Introducción a Arquitectura de las Computadoras

Diego Feroldi

Arquitectura del Computador*
Departamento de Ciencias de la Computación
FCEIA-UNR

*Actualizado 15 de agosto de 2022 (D. Feroldi, feroldi@fceia.unr.edu.ar)

Índice

1. Definición de computadora

> *"Una computadora digital es una máquina que puede resolver problemas ejecutando las instrucciones que recibe de las personas"*[1].

La anterior es una de las tantas definiciones posibles de qué es una computadora. Esta definición indica que a pesar de las enormes prestaciones de las computadoras actuales, en su nivel más elemental las computadoras resuelven operaciones muy simples. En efecto, es importante destacar que los circuitos integrados de una computadora solo pueden reconocer y ejecutar un conjunto muy limitado de instrucciones sencillas. A lo largo de esta asignatura veremos como se realizan estas operaciones. En primer lugar, veamos un poco cómo están constituidas las computadoras.

2. Breve historia de la computación

Se han construido cientos de tipos diferentes de computadoras desde sus orígenes. Algunas con gran impacto y otras con menor incidencia. El objetivo de esta sección es dar un pantallazo muy breve sobre los orígenes de la computación marcando algunos hitos relevantes.

2.1. Computadoras mecánicas

- Blaise Pascal (1623-1662). Dispositivo construido en 1642 construido totalmente mecánico con engranajes. Solo podía restar y sumar.
- Goofried von Leibnitz (1646-1716). Construyó otra máquina totalmente mecánica que además podía multiplicar y dividir.
- Charles Babbage (1792-1871). Construyó una máquina diseñada para ejecutar un solo algoritmo con el objetivo de calcular tablas numéricas útiles para la navegación. Perforaba sus resultados en una placa de cobre.
- Luego desarrolló la máquina analítica para poder ejecutar diferentes algoritmos. Tenía cuatro componentes: el almacén (memoria), el molino (unidad de cómputo), la sección de entrada (lector de tarjetas perforadas) y la sección de salida (salidas perforadas e impresas). Podía sumar, restar multiplicar y dividir operandos.
- Ada Lovelace (1815-1852). Primera programadora de computadoras del mundo.
- Konrad Zuse (1910-1995). Construyó una serie de máquinas calculadoras automáticas empleando contactores electromagnéticos[1].
- John Atanasoff (1903-1995). Diseñó una máquina que utilizaba aritmética binaria y tenía una memoria de condensadores empleando un proceso que denominó "refrescar la memoria". Nunca funcionó por problemas de implementación aunque el concepto era correcto.
- George Stibbitz (1904-1995). Realizó una máquina más primitiva que la de Atanasoff aunque si funcionó.

[1] También denominados relés o relevadores. Un relé es un dispositivo electromagnético que funciona como un interruptor controlado por un circuito eléctrico en el que, por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independientes.

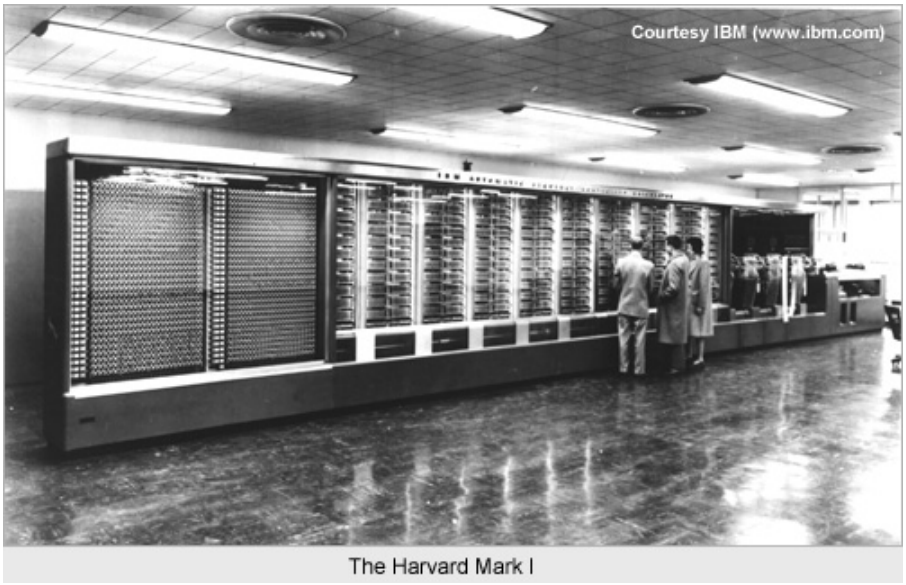


Figura 1: Imagen de la computadora Mark I (fuente:).

- Howard Aiken (1900-1973). Construyó con relés la máquina de propósito general que Babbage no había podido construir con ruedas dentadas. La primera máquina de Aiken (Mark I) se completó en Harvard en 1944. Tenía 72 palabras de 23 dígitos decimales cada una y un tiempo de instrucción de 6 segundos. Las entradas y salidas se efectuaban con cintas de papel perforadas (Ver Figura 1). Para cuando se terminó de desarrollar la Mark II las máquinas de relés ya eran obsoletas.

2.2. Computadoras con válvulas de vacío

La válvula electrónica, válvula de vacío, tubo de vacío o bulbo, es un componente electrónico utilizado para amplificar, conmutar, o modificar una señal eléctrica mediante el control del movimiento de los electrones en un espacio "vacío" a muy baja presión, o en presencia de gases especialmente seleccionados. La válvula originaria fue el componente crítico que posibilitó el desarrollo de la electrónica durante la primera mitad del siglo XX, incluyendo la expansión y comercialización de la radiodifusión, televisión, radar, audio, redes telefónicas, computadoras analógicas y digitales, control industrial, etc. Algunas de estas aplicaciones son anteriores a la válvula, pero experimentaron un crecimiento explosivo gracias a ella [2].

A continuación se enumeran algunos de los avances más notables de la computación utilizando esta tecnología:

- Alan Turing (1912-1954). Contribuyó a la creación de COLOSSUS, la primera computadora electrónica, desarrollada por el gobierno inglés.
- John Mauchley (1907-1980). Construyó ENIAC en 1946, la primera computadora moderna (Ver Figura 2). Tenía 18000 válvulas y 1500 relés. Pesaba 30 toneladas y consumía 140 kW. En la Figura 3 se puede apreciar una válvula de vacío.
- EDSAC. Máquina sucesora de ENIAC en 1949.
- John von Neumann (1903-1957). Desarrolló en 1952 la máquina IAS donde se emplea el diseño que ahora se conoce como **máquina de Von Neumann**, la cual tiene cinco partes básicas: la memoria, la unidad aritmética lógica, la unidad de control y el equipo de entrada y salida. Este diseño sigue siendo la base de casi todas las computadoras digitales aun hoy día.

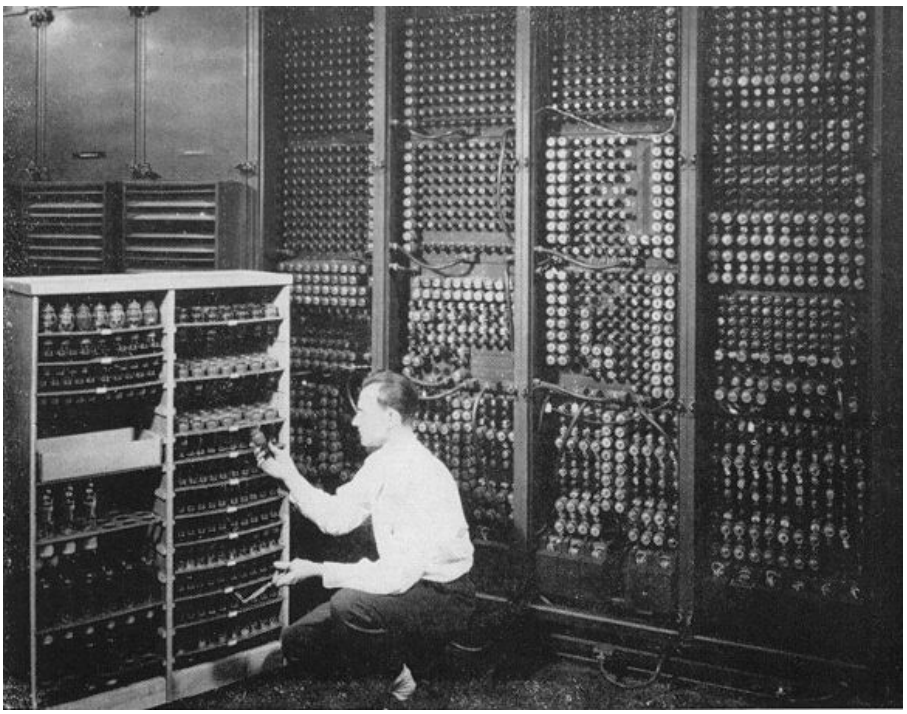


Figura 2: Imagen de la computadora ENIAC (fuente: [3]).

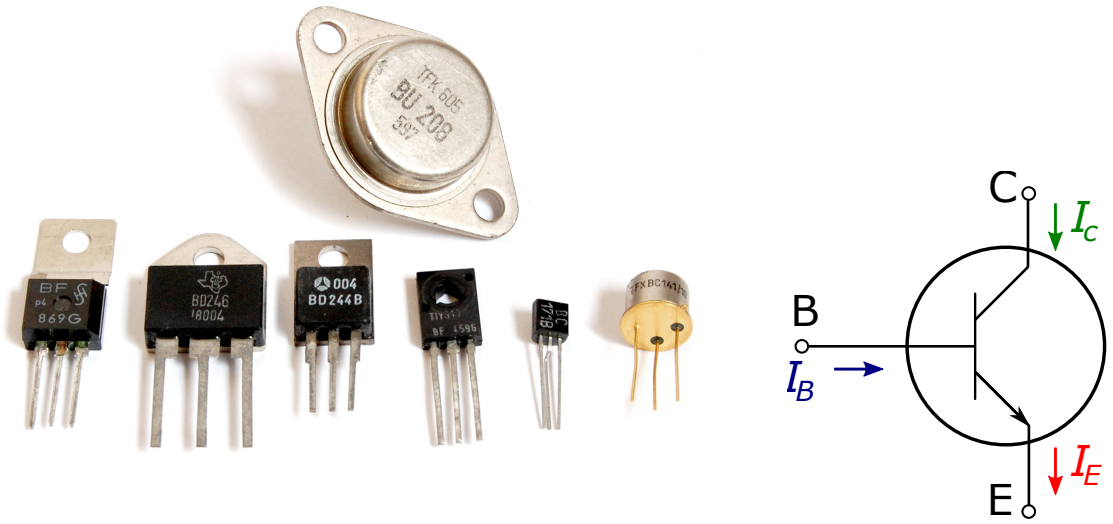
Figura 3: Imagen de una válvula electrónica (fuente: [2]).

- Whirlwind I (1951). Diseñada en el MIT. Tenía palabras de 16 bits y estaba diseñada para el control en tiempo real.

2.3. Computadoras con transistores

En 1948, John Bardeen, Walter Brattain y William Schocley inventan el transistor trabajando en los Labotarios Bell. El transistor es un dispositivo electrónico semiconductor utilizado para entregar una señal de salida en respuesta a una señal de entrada. Puede cumplir funciones de amplificador, oscilador, interruptor o rectificador. Actualmente se encuentra prácticamente en todos los aparatos electrónicos de uso diario tales como radios, televisores, reproductores de audio y video, relojes de cuarzo, computadoras, lámparas fluorescentes, tomógrafos, teléfonos celulares, aunque casi siempre dentro de los llamados circuitos integrados [4]. Los transistores pueden ser utilizados como interruptores, al igual que las válvulas, pero con mucho menor tamaño, costo y disipación de calor. En la Figura 4(a) se pueden apreciar diferentes tipos de

transistores mientras que en la Figura 4(b) se ilustra un esquema de un transistor tipo NPN, dónde de observan las tres partes fundamentales de un transistor: la *base*, el *colector* y el *emisor*.



(a) Imagen de algunos tipos de transistores.

(b) Diagrama de Transistor tipo NPN.

Figura 4: Transistores (fuente: [4]).

El impacto del transistor ha sido enorme, pues además de iniciar la industria de los semiconductores, ha sido el precursor de otros inventos como son los circuitos integrados, los dispositivos optoelectrónicos y los microprocesadores. Sin embargo, los cambios más notables han sido en la computación. Según palabras de Albert P. Malvino [5], el transistor no se creó para mejorar la computación sino que él la creó. Antes de 1950, una computadora ocupaba todo un salón y costaba millones de dólares. Luego, a partir del uso del transistor se pudo reducir notablemente el tamaño y el costo. Hoy, una computadora cabe en un bolsillo con un costo muy bajo y enormes prestaciones.

A continuación se enumeran algunos de los avances más notables de la computación utilizando esta tecnología:

- TX-0 (1956). Primera computadora transistorizada, desarrollada en el Lincoln Laboratory del MIT.
- PDP-1 (1960). Primera microcomputadora. 4K de palabras de 18 bits y tiempo de ciclo de 5 μs.
- 1401 (1961). Desarrollada por IBM y orientada a la contabilidad comercial (Ver Figura 5).
- 7094 (1962). Desarrollada por IBM y orientada a la computación científica.
- 6600 (1964). Desarrollada por CDC. Primera supercomputadora científica.

2.4. Computadoras con circuitos integrados

En 1958, Robert Noyce inventó el circuito integrado de silicio, lo cual hizo posible colocar docenas de transistores en un solo chip. Un circuito integrado, también conocido como chip o microchip, es una estructura de pequeñas dimensiones de material semiconductor, normalmente silicio, de algunos milímetros cuadrados de superficie, sobre la que se fabrican circuitos

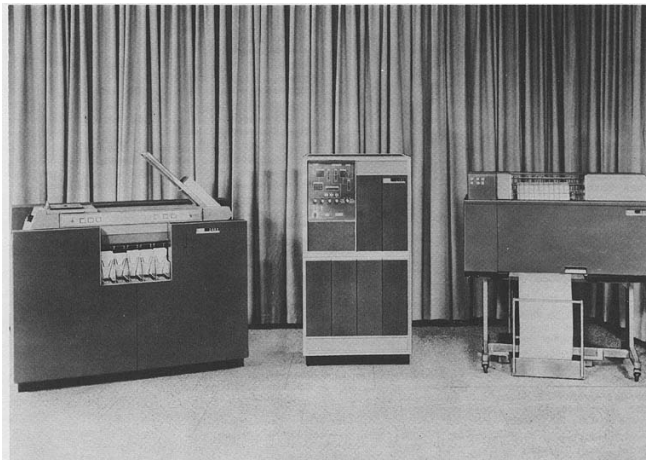


Figura 5: Imagen de la computadora IBM 1401 (fuente: [6]).

electrónicos generalmente mediante fotolitografía y que está protegida dentro de un encapsulado de plástico o de cerámica. El encapsulado posee conductores metálicos apropiados para hacer conexión entre el circuito integrado y un circuito impreso [7].

La integración de grandes cantidades de pequeños transistores dentro de un pequeño espacio fue un gran avance en la elaboración manual de circuitos utilizando componentes electrónicos discretos. La capacidad de producción masiva de los circuitos integrados, así como la fiabilidad y acercamiento a la construcción de un diagrama a bloques en circuitos, aseguraba la rápida adopción de los circuitos integrados estandarizados en lugar de diseños utilizando transistores discretos.

A continuación se enumeran algunos de los avances más notables de la computación utilizando esta tecnología:

- System/360 (1964). Desarrollada por IBM como familia de productos tanto para computación científica como comercial. Una importante innovación fue la multiprogramación (Ver Figura 6).
- PDP-8 (1965). Primera minicomputadora con mercado masivo (50000 unidades vendidas).
- PDP-11 (1970). Dominó el mercado de las minicomputadoras en los años setenta.

2.5. Computadoras con integración a muy gran escala

La integración a escala muy grande o VLSI (sigla en inglés de very-large-scale integration) es el proceso de crear un circuito integrado compuesto por cientos de miles de transistores en un único chip. VLSI comenzó a usarse en los años 70, como parte de las tecnologías de semiconductores y comunicación que se estaban desarrollando [8].

- Intel 8080 (1974). Primera computadora de propósito general de 8 bits en un chip. Corría a 2 MHz y se le considera el primer diseño de microprocesador verdaderamente usable. En la Figura 7 se puede apreciar el microprocesador Intel 8080:
- Apple II (1977). Primera serie de microcomputadores de producción masiva hecha por la empresa Apple Computer (Ver Figura 8). Arquitectura de 8 bits.
- 8086 (1978). Uno de los primeros chips individuales. Microprocesador de 16 bits. 29 K transistores. El 8088, una variante del 8086 con un bus externo de 8 bits fue utilizado en computadora personal original de IBM.

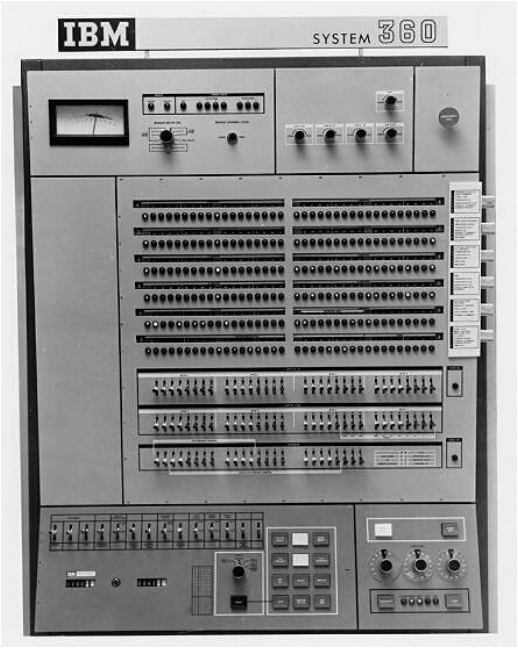


Figura 6: Imagen de la computadora IBM 360.

Figura 7: Microprocesador Intel 8080.

Figura 8: Computadora Apple II.

- 8087 (1980) Intel introdujo el coprocesador 8087 de punto flotante (45 K transistores) para operar junto a un procesador 8086 o 8088 ejecutando la instrucciones de punto flotante. El 8087 estableció el modelo de punto flotante para la línea x86, a menudo denominado "x87".

- IBM Personal Computer (1981). Se convirtió en la computadora más vendida de la histo-

ria. Venía equipada con el sistema operativo MS-DOS provisto por la compañía Microsoft Corporation:

- 80286 (1982). Agregó más modos de direccionamiento de memoria. Formó la base de la computadora personal IBM PC-AT, la plataforma original para MS Windows. 134 K transistores
- RISC-I (1982). Desarrollada dentro del proyecto RISC en la Universidad de Berkeley bajo la dirección de David A. Patterson. RISC, del inglés Reduced Instruction Set Computer, en español Computador con Conjunto de Instrucciones Reducidas, propone reemplazar arquitecturas complejas (CISC) por otras mucho más sencillas pero más rápidas.
- R2000 (1985). Primer diseño MIPS por John L. Hennessy, el cual mejoró drásticamente el rendimiento mediante el uso de la segmentación.
- i386 (1985). Expandió la arquitectura a 32 bits. 275 K transistores.
- i486 (1989). mejoró el desempeño de la línea x86 e integró la unidad de punto flotante en el chip pero no tuvo cambios significativos en el conjunto de instrucciones. 1.2 M transistores.
- Pentium (1993). Mejoró el desempeño pero solo agregó extensiones menores al conjunto de instrucciones. 3.1 M transistores.
- PentiumPro (1995). Introdujo un diseño de procesador radicalmente nuevo, conocido internamente como la microarquitectura P6. Se agregó una clase de instrucciones de movimiento condicional al conjunto de instrucciones. 5.5 M transistores.
- Pentium II (1997). Continuación of microarquitectura P6. 7 M transistores.
- Pentium III (1999). Introdujo el SSE, una clase of instrucciones para manipular vectores de datos enteros o de punto flotante. 8.2 M transistores pero versiones posteriores llegaron a 24 M transistores.
- Pentium 4 (2000). Extendió SSE a SSE2, agregando nuevos tipos de datos (incluyendo punto flotante de doble precisión), junto con muchas más instrucciones para estos tipos de datos. 42 M transistores
- Pentium 4E (2004). Se agregó *hyperthreading*, un método para ejecutar dos programas simultáneamente en un solo procesador, así como EM64T, la implementación de Intel de una extensión de 64 bits a IA32 desarrollada por Advanced Micro Devices (AMD), a la que nos referimos como x86-64. 125 M transistores.
- Core 2 (2006). Regresó a una microarquitectura similar a P6. Primer microprocesador Intel multinúcleo, donde se implementan múltiples procesadores en un solo chip. No es compatible con *hyperthreading*. 291 M transistores.
- Core i7 (2008). Incorpora tanto *hyperthreading* como *multi-core*, con la versión inicial que admite dos programas en ejecución en cada núcleo y hasta cuatro núcleos en cada chip. 781 M transistores

3. Definición de Arquitectura de Computadores

En el pasado, el término arquitectura del computador o arquitectura de computadores a menudo se refería solo al diseño del conjunto de instrucciones. Otros aspectos del diseño de la computadora fueron llamados *implementación*, a menudo insinuando que la implementación no es interesante o menos desafiante. Sin embargo, el trabajo del arquitecto o diseñador es mucho más que el diseño del conjunto de instrucciones, y los obstáculos técnicos en los otros aspectos del proyecto son probablemente más desafiantes que los encontrados en el diseño del conjunto de instrucciones.

Los principales objetivos en el diseño de una Arquitectura de Computadora son los siguientes:

- Maximizar el rendimiento,
- Reducir el costo,
- Reducir el consumo energético,
- Garantizar la confiabilidad.

Estos objetivos pueden ser contradictorios y en general se terminará adoptando una solución de compromiso. El objetivo de esta asignatura no es el diseño de una computadora, sino entender el funcionamiento desde una perspectiva de programador. En este sentido nos centraremos principalmente en el conjunto de instrucciones, la interacción con la memoria, cómo se representan los datos, etc. Por lo tanto, las preguntas clave al abordar una nueva arquitectura son:

- ¿Cuál es la longitud de la palabra de datos?
- ¿Cuáles son los registros?
- ¿Cómo se organiza la memoria?
- ¿Cuáles son las instrucciones?

4. Descripción básica de la arquitectura de una computadora

Los componentes básicos de una computadora incluyen una Unidad Central de Procesamiento (CPU), almacenamiento primario o memoria de acceso aleatorio (RAM, por su sigla en inglés *Random Access Memory*), almacenamiento secundario, dispositivos de entrada/salida (pantalla, teclado, mouse, etc.) y una interconexión entre los diferentes componentes denominada Bus. Un diagrama muy básico de la arquitectura de una computadora se muestra en la Figura 9.

Esta arquitectura se conoce típicamente como Arquitectura de von Neumann, o Arquitectura de Princeton, y fue descrita en 1945 por el matemático y físico John von Neumann. Los programas y los datos generalmente se almacenan en un almacenamiento secundario (por ejemplo, un disco rígido o de estado sólido). Cuando se ejecuta un programa, tanto los programas como los datos deben copiarse desde el almacenamiento secundario hacia el almacenamiento primario o memoria principal (RAM). Luego, la CPU ejecuta el programa desde el almacenamiento primario.

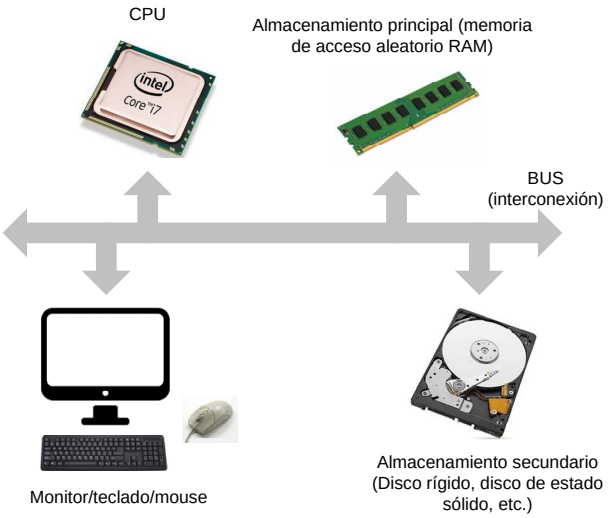


Figura 9: Esquema básico de la arquitectura de una computadora.

4.1. Procesador

La unidad central de procesamiento (CPU), o simplemente el procesador, es el "motor" que interpreta (o ejecuta) las instrucciones almacenadas en la memoria principal. Estos son algunos ejemplos de las operaciones simples que la CPU podría realizar a pedido de una instrucción:

- **Cargar**: Copiar información de la memoria principal en un registro, sobrescribiendo los contenidos anteriores del registro.
- **Almacenar**: Copiar información de un registro a una ubicación en la memoria principal, sobrescribiendo el contenido anterior de esa ubicación.
- **Operar**: Realizar operaciones con el contenido de registros de la ALU y almacenar el resultado en un registro, sobrescribiendo el contenido anterior de ese registro.
- **Saltar**: Modificar la secuencia de ejecución de instrucciones, modificando el contenido del contador de programa (PC).

4.1.1. ALU

La CPU incluye varias unidades funcionales, incluida la unidad aritmética lógica (ALU, por su sigla en inglés *Aritmetic Logic Unit*), que es la parte del chip que realmente realiza los cálculos aritméticos y lógicos. La función de la ALU es calcular nuevos datos y valores de dirección. A nivel de la ALU solo hay disponibles algunas pocas operaciones muy simple:

- operaciones aritméticas entre operandos,
- operaciones lógica de bits,
- operaciones de desplazamiento de bits.

Estas operaciones trabajan en torno a la memoria principal, el archivo de registros y la unidad aritmética/lógica (ALU).

4.1.2. Registros

Para que la ALU pueda operar, el chip también contiene registros y memoria caché. Un registro de la CPU, o simplemente un registro, es un almacenamiento temporal o una ubicación

de trabajo integrada en la propia CPU (separada de la memoria). El intercambio de datos entre la CPU y la memoria es una parte crucial de los cálculos en una arquitectura von Neumann: las instrucciones deben recuperarse de la memoria, los operandos también deben recuperarse de la memoria y algunas instrucciones almacenan resultados también en la memoria. Esto crea un cuello de botella y conduce a una pérdida de tiempo de la CPU cuando espera la respuesta de datos del chip de memoria. Por ello, para evitar una espera constante, el procesador está equipado con sus propias celdas de memoria, llamadas registros. Estos son pocos pero rápidos.

4.1.3. Memoria Caché

Por otra parte, la memoria caché es un pequeño subconjunto del almacenamiento primario o RAM ubicado en el chip de la CPU. Los registros de la CPU y la memoria caché se describirán con mayor detalle en la Sección 5. Cabe señalar que el diseño interno y la configuración de un procesador moderno es sumamente complejo. Este apunte proporciona una vista de alto nivel muy simplificada de algunas unidades funcionales clave dentro de una CPU.

4.1.4. Contador de programa

En el núcleo de la CPU hay un dispositivo de almacenamiento (o registro) del tamaño de una palabra llamado contador de programa (PC). En cualquier momento, el PC apunta a (contiene la dirección de) alguna instrucción en lenguaje máquina en la memoria principal. Desde el momento en que se aplica la alimentación al sistema, hasta el momento en que se apaga la alimentación, un procesador ejecuta repetidamente la instrucción señalada por el contador del programa y actualiza el contador del programa para que apunte a la siguiente instrucción. El funcionamiento de un procesador se puede interpretar de acuerdo con un modelo de ejecución de instrucciones muy simple. En este modelo, las instrucciones se ejecutan en secuencia estricta, y ejecutar una sola instrucción implica realizar una serie de pasos: el procesador lee la instrucción de la memoria señalada por el contador del programa (PC), interpreta los bits en la instrucción, realiza una operación simple dictada por la instrucción y luego actualiza el PC para que apunte a la siguiente instrucción, que puede o no ser contigua en memoria a la instrucción que se acaba de ejecutar.

4.2. Memoria principal

La memoria principal es un dispositivo de almacenamiento temporal que contiene tanto el programa como los datos que manipula mientras el procesador ejecuta el programa[2]. Físicamente, la memoria principal consiste en una colección de chips de memoria dinámica de acceso aleatorio (DRAM, por su sigla en inglés *Dynamic Random Access Memory*). A nivel lógico, la memoria está organizada como una matriz lineal de bytes[3], cada uno con su propia dirección única (índice de matriz) que comienza en cero. En general, cada una de las instrucciones de la máquina que constituyen un programa puede consistir en un número variable de bytes.

El almacenamiento primario o la memoria principal también se conoce como memoria volátil dado que cuando se corta la energía, la información no se retiene y, por lo tanto, se pierde. Por el contrario, el almacenamiento secundario (por ejemplo, el disco rígido) se denomina memoria no volátil ya que la información se retiene cuando la computadora se apaga.

[2]Esta es una de las principales características de la computadoras basadas en el concepto de *Máquina de Von Neumann*.

[3]Un conjunto ordenado de ocho bits se denomina byte. A su vez, un bit (acrónimo de *Binary digit*) es la mínima unidad de información en un sistema binario. Un bit o dígito binario puede representar uno de estos dos valores: 0 o 1. Este tema se verá con mayor detalle en el apunte **Representación Computacional de Datos**.

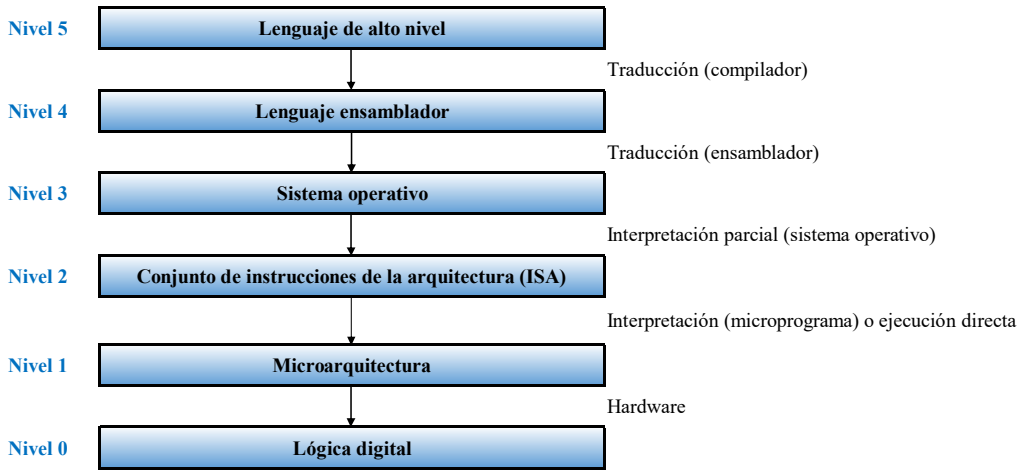


Figura 10: Modelo de máquina multinivel.

Los tamaños de los elementos de datos almacenados tienen correspondencia con las variables utilizadas en los lenguajes de alto nivel. Por ejemplo, en una máquina x86-64 que ejecuta Linux, los datos en lenguaje C de tipo `char` requieren 1 byte, los `short` requieren dos bytes, los `int` 4 bytes, los tipo `long` 8 bytes, los `float` 4 bytes y los `double` 8 bytes.

4.3. Buses

A lo largo de todo el sistema existe una colección de conductores eléctricos llamados buses que transportan bytes de información de un lado a otro entre los componentes. Los buses suelen estar diseñados para transferir fragmentos de información de tamaño fijo conocidos como palabras. El número de bytes en una palabra (el tamaño de la palabra) es un parámetro fundamental del sistema que varía de un sistema a otro. La mayoría de las máquinas actuales tienen tamaños de palabra de 4 bytes (32 bits) u 8 bytes (64 bits). En este curso nos centraremos en arquitecturas de 64 bits.

4.4. Dispositivos de entrada/salida

Los dispositivos de entrada/salida (*I/O devices*) son la conexión del sistema al mundo externo. La computadora del ejemplo de la Figura 9 tiene cuatro dispositivos de E/S: un teclado y un mouse para la entrada del usuario, una pantalla para la salida del usuario y una unidad de disco (o simplemente un disco rígido) para el almacenamiento a largo plazo de datos y programas. Cada dispositivo de E/S está conectado al bus de E/S mediante un controlador o un adaptador. Los controladores son conjuntos de chips en el propio dispositivo o en la placa de circuito impreso principal del sistema (a menudo denominada placa base). Un adaptador es una tarjeta que se conecta a una ranura en la placa base. En ambos casos, el propósito es transferir información de ida y vuelta entre el bus de E/S y un dispositivo de E/S.

4.5. Conjunto de instrucciones

El término conjunto de instrucciones de la arquitectura (ISA, por las siglas en inglés *Instruction Set Architecture*) se refiere al conjunto de instrucciones real disponible para controlar la CPU. El ISA actúa como una interfaz entre el hardware y el software, especificando lo que el procesador es capaz de hacer y cómo lo hace. La Figura 10 muestra cómo se inserta el ISA dentro de un esquema con diferentes niveles de abstracción en los que se puede describir una computadora.

El ISA define los tipos de datos admitidos, los registros, cómo el hardware administra la memoria principal, características clave (como la memoria virtual), qué instrucciones puede ejecutar un microprocesador y el modelo de entrada/salida de múltiples implementaciones de ISA. El ISA se puede ampliar agregando instrucciones u otras capacidades, o agregando soporte para direcciones y valores de datos más grandes.

Un ISA se puede clasificar de varias maneras diferentes. Una clasificación común es por complejidad arquitectónica. Una computadora con conjunto de instrucciones complejas (CISC, por la sigla en inglés *Complex Instruction Set Computer*) tiene muchas instrucciones especializadas, algunas de las cuales rara vez se usan en programas prácticos. Una computadora con conjunto de instrucciones reducido (RISC, por la sigla en inglés *Reduced Instruction Set Computer*) simplifica el procesador al implementar de manera eficiente solo las instrucciones que se usan con frecuencia en los programas, mientras que las operaciones menos comunes se implementan como subrutinas, lo que hace que el tiempo de ejecución del procesador adicional resultante se compense con el uso poco frecuente.

En esta asignatura nos vamos a focalizar en primer lugar en la arquitectura x86-64. El x86-64 es un diseño de CPU de tipo CISC. Esto se refiere a la filosofía de diseño del procesador interno. Los procesadores CISC generalmente incluyen una amplia variedad de instrucciones (a veces superpuestas), diferentes tamaños de instrucciones y una amplia gama de modos de direccionamiento. El término se acuñó retroactivamente en contraste con las arquitruras tipo RISC. Una de las arquitecturas tipo RISC es la denominada ARM, la cual estudiaremos al final del curso en el apunte *Conceptos básicos del ensamblador y arquitectura ARM*.

4.6. Microarquitectura

El nivel 1 del modelo de máquina multinivel mostrado en la Figura 10 se denomina microarquitectura. La microarquitectura se refiere a cómo se realiza la conexión entre la lógica y la arquitectura. La microarquitectura es la disposición específica de registros, ALU, máquinas de estados finitos (FSM), memorias y otros bloques de construcción lógicos necesarios para implementar una arquitectura. Una arquitectura particular, como x86-64, puede tener muchas microarquitecturas diferentes, cada una con diferentes rendimiento, costo y complejidad. Todos las microarquitecturas de una determinada arquitectura ejecutan los mismos programas, pero sus diseños internos pueden variar ampliamente. En esta asignatura no abordaremos las microarquitecturas.

5. Jerarquía de memoria

Los dispositivos de almacenamiento en cada sistema informático están organizados utilizando una jerarquía de memoria similar a la Figura 11. A medida que avanzamos desde la parte superior de la jerarquía hacia la parte inferior, los dispositivos se vuelven más lentos, más grandes y menos costosos por byte. Los registros[4] ocupan el nivel superior en la jerarquía, que se conoce como nivel 0 o L0. En la figura se muestran tres niveles de almacenamiento en caché de L1 a L3, ocupando los niveles de jerarquía de memoria 1 a 3. La memoria principal ocupa el nivel 4, y así sucesivamente. La idea principal de una jerarquía de memoria es que el almacenamiento en un nivel sirve como caché para el almacenamiento en el siguiente nivel inferior. Por lo tanto, los registros son un caché para el caché L1. Los cachés L1 y L2 son cachés para L2 y L3, respectivamente. El caché L3 es un caché para la memoria principal, que a su vez es un caché para el disco. La caché L1 se divide en dos tipos: una de datos y otra de instrucciones.

[4]Un registro es una memoria de alta velocidad y poca capacidad, integrada en el microprocesador, que permite guardar transitoriamente y acceder a valores muy usados, generalmente en operaciones matemáticas. Los registros del procesador los veremos en detalles en los apuntes **Lenguaje Ensamblador y Arquitectura x86-64** y **Conceptos básicos del ensamblador y arquitectura ARM**.

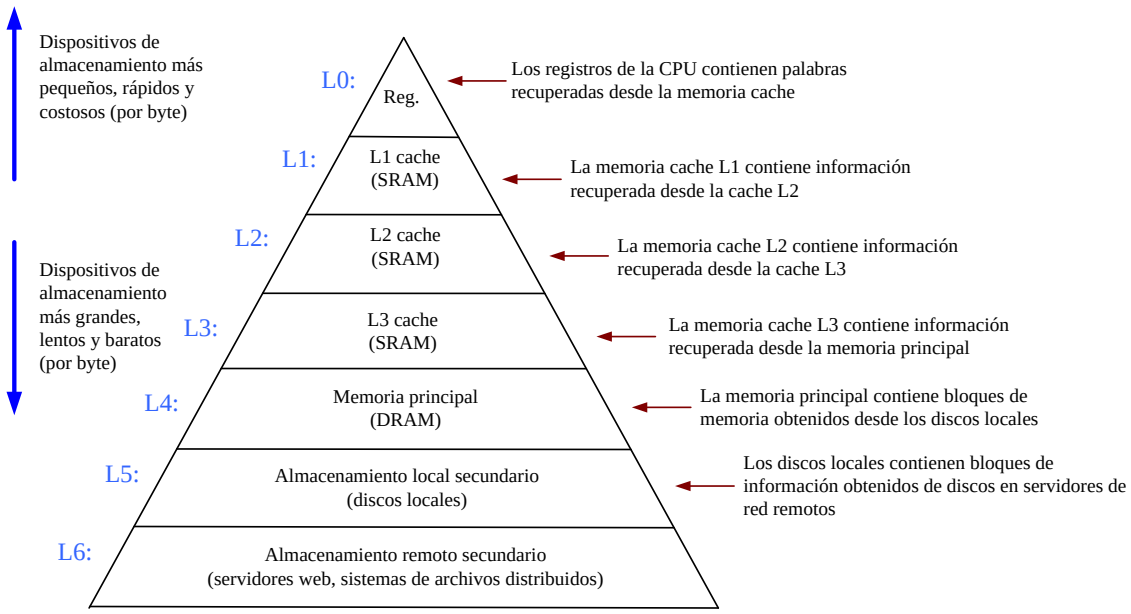


Figura 11: Ejemplo de jeraquía de memoria (fuente: [9]).

La tendencia actual es colocar los tres niveles de caché en el procesador y que cada nueva serie de procesadores tenga tamaños de caché más grandes

6. Ejecución de un programa

Para comprender qué sucede cuando ejecutamos un programa (vamos a ejemplificar con el típico `hello.c`), necesitamos comprender la organización del hardware de una computadora típica. Para ello en la Figura 12 vemos una versión un poco más detallada de la Figura 9. Esta imagen en particular sigue el modelo de la familia de microprocesadores Intel. Simplificando y omitiendo muchos detalles que se verán con más detalle más adelante, podemos describir el proceso de la siguiente manera. Inicialmente, el programa *shell*[5] está ejecutando sus instrucciones, esperando que escribamos un comando. A medida que escribimos los caracteres `./hello` en el teclado, el programa *shell* lee cada uno en un registro y luego lo almacena en la memoria. Cuando presionamos la tecla `Enter` en el teclado, el *shell* sabe que hemos terminado de escribir el comando. Luego, el *shell* carga el archivo ejecutable `hello` ejecutando una secuencia de instrucciones que copia el código y los datos del archivo objeto `hello` del disco a la memoria principal. Los datos incluyen la cadena de caracteres `"hola, mundo\n"` que finalmente se imprimirá. Usando una técnica conocida como acceso directo a la memoria (DMA, por la sigla en inglés *Direct Memory Access*), los datos viajan directamente desde el disco a la memoria principal, sin pasar por el procesador. Una vez que el código y los datos en el archivo de objeto `hello` se cargan en la memoria, el procesador comienza a ejecutar las instrucciones en lenguaje de máquina en la rutina principal del programa `hello`. Estas instrucciones copian los bytes de la cadena `"hola, mundo\n"` de la memoria a los registros de la CPU, y desde allí al dispositivo de visualización, donde se muestran en la pantalla.

Un punto importante a destacar es que una computadora pasa mucho tiempo moviendo información de un lugar a otro. Las instrucciones de máquina en el programa `hello` se almacenan originalmente en el disco. Cuando se carga el programa, se copian en la memoria principal. A medida que el procesador ejecuta el programa, las instrucciones se copian de la memoria principal al procesador. De manera similar, la cadena de datos `"hola, mundo \n"`, originalmente en el disco, se copia en la memoria principal y luego se copia desde la memoria

[5] Intérprete de órdenes o intérprete de comandos.

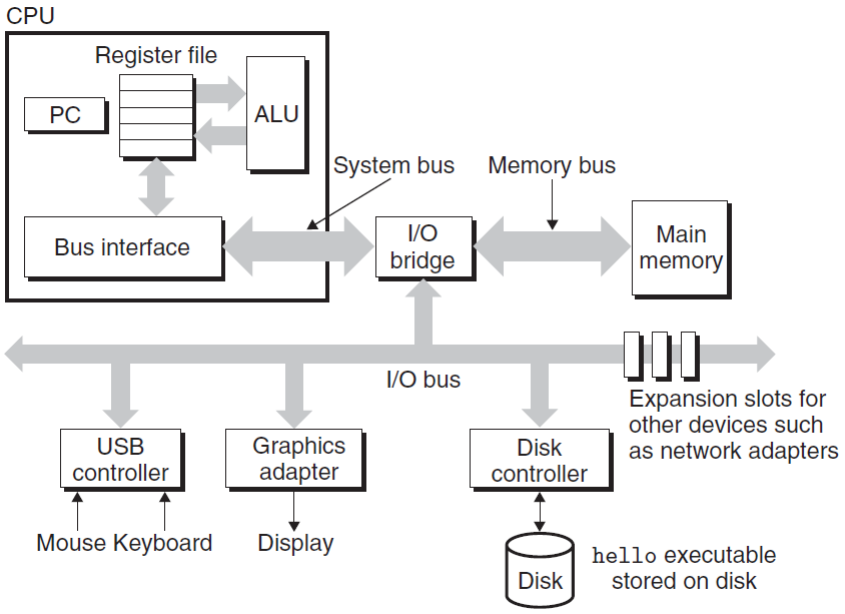


Figura 12: Organización de hardware de una computadora típica. CPU: Unidad central de procesamiento, ALU: Unidad aritmética/lógica, PC: Contador de programas, USB: Bus serie universal (fuente: [9]).

principal al dispositivo de visualización. Gran parte de estas operaciones de copia constituyen una sobrecarga que ralentiza el "trabajo real" del programa. Por lo tanto, un objetivo principal para los diseñadores de sistemas es hacer que estas operaciones de copia se ejecuten lo más rápido posible.

Debido a las leyes físicas, los dispositivos de almacenamiento más grandes son más lentos que los dispositivos de almacenamiento más pequeños. Además, los dispositivos más rápidos son más caros de construir que sus homólogos más lentos. Por ejemplo, la unidad de disco en un sistema típico puede ser 1.000 veces más grande que la memoria principal, pero el procesador puede tardar 10.000.000 veces más en leer una palabra del disco que de la memoria. De manera similar, los registros de la CPU almacenan solo unos pocos bytes de información, a diferencia de miles de millones de bytes en la memoria principal. Sin embargo, el procesador puede leer datos de los registros casi 100 veces más rápido que de la memoria. Aún más problemático, a medida que la tecnología de semiconductores progresa a lo largo de los años, esta brecha en la memoria del procesador continúa aumentando. Es más fácil y económico hacer que los procesadores funcionen más rápido que hacer que la memoria principal funcione más rápido.

Para lidiar con la brecha de memoria del procesador, los diseñadores del sistema incluyen dispositivos de almacenamiento más pequeños y más rápidos llamados memorias de caché (o simplemente cachés) que sirven como áreas de almacenamiento temporales para la información que el procesador probablemente necesitará en el futuro cercano. Los cachés se implementan con una tecnología de hardware conocida como memoria estática de acceso aleatorio (SRAM, *Static random-access memory*). Los sistemas más nuevos y potentes incluso tienen tres niveles de caché: L1, L2 y L3. La idea detrás del almacenamiento en caché es que un sistema puede obtener el efecto de una memoria muy grande y muy rápida explotando la localidad, la tendencia de los programas a acceder a datos y códigos en regiones localizadas.

7. Representación de programas a nivel de máquina

Las computadoras ejecutan código de máquina, secuencias de bytes que codifican las operaciones de bajo nivel que manipulan datos, administran la memoria, leen y escriben datos en

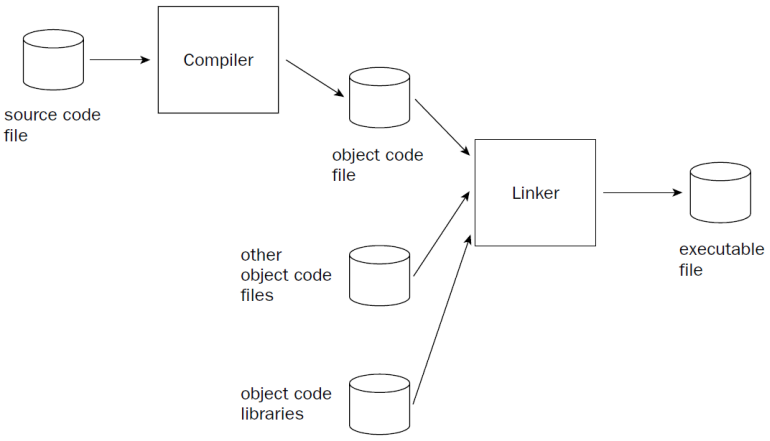


Figura 13: Proceso de compilación (fuente: [10]).

dispositivos de almacenamiento y se comunican a través de redes. Un compilador genera código de máquina a través de una serie de etapas, basadas en las reglas del lenguaje de programación, el conjunto de instrucciones de la máquina de destino y las convenciones seguidas por el sistema operativo. Por ejemplo, el compilador *gcc* genera su salida en forma de código Assembler[6], una representación textual del código de máquina que proporciona las instrucciones individuales en el programa. Luego, *gcc* invoca tanto un ensamblador como un enlazador para generar el código de máquina ejecutable a partir del código Assembler. La Figura 13 ilustra este proceso.

Es interesante analizar el código de máquina y sobre todo su representación legible por humanos como es el código Assembler.

Ejemplo

Un código equivalente del clásico programa en lenguaje C que imprime por pantalla la frase `¡Hola Mundo!`*:*

```
#include<stdio.h>
int main()
{
  printf("¡Hola Mundo!\n");
  return 0;
}
```

es el siguiente:

```
0x0000000000401122 <+0>:    48 c7 c7 30 40 40 00    movq  $0x404030, %rdi
0x0000000000401129 <+7>:    48 c7 c0 00 00 00 00    movq  $0x0, %rax
0x0000000000401130 <+14>:   e8 fb fe ff ff          callq 0x401030 <printf@plt>
0x0000000000401135 <+19>:   c3                      retq
```

En el código precedente podemos visualizar a la izquierda las direcciones de memoria de las intrucciones, luego en el centro el equivalente en lenguaje de máquina escrito en formato hexadecimal mientras que en la parte derecha vemos el equivalente en lenguaje Assembler. Si bien todavía no sabemos Assembler, es evidente que esta representación es mucho más legible y comprensible para un humano que el código de máquina puro.

[6] Assembler, o Ensamblador en castellano, es un tema que veremos en detalle en el apunte 3 *Ensamblador y Arquitectura x86-64*

El lenguaje ensamblador es específico de la máquina. Por ejemplo, el código escrito para un procesador x86-64 no se ejecutará en un procesador diferente, por ejemplo en un procesador RISC (popular en tabletas y teléfonos inteligentes). El lenguaje ensamblador es un lenguaje de "bajo nivel" y proporciona la interfaz de instrucción básica para el procesador de la computadora. Para un programador, el lenguaje ensamblador es lo más cercano al procesador. Los programas escritos en un lenguaje de alto nivel se traducen al lenguaje ensamblador para que el procesador ejecute el programa. El lenguaje de alto nivel es una abstracción entre el lenguaje y las instrucciones reales del procesador. El lenguaje ensamblador le brinda al programador control directo de los recursos del sistema. Esto implica establecer registros del procesador, acceder a ubicaciones de memoria e interactuar con otros elementos de hardware. Esto requiere una comprensión significativamente profunda de cómo funcionan exactamente el procesador y la memoria.

Al programar en un lenguaje de alto nivel como C, no necesitamos preocuparnos por la implementación detallada de nuestro programa a nivel de máquina. Por el contrario, al escribir programas en código Assembler (como se hizo en los primeros días de la informática), un programador debe especificar las instrucciones de bajo nivel que el programa utiliza para llevar a cabo un cálculo. La mayoría de las veces, es mucho más productivo y confiable trabajar en el nivel más alto de abstracción proporcionado por un lenguaje de alto nivel. La verificación de tipo proporcionada por un compilador ayuda a detectar muchos errores de programa y asegura que hagamos referencia y manipulemos los datos de manera consistente. Con los compiladores y optimizadores modernos, el código generado suele ser al menos tan eficiente como lo que escribiría a mano un programador experto en lenguaje ensamblador. Lo mejor de todo es que un programa escrito en un lenguaje de alto nivel se puede compilar y ejecutar en varias máquinas diferentes, mientras que el código de ensamblaje es altamente específico de la máquina.

Entonces, ¿por qué deberíamos aprender Assembler? El lenguaje ensamblador tiene varios beneficios:

- Velocidad: Los programas en lenguaje ensamblador son generalmente los programas más rápidos.
- Espacio: Los programas en lenguaje ensamblador suelen ser los más pequeños.
- Capacidad: Se pueden hacer cosas en lenguaje ensamblador que son difíciles o imposibles en HLL[7].
- Conocimiento: El conocimiento del lenguaje ensamblador ayuda a escribir mejores programas, incluso cuando use HLL.

Aunque los compiladores hacen la mayor parte del trabajo en la generación de código Assembler, poder leerlo y comprenderlo es una habilidad importante para los programadores avanzados. Al invocar al compilador con los parámetros de línea de comandos apropiados, el compilador generará un archivo que muestra su salida en forma de código de ensamblaje. Al leer este código, podemos entender las capacidades de optimización del compilador y analizar las ineficiencias subyacentes en el código. Los programadores que buscan maximizar el rendimiento de una sección crítica de código a menudo prueban diferentes variaciones del código fuente, cada vez que compilan y examinan el código de ensamblaje generado para tener una idea de qué tan eficientemente se ejecutará el programa. Además, hay momentos en que la capa de abstracción proporcionada por un lenguaje de alto nivel oculta información sobre el comportamiento en tiempo de ejecución de un programa que debemos comprender. Por ejemplo, cuando se escriben programas concurrentes utilizando un paquete de subprocesos, es importante saber qué región de la memoria se usa para contener las diferentes variables del programa. Esta

[7]HLL, programas de alto nivel por su sigla en inglés *high level languages*.

información es visible a nivel de código Assembler. Como otro ejemplo, muchas de las formas en que se pueden atacar los programas, permitiendo que gusanos y virus infesten un sistema, implican matices de la forma en que los programas almacenan la información en tiempo de ejecución. Muchos ataques implican la explotación de las debilidades en los programas del sistema para sobrescribir la información y, por lo tanto, tomar el control del sistema. Comprender cómo surgen estas vulnerabilidades y cómo protegerse de ellas requiere un conocimiento de la representación de los programas a nivel de máquina. La necesidad de que los programadores aprendan código Assembler ha cambiado con el paso de los años de poder escribir programas directamente Assembler a ser capaz de leer y comprender el código generado por los compiladores. Finalmente, podríamos concluir diciendo que la razón principal para aprender el lenguaje ensamblador radica más en entender cómo funciona una computadora en lugar de desarrollar programas grandes.

Referencias

[1] Andrew S Tanenbaum. *Organización de computadoras: un enfoque estructurado*. Pearson educación, 2000.

[2] Wikipedia. Válvula termoiónica — Wikipedia, La enciclopedia libre. `https://es.wikipedia.org/w/index.php?title=V%C3%A1lvula_termoi%C3%B3nica&oldid=121814534`, 2019. [Internet; descargado 27-enero-2020].

[3] Historia de la Informática. Proyecto ENIAC, 2011. [Internet; descargado 31-enero-2022].

[4] Wikipedia. Transistor — Wikipedia, La enciclopedia libre. `https://es.wikipedia.org/w/index.php?title=Transistor&oldid=122923620`, 2020. [Internet; descargado 27-enero-2020].

[5] Albert Paul Malvino and David J. Batesr. *Principios de Electrónica*. Mcgraw-Hill, 2007.

[6] Wikipedia. IBM 1401 — Wikipedia, La enciclopedia libre, 2021. [Internet; descargado 31-enero-2022].

[7] Wikipedia. Circuito integrado — Wikipedia, La enciclopedia libre. `https://es.wikipedia.org/w/index.php?title=Circuito_integrado&oldid=122233649`, 2019. [Internet; descargado 27-enero-2020].

[8] Wikipedia. Integración a muy gran escala — Wikipedia, La enciclopedia libre. `https://es.wikipedia.org/w/index.php?title=Integraci%C3%B3n_a_muy_gran_escala&oldid=117810308`, 2019. [Internet; descargado 27-enero-2020].

[9] Randal E Bryant and David Richard O'Hallaron. *Computer systems: a programmer's perspective*, volume 281. Prentice Hall Upper Saddle River, second edition, 2003.

[10] Richard Blum. *Professional assembly language*. John Wiley & Sons, 2007.

[11] Ed Jorgenson. *X86-64 Assembly Language Programming with Ubuntu*. 2019.

[12] John L Hennessy and David A Patterson. *Computer architecture: A quantitative approach*. Elsevier, 2011.

[13] Randall Hyde. *The art of assembly language*. No Starch Press, 2003.

[14] Igor Zhirkov. *Low-Level Programming: C, Assembly, and Program Execution on Intel® 64 Architecture*. Apress, 2017.

[15] Sarah L Harris and David Harris. *Digital design and computer architecture*. Morgan Kaufmann, 2015.