóricos con la experiencia práctica, simplificando abstracciones y facilitando la labor docente [11][12][13][14][31].

3.3 Criterios de evaluación

Los criterios de evaluación se han definido cuidadosamente para asegurar una valoración integral de los simuladores. Estos criterios incluyen:

- **Usabilidad**: Se refiere a la capacidad ser usado del producto.
  - **Indicadores**:
    - Facilidad de aprendizaje: tiempo requerido para familiarizarse con la herramienta.
    - Interfaz de usuario: claridad y organización.
    - Documentación y ayuda: disponibilidad y accesibilidad de tutoriales y guías.
  - **Escala**: difícil-media-fácil.

- **Editor**: Funcionalidades que ofrece el editor para escribir y depuración de código en lenguaje ensamblador.
  - **Indicadores**:
    * Capacidad de edición: funcionalidades como resaltado de sintaxis, puntos de interrupción (breakpoints).
    * Errores de sintaxis: detección y aviso de errores.
    * Almacenamiento: opciones para guardar y cargar programas.
  - **Escala**: baja-media-alta.
- **Documentación**: Disponibilidad de la documentacion para dar soporte al aprendizaje.
  - **Indicadores**:
    * Manual de usuario: disponibilidad y calidad.
    * Tutoriales: existencia y utilidad de tutoriales de aprendizaje.
    * Repertorio de instrucciones: exhaustividad y claridad en la explicación de instrucciones.
  - **Escala**: mínima-media-completa.
- **Ejecución de simulación**: Facilidad para controlar y observar el flujo de ejecución de sus programas.
  - **Indicadores**:
    * Control de la simulación: opciones para pausar, reanudar, y retroceder la ejecución.
    * Visualización: claridad en la representación del flujo de ejecución.
    * Configurabilidad: opciones para ajustar la velocidad del reloj de la CPU y otros parámetros.
  - **Escala**: baja-media-alta.
- **Nivel de especificación de la Organización y Arquitectura del sistema simulado**: Precisión con la que el simulador representa la arquitectura x86. Nivel de implementación del set de instrucciones, memoria, módulos de E/S, etc.
  - **Indicadores**:
    * Precisión de la arquitectura: fidelidad en la representación de la arquitectura x86.
    * Implementación del set de instrucciones: completitud y exactitud.
    * Soporte de módulos: inclusión y funcionalidad de memoria, módulos de E/S, etc.
  - **Escala**: mínima-media-completa.
- **Características del desarrollo del producto software**: se basa en las características propias del producto software.

  - **Indicadores**:
    * Tipo de licencia: open source o privativa.
    * Actualizaciones: frecuencia y fecha de la última versión.
    * Plataforma: si es una aplicación web o de escritorio.
  - **Escala**: mala-buena-muy buena.

- **Cobertura de los contenidos preestablecidos en la currícula**: Cobertura de los tópicos de la asignatura Arquitectura de Computadoras.
  - **Indicadores**:
    * Correspondencia con el currículum: alineación con los tópicos de la asignatura Arquitectura de Computadoras.
    * Profundidad de los temas: nivel de detalle en la cobertura de los contenidos curriculares.
  - **Escala**: baja-media-alta.

3.4 Selección de simuladores

A partir de una exhaustiva exploración en internet sobre herramientas de simulación de la arquitectura x86 utilizadas para la enseñanza, se identificaron los siguientes simuladores: Assembly debugger (x86), Simple 8-bit Assembler Simulator, Microprocessor Simulator, Simulador de ensamblador de 16 bits, Emu8086, VonSim, Orga1 y Qsim.

La selección se basa en una evaluación preliminar que considera se seleccionaron tres considerando el tiempo de evaluación de cada uno y la cantidad de criterios cumplidos por cada simulador, buscando aquellos que ofrecen un balance adecuado entre funcionalidad, usabilidad, documentación y alineación con los contenidos curriculares de la asignatura Arquitectura de Computadoras. De estos simuladores, se seleccionaron tres que, a priori, contemplaban la mayor cantidad de criterios a evaluar: Emu8086, VonSim y Simple 8-bit Assembler Simulator.

3.5 Participantes en la evaluación

La evaluación de los simuladores aplicando los criterios preestablecidos fue llevada a cabo por 3 docentes de la asignatura arquitectura de computadoras [Marcelo A. Colombani, José M. Ruiz, Amalia G. Delduca], quienes aportaron su experiencia en el uso de simuladores y su aplicabilidad en la enseñanza de arquitectura de computadoras. Además, se contó con la colaboración de 1 asesor externo [Marcelo A. Falappa], quienes ofrecieron una visión externa y validaron el proceso de evaluación y los resultados obtenidos.

3.6 Análisis comparativo

A continuación, se presenta un análisis detallado de los simuladores seleccionados basado en los criterios de evaluación definidos:

3.6.1 Simple 8-bit Assembler Simulator

- **Usabilidad**: Nivel medio. Presenta todos los componentes en una sola pantalla, lo cual puede ser abrumador para usuarios principiantes.
- **Editor**: Nivel bajo. Incluye aviso de errores de sintaxis al ensamblar, sin resaltado de sintaxis ni breakpoints. No permite guardar o cargar programas.
- **Documentación**: Nivel mínimo. Consta solo de un manual de instrucciones implementadas.
- **Ejecución de simulación**: Nivel medio. Permite configurar la velocidad del reloj de CPU y ofrece controles básicos de simulación.
- **Nivel de especificación**: Nivel mínimo. Simplifica la arquitectura x86 en un CPU de 8 bits, con memoria de 256 bytes y sin soporte para IN y OUT.
- **Desarrollo del producto**: Nivel bueno. Licencia MIT, última versión en 2015, desarrollado sobre plataforma web.
- **Cobertura de contenidos**: Nivel bajo. No implementa memoria independiente para módulos de entrada y salida, rutinas de tratamiento de interrupciones ni ciclo de instrucción.

3.6.2 VonSim

- **Usabilidad**: Nivel medio. Presenta componentes mediante solapas, lo cual puede ser abrumador para usuarios iniciales.
- **Editor**: Nivel medio. Incluye aviso de errores de sintaxis, resaltado de sintaxis y breakpoints mediante interrupción por software.
- **Documentación**: Nivel medio. Posee manual de uso y tutorial interactivo.
- **Ejecución de simulación**: Nivel medio. Permite configurar la velocidad del reloj de CPU y ofrece controles básicos de simulación.
- **Nivel de especificación**: Nivel medio. Representa una simplificación del procesador 8088 con arquitectura de 16 bits y memoria direccionable de 16 KiB.
- **Desarrollo del producto**: Nivel muy bueno. Licencia GNU Affero General Public License v3.0, última versión en 2020, desarrollado sobre plataforma web con extensa evidencia de uso académico.
- **Cobertura de contenidos**: Nivel medio. Implementa dispositivos internos y externos, pero no desarrolla contenidos visuales para ciclo de instrucción y medidas de rendimiento.

3.6.3 Emu8086

- **Usabilidad**: Nivel fácil. Presenta inicialmente el editor y permite activar componentes del simulador a medida que se cargan programas.
- **Editor**: Nivel alto. Incluye aviso de errores de sintaxis, resaltado de sintaxis y opciones de breakpoints. Permite guardar y cargar programas.
- **Documentación**: Nivel completo. Incluye manual de instrucciones, tutorial de aprendizaje y manual de uso.
- **Ejecución de simulación**: Nivel alto. Permite configurar la velocidad del reloj de CPU y ofrece controles avanzados de simulación, incluyendo "step back".
- **Nivel de especificación**: Nivel completo. Representa detalladamente la arquitectura del procesador 8086 con memoria direccionable de 1 MiB y soporte para interrupciones por software y hardware.
- **Desarrollo del producto**: Nivel bueno. Licencia privativa, última versión en 2023, desarrollado sobre plataforma de escritorio.
- **Cobertura de contenidos**: Nivel alto. Soporta todos los modos de direccionamiento y permite emular el booteo de una IBM PC desde el floppy disk, entre otros.

Cuadro 3.1: Comparativa según criterios de evaluación preestablecidos

Criterio.de.Evaluacion	Simple.8.bit.Assembler.Simulator	VonSim	Emu8086
Usabilidad	Medio	Medio	Fácil
Editor	Bajo	Medio	Alto
Documentación	Mínima	Media	Completa
Ejecución de simulación	Medio	Medio	Alta
Nivel de especificación x86	Mínima	Media	Completa
Características del producto	Buena	Muy buena	Buena
Cobertura de contenidos	Baja	Media	Alta

3.7 Resultados

La asignatura promueve el uso de simuladores para apoyar la enseñanza y el aprendizaje, permitiendo aplicar los contenidos desarrollados en máquinas reales. Emu8086 es la herramienta más adecuada para esta finalidad, ya que facilita la implementación de programas en hardware real. Sin embargo, su dependencia de MS-DOS complica su ejecución en sistemas operativos actuales, requiriendo el uso de emuladores de MS-DOS, lo que añade complejidad al proceso de enseñanza y aprendizaje.

Desde 2018, la asignatura utiliza la versión 4.08 de Emu8086. La herramienta tiene un periodo de evaluación gratuito de 14 días, después del cual se debe adquirir una

licencia. Esto es un inconveniente, ya que se busca que los estudiantes puedan acceder a las herramientas de forma libre y gratuita.

Utilizar lenguaje NASM (Netwide Assembler) garantiza soporte tanto para Linux como Windows a través de herramientas libres como GCC (GNU Compiler Collection), generando programas para la arquitectura x86 de 16, 32 y 64 bits.

Emu8086 destaca por su interfaz dinámica, que muestra componentes como la pila, flags, teclado y pantalla solo cuando son necesarios, a diferencia de otros simuladores que presentan todos sus componentes desde el inicio.

En el criterio de evaluación dos, Emu8086 se destaca por su editor, que incluye puntos de ruptura para detener la ejecución del programa en un punto específico y retroceder a la intruccion anterior. Además, permite guardar y recuperar programas directamente desde el editor.

En el criterio de evaluación tres, Emu8086 se destaca por ofrecer tres tipos de documentación: un repertorio de instrucciones con ejemplos para cada tipo de instrucción, un manual que explica las partes de la herramienta y un tutorial para aprender a programar en ensamblador.

En el criterio de evaluación cuatro, Emu8086 se destaca por ofrecer una mayor cantidad de controladores para gestionar el flujo de ejecución, como la capacidad de retroceder la ejecución de una instrucción y recargar el programa actual.

En el criterio de evaluación cinco el emu8086 se destaca del resto debido a que ofrece una mayor especificidad de la arquitectura x86, además implementa interrupciones del sistema operativo MS-DOS, a través del cual se pueden ejecutar los programas en una máquina real.

En el criterio de evaluación seis VonSim se destaca del resto debido a que es licencia libre y posee una comunidad que respalda el proyecto.

En cuanto al último criterio, ninguna de las herramientas evaluadas cubre todos los contenidos que se pretende desarrollar con la ayuda de una herramienta, quedando excluido pasos del ciclo de instrucción y medidas de rendimientos (tiempo de CPU y CPI: ciclo por instrucción).

En resumen: - Emu8086 se destaca por su alta usabilidad, documentación completa, y precisión en la simulación de la arquitectura para la enseñanza de la arquitectura x86. Sin embargo, su licencia privativa y dependencia de MS-DOS representan desafíos importantes. - VonSim ofrece una buena solución con licencia libre y una interfaz más amigable, pero su nivel de especificación y cobertura de contenidos es medio, lo cual podría limitar su efectividad en el curso. - Simple 8-bit Assembler Simulator tiene limitaciones significativas en usabilidad, documentación, y especificación, lo que lo hace menos adecuado para una enseñanza completa de la arquitectura x86.

Del análisis realizado, se concluye que, aunque todos los simuladores evaluados presentan ventajas y desventajas, ninguno de los simuladores analizados parece adecuado para la asignatura sugiere que los simuladores actuales no cumplen completamente con los requisitos de complejidad y cobertura necesarios para la enseñanza exhaustiva de la arquitectura x86. Se recomienda usar Emu8086 mientras se desarrolla un simulador que combine sus mejores características con una licencia libre, con soporte para sistemas operativos modernos y una mayor cobertura de contenidos curriculares para la asignatura..

3.7.1 Publicación

El resultado de esta comparativa fue publicado en el XVII Congreso de Tecnología en educación y Educación en Tecnología año 2022 bajo el título de "Herramientas de software para dar soporte en la enseñanza y aprendizaje de la arquitectura x86" [30].

Durante la elaboración de este análisis comparativo establecí contacto con uno de los docentes de la Universidad Nacional de la Plata (UNLP) que habían desarrollado un simulador web VonSim, con el acuerdo de docente se generó una solicitud de incorporación de cambios (pull request) en el repositorio de Github del simulador VonSim y en agosto-2023 salió una nueva versión donde se implementan animaciones de la ejecución de instrucciones y documentación on-line.

Capítulo 4

Diseño y construcción del simulador

El objetivo principal de esta tesis es construir una herramienta de simulación de la arquitectura x86 para apoyar la enseñanza de arquitectura de computadoras. Para lograr este objetivo, se plantean los siguientes objetivos específicos:

1. Estudiar y evaluar diferentes herramientas actuales de simulación destinadas a dar apoyo a la enseñanza de la arquitectura x86.

   Se cumplió el primer objetivo mediante una revisión exhaustiva de las herramientas de simulación actuales destinadas a apoyar la enseñanza de la arquitectura x86.

2. Construir una herramienta de apoyo para impartir los contenidos de la asignatura Arquitectura de Computadoras, para ello debe cumplir:

   - Una visión global de la estructura y funcionamiento de la computadora.
   - Generación y ejecución de programas en ensamblador.
   - Repertorio de instrucciones x86 reducido y habilitado progresivamente.
   - Simulación visual e interactiva de micropasos de instrucciones.
   - Gestión de interrupciones y comunicación con periféricos.
   - Medidas de rendimiento de ejecución de programas.

Para cumplir con el segundo objetivo, se diseñó y construyó una herramienta de apoyo que cumple con los requisitos previamente mencionados para impartir los contenidos de la asignatura Arquitectura de Computadoras.

Construir una herramienta de apoyo para impartir los contenidos de la asignatura Arquitectura de Computadoras, para ello debe cumplir, se puede adpatar

Para lograr esto, se comenzamos por definir los requisitos de la herramienta.

4.1 Requisitos de la herramienta

Para cumplir con el segundo objetivo específico, la herramienta debe cumplir con los siguientes requisitos:

- **Visión global de la computadora:** Debe mostrar la estructura completa de la computadora (CPU, bus, memoria y E/S) durante la ejecución de programas, destacando componentes relevantes.
- **Generación y ejecución de programas en ensamblador:** Permitir tanto la ejecución paso a paso como completa, facilitando la comprensión de cada instrucción.
- **Repertorio de instrucciones x86 reducido:** Seleccionar un subconjunto esencial de instrucciones x86, habilitándolas progresivamente según avance el contenido de la asignatura.
- **Simulación visual e interactiva de micropasos:** Utilizar el lenguaje de transferencia entre registros (RTL) para describir el flujo de datos, facilitando la comprensión del ciclo básico de una instrucción. Correr un programa y que se ilumine cada componente durante la ejecución de un programa.
- **Gestión de interrupciones:** Incorporar un vector de interrupción predefinido para interactuar con el teclado y monitor.
- **Comunicación con módulos de E/S y periféricos:** Incluir instrucciones IN y OUT, y un módulo de E/S simplificado basado en el modo 2 del Intel 8255.
- **Medidas de rendimiento:** Proveer información sobre tiempo de ciclo, tiempo de CPU y CPI de un programa.

La herramienta debe ofrecer una visión global de la estructura y funcionamiento de la computadora, permitir la generación y ejecución de programas escritos en lenguaje ensamblador, ya sea paso a paso por instrucción o completa, y ofrecer un repertorio de instrucciones x86 reducido donde se habiliten las instrucciones a medida que se desarrolle el contenido en la asignatura.

Además, la herramienta debe simular de manera visual e interactiva los micropasos que conlleva el ciclo básico de una instrucción, permitir la gestión básica de interrupciones permitiendo la interacción con el teclado y la pantalla, permitir la comunicación con los módulos de entrada y salida e interacciones con los periféricos, y ofrecer medidas de rendimiento sobre la ejecución de un programa.

Definición de Requisitos: - Visión global de la estructura y funcionamiento de la computadora. - Generación y ejecución de programas en ensamblador. - Repertorio

de instrucciones x86 reducido y habilitado progresivamente. - Simulación visual e interactiva de micropasos. - Gestión de interrupciones y comunicación con periféricos. - Medidas de rendimiento de ejecución de programas.

Diseño de la Herramienta: - Considerar la facilidad de uso para estudiantes y profesores. - Utilizar gráficos y diagramas para mejorar la comprensión.

Construcción de la Herramienta: - Implementar la herramienta siguiendo las especificaciones de diseño. - Realizar pruebas exhaustivas para asegurar que la herramienta cumple con todos los requisitos. - Solucionar cualquier problema identificado durante las pruebas antes del lanzamiento.

Para cumplir con el segundo objetivo específico, es esencial diseñar y construir una herramienta de apoyo que sea fácil de usar para estudiantes y profesores, cumpliendo con los requisitos definidos, y realizar pruebas rigurosas para asegurar su funcionalidad y usabilidad.

La herramienta debe ofrecer una visión global de la estructura y funcionamiento de la computadora, permitir la generación y ejecución de programas escritos en lenguaje ensamblador, ya sea paso a paso por instrucción o completa, y ofrecer un repertorio de instrucciones x86 reducido donde se habiliten las instrucciones a medida que se desarrolle el contenido en la asignatura. Además, la herramienta debe simular de manera visual e interactiva los micropasos que conlleva el ciclo básico de una instrucción, permitir la gestión básica de interrupciones permitiendo la interacción con el teclado y la pantalla, permitir la comunicación con los módulos de entrada y salida e interacciones con los periféricos, y ofrecer medidas de rendimiento sobre la ejecución de un programa.

Una vez que se hayan definido los requisitos, se puede comenzar a diseñar la herramienta. Se debe tener en cuenta la facilidad de uso para los estudiantes y profesores. La herramienta debe ser intuitiva y fácil de navegar. Se pueden utilizar gráficos y diagramas para ayudar a los estudiantes a comprender mejor la estructura y funcionamiento de la computadora.

Después de diseñar la herramienta, se debe construir y probar. Se deben realizar pruebas exhaustivas para asegurarse de que la herramienta cumpla con todos los requisitos y sea fácil de usar. Si se encuentran problemas durante las pruebas, se deben solucionar antes de lanzar la herramienta.

En resumen, para cumplir con el segundo objetivo específico, se debe diseñar y construir una herramienta de apoyo para impartir los contenidos de la asignatura Arquitectura de Computadoras. La herramienta debe cumplir con los requisitos mencionados anteriormente y debe ser fácil de usar para los estudiantes y profesores. Para lograr esto, se puede comenzar por definir los requisitos de la herramienta, diseñar la herramienta, construir y probar.

4.2 Desarrollo y Pruebas

- **Diseño de la Herramienta:** Comenzar con la definición detallada de los requisitos, seguido de un diseño centrado en la facilidad de uso para estudiantes y profesores. Se utilizarán gráficos y diagramas para mejorar la comprensión.
- **Construcción de la Herramienta:** Implementar la herramienta siguiendo las especificaciones de diseño, asegurando la integridad y funcionalidad de cada componente.
- **Pruebas Exhaustivas:** Realizar pruebas rigurosas para asegurar que la herramienta cumple con todos los requisitos. Las pruebas incluirán:
  - Validación de la funcionalidad completa de la herramienta.
  - Evaluación de la usabilidad por parte de estudiantes y profesores.
  - Identificación y solución de problemas antes del lanzamiento.

4.3 Portabilidad

Para asegurar la portabilidad, el simulador se implementará como una aplicación web que puede ejecutarse en cualquier navegador web, garantizando su uso multiplataforma.

4.4 Simplicidad

La herramienta será diseñada con una interfaz intuitiva y fácil de navegar, minimizando la curva de aprendizaje para los usuarios.

4.5 Mantenibilidad

El código de la herramienta será modular y bien documentado, facilitando futuras actualizaciones y mantenimiento. Además, se implementarán prácticas de desarrollo sostenibles para asegurar su longevidad.

4.6 Escalabilidad

Desde la asignatura se incentiva el uso de simuladores para dar apoyo a los proceso de enseñanza y aprendizaje, pero también se incentiva que los contenidos desarrollados

puedan volcarse en máquinas reales, en este sentido consideramos que el enfoque planteado por la herramienta emu8086 es el más adecuado para la asignatura, ya que facilita mecanismos para implementar los programas en máquinas reales. Sin embargo presenta el inconveniente que genera ejecutables dependientes del sistema operativo MS-DOS, la mayoría de los sistemas operativos actuales no permiten la ejecución de dichos programas, obligando a la utilización de emuladores de MS-DOS para poder correrlos, siendo esto otro elemento más que se incorpora a los procesos de enseñanza y aprendizaje. Utilizar lenguaje NASM (Netwide Assembler) garantiza soporte tanto para Linux como Windows a través de herramientas libres como GCC (GNU Compiler Collection), generando programas para la arquitectura x86 de 16, 32 y 64 bits.

4.7 Memoria

La memoria contiene 256 posiciones y cada una contiene un byte. Por lo tanto, el bus de direcciones y de datos son de 8 bits.

4.8 Repertorio de instrucciones propuesto

El repertorio de instrucciones x86 propuesto para facilitar el aprendizaje y la enseñanza de la arquitectura x86 para aquellos que están recién familiarizándose con este conjunto de instrucciones, es un repertorio ficticio basado en la arquitectura x86, para facilitar el aprendizaje se decidió simplificar los conceptos a un repertorio de 8 bits para un mejor entendimiento. Aunque el conjunto de instrucciones real de x86 es mucho más amplio y complejo, este enfoque básico sentará las bases para comprender el formato de instrucciones, los modos de direccionamiento y el ciclo de búsqueda y ejecución.

4.9 Repertorio de instrucciones

El repertorio x86 es reducido (8 bits)

Instrucciones	Código operación	Acción
Transferencia de datos	← **MOV {0,1,2}**	0. Copiar entre registros 1. Almacenar en memoria 2. Cargar a registro

Instrucciones	Código operación	Acción
Procesamiento de datos	**+ ADD {3, 4, 5} - SUB {6, 7, 8} * CMP {9, 10, 11}**	**Operación aritmética**: operando1 ← operando1 OPE operando2 ; actualiza el destino **Comparación**: operando1 - operando2 ;realiza la operación de sustraer pero no actualiza el destino.
Control de flujo	**↑ JMP / Jxx / CALL/ RET {12}**	Salto incondicional JMP Saltos condicionales Jxx Llamadas a rutinas CALL y retorno RET
Manejo de pila y E/S	**PUSH / POP / OUT / IN {13}**	Poner en la pila PUSH Retirar de la pila POP Enviar un byte al puerto del dispositivo de E/S Recibir un byte del dispositivo de E/S
Manejo de interrupción	**INT / IRET {14}**	Llamar a una rutina de tratamiento de interrupción INT Retornar de una rutina de tratamiento de interrupción IRET
Control del CPU	**NOP / HLT {15}**	No opera NOP Detiene el CPU HLT

4.9.1 Formato de la instrucción

Formato instrucción		Acción
Código op. byte 1 4 bits	**2 bits**	**2 bits**

	Formato instrucción			Acción
MOV	**0**	*rd*	*rf*	**Registro a registro**: Copiar entre registros: rd ← rf
	1	*{0,1,2,3}*	*rf / {0,1, 2}*	** Registro/valor a memoria**: *mem ← rf mem* BL ← *rf mem* $Dir + BL$ ← *rf 3.0 mem ← valor 1 mem* BL ← *valor 3.2 mem* $Dir + BL$ ← *valor*

Formato instrucción			Acción
2	*rd*	*{0,1,2,3}*	** Memoria/valor a registro**: *0.* $rd \leftarrow mem$ $rd \leftarrow mem$ BL $rd \leftarrow mem$ $Dir + BL$ $rd \leftarrow valor$

	Formato instrucción			Acción
ADD / SUB / CMP	**{3..11}**	*rd*	*rf*	Todas las instrucciones de procesamiento de datos tienen los mismos modos de direccionamiento que la instrucción MOV. Operación aritmética: **operando1 ← operando1 OPE operando2** ; actualiza el destino Comparación: **operando1 - *operando2*** ;realiza la operación de sustraer pero no actualiza el destino. Operando2 puede ser registro/memoria/valor inmediato Operando1 puede ser registro o memoria

Formato instrucción	Acción
rr	
rd	‘

	Formato instrucción		Acción
JMP/ Jxx / CALL / RET	**12**	*{0,1,2,3, 4,5,6,7,8,9,10,11,12,13,14}*	**saltos/rutinas *RE: el registro de estado contiene los siguientes flags o banderas: Z =cero C = acarreo (Carry) S = signo (sign) O = desbor-damiento (Over-flow) Incondi-cional: Sin condi-cional* 0. JMP: IP ↑ IP + desp *Condi-cional: Si se cumple la condición salta* 1. JZ/JE: Z == 1 2. JNZ/JNE: Z == 0 3. JC: Si C == 1 4. JNC: Si C == 0 5. JS: Si S == 1 6. JNS: Si S == 0 7. JO: Si O == 1 8.**

	Formato instrucción			Acción
PUSH / POP / OUT / IN	**13**	*{0,1, 2, 3}*	*rf/rd*	**Pila y E/S *0.* PUSH *rf: Memoria* *SP* ↑ *rf SP* ↑ *SP - 1 1.* POP *rd: SP* ↑ *SP + 1 rd* ↑ *Memoria* *SP* *2.* E/S *puerto* ↑ DL *3.* DL ↑E/S *puerto*

	Formato instrucción		Acción
INT / IRET	**14**	*ID_INT /* -	**Interrupción *0.* INT Nro_INT*: Memoria* SP ↑ *IP SP* ↑ *SP - 1 Memoria* SP ↑ *RE SP* ↑ *SP - 1 IP* ↑ *dirección* *IP* ↑ *Vector_Interrupción* ID_INT *STI (Setear flag interrupción) 1.* IRET*: SP* ↑ *SP + 1 RE* ↑ *Memoria* SP *SP* ↑ *SP +1 IP* ↑ *Memoria* SP *CLI (restablecer flag interrupción)*

	Formato instrucción			Acción
NOP / HLT	**15**	-	**{0,1}	** Control *0.* NOP*: IP ↑ IP + 1 1.* HLT*: Detiene el CPU*

4.9.2 Registros (Banco de Registros)

Números de registros (r)		
registro	**binario**	**decimal**
AL	00	0
BL	01	1
CL	10	2
DL	11	3

4.9.3 Modos de direccionamientos

Los siguientes tipos de direccionamiento son utilizados en las instrucciones de esta CPU para referenciar a los operandos involucrados en la instrucción:

- Registro a registro: los operandos de la instrucción son registros.
- Directo: en la instrucción se indica la dirección de memoria en la que está contenido el operando.
- Indirecto: dirección de memoria donde está el operando viene determinada por el contenido del registro BL
- Inmediato: el operando fuente de la instrucción es un valor contenido en la misma instrucción.

La mayoría de las instrucciones básicas tienen solo las siguientes formas:

- **MOV *reg, reg***

- **MOV *reg, mem***
- **MOV *reg, inm***
- **MOV *mem, reg***
- **MOV *mem, inm***

4.9.4 Transferencia MOV (COPIAR)

MOV d, f

Tipo de operandos d y f: pueden ser registros, dirección de memoria, dirección indirecta por registro (BL) o valor. El operando valor no puede ser destino.

rf: registro fuente rd: registro destino

Los 4 primeros bits de la instrucción se descompone en: Código operación 3 bis y 1 bit se corresponde al tipo de destino, si es cero representa registro y si es 1 representa memoria.

Formato instrucción MOV Ejemplo Acción Direccionamiento

4.9.5 Ciclo de la instrucción (Etapas de captación y ejecución)

Captación

```
1.  MAR = IP
2.  MDR = read(Memoria[MAR])
    IP = IP + 1
3.  IR  = MDR
```

Ejecución

Instrucciones Operación Nemónico Acción Transferencia de datos Mover MOV od, of Copiar: operando-destino (od) operando-fuente (of)

Procesamiento de datos + Sumar
- Restar Comparar ADD od, of SUB od, of CMP od, of od = od + of

od od - of
od - of

Control de flujo Salto incondicional Salto condicional Detener JMP d Jxx d HLT Salto incondicional a dirección destino (d). Saltos condicionales en base a las banderas (flags) a d. Detiene el CPU.

Formato de Instrucciones y modos de direccionamiento

El formato de las instrucciones propuesto para la enseñanza de la arquitectura es una simplificación de la arquitectura x86, siendo este último un set CISC (conjunto de repertorio de instrucciones complejas) las instrucciones tienen diferente tamaño para poder aplicar estos conceptos a la enseñanza se representa un repertorio de instrucciones simplificado, podemos clasificar las instrucciones según los modos de direccionamientos que indica de donde proviene un operando: Operando tipo r Registros del CPU [d] Contenido de la dirección de memoria i Valor inmediato [r] Contenido de la dirección de memoria dada por el registro

4.10 Procesador `VonSim8`

`VonSim8` es un procesador diseñado e implementado sobre la herramienta *Logisim*. Este cuenta con las siguientes características:

Arquitectura CPU	**Características**
	- Arquitectura *von Neumann*, memoria de datos e instrucciones compartida. - 8 registros de propósito general, `R0` a `R7`. - 1 registro de propósito específico `IP`. - Tamaño de palabra de 8 bits e instrucciones de 16 bits. - Memoria de 256 palabras de 8 bits. - Bus de 8 bits. - Diseño microprogramado.

4.11 Instrucciones

Las instrucciones están codificadas en 16 bits. Los primeros 5 bits identifican el `opcode` de la instrucción, el resto de los bits indican los parámetros. Existen 4 posibles codificaciones de parámetros.

Caso	Codificación	Parámetros
A	`OOOO XXYY-----`	`XX` = Registro X, `YY` = Registro Y o inmediato
B	`OOOO XX--------`	`XX` = Registro X
C	`OOOO ---MMMMMMMM`	`MMMMMMMM` = Dirección de memoria o Inmediato
D	`OOOO XXMMMMMMMM`	`XXX` = Registro X, `MMMMMMMM` = Dir. de memoria o Imm.

Considerando:

- `Rx` o `Ry`: Índices de registros, número entre `0` y `7`.
- `M`: Dirección de memoria o valor inmediato, número de 8 bits.
- `t`: Valor inmediato de desplazamiento, número entre `0` y `7`. Se codifica como `YYY`.
- En la columna de codificación, los bits indicados con `-` son reservados y deben valer cero.
- Las instrucciones de `opcode`: 9, 10, 11, 12, 13, 14, 15, 28, 29 y 30 son instrucciones reservadas.

Las instrucciones soportadas por la arquitectura son las siguientes:

Instrucción	Acción	Codificación
`ADD  Rx, Ry`	`Rx` ← `Rx + Ry`	`00001 XXXYYY-----`
`SUB  Rx, Ry`	`Rx` ← `Rx - Ry`	`00011 XXXYYY-----`
`CMP  Rx, Ry`	Modifica *flags* de `Rx - Ry`	`00111 XXXYYY-----`
`MOV  Rx, Ry`	`Rx` ← `Ry`	`01000 XXXYYY-----`
`MOV  [M], Rx`	`Mem[M]` ← `Rx`	`10000 XXXMMMMMMMM`
`MOV Rx, [M]`	`Rx` ← `Mem[M]`	`10001 XXXMMMMMMMM`
`MOV  [Rx], Ry`	`Mem[Rx]` ← `Ry`	`10010 XXXYYY-----`
`MOV Rx, [Ry]`	`Rx` ← `Mem[Ry]`	`10011 XXXYYY-----`
`JMP M`	`PC` ← `M`	`10100 ---MMMMMMMM`
`JC M`	Si `flag C=1` entonces `PC` ← `M`	`10101 ---MMMMMMMM`
`JZ M`	Si `flag Z=1` entonces `PC` ← `M`	`10110 ---MMMMMMMM`

Las instrucciones soportadas por la arquitectura son las siguientes:

4.12 Assembler

#	Mnemonics	Action
0	`MOV Rx, Ry`	`Ra = Rb`
1	`ADD Rx, Ry`	`Ra += Rb`
2	`SUB Rx, Ry`	`Ra -= Rb`
3	`CMP Rx, Ry`	`Ra -= Rb`
4	`MOV Rx, [Rb]`	`Ra = Mem[Rb]`
5	`MOV [Rb], Rx`	`Mem[Rb] = Ra`
7	`INP Ra`	`Ra = Inp`
8	`JEQ Ra, value|label`	`PC = value|label, Ra == 0`
9	`JNE Ra, value|label`	`PC = value|label, Ra != 0`
a	`JGT Ra, value|label`	`PC = value|label, Ra > 0`
b	`JLT Ra, value|label`	`PC = value|label, Ra < 0`
c	`MOV Ra, value|label`	`Ra = Mem[value|label]`
d	`MOV value|label, Ra`	`Mem[value|label] = Ra`
e	`MOV Ra, value|label`	`Ra = value|label`
f	`JMP value|label`	`PC = value|label`

4.13 Registros

##	Name	Description
00	R0	User data
01	R1	User data
10	R2	User data (output pins A0-A7)
11	R3	User data (output pins B0-B7)

4.14 Componentes

La arquitectura está compuesta por 6 componentes interconectados. El circuito identificado como `microOrgaSmall` los integra en un `dataPath` sobre el lado izquierdo del mismo. El lado derecho presenta la visualización del estado de los registros.

Los componentes de la arquitectura son: `Registers` (Banco de Registros), `PC` (Contador de Programa), `ALU` (Unidad Aritmético Lógica), `Memory` (Memoria), `Decode` (Decodificador de Instrucciones) y `ControlUnit` (Unidad de Control).

Cada uno de estos componentes es controlado por medio del conjunto de entradas y salidas descritas a continuación:

4.15 Ejecución de Instrucciones

Las instrucciones en `OrgaSmall` se ejecutan en varios ciclos de reloj, donde cada ciclo implica una acción específica. A continuación, se describe la secuencia de acciones para la ejecución de las instrucciones de la arquitectura.

1. **Ciclo de Búsqueda**:
    - `PC` envía la dirección actual al bus.
    - `Memory` coloca la instrucción en el bus de datos.
    - `Decode` carga la instrucción en sus registros internos.
    - `PC` incrementa su valor.
2. **Ciclo de Decodificación**:
    - `Decode` divide la instrucción en `opcode`, registros e inmediato.
    - `ControlUnit` interpreta el `opcode` y emite señales correspondientes.
3. **Ciclo de Ejecución**:
    - `ControlUnit` habilita y controla los componentes para ejecutar la instrucción.
    - `ALU` realiza la operación aritmética o lógica.
    - `Registers` y `Memory` realizan operaciones de lectura/escritura según sea necesario.

Este flujo se repite para cada instrucción, permitiendo la ejecución de programas complejos en la arquitectura `OrgaSmall`.

Bibliografía

[1] M. A. Colombani, M. A. Falappa, A. G. Delduca, and J. M. Ruiz, "PID novel 7065: Enseñanza/aprendizaje de asignatura Arquitectura de Computadoras con herramientas de simulación de sistemas de cómputos." Feb. 2022. Accessed: Jul. 10, 2024. Available: https://proyectos.uner.edu.ar/aplicacion.php?ah=st668e6d47663eb&ai=gestion_extinv||2300

[2] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-event system simulation*, 5th ed. Prentice Hall, 2010.

[3] A. M. Law, *Simulation Modeling & Analysis*, 5th ed. New York, NY, USA: McGraw-Hill, 2015.

[4] S. Robinson, *Simulation: The Practice of Model Development and Use*, 2nd edition. 2014.

[5] C. Lion, "Los simuladores. Su potencial para la enseñanza universitaria," *Cuadernos de Investigación Educativa*, vol. 2, no. 12, pp. 53–66, 2005.

[6] G. Contreras, R. G. Torres, and M. S. R. Montoya, "Uso de simuladores como recurso digital para la transferencia de conocimiento," *Apertura: Revista de Innovación Educativa*, vol. 2, no. 1, pp. 86–100, 2010.

[7] A. Garcia-Garcia, J. C. Saez, J. L. Risco-Martin, and M. Prieto-Matias, "PB-BCache: An open-source parallel simulator for rapid prototyping and evaluation of cache-partitioning and cache-clustering policies," *Journal of Computational Science*, vol. 42, p. 101102, 2020.

[8] B. Nova, J. C. Ferreira, and A. Araújo, "Tool to support computer architecture teaching and learning," in *Engineering Education (CISPEE), 2013 1st International Conference of the Portuguese Society for*, IEEE, 2013, pp. 1–8.

[9] B. Mustafa, "Evaluating A System Simulator For Computer Architecture Teaching And Learning Support," *Innovation in Teaching and Learning in Information and Computer Sciences*, vol. 9, no. 1, pp. 100–104, 2010, doi: 10.11120/ital.2010.09010100.

[10] Z. Radivojevic, M. Cvetanovic, and J. Đordevic, "Design of the simulator for teaching computer architecture and organization," in *2011 Second Eastern European*

Regional Conference on the Engineering of Computer Based Systems, IEEE, 2011, pp. 124–130.

[11] B. Nikolic, Z. Radivojevic, J. Djordjevic, and V. Milutinovic, "A Survey and Evaluation of Simulators Suitable for Teaching Courses in Computer Architecture and Organization," *IEEE Transactions on Education*, vol. 52, no. 4, pp. 449–458, Nov. 2009, doi: 10.1109/TE.2008.930097.

[12] R. Hasan and S. Mahmood, "Survey and evaluation of simulators suitable for teaching for computer architecture and organization Supporting undergraduate students at Sir Syed University of Engineering & Technology," in *Control (CONTROL), 2012 UKACC International Conference on*, IEEE, 2012, pp. 1043–1045.

[13] J. L. Hennessy and D. A. Patterson, *Computer architecture: A quantitative approach*, Fifth Edition. Elsevier, 2012.

[14] W. Stallings, *Computer organization and architecture: Designing for performance*, Eleventh Edition. Pearson, 2013.

[15] Intel, "64 and IA-32 architectures software developers manual," *325462-060US*, vols. 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C and 3D, p. 4670, 2016, Available: https://software.intel.com/en-us/articles/intel-sdm

[16] AMD, "Developer Guides, Manuals & ISA Documents." Apr. 2019. Accessed: Apr. 21, 2019. Available: https://developer.amd.com/resources/developer-guides-manuals/

[17] P. Abel, *IBM PC Assembly Language and Programming*, Fifth Edition. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[18] D. Skrien, "CPU Sim 3.1: A tool for simulating computer architectures for computer organization classes," *Journal on Educational Resources in Computing (JERIC)*, 2001.

[19] J. L. Peterson, *Petri net theory and the modeling of systems*. Prentice Hall PTR, 1981.

[20] B. Zeigler, H. Prähofer, and T. G. Kim, "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems," vol. 2, Jan. 2000.

[21] B. P. Zeigler, A. Muzy, and E. Kofman, *Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations*. Academic Press, 2018.

[22] D. A. Patterson and J. L. Hennessy, *Computer organization and design arm edition: The hardware software interface*. Morgan kaufmann, 2016.

[23] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, "The risc-v instruction

set manual, volume i: User-level isa," *CS Division, EECE Department, University of California, Berkeley*, p. 28, 2014.

[24] D. A. Patterson and J. L. Hennessy, *Computer organization and design risc-v edition: The hardware software interface*. Morgan Kaufmann, 2017.

[25] K. R. Irvine and L. B. Das, *Assembly language for x86 processors*. Prentice Hall, 2011.

[26] B. International, *Turbo assembler user's guide*. Borland International, 1993.

[27] M. Corporation, *Microsoft macro assembler 6.1 reference*. Microsoft Press, 1992.

[28] The NASM Project, *The netwide assembler (nasm) manual*. 2023.

[29] R. Hyde, *The art of assembly language*. No Starch Press, 2010.

[30] M. A. Colombani, J. M. Ruiz, A. G. Delduca, and M. A. Falappa, "Herramientas de software para dar soporte en la enseñanza y aprendizaje de la arquitectura x86," 2022. Accessed: Jul. 10, 2024. Available: http://sedici.unlp.edu.ar/handle/10915/139908

[31] P. BEHROOZ, *Computer Architecture: From Microprocessors to Supercomputers*. Oxford University Press Inc, 2005.