Introducción a Arquitectura de las Computadoras

Diego Feroldi

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

*Actualizado 16 de agosto de 2024 (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".*
>
> **Andrew S. Tanenbaum**
> Organización de computadoras: un enfoque estructurado, 2000.

La anterior es solo una de las muchas definiciones posibles de lo que es una computadora. Esta definición sugiere que, a pesar de las enormes prestaciones de las computadoras actuales, en su nivel más elemental, estas máquinas resuelven operaciones muy simples. De hecho, es importante destacar que los circuitos integrados de una computadora solo pueden reconocer y ejecutar un conjunto muy limitado de operaciones básicas. A lo largo de esta asignatura, exploraremos cómo se realizan estas operaciones.

En primer lugar, examinaremos cómo están constituidas las computadoras. Además, haremos un breve repaso de su evolución histórica, explorando cómo han avanzado desde sus orígenes hasta convertirse en las potentes herramientas que utilizamos hoy en día.

2. Definición de Arquitectura del Computador

La **arquitectura del computador** se refiere a la organización de los componentes que conforman un sistema informático y las operaciones que guían su funcionamiento. Los principales objetivos en el diseño de la arquitectura de una computadora son:

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

Estos objetivos pueden ser contradictorios y, en general, se debe adoptar una solución de compromiso. El objetivo de esta asignatura no es diseñar una computadora, sino entender su funcionamiento desde la perspectiva de un programador. En este sentido, nos centraremos principalmente en el conjunto de instrucciones, la interacción con la memoria, la representación de los datos, entre otros aspectos.

Por lo tanto, las preguntas clave al abordar una nueva arquitectura son:

- ¿Cuál es la longitud de la palabra de datos?
- ¿Cuáles son y cuántos registros existen?
- ¿Cómo está organizada la memoria?
- ¿Cuáles son las instrucciones disponibles?

3. 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 (disco duro, disco de estado

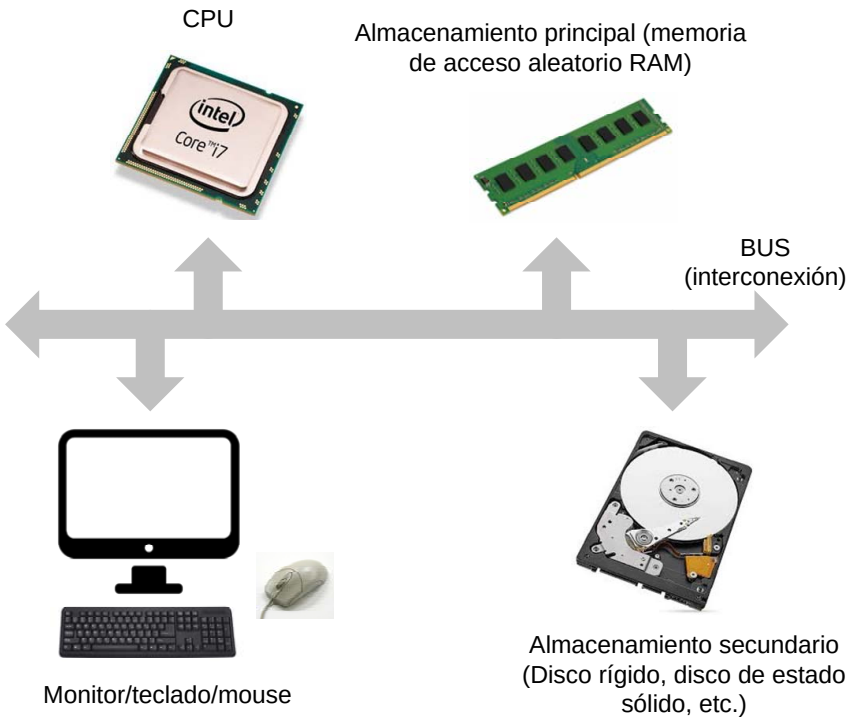


Figura 1: Componentes básicos de la arquitectura de una computadora.

sólido, memoria USB, etc.), dispositivos de entrada/salida (pantalla, teclado, mouse, etc.) y una interconexión entre los diferentes componentes denominada buses. Un diagrama muy básico de la arquitectura de una computadora se muestra en la Figura 1.

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.

3.1. Arquitectura de von Neumann

En 1946, von Neumann y sus colegas comenzaron el diseño de una nueva computadora, conocida como la computadora IAS, en el Instituto de Estudios Avanzados de Princeton. Aunque la computadora IAS no se completó hasta 1952, es el prototipo de todas las computadoras de propósito general posteriores. Un enfoque de diseño fundamental implementado por primera vez en la computadora IAS es conocido como el concepto de *programa almacenado*.

La Figura 2 muestra la estructura de la computadora IAS. Consiste en:

- Una memoria principal, que almacena tanto datos como instrucciones.
- Una unidad aritmética-lógica (ALU) capaz de operar con datos binarios.
- Una unidad de control, que interpreta las instrucciones en la memoria y las ejecuta.
- Equipos de entrada/salida (I/O) operados por la unidad de control.

Los programas y los datos 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 (o memoria principal), realizando repetitivamente un ciclo de instrucciones. Con raras excepciones, todas las computadoras actuales tienen esta misma estructura y función general, por lo que se las conoce como *arquitectura de von Neumann*.

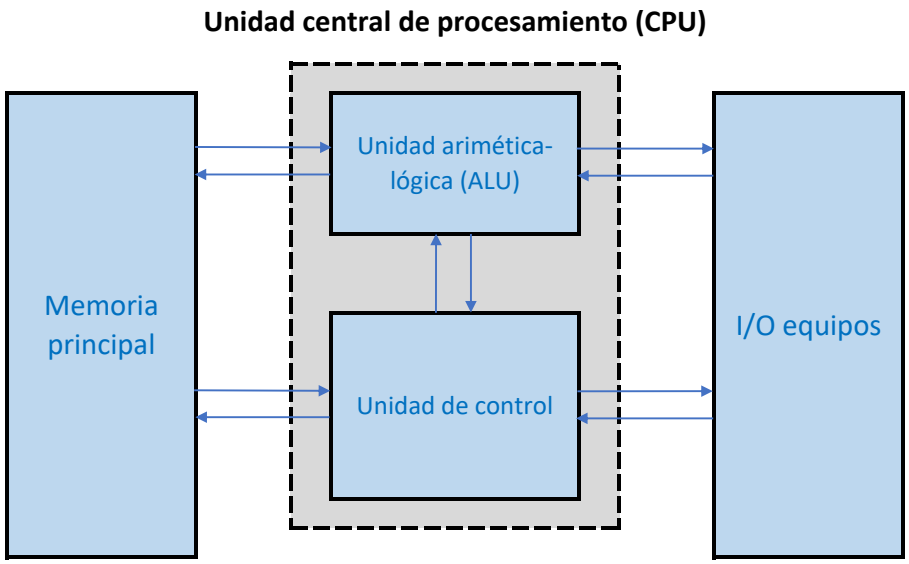


Figura 2: Estructura de la computadora IAS.

3.2. Unidad Central de Procesamiento (CPU)

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).

3.2.1. 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).

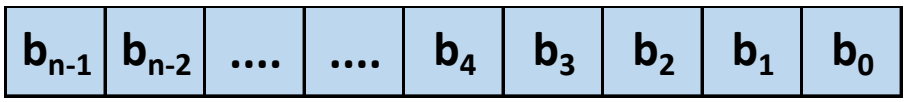


Figura 3: Esquema de un registro de n bits.

3.2.2. Registros

Para que la Unidad Aritmética Lógica (ALU) pueda operar, el chip también contiene registros y memoria caché. Un registro de la CPU, o simplemente registro, es un almacenamiento temporal o una ubicación de trabajo integrada en la propia CPU, separada de la memoria principal. 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 obtenerse de la memoria, los operandos también deben obtenerse de la memoria, y algunas instrucciones almacenan resultados en la memoria.

Este proceso puede generar un cuello de botella debido a la diferencia de velocidades entre la CPU y la memoria, lo que provoca que la CPU pierda tiempo mientras espera la respuesta de datos del chip de memoria. Para evitar esta espera constante, el procesador está equipado con sus propias celdas de memoria, llamadas registros. Aunque la cantidad de registros es limitada, son extremadamente rápidos, lo que mejora significativamente el rendimiento del procesamiento.

Las principales características de los registros son las siguientes:

- **Velocidad**: Son extremadamente rápidos en comparación con la memoria principal (RAM), ya que están ubicados dentro del propio procesador.
- **Capacidad**: Tienen una capacidad limitada, generalmente de unos pocos bits o bytes (por ejemplo, 8, 16, 32 o 64 bits dependiendo de la arquitectura).
- **Función**: Se utilizan para almacenar datos temporales como resultados de operaciones aritméticas, direcciones de memoria, o valores intermedios durante la ejecución de programas.

En la Figura 3 se presenta un diagrama esquemático de un registro compuesto por n celdas de memoria. Cada celda de memoria está formada por un circuito electrónico biestable, comúnmente conocido como *flip-flop*, el cual tiene la capacidad de permanecer en uno de dos estados posibles, permitiendo así el almacenamiento de un bit de información[1].

3.2.3. Memoria Caché

La memoria caché es un pequeño subconjunto del almacenamiento primario que se encuentra integrado en el chip de la CPU. Está diseñada para combinar la alta velocidad de acceso de los registros de la CPU, que son costosos y rápidos, con el gran tamaño de la memoria principal, que es menos costosa pero más lenta. De este modo, se establece una jerarquía en la que la memoria principal es relativamente grande y lenta, mientras que la memoria caché es más pequeña pero mucho más rápida.

La memoria caché contiene copias de porciones de la memoria principal. Cuando el procesador intenta leer una palabra de memoria, primero verifica si esa palabra está en la caché. Si es así, la palabra se entrega al procesador de inmediato. Si no está en la caché, se lee un bloque de la memoria principal, que consta de un número fijo de palabras, y se carga en la caché antes de entregar la palabra solicitada al procesador. Este proceso se beneficia del fenómeno de localidad, que sugiere que cuando se carga un bloque de datos en la caché para satisfacer

[1]El concepto de bit se presentará en detalle en el Apunte **Representación Computacional de Datos.**

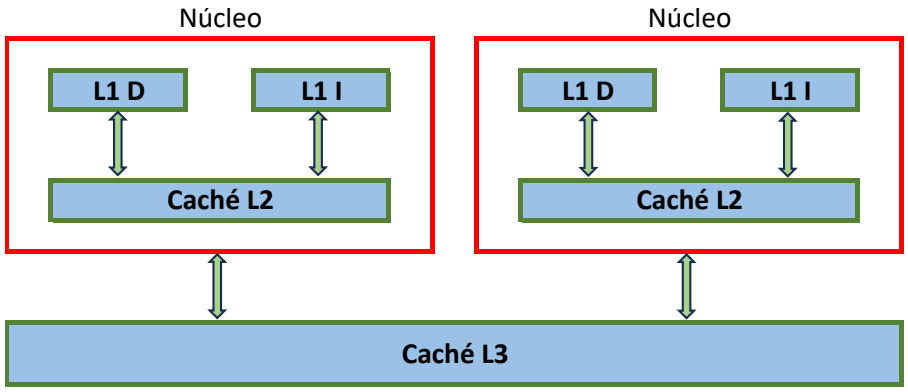


Figura 4: Esquema jerárquico de la memoria caché en una CPU de dos núcleos.

una referencia, es probable que se realicen referencias futuras a esa misma ubicación o a otras palabras dentro del mismo bloque[2].

La memoria caché tiene su propia jerarquía, que se organiza en tres niveles:

- **Caché L1 (Nivel 1)**: Es la memoria caché más cercana al núcleo del procesador y, por lo tanto, la más rápida. Suele estar dividida en dos partes: una para instrucciones y otra para datos. Debido a su proximidad y alta velocidad, la caché L1 es relativamente pequeña, normalmente entre 16 KB y 64 KB por núcleo.
- **Caché L2 (Nivel 2)**: Se encuentra un nivel por debajo de la caché L1 y es más grande, generalmente entre 256 KB y 1 MB por núcleo. Aunque es más lenta que la L1, sigue siendo mucho más rápida que la memoria principal (RAM). La caché L2 puede estar compartida entre varios núcleos o ser exclusiva para cada uno, dependiendo del diseño del procesador.
- **Caché L3 (Nivel 3)**: Es el nivel más grande y lento de la jerarquía de caché. Suele estar compartida por todos los núcleos del procesador y su tamaño varía, pudiendo llegar a varios megabytes (de 4 MB a 64 MB o más). Aunque es más lenta que las cachés L1 y L2, la L3 sigue siendo más rápida que acceder a la RAM, por lo que juega un papel crucial en la mejora del rendimiento del sistema.

Cada nivel de caché actúa como un intermediario para el siguiente, con el objetivo de reducir los tiempos de acceso a los datos y mejorar la eficiencia general del procesamiento. La Figura 4 muestra esquemáticamente la jerarquía de una memoria caché típica en una CPU de dos núcleos.

3.2.4. Contador de programa

El contador de programa (conocido en inglés como *Program Counter* o PC) es un registro en la CPU que contiene la dirección de memoria de la siguiente instrucción que debe ser ejecutada. Es uno de los registros más importantes en la arquitectura del computador, ya que controla la secuencia de ejecución de las instrucciones. En todo momento, el PC apunta a una instrucción en lenguaje máquina almacenada en la memoria principal. Desde que el sistema se enciende hasta que se apaga, el procesador ejecuta repetidamente la instrucción señalada por el contador de programa y actualiza el PC para que apunte a la siguiente instrucción.

[2]Los registros de la CPU y la memoria caché se describirán con mayor detalle en la Sección 5 y otros apuntes. Sin embargo, cabe señalar que el diseño interno y la configuración de un procesador moderno son sumamente complejos, y esta asignatura proporciona un enfoque de alto nivel muy simplificado de algunas unidades funcionales clave dentro de una CPU.

El funcionamiento de un procesador se puede interpretar mediante un modelo de ejecución de instrucciones muy simple. En este modelo, las instrucciones se ejecutan en secuencia estricta, y la ejecución de una sola instrucción implica una serie de pasos: el procesador lee la instrucción de la memoria a la que apunta el contador de programa (PC), interpreta los bits de la instrucción, realiza la operación simple dictada por la instrucción y luego actualiza el PC para que apunte a la siguiente instrucción. Esta siguiente instrucción puede estar o no contigua en memoria a la instrucción que acaba de ejecutarse.

3.3. Memoria principal

La memoria principal es un dispositivo de almacenamiento temporal que contiene tanto el programa como los datos que se están manipulando mientras el procesador ejecuta el programa. Esta característica es fundamental en las computadoras basadas en el concepto de *Máquina de Von Neumann*. Físicamente, la memoria principal está constituida por una colección de chips de memoria dinámica de acceso aleatorio (DRAM, por sus siglas 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 (dirección de memoria).

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.

3.4. Memoria secundaria

El principal problema de las memorias RAM es su volatilidad. Esto significa que la memoria RAM pierde toda la información almacenada cuando la computadora se apaga o se reinicia. A diferencia de los dispositivos de almacenamiento no volátiles, como los discos duros o las SSD, que conservan los datos incluso cuando la energía se desconecta, la memoria RAM solo retiene los datos mientras está alimentada. Esta característica limita su uso a tareas que requieren acceso rápido a datos temporales durante el funcionamiento de la computadora, pero no es adecuada para el almacenamiento permanente de información.

Un disco duro es un dispositivo de almacenamiento de datos magnético que se utiliza en las computadoras para guardar permanentemente archivos y programas. Una de las principales diferencias entre el disco duro y la memoria RAM es la velocidad. La memoria RAM proporciona un acceso rápido y aleatorio a los datos y programas, lo que permite al procesador acceder a ellos de manera eficiente. En comparación, los discos duros son significativamente más lentos en términos de velocidad de acceso, ya que el brazo de lectura/escritura debe moverse físicamente para acceder a los datos almacenados en los platos giratorios.

Un SSD (Solid State Drive) es un dispositivo de almacenamiento de datos que utiliza memoria flash para guardar permanentemente archivos y programas en la computadora. A diferencia de los discos duros tradicionales, que emplean platos magnéticos giratorios y cabezales de lectura/escritura, una SSD no tiene partes móviles y utiliza chips de memoria flash para almacenar y acceder a los datos.

La tecnología SSD ofrece un rendimiento superior en comparación con los discos duros debido a la ausencia de partes mecánicas móviles. Los datos se almacenan en chips de memoria flash, que permiten un acceso y lectura mucho más rápidos. Como resultado, las SSD proporcionan un mejor rendimiento en términos de velocidad de lectura/escritura y tiempo de acceso.

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

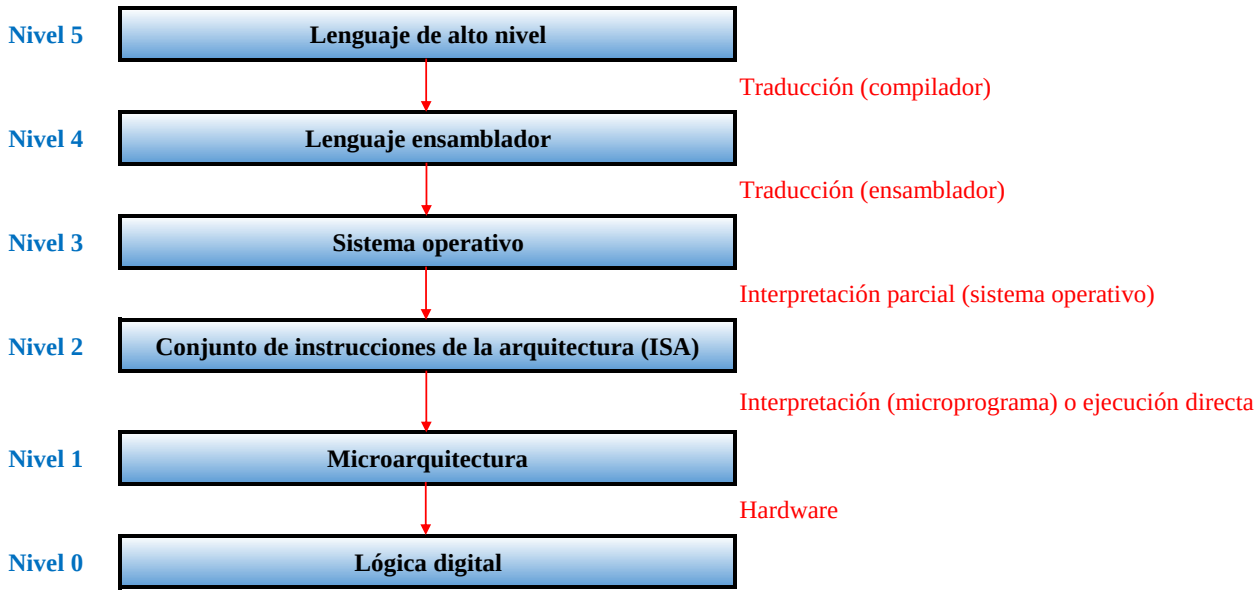


Figura 5: Modelo de máquina multinivel.

3.5. 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.

3.6. Dispositivos de entrada/salida

Los dispositivos de entrada/salida (I/O *devices*) son la conexión del sistema con el mundo externo. La computadora del ejemplo en la Figura 1 tiene cuatro dispositivos de E/S: un teclado y un mouse para la entrada del usuario, una pantalla para la salida, y una unidad de disco (o 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 integrados en el propio dispositivo o en la placa de circuito impreso principal del sistema, conocida comúnmente como placa base. Un adaptador, en cambio, 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.

3.7. 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 5 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*.

3.8. Microarquitectura

El nivel 1 del modelo de máquina multinivel mostrado en la Figura 5 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.

4. Breve historia de la computación

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

4.1. 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

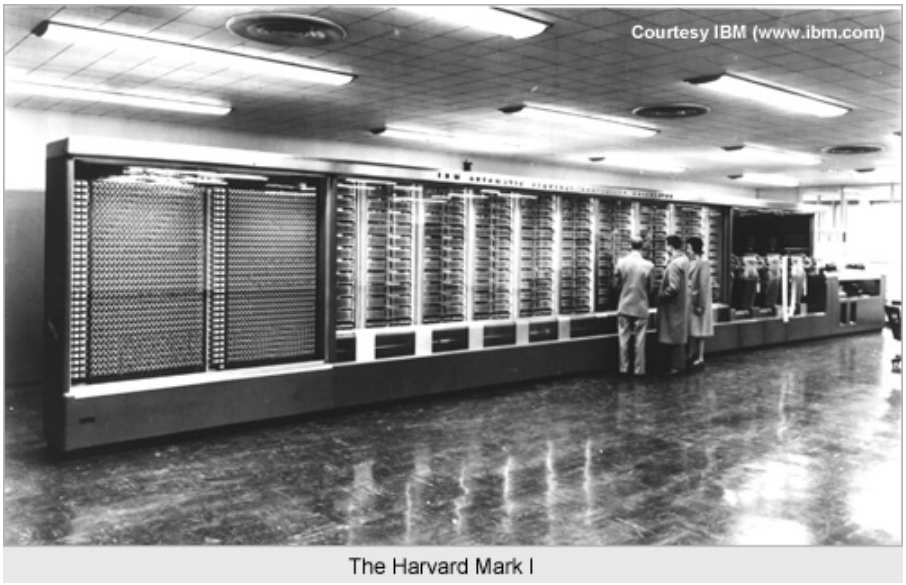


Figura 6: Imagen de la computadora Mark I (fuente:[2]).

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[4].
- 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.
- 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 6). Para cuando se terminó de desarrollar la Mark II las máquinas de relés ya eran obsoletas.

4.2. 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

[4]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.

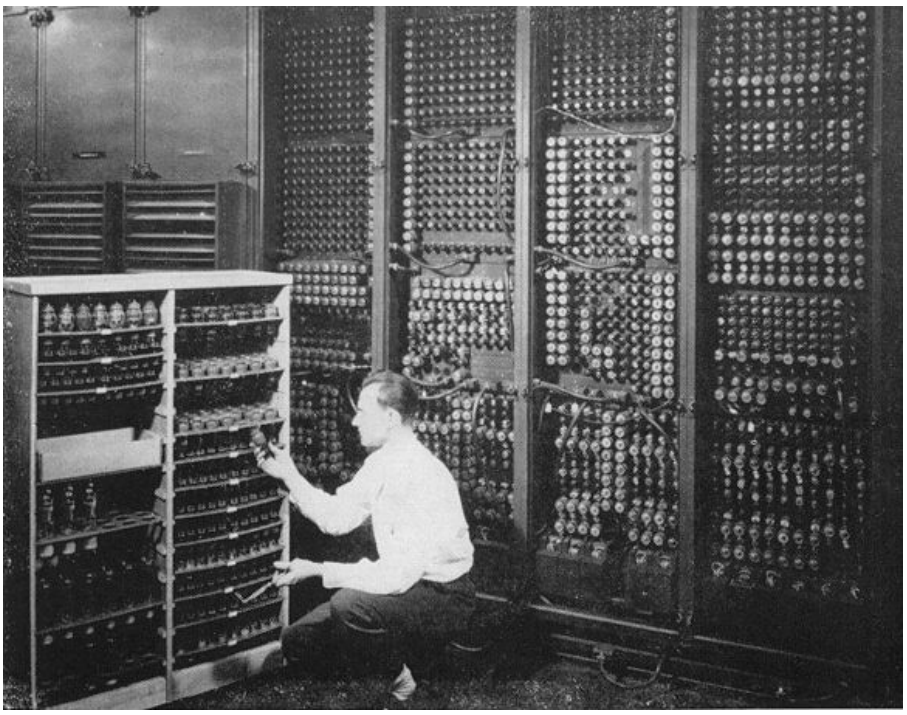


Figura 7: Imagen de la computadora ENIAC (fuente: [4]).

analógicas y digitales, control industrial, etc. Algunas de estas aplicaciones son anteriores a la válvula, pero experimentaron un crecimiento explosivo gracias a ella [3].

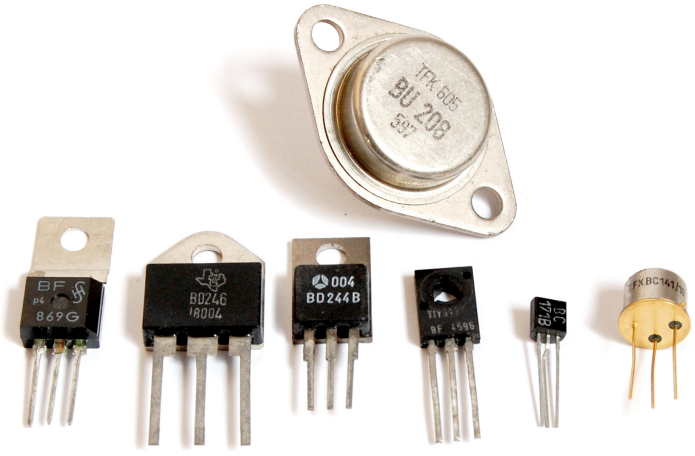
A continuación se enumeran algunos de los avances más notables de la computación utilizando esta tecnología:

- Alan Turing (1912-1954). Contribuyó a la creación de COLOSSUS, la primera computadora electrónica, desarrollada por el gobierno inglés.
- John Mauchley (1907-1980). Construyó la ENIAC (Electronic Numerical Integrator And Computer) en 1946, la primera computadora digital, electrónica, de propósito general y programable (Ver Figura 7). Tenía 18000 válvulas y 1500 relés. Pesaba 30 toneladas y consumía 140 kW. Trabajaba en sistema decimal. En la Figura 8 se puede apreciar una válvula de vacío.
- EDSAC (Electronic Delay Storage Automatic Calculator). Máquina sucesora de ENIAC en 1949. Trabajaba en sistema binario.
- EDVAC (Electronic Discrete Variable Automatic Computer). Fue una de las primeras computadoras electrónicas (1951). A diferencia de la ENIAC, era binaria en lugar de decimal y fue diseñada para ser una computadora de programa almacenado. El diseño de la EDVAC está basado en el concepto de Máquina de Turing.
- 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.
- Whirlwind I (1951). Diseñada en el MIT. Tenía palabras de 16 bits y estaba diseñada para el control en tiempo real.

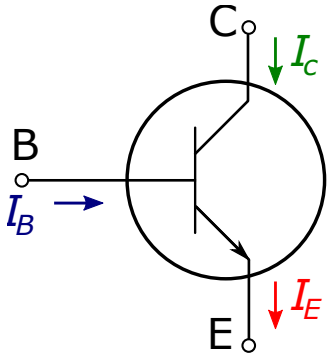
Figura 8: Imagen de una válvula electrónica (fuente: [3]).

4.3. 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 [5]. 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 9(a) se pueden apreciar diferentes tipos de transistores mientras que en la Figura 9(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 9: Transistores (fuente: [5]).

El impacto del transistor ha sido enorme, ya que, además de iniciar la industria de los semiconductores, ha sido el precursor de otros inventos como los circuitos integrados, los dis-

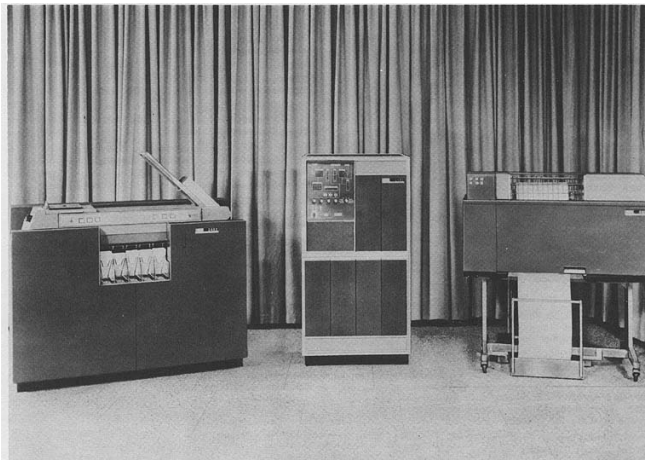


Figura 10: Imagen de la computadora IBM 1401 (fuente: [7]).

positivos optoelectrónicos y los microprocesadores. Sin embargo, los cambios más notables se han producido en el campo de la computación. Según Albert P. Malvino [6], *"el transistor no se creó para mejorar la computación, sino que fue él quien la creó"*.

Antes de 1950, una computadora ocupaba toda una sala y costaba millones de dólares. Sin embargo, con la incorporación del transistor, fue posible reducir notablemente tanto el tamaño como el costo de estas máquinas. Hoy en día, una computadora cabe en un bolsillo, tiene un costo muy bajo y ofrece 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 10).
- 7094 (1962). Desarrollada por IBM y orientada a la computación científica.
- 6600 (1964). Desarrollada por CDC. Primera supercomputadora científica.

4.4. 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 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 [8].

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

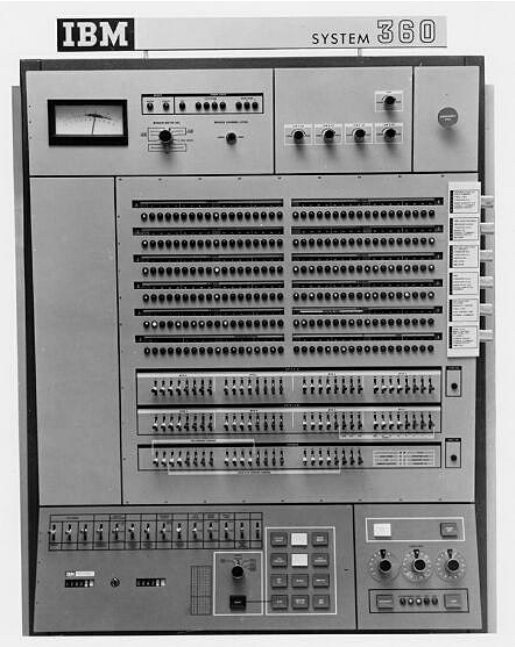


Figura 11: Imagen de la computadora IBM 360 (fuente: [9]).

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 11).
- PDP-8 (1965). Primera minicomputadora con mercado masivo (50000 unidades vendidas).
- PDP-11 (1970). Dominó el mercado de las minicomputadoras en los años setenta.

4.5. 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 [10].

- 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 12 se puede apreciar el microprocesador Intel 8080:

Figura 12: Microprocesador Intel 8080 (fuente: [11]).

- Apple II (1977). Primera serie de microcomputadores de producción masiva hecha por la empresa Apple Computer (Ver Figura 13). Arquitectura de 8 bits.

Figura 13: Computadora Apple II (fuente: [12]).

- 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.
- 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 historia. 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. Registros de 16 bits. 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

5. Jerarquía de memoria

> *"Idealmente, se desearía una capacidad de memoria indefinidamente grande de manera que cualquier palabra en particular estuviera disponible de inmediato. Sin embargo, nos vemos obligados a reconocer la posibilidad de construir una jerarquía de memorias, cada una de las cuales tiene una capacidad mayor que la anterior, pero que es menos rápidamente accesible."*
>
> **A. W. Burks, H. H. Goldstine, y J. von Neumann**
> Discusión preliminar sobre el diseño lógico de un instrumento de cálculo electrónico, 1946.

Los dispositivos de almacenamiento en cada sistema informático están organizados utilizando una jerarquía de memoria, como se ilustra en la Figura 14. A medida que descendemos desde la parte superior de la jerarquía hacia la inferior, los dispositivos se vuelven más lentos, más grandes y menos costosos por byte. En contraste, al acercarnos a la parte superior de la pirámide, encontramos dispositivos más rápidos y costosos.

Los registros[5] ocupan el nivel superior en la jerarquía, conocido como nivel 0 o L0. Luego, en la figura se muestran tres niveles de caché, de L1 a L3, que corresponden a los niveles de jerarquía de memoria del 1 al 3. La memoria principal ocupa el nivel 4, y los niveles inferiores siguen en orden descendente.

La idea principal de la jerarquía de memoria es que el almacenamiento en un nivel actúa como caché para el almacenamiento en el nivel inferior. Así, los registros sirven como caché para

[5]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 se verán en detalle en los apuntes **Lenguaje Ensamblador y Arquitectura x86-64** y **Conceptos básicos del ensamblador y arquitectura ARM**.

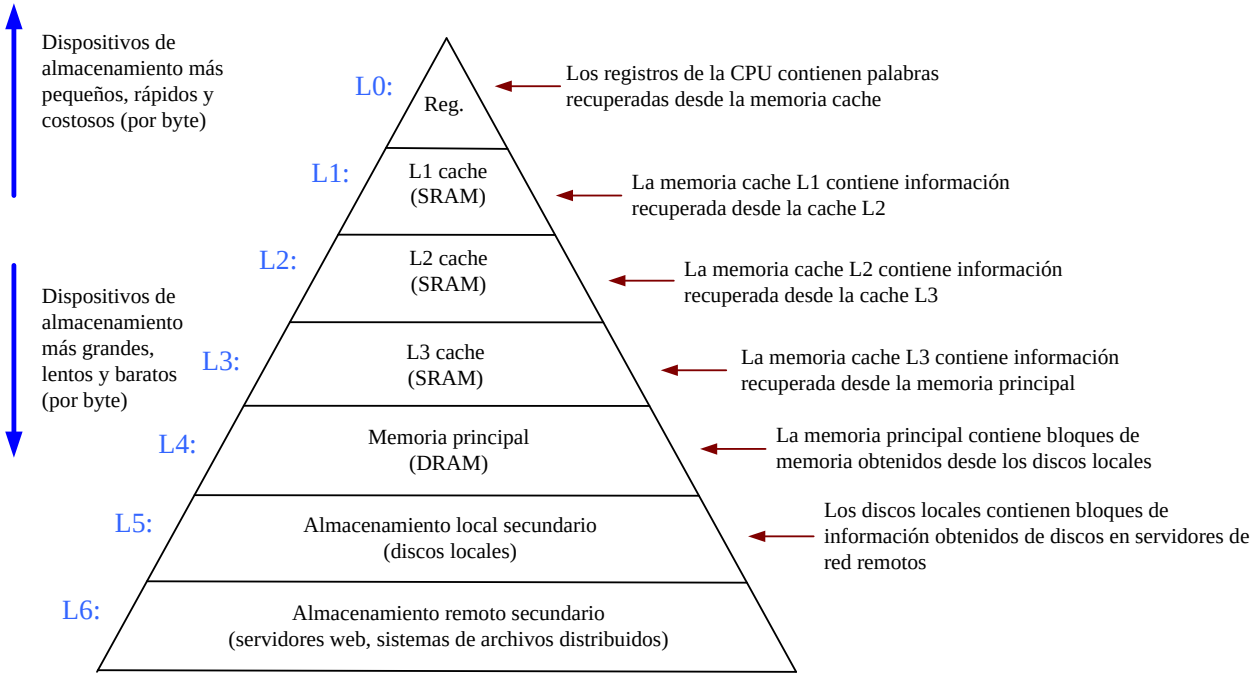


Figura 14: Ejemplo de jeraquía de memoria.

el caché L1. Los cachés L1 y L2 funcionan como cachés para L2 y L3, respectivamente. El caché L3 actúa como caché para la memoria principal, que a su vez sirve como caché para el disco. La caché L1 se divide en dos tipos: una para datos y otra para instrucciones. La tendencia actual es integrar los tres niveles de caché en el procesador, y cada nueva generación de procesadores tiende a tener tamaños de caché más grandes.

6. 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 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 15 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 posible código equivalente del clásico programa en lenguaje C

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

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

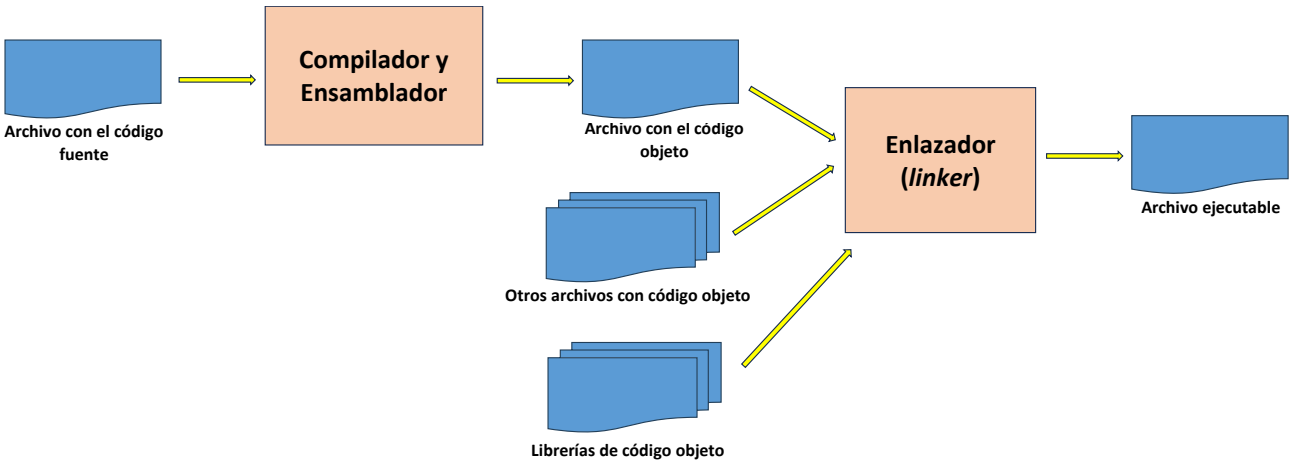


Figura 15: Proceso de compilación.

```
  return 0;
}
```

que imprime en pantalla la frase `¡Hola Mundo!` *es el siguiente:*

```
.data
msg: .asciz "!Hola mundo!\n"

.text
.global main
main:
0x0000000000401126 <+0>:     55                          push   %rbp
0x0000000000401127 <+1>:     48 c7 c7 30 40 40 00        movq   $msg, %rdi
0x000000000040112e <+8>:     48 c7 c0 00 00 00 00        movq   $0x0,%rax
0x0000000000401135 <+15>:    e8 f6 fe ff ff              call   printf
0x000000000040113a <+20>:    31 c0                       xorl   %eax, %eax
0x000000000040113c <+22>:    5d                          pop    %rbp
0x000000000040113d <+23>:    c3                          ret
```

En el código anterior, podemos observar a la izquierda las direcciones de memoria de las instrucciones, en el centro su equivalente en lenguaje de máquina escrito en formato hexadecimal, y a la derecha el equivalente en lenguaje ensamblador (Assembler). Aunque aún no dominamos el lenguaje ensamblador, es evidente que esta representación es mucho más legible y comprensible para un humano que el código de máquina puro.

El lenguaje ensamblador es específico de cada máquina. Por ejemplo, el código escrito para un procesador x86-64 no se ejecutará en un procesador diferente, como un procesador ARM, que es 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 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 pueda ejecutarlos. En este sentido, el lenguaje de alto nivel actúa como una abstracción entre el código y las instrucciones reales del procesador.

El ensamblador ofrece al programador control directo sobre los recursos del sistema, lo que implica configurar registros del procesador, acceder a ubicaciones de memoria e interactuar con otros componentes de hardware. Esto requiere una comprensión profunda del funcionamiento

del procesador y la memoria.

Al programar en un lenguaje de alto nivel como C, no es necesario preocuparse por la implementación detallada del programa a nivel de máquina. En cambio, al escribir programas en código ensamblador, como se hacía en los primeros días de la informática, el programador debe especificar las instrucciones de bajo nivel que el programa utilizará para realizar un cálculo. La mayoría de las veces, resulta mucho más productivo y confiable trabajar en un nivel más alto de abstracción proporcionado por un lenguaje de alto nivel. La verificación de tipos realizada por un compilador ayuda a detectar muchos errores de programación y asegura que los datos se manejen de manera consistente.

Con los compiladores y optimizadores modernos, el código generado suele ser tan eficiente como el que escribiría manualmente un programador experto en ensamblador. Además, un programa escrito en un lenguaje de alto nivel puede compilarse y ejecutarse en múltiples plataformas, mientras que el código ensamblador es altamente específico de la máquina para la que fue desarrollado.

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 lenguajes de alto nivel.
- **Conocimiento**: El conocimiento del lenguaje ensamblador ayuda a escribir mejores programas, incluso cuando se use lenguajes de alto nivel.

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 información es visible a nivel de código Assembler.

Otro ejemplo es que 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] Historia de la Informática. Harvard MARK I (1944), 2015. [Internet; descargado 2-agosto-2023].

[3] Colaboradores de 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].

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

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

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

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

[8] Colaboradores de 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].

[9] Colaboradores de Wikipedia. IBM System/360, 2015. [Internet; descargado 2-agosto-2021].

[10] Colaboradores de 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].

[11] Elprocus. Introduction to 8080 Microprocessor and its Architecture, 2015. [Internet; descargado 2-agosto-2023].

[12] Colaboradores de Wikipedia. Apple II, 2023. [Internet; descargado 2-agosto-2023].

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

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

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

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

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

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

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

[20] William Stallings. *Computer organization and architecture: designing for performance*. Pearson Education India, 11th edition, 2022.