



TEMAS REPASO COC

Una **Computadora** es una máquina digital programable y sincrónica, con cierta capacidad de cálculo numérico y lógico, controlada por un programa almacenado y con posibilidad de comunicación con el mundo exterior.

Es **DIGITAL** porque dentro de la computadora las señales eléctricas que se manejan y la información que se procesa se representa en forma discreta, por medio de dos valores (0 y 1).

Es **SINCRÓNICA** porque realiza las operaciones coordinada por un reloj central que envía señales de sincronismo a todos los elementos que componen la computadora.

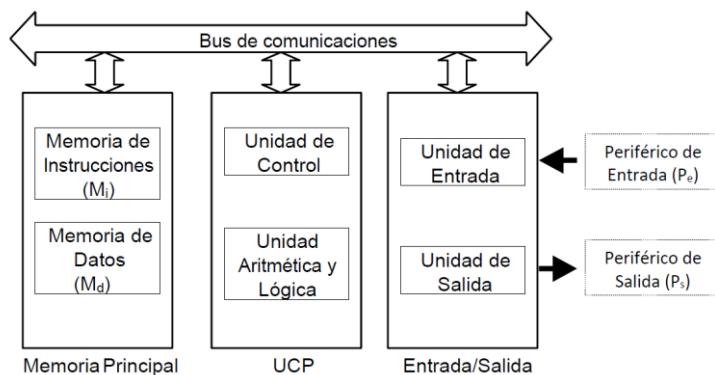
Internamente posee una capacidad de cálculo numérico y lógico, en un subsistema denominado Unidad Aritmético-Lógica o **ALU**.

Está **CONTROLADA POR PROGRAMA** (a diferencia a una calculadora), significa que internamente se tienen órdenes o instrucciones almacenadas, que la computadora podrá obtener, interpretar y ejecutar.

Además, está **COMUNICADA** con el mundo exterior. Esto significa que podrá realizar operaciones de ingreso o egreso de valores desde y hacia el mundo real, utilizando dispositivos periféricos.

- **Componentes y funcionamiento básico de una computadora**

La mayoría de las computadoras actuales de propósito general presentan una estructura interna basada en la arquitectura definida por **JOHN VON NEUMANN**. Esta estructura interna debe contener aquellos componentes que permitan realizar el procesamiento de datos útiles para el problema a resolver. Dado que se utilizará un programa que controlará la sucesión de pasos a seguir, será necesario no solamente tener una unidad de cálculo sino también una unidad de memoria. Podrá también, ser necesario interactuar con el mundo exterior, tanto para obtener datos como para entregar resultados, por lo que unidades que se encarguen de la entrada y salida de valores podrán estar presentes.



La memoria principal se divide conceptualmente en **memoria de instrucciones** donde residen las órdenes que la computadora debe interpretar y ejecutar, y **memoria de datos** donde se almacena la información con la cual la computadora realizará los procesos (cálculos, decisiones, actualizaciones) que sean necesarios para la resolución del problema.

Las líneas de comunicación indicadas como bus de comunicaciones normalmente permiten el paso de tres grandes categorías de información que son independientes entre si:

- **Bus de control:** mediante él se indica el tipo de operación a realizar, se sincronizan los dispositivos, se verifica si la contraparte está lista, se autoriza el uso de otro bus, entre otros.
- **Bus de datos:** a través de este canal se transfieren los datos, una vez que se establecieron las señales de control correspondientes y que se estableció con qué dispositivo o dirección de memoria se desea comunicar. Cuanto mayor capacidad tenga este bus, más datos se pueden transferir en cada instante de tiempo.
- **Bus de direcciones:** a través de este canal se transfiere con qué dirección de memoria se desea comunicar.

 La combinación de la unidad de control UC, la unidad de cálculo ALU y un conjunto de REGISTROS se la llama **unidad central de procesamiento CPU**. La ALU se ocupa de realizar el cómputo, y los datos con los que se trabaja siempre están en registros: a esta memoria interna deben viajar las instrucciones y los datos desde la memoria principal. Luego, la UC decodifica las instrucciones y orquesta las operaciones para ejecutarlas.

Algunos registros especiales son:

- **Contador de programa:** este registro guarda la dirección en la que está almacenada la próxima instrucción a ejecutar, y se incrementa en uno cada vez que se hace una lectura de instrucciones. De esta manera, el procesador sabe a dónde ir a leer la instrucción.
- **Registro de instrucción:** la instrucción se lee desde la memoria principal y se va almacenando en este registro para que la UC decida qué hacer.
- **Registro de direccionamiento:** en este registro se coloca el número de dirección al que ir a buscar un dato. Esta dirección es indicada por la propia instrucción o por algún otro modo de direccionamiento. Lo importante es no interferir con el contador del programa porque la máquina podría "perderse".

- **Funcionamiento básico**

1. Buscar la próxima instrucción a ejecutar en la memoria de instrucciones Mi
2. Interpretar qué hacer en la Unidad de Control (UC).

3. Ejecutar las operaciones interpretadas por UC, utilizando la ALU de ser necesario. Estas operaciones pueden comprender lectura/escritura de la memoria de datos o entrada/salida por los periféricos.
4. Esto representa una secuencia infinita de pasos.

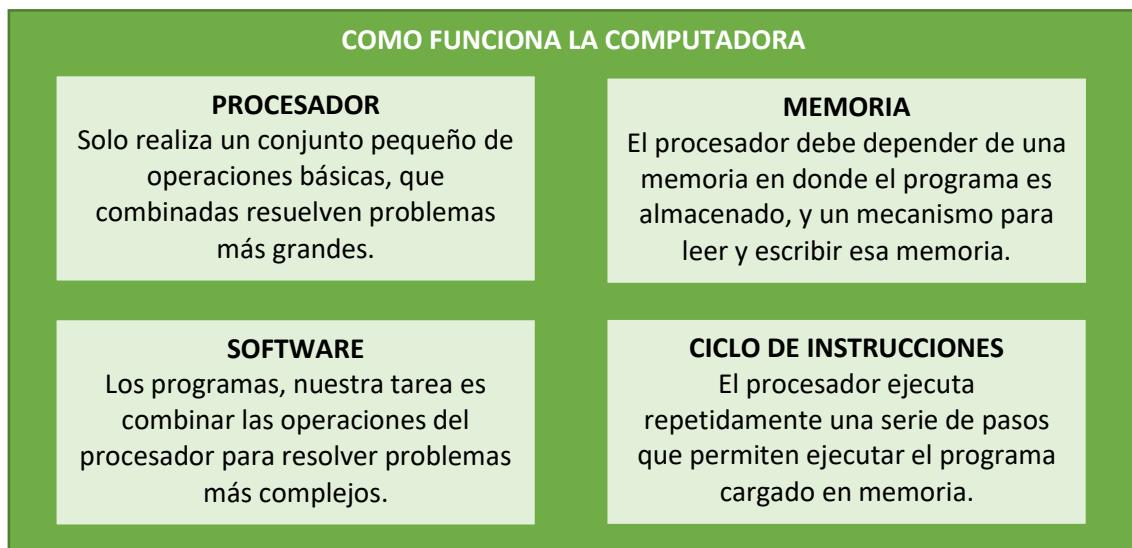
- **Algunos conceptos**

El **HARDWARE** se refiere a las componentes físicas de la computadora.

El **SOFTWARE** comprende los programas que se ejecutan sobre la computadora.

Un **BIT** (dígito binario) es la unidad de información más pequeña. Solo puede tener uno de dos valores: encendido o apagado (0 o 1, sí o no, etc.).

La Unidad Central de Procesamiento **CPU** es la encargada de interpretar y llevar a cabo las instrucciones de los programas.



Antes existía lo que se conoce como programación en Hardware, y cuando cambiaban las tareas debíamos cambiar el hardware. Luego, con la operación en Software en cada paso se efectúa alguna operación sobre los datos. Para cada paso se necesita un nuevo conjunto de señales de control, proporcionadas por las instrucciones. De este modo aparece un nuevo concepto de programación puesto que ya no es necesario cambiar el hardware cada vez que quiera realizar una tarea distinta.

- **Software**

La producción de sistemas de software, constituye el puente útil entre el usuario y la computadora. Hay un modelo por capas desde la máquina hasta el usuario:

1. HARDWARE: es la primera capa, puede ser un artefacto muy elaborado desde el punto de vista tecnológico, pero **totalmente inútil si no se lo “carga” con software**.
2. SISTEMA OPERATIVO: es la segunda capa, nos permite comunicarnos con la computadora y utilizar eficientemente sus recursos. Se subdivide en tres niveles: el primer nivel del sistema operativo es el que nos permite que al encender la máquina haya funciones “vitales” incorporadas al hardware, estas vienen incorporadas con el hardware y se denomina Sistema Operativo residente o BIOS; el segundo nivel del sistema operativo trata de ser “portable”, es decir agregar funciones que sean útiles al usuario del sistema operativo sobre cualquier máquina, y normalmente se “cargan” desde disco al ser solicitadas; el tercer nivel del sistema operativo se refiere esencialmente a las funciones de administración de recursos de la o las máquinas que controla el usuario: administrar la memoria principal, los dispositivos de almacenamiento, etc.
3. UTILITARIOS BÁSICOS: es la tercera capa y se refiere a los programas que nos acercan soluciones a problemas muy básicos del mundo real como procesadores de texto, planillas de cálculo, etc. En general, estas aplicaciones de software se construyen alrededor de metáforas visuales del mundo real, extendiendo de algún modo las habilidades naturales del usuario (por ejemplo, tener un procesador de textos con corrector ortográfico).
4. LENGUAJES DE PROGRAMACIÓN DE APLICACIONES: en la cuarta capa tenemos los lenguajes de programación de aplicaciones (tales como Pascal, C, Java, C++, ADA, Basic, Fortran, Delphi, etc.).
5. LENGUAJES ORIENTADOS A LA APLICACIÓN: en la quinta capa se trata de acercar aún más la forma de expresar los problemas y su solución al mundo del usuario. Estos lenguajes permiten resolver en forma sencilla alguna clase de problemas, no exigiendo una preparación especial del usuario.
6. SISTEMA DE SOFTWARE DE PROPÓSITO GENERAL: en la sexta capa tenemos estos sistemas, tales como los sistemas contables, de liquidación de sueldos, de facturación, etc.
7. SISTEMA DE SOFTWARE DEDICADOS: en la séptima capa se trata de desarrollar un producto “a medida” para una determinada organización, empresa o máquina. Por ejemplo, los controladores de un robot, de una máquina fotográfica o de un lavarropas.
8. USUARIO DEL MUNDO REAL.

¿Qué es el sistema operativo?

En el sistema operativo se incorporan las funciones de control del hardware de una computadora, de administración de sus recursos físicos y de sus usuarios, así como el control efectivo de la ejecución de los programas que en ella se carguen.

El funcionamiento del Sistema Operativo implica de existencia de al menos un programa que está permanentemente ejecutándose junto con nuestras aplicaciones.

Algunas de las tareas que hace el SO son: comunicación con los periféricos, control de autorización de usuarios, control de la ejecución de programas, control de errores, administración de memoria, entre otros.

El modo en que el sistema operativo se comunica con el usuario constituye la **interfaz** del mismo.

VIMA (WIMP en inglés) significa **Ventanas, Iconos, Menús y Apuntadores**, como interfaz de usuario tiene una serie de ventajas: son intuitivas ya que el usuario no necesita estudiar un manual de comandos para comprender lo que la imagen le muestra en un menú; son consistentes puesto que toda una gama de aplicaciones tiene la misma forma de interfaz; facilitan el autoaprendizaje al ser repetitivas; incorporan mecanismos de seguridad, como impedir determinados errores mediante mensajes y bloqueos para el usuario o permitir “volver atrás”, de modo de corregir alguna secuencia incorrecta de acciones; incrementan la flexibilidad, por ejemplo, al usar simultánea o alternativamente el teclado o el mouse.

¿Qué es lo primero que hace el procesador al encender la máquina?

Buscar una instrucción en la memoria, decodificarla, buscar los operandos, ejecutar la operación, almacenar el resultado, verificar interrupciones, repetir el ciclo con la siguiente dirección de memoria. El procesador siempre carga la primera instrucción desde una dirección preestablecida. De esta forma ejecuta un firmware que se encarga de inicializar dispositivos y ceder la ejecución al SO.

- **Componentes de la computadora**

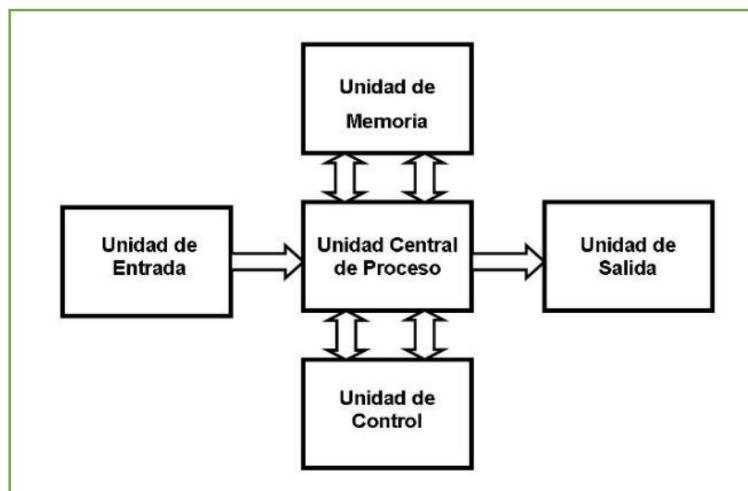
Las computadoras en realidad solo hacen cuatro cosas: recibir entradas (aceptan información desde el mundo exterior), producir salidas (dan información al mundo exterior), procesar información (llevan a cabo operaciones aritméticas o lógicas con la información) y almacenar información (mueven y almacenan información en la memoria).

Virtualmente todos los diseños de computadoras contemporáneas están basados en los conceptos desarrollados por John Von Neumann. Tal diseño es conocido como la **Arquitectura Von Neumann**, y se basa en tres conceptos claves:

- Los datos e instrucciones están almacenados en una única memoria de lectura-escritura constituida por celdas de igual tamaño.
- Los contenidos de las celdas de la memoria son identificables por posición, sin importar el tipo de los datos guardados en ese lugar.
- La ejecución ocurre de manera secuencial (a menos que se modifique explícitamente) de una instrucción a la siguiente.

El modelo de Von Newmann consta de 5 elementos principales:

- Unidad de Entrada: provee las instrucciones y los datos.
- Unidad de Memoria: en donde se almacenan los datos e instrucciones.
- Unidad Aritmético-Lógica: es la que procesa los datos.
- Unidad de Control: es la que dirige la operación.
- Unidad de Salida: donde se envían los resultados.



LA UNIDAD CENTRAL DE PROCESAMIENTO

Toda computadora tiene una CPU que interpreta y lleva a cabo las instrucciones de los programas, efectúa manipulaciones aritméticas y lógicas con los datos y se comunica con las demás partes del sistema de cómputo. En las computadoras personales se utilizan varios chips de CPU distintos. Aunque hay variantes en cuanto al diseño de estos chips, existen dos factores relevantes para el usuario: la compatibilidad (no todo el software es compatible con todas las CPU) y la velocidad (en una computadora está determinada en gran parte por la velocidad de su reloj interno, el dispositivo cronométrico que produce pulsos eléctricos para sincronizar las operaciones. Por lo general, las computadoras se describen en términos de su velocidad de reloj, medida en Hertz. La velocidad está determinada también por la arquitectura del procesador, esto es, el diseño que establece de qué manera están colocados en el chip los componentes individuales de la CPU).

Los principales componentes de la CPU son una Unidad Aritmética y Lógica (ALU) y una Unidad de Control (UC). La ALU realiza la computación real o procesamiento de datos. La UC controla la operación de la ALU. Además, existe una mínima memoria interna de la CPU, consistente de un

conjunto de lugares de almacenamiento, llamados registros. También hay un bus interno de CPU, necesario para transferir datos entre los distintos registros y la ALU, dado que la ALU de hecho opera sólo sobre datos en la memoria interna de la CPU.

Los registros de la CPU sirven para dos funciones:

- Registros visibles al usuario. Permiten al programador en lenguaje de máquina minimizar las referencias a memoria principal optimizando el uso de los registros, que son de acceso más rápido que aquella.
- Registros de control y estado. Son utilizados por la UC para controlar la operación de la CPU, y por programas del SO para controlar la ejecución de los programas. Ejemplo: el Contador de Programa (que contiene la ubicación de la próxima instrucción a ser buscada y ejecutada) y el Registro de Instrucción (que contiene la última instrucción buscada).

LA MEMORIA

La CPU sólo puede manejar una instrucción y unos cuantos datos a la vez. La computadora tiene que “recordar” el resto del programa y los datos hasta que el procesador esté listo para usarlos. La CPU está conectada con el resto de los componentes del sistema a través de tres buses distintos. Se tiene entonces, un medio para identificar: *el bus de direcciones*, un medio para transportar el dato propiamente dicho: *el bus de datos*, y un medio para controlar el intercambio de información: *el bus de control*.

La CPU saca información de la misma instrucción que debe ejecutar, así sabe, por ejemplo, en qué sentido deberían viajar los datos (lectura o escritura) y envía en consecuencia las señales adecuadas por el bus de control. La CPU deberá también conocer cuando enviar esas señales para trabajar en forma conjunta y ordenada. Se denomina tiempo de acceso al tiempo que tarda un elemento de memoria en cumplir efectivamente una orden de lectura o escritura.

El denominado ‘tamaño del bus de direcciones’, determina cuántos bits tienen las direcciones que identifican cada celda de memoria identificada con una dirección de memoria.

En general para identificar N diferentes posiciones de memoria, se necesitará que n (el número de bits del bus de direcciones) sea tal que se cumpla: $N \leq 2^n$

Ejemplo 1 ¿Cuántos bits deberán tener las direcciones para identificar 250 posiciones de memoria diferentes?

$N = 250 \leq 2^n$ si $n = 8$, $2^8 = 256$ y se cumple la desigualdad. Respuesta: 8 bits.

Ejemplo 2 Escriba en binario la dirección más pequeña y la más grande para el bus de direcciones del ej. anterior. ¿Puede determinar su valor en decimal?

Dirección más pequeña = 00000000 | Dirección más grande = 11111111

Identificamos cada posición de memoria con un número binario llamado dirección. La cantidad de bits almacenados en ella (llamado dato) se conoce como “unidad mínima direccionable”. Si la posición de memoria puede contener 8 bits (1 byte) de información decimos que la unidad mínima direccionable es el byte.

RAM Y ROM

La RAM (Random Access Memory o Memoria de Acceso Aleatorio) es el tipo más común de almacenamiento primario o memoria de la computadora. Los chips RAM contienen circuitos que sirven para almacenar temporalmente instrucciones de programas y datos. Un chip de RAM está dividido en posiciones o celdas de igual tamaño, identificadas por una dirección única, de manera que el procesador puede distinguirlas y ordenar que se guarde o recupere información de ella. La RAM es una memoria volátil: si se interrumpe la energía eléctrica, la computadora olvida inmediatamente todo lo que estaba recordando en la RAM.

La memoria no volátil se denomina ROM (Read-Only Memory o Memoria Sólo de Lectura), la computadora puede leer información de ella, pero no escribir nueva información. Todas las computadoras modernas cuentan con dispositivos de ROM que contienen las instrucciones de arranque y otra información crítica. La información en la ROM se graba permanentemente cuando nace la computadora, de modo que siempre está disponible cuando ésta opera, pero no puede cambiarse a menos que se reemplace el chip de ROM.

En RAM y ROM el tiempo de acceso es constante sin importar la ubicación relativa de las celdas. Adicionalmente, existen otros medios donde almacenar información y que constituyen una forma de memoria externa, como por ejemplo los discos rígidos.

Hay una relación entre las tres características clave de la memoria (costo, capacidad y tiempo de acceso): a menor tiempo de acceso, mayor costo por bit; a mayor capacidad, menor costo por bit; a mayor capacidad, mayor tiempo de acceso.

BUSES Y ENTRADA / SALIDA

Por lo general, los buses tienen 8, 16 o 32 cables; dado que por cada cable puede fluir un bit a la vez, un bus con 16 cables se denomina bus de 16 bits, ya que puede transmitir 16 bits de información al mismo tiempo (por distintos caminos); transmite el doble de información que un bus de 8 bits. Entonces, los buses más anchos pueden transmitir información con más rapidez que los angostos. Además de la CPU y un conjunto de módulos de memoria, el tercer elemento clave de un sistema de cómputo es un conjunto de módulos de entrada y/o salida (E/S). Cada módulo realiza la interfase con el bus del sistema y controla uno o más dispositivos periféricos.

Un módulo de E/S es la entidad responsable de controlar uno o más dispositivos externos y de intercambiar datos entre estos dispositivos y la memoria principal y/o los registros de la CPU.

- **El ciclo de instrucción**

La función básica realizada por una computadora es la ejecución de programas. Hay que entender el proceso de ejecución: el punto de vista más simple es considerar el procesamiento de una instrucción como consistente de dos pasos: la CPU lee (*búsqueda*) las instrucciones desde la memoria una a la vez, y las completa (*ejecución*).

La corrida de un programa consiste en la repetición de los pasos de búsqueda y ejecución. La ‘búsqueda’ de instrucción es una operación común para cada instrucción, y consiste en leer información de, al menos, una posición de memoria. La ‘ejecución’ puede involucrar varias operaciones y depende de la naturaleza de la instrucción.

El procesamiento requerido para una sola instrucción es llamado ciclo de instrucción. Usando la descripción simplificada, los dos pasos son el ciclo de búsqueda y el ciclo de ejecución. El ciclo se detiene sólo si la máquina es apagada, si ocurre algún error irrecuperable, o se encuentra una instrucción de programa que detenga la computadora.

En el comienzo de cada ciclo de instrucción, la CPU busca una instrucción desde la memoria. En una CPU típica, se usa un registro llamado contador de programa para conocer la ubicación desde la cual la próxima instrucción debe ser buscada. A menos que se diga otra cosa, la CPU siempre incrementa el contador de programa después de cada búsqueda de instrucción, de modo de quedar listo para buscar la próxima en secuencia (es decir, la instrucción ubicada en la siguiente posición de memoria). Esta secuencia podrá ser alterada y deberemos indicarla en modo especial.

La instrucción buscada es cargada en un registro de la CPU conocido como Registro de Instrucción. La instrucción está en la forma de un código binario que especifica qué acción debe tomar la CPU; ésta interpreta la instrucción y realiza la acción requerida. En general, estas acciones caen en 4 categorías:

- CPU - Memoria: los datos pueden ser transferidos desde la UCP a la memoria o desde la memoria a la CPU.
- CPU - E/S: los datos pueden ser transferidos hacia o desde el mundo exterior por una transferencia entre la CPU y el módulo de E/S.
- Procesamiento de datos: la CPU puede realizar alguna operación aritmética o lógica sobre los datos.
- Control: una instrucción puede especificar que la secuencia de ejecución sea alterada (por ejemplo, con una operación de salto o jump). Por ejemplo, si la CPU busca una instrucción

de la posición 149 que especifica que la próxima instrucción sea buscada en la posición 182, la CPU recordará esto poniendo el 182 en el contador de programa. Así, en el próximo ciclo de búsqueda, la instrucción será buscada en la posición 182 en vez de la 150 como sería en la secuencia sin alterar.

La ejecución de una instrucción puede involucrar una combinación de estas acciones.

Además, todas las computadoras proveen un mecanismo por el cual otros módulos, de E/S o memoria, pueden interrumpir el procesamiento normal de la CPU. Las interrupciones se proveen principalmente como una manera de mejorar la eficiencia de procesamiento.

- **Representación numérica**

Los datos e informaciones que se manejan internamente en un sistema informático se pueden representar, según sus características, en:

CÓDIGOS ALFANUMÉRICOS (ASCII): en general cada carácter se maneja internamente en una computadora por medio de un conjunto de 8 bits (1 byte) mediante un sistema de codificación binario que denominaremos código de caracteres.

REPRESENTACIONES NUMÉRICAS: un sistema de numeración se caracteriza fundamentalmente por su *base*, que es el número de símbolos distintos que utiliza, y un coeficiente que determina cuál es el valor de cada símbolo dependiendo de la posición que ocupe.

Teorema Fundamental de la Numeración

Se trata de un teorema que relaciona una cantidad expresada en cualquier sistema de numeración posicional con la misma cantidad expresada en el sistema decimal. El Teorema Fundamental de la Numeración dice que el valor decimal de una cantidad expresada en otro sistema de numeración, está dado por la fórmula:

$$N = \sum_{i=-d}^n X_i x B^i$$

SISTEMA DECIMAL: sistema posicional que utiliza 10 símbolos (del 0 al 9).

SISTEMA BINARIO: es el sistema de numeración que utiliza internamente el hardware de las computadoras actuales. La base o número de símbolos que utiliza el sistema binario es 2 (siendo los símbolos 0 y 1).

SISTEMA HEXADECIMAL: es un sistema posicional pero que utiliza dieciséis símbolos (0 al 9 y de A al F).

Rango de representación

Se denomina *rango de representación* en un sistema determinado, al conjunto de números representables con el mismo. Un sistema de base b y números de n dígitos tiene un rango igual a b^n .

PRÁCTICA PERIFÉRICOS



- **Pantalla alfanumérica:** sólo pueden mostrar caracteres alfanuméricos y para esto se dividen en filas y columnas. En cada celda se almacena un carácter codificado en binario. Para esto se utiliza la tabla ASCII que asigna 8 bits a cada carácter. Puede suceder que se especifique un color para cada carácter, entonces necesito bits adicionales. Como sólo se puede elegir un color por vez y por carácter serán $\log_2(\text{cantidad colores})$. Si se agregan atributos por cada carácter (subrayado, negrita, etc.) se necesitará 1 bit adicional para cada atributo (puedo tener más de uno a la vez).

Tamaño de memoria para la pantalla alfanumérica

Columnas x filas x (8 bits ASCII + bits color + cantidad atributos)

- **Pantalla gráfica:** este tipo de pantalla se divide en pixeles. La cantidad de pixeles estará dada por la resolución (ancho x alto, por ejemplo, 1024x768). Cada punto puede ser de un color, e indicando el color de cada pixel en toda la pantalla se formará la imagen. Existen diferentes profundidades de color, por ejemplo: **monocromo** (blanco o negro, 1 bit); **escala de grises** (256 tonos de gris, 1 byte por pixel); **True Color** (24 bits por pixel).

Tamaño de memoria para la pantalla gráfica

Ancho x alto x bits color

- **Impresoras y escáneres:** estos dispositivos sirven para transferir información desde (escáner) y hacia (impresora) un medio impreso como el papel. Las características del medio y las capacidades del dispositivo definirán el tamaño de la información. La cantidad de puntos o de caracteres dependerá de la resolución del dispositivo. Además, existen impresoras que trabajan en modo texto, y cada unidad, en lugar de ser un punto es un carácter lo que agrega 8 bits (ASCII) a cada unidad de dato.

Ejemplo 1: calcular el tamaño de memoria para un escáner monocromo con una resolución de 300ppp al escanear una hoja de 10x15 pulgadas.

$$(300 \times 10) \times (300 \times 15) \times 1 \text{ bit (monocromo)} = 13500000 \text{ bits}$$

Ejemplo 2: calcular el tamaño de memoria para una impresora con una resolución de 150 puntos por centímetro que trabaja en True Color para imprimir una hoja 10x15cm.

$$(150 \times 10) \times (150 \times 15) \text{ pixels} \times 3 \text{ bytes / pixels (24 bits True Color)} = 10125000 \text{ bytes}$$

Ejemplo 3: calcular el tamaño de memoria para una impresora que imprime en alfanumérico con 150 filas y 100 caracteres por fila en True Color.

$$(150 \times 10) \times (150 \times 15) \text{ caracteres} \times (1 + 3 \text{ bytes}) / \text{caracter (TC + ASCII)} = 13500000 \text{ bytes}$$

- **Disco duro:** este es un dispositivo externo de almacenamiento secundario. Vamos a asumir que las dos caras de cada plato son utilizables. Todas las caras tienen la misma cantidad de pistas, todas las pistas tienen la misma cantidad de sectores y todos los sectores tienen la misma cantidad de bits. Esto es importante ya que la pista más interna es más corta que la más externa, lo cual implica que los sectores son más densos en el interior, pero siempre todos almacenan la misma cantidad de bytes. Por lo general un sector tiene 512 bytes y esta es la mínima unidad que se puede leer / escribir en el disco.

Tamaño de memoria para disco duro

$$\frac{\text{X platos} \times \text{2 caras} \times \text{X pistas} \times \text{X sectores} \times \text{512 bytes}}{\text{plato} \quad \text{cara} \quad \text{pista} \quad \text{sector}}$$

Ejemplo: calcular el tamaño de memoria para un disco duro de 2 platos con 300 pistas por cara y 1000 sectores por pista.

$$2 \text{ platos} \times 2 \text{ caras} / \text{plato} \times 300 \text{ pistas} / \text{cara} \times 1000 \text{ sectores} / \text{pista} \times 512 \text{ bytes} / \text{sector} = 614,4 \text{ MB}$$

Cosas a tener en cuenta:

1. La velocidad de rotación es una característica importante del disco porque afecta al desempeño general.
2. Si debemos hacer cálculos que impliquen a la velocidad de rotación diremos que en cada vuelta se puede leer una pista completa.
3. Si el valor que conocemos es el radio o diámetro de una pista y la densidad de sectores en ese extremo hay que usar la fórmula de la circunferencia.

Ejemplo: calcular cantidad de sectores por pista, si tengo 10 sectores por centímetro y una pista de radio de 2 centímetros.

$$10 \text{ sectores} / \text{cm} \times (2\pi \times \text{radio}) \text{ cm} = 10 \times 2\pi \times 2 \text{ sectores} = 40\pi \text{ sectores} \approx 125 \text{ sectores}$$

SISTEMAS DE REPRESENTACIÓN EN PUNTO FIJO

BSS – Binario sin signo

- Con n bits represento 2^n números.
- Rango de representación: $0 \leq x \leq 2^n - 1$
- Números negativos o fraccionarios no pueden ser representados en este sistema.

Ejemplo BSS restringido a 3 bits

Represento $2^3 = 8$ números / Rango de representación: $0 \leq x \leq 7$

0=000 | 1=001 | 2=010 | 3=011 | 4=100 | 5=101 | 6=110 | 7=111

BCS – Binario con signo

- Con n bits represento 2^n números.
- Rango de representación: $-(2^{n-1} - 1) \leq x \leq 2^{n-1} - 1$
- Tengo dos ceros: 00...0 / 11...1
- Números positivos igual que el BSS, números negativos empiezan con 1.

Ejemplo BCS restringido a 3 bits

Represento $2^3 = 8$ números / Rango de representación: $-3 \leq x \leq 3$

-3=111 | -2=110 | -1=101 | 0=000 y 111 | 1=001 | 2=010 | 3=011

TECNICA DE COMPLEMENTOS: el complemento a un número N de un número A ($A < N$) es igual a la cantidad que le falta a A para ser N. Complemento a N de A = $N - A$

El complemento a un número N de $(N - A)$ es A. Complemento a N de $(N - A) = N - (N - A) = A$

Ca1 – Complemento a 1 (Complemento a la base reducida)

- Complemento a la base denominada $N=b^n-1$
- Con n bits represento 2^n números.
- Rango de representación: $-(2^{n-1} - 1) \leq x \leq 2^{n-1} - 1$
- Tengo dos ceros: 00...0 / 11...1
- Números positivos igual que el BSS, números negativos calculo el número positivo y luego cambio los 0 por 1 y viceversa.

Ejemplo Ca1 restringido a 3 bits

Represento $2^3 = 8$ números / Rango de representación: $-3 \leq x \leq 3$

-3=100 | -2=101 | -1=110 | 0=000 y 111 | 1=001 | 2=010 | 3=011

- OTRA FORMA: pasar negativo de decimal a binario: $-(2^{n-1} - 1) + |\text{num}| \rightarrow$ el resultado lo paso a binario. Por ejemplo, en 8 bits: $-56 = -(2^{8-1} - 1) + 56 = -2^7 + 56 = -71 = 11000111$.

puedo pasar de binario a negativo: el primer digito es ahora $-(2^{n-1}-1)$ y el resto de los dígitos con pesos positivos. Por ejemplo, en 3 bits: $101 = -(2^{3-1}-1) + 2^0 = -3+1 = -2$

Ca2 – Complemento a 2 (Complemento a la base)

- Complemento a la base denominada $N=b^n$
- Con n bits represento 2^n números.
- Rango de representación ASIMÉTRICO: $-(2^{n-1}) \leq x \leq 2^{n-1}-1$
- Tengo un cero: 00...0
- Números positivos igual que el BSS, números negativos calculo el Ca1y sumo 1.

Ejemplo Ca2 restringido a 3 bits

Represento $2^3=8$ números / Rango de representación: $-4 \leq x \leq 3$

$-4=100$ ($100 \rightarrow 011+1 \rightarrow 100$) | $-3=101$ ($011 \rightarrow 100+1 \rightarrow 101$) | $-2=110$ ($010 \rightarrow 101+1 \rightarrow 110$) |
 $-1=111$ ($001 \rightarrow 110+1 \rightarrow 111$) | 0=000 | 1=001 | 2=010 | 3=011

- OTRA FORMA: pasar negativo de decimal a binario: $-(2^{n-1})+|num| \rightarrow$ el resultado lo paso a binario. Por ejemplo, en 8 bits: $-56 = -(2^{8-1}) + 56 = -2^8 + 56 = -72 = 11000110$.
puedo pasar de binario a negativo: el primer digito es ahora $-(2^{n-1})$ y el resto de los dígitos con pesos positivos. Por ejemplo, en 3 bits: $101 = -(2^{3-1}) + 2^0 = -4+1 = -3$

TECNICA DE EXCESO: la representación de un número A es la que corresponde a la SUMA del mismo y un valor constante E. Exceso E de A = A + E

Dado un valor A, para obtener el número representado tengo que RESTAR el valor del exceso.

$A = \text{Exceso E de A} - E$

Ex2 – Exceso en base 2

- El número a representar viene dado por su valor más el valor del exceso (n bits, $E=2^{n-1}$).
- Con n bits represento 2^n números.
- Rango de representación ASIMÉTRICO: $-(2^{n-1}) \leq x \leq 2^{n-1}-1$
- Un cero: 10...0
- Si empieza con 0 es negativo y si empieza con 1 es positivo (AL REVÉS).

Ejemplo Ex2 restringido a 3 bits

Represento $2^3=8$ números / Rango de representación: $-4 \leq x \leq 3$ / Exceso $2^{3-1}=4$

$-4=000$ ($-4+4=0 \rightarrow 000$) | $-3=001$ ($-3+4=1 \rightarrow 001$) | $-2=010$ ($-2+4=2 \rightarrow 010$) |
 $-1=011$ ($-1+4=3 \rightarrow 011$) | 0=100 ($0+4=4 \rightarrow 100$) | 1=101 ($1+4=5 \rightarrow 101$) | 2=110 ($2+4=6 \rightarrow 110$) |
3=111 ($3+4=7 \rightarrow 111$)

NÚMEROS EN PUNTO FIJO

Se considera que todos los números a representar tienen exactamente la misma cantidad de dígitos y la coma fraccionaria está siempre ubicada en el mismo lugar.

Ejemplo 11,00 | 10,01 | 00,10 → 4 bits: 2 para la parte entera y dos para la fraccionaria.

En la computadora no se guarda la COMA, se supone que está en un lugar determinado.

RANGO Y RESOLUCION EN PUNTO FIJO

El RANGO es la diferencia entre el número mayor y el menor [nº menor; nº mayor]

La resolución es la diferencia entre dos números consecutivos. En los sistemas de punto fijo la resolución es constante.

Ejemplo en un Sistema Punto Fijo con 3 bits parte entera en Ca2 y dos bits fraccionaria en BSS:

Rango [111,11 ; 011,11] → [-4,75 ; 3,75] / Resolución 111,11-111,10=000,01=0.25

REPRESENTACIÓN Y ERROR

Al convertir un número decimal a binario tenemos dos casos:

- Sin restricción: en la cantidad de bits a usar, la representación no tendrá errores.
- Con restricción de bits: la representación puede tener un error dado por $e = |x - n|$, donde x es el número que quiero representar y n es el número más cercano a x que puedo representar. Además, $e^{\max} = \text{resolución}/2$

PASAR DECIMAL FRACCIONARIO A BINARIO

Cuando se trata de pasar a binario un número fraccionario SE MULTIPLICA POR 2. Serán sucesivas multiplicaciones que se acotarán según la cantidad de bits con los que contemos para la representación. Las partes enteras de esas multiplicaciones parciales formarán al finalizar el número binario resultante. Las partes fraccionarias parciales son las que se irán multiplicando por 2 sucesivamente. Por ejemplo, para 0,2:

0,2 * 2 = 0,4 me quedo con el entero 0 del resultado

0,4 * 2 = 0,8 me quedo con el entero 0 del resultado

0,8 * 2 = 1,6 me quedo con el entero 1 del resultado

0,6 * 2 = 1,2 me quedo con el entero 1 del resultado

0,2 * 2 = 0,4 ... Comienzan a repetirse los 4 cálculos anteriores

Entonces para la representación de la parte fraccionaria se toman los valores enteros de los resultados parciales, en el orden calculado: 0.2 = .0011 0011 ... 0011 (periódico).

SISTEMA BCD Y BCD EMPAQUETADO (Sistema Decimal codificado en Binario)

En este sistema, los dígitos decimales se convierten uno a uno en binario. Para representar un dígito decimal necesito 4 bits, y se asocia cada dígito con su valor en binario puro.

0=0000 | 1=0001 | 2=0010 | 3=0011 | 4=0100 | 5=0101 | 6=0110 | 7=0111 | 8=1000 | 9=1001

- **BCD Desempaquetado:** se aplican en E/S y periféricos, los números se codifican usando un byte por dígito. Pueden ser SIN SIGNO (4 bytes para el binario puro y los otros 4 se completan con 1111, ejemplo: 5 = 1111 0101 = F5); o CON SIGNO (los 4 bits que acompañan al ÚLTIMO dígito son reemplazados por C=1100 + | D=1101 -, por ejemplo: 4 = **1100** 0100 = C4 | -23 = 1111 0010 **1101** 0011 = F2D3)
- **BCD Empaquetado:** se aplica en cálculo, se reservan 4 bits por dígito. NUNCA cortar los bits, siempre escribir de a pares, si la cantidad de dígitos del número a representar es impar, completo con 0000. Por ejemplo: 125 = **0000** 0001 0010 0101 = 0125. El BCD Empaquetado también puede ser CON SIGNO, en este caso agrego el signo al final. Por ejemplo: -25 = 0000 0010 0101 **1101** = 025D | +8 = 1000 **1100** = 8C

Para sumar en BCD tenemos dos casos: RES \leq 9 \rightarrow No hay problema | RES $>$ 9 \rightarrow SUMO 6 a ese dígito (genero “acarreo” porque hay seis combinaciones no usadas).

Por ejemplo:

$$\begin{array}{r} \begin{array}{r} 26 \\ + 15 \\ \hline 41 \end{array} & \xrightarrow{\hspace{1cm}} & \begin{array}{r} +1 \\ 0010\ 0110 \\ + 0001\ 0101 \\ \hline 1011 \\ + 0110 \\ \hline 0100\ 0001 \end{array} \end{array}$$

SISTEMA BCH (Sistema Hexadecimal codificado en Binario)

Los dígitos decimales se convierten uno a uno en binario. Para representar un dígito hexadecimal se usarán siempre 4 bits, asociando cada dígito con su valor en binario puro.

0=0000 | 1=0001 | 2=0010 | 3=0011 | 4=0100 | 5=0101 | 6=0110 | 7=0111

8=1000 | 9=1001 | A=1010 | B=1011 | C=1100 | D=1101 | E=1110 | F=1111

SUMA Y RESTA BINARIA

En el Sistema Binario, cada nueva posición tiene **un** valor dos veces más grande que el anterior.

| En la suma | | | | En la resta | | | |
|------------|----------|----------|----------|-------------|----------|----------|----------|
| + 0 | + 0 | + 1 | + 1 | - 0 | - 1 | - 1 | - 0 |
| <u>0</u> | <u>1</u> | <u>0</u> | <u>1</u> | <u>0</u> | <u>1</u> | <u>0</u> | <u>1</u> |

CAMBIO DE BASES

| | $BASE_2$ | $BASE_{10}$ | $BASE_{16}$ |
|-------------|---|--|--|
| $BASE_2$ | | Teorema Fundamental de la Numeración $N_{10} = \sum_{i=n}^m d_i 2^i$ | TABLA: formo grupos de 4 bits desde la derecha y asingo hexa. 11 1110 ₂ = 3E ₁₆ |
| $BASE_{10}$ | Divido x_{10} por 2 5 2 → 5 ₁₀ = 101 ₂ 1 2 2 0 1 | | Divido x_{10} por 16 20 16 → 20 ₁₀ = 14 ₁₆ 4 1 |
| $BASE_{16}$ | TABLA: reemplazo cada dígito hexadecimal por su binario. 4A ₁₆ = 0100 1010 ₂ | Teorema Fundamental de la Numeración $N_{10} = \sum_{i=n}^m d_i 16^i$ | |

FLAGS

Las FLAGS son bits de condición que el procesador establece de modo automático acorde al resultado de cada operación realizada. Sus valores permitirán tomar decisiones como: realizar o no una transferencia de control o determinar relaciones entre números.

| Z (cero) | N (negativo) | C (carry) | O (overflow) |
|------------------------|---------------------|----------------------|---------------------|
| 1 → todos los bits = 0 | 1 → número negativo | 1 → acarreo / borrow | 1 → desborde en BCS |
| 0 → algún bit es 1 | 0 → número positivo | 0 → otro caso | 0 → otro caso |

- Si hay OVERFLOW [O] → Ca2 es incorrecto
- Si hay CARRY o BORROW [C] → BSS es incorrecto

Cuando hay Overflow

$$\begin{array}{ll} + + + = - & + - - = - \\ - + - = + & - - + = + \end{array}$$

REPRESENTACIÓN ALFANUMÉRICA

Se pueden representar: letras (mayúsculas y minúsculas), dígitos decimales (0...9), signos de puntuación, caracteres especiales, caracteres u órdenes de control. Algunos códigos son:

- **FIELDATA:** 26 letras mayúsculas + 10 dígitos + 28 caracteres especiales. Como hay 64 combinaciones en total → Código de **6 bits**.
- **ASCII:** FIELDATA + minúsculas + control. Como hay en total 127 combinaciones → Código de **7 bits**.
- **ASCII extendido:** ASCII + multinacional + semigráficos + matemática → Código de **8 bits**.
- **EBCDIC [Extended BCD Interchange Code]:** es parecido al ASCII, pero de IBM. → Código de **8 bits**.

SISTEMA EN PUNTO FLOTANTE

Este sistema nos permite tener un rango mayor de números, es decir, podemos representar números de “más pequeños” a “más grandes”, con la misma cantidad de bits, lo cual era una limitación para el sistema en Punto Fijo.



La base [B] es implícita y no necesita almacenarse, puesto que es la misma para todos los números. La ventaja que tiene este sistema es que necesito menos bits para almacenar la mantisa y el exponente, que para almacenar el número completo. Tanto la M como el E, están representados en algún sistema Punto Fijo ya conocido (BSS, BCS, Ca1, Ca2, Ex2).

DIFERENCIAS ENTRE SISTEMAS PUNTO FIJO Y PUNTO FLOTANTE

- El rango en Punto Flotante es mayor.
- La cantidad de combinaciones binarias es la misma en ambos sistemas.
- En Punto Flotante, la resolución no es constante a lo largo del intervalo.

MANTISA SIN NORMALIZAR, NORMALIZADA, O NORMALIZADA BIT IMPLÍCITO

- Número **SIN NORMALIZAR**: podemos tener distintas representaciones para un mismo número, por ejemplo: $2^{-1}x 2^0 = 2^0x 2^{-1} = 2^2x 2^{-3} \dots$ etc.
- **NORMALIZACIÓN**: nos sirve para tener un único par de valores de mantisa y exponente para un número **[SIN 0]**. EN EL SISTEMA FRACCIONARIO NORMALIZADO TODAS LAS MANTISAS EMPIEZAN CON 0,1.
- **NORMALIZACIÓN BIT IMPLÍCITO**: en este caso, no almaceno el uno más significativo de la mantisa, este es el bit implícito **[SIN 0]**.

MANTISA ENTERA O FRACCIONARIA

La mantisa puede ser entera o fraccionaria, y esto me lo especifican en el enunciado. Por ejemplo, si me dan el número 0011 010 con 4 bits de mantisa fraccionaria BSS y 3 de exponente BCS, la mantisa [M] será $0,011 = 2^{-2}+2^{-3}$ y el exponente [E] será 010 = 2, entonces reemplazamos $M \times 2^E = (2^{-2}+2^{-3}) \times 2^2 = 2^0+2^{-1} = 1,5$

RANGO Y RESOLUCIÓN EN SISTEMA PUNTO FLOTANTE

Al igual que en el Sistema Punto Fijo, el RANGO es la diferencia entre el mayor y el menor número, se denota **[mayor; menor]**. Siempre será: mínimo → menor número en mantisa y mayor número en exponente | máximo → mayor número en mantisa y menor número en exponente.

exponente. Además, si la mantisa es normalizada con signo, voy a tener dos intervalos de números, uno positivo y otro negativo, por lo tanto, voy a tener dos mínimos y dos máximos, uno por cada intervalo.

La RESOLUCIÓN es la diferencia entre dos representaciones sucesivas, y a diferencia del Sistema en Punto Fijo, en el caso del Sistema Punto Flotante varía a lo largo del rango. Podemos tener 4 resoluciones distintas:

INFERIOR NEGATIVA | SUPERIOR NEGATIVA | INFERIOR POSITIVA | SUPERIOR POSITIVA

En una recta se ven:



Tener en cuenta que cuando la mantisa fraccionaria tiene muchos dígitos, para buscar máximos número de mantisa [M] con n dígitos hacemos $M = \pm (1-2^{-n})$. Por ejemplo, para una mantisa fraccionaria 6 dígitos BCS y un bit de signo, el número máximo que puedo representar con la mantisa es $M = \pm (1-2^{-6})$. (Después tengo que ver el exponente)

SUMAS EN SISTEMA PUNTO FLOTANTE Y CORRIMIENTO DEL EXPONENTE

Para realizar operaciones en el Sistema Punto Flotante es necesario que, para sumar las mantisas, los exponentes de cada número sean iguales, en el caso de que no se cumpla esto, debo hacer un corrimiento de alguno de los dos exponentes (o ambos de ser necesario), para poder realizar la operación correctamente. Por ejemplo, suma en Punto Flotante con mantisa BSS 8 bits y exponente en BCS 8 bits. $00001111\ 00000011 + 00001000\ 00000010$

$$00001111\ 00000011 \rightarrow E = 3$$

$$00001000\ 00000010 \rightarrow E = 2 \rightarrow M \times 2^E = 2^3 \times 2^2 = 2^2 \times 2^3 \rightarrow M = 00000100$$

Luego sumo:

$$\begin{array}{r}
 + 00001111\ 00000011 \\
 + 00000100\ 00000011 \\
 \hline
 00010011\ 00000011
 \end{array} \rightarrow (2^4+2^1+2^0) \times 2^3 = 2^7+2^4+2^3 = 152$$

APROXIMACIÓN, ERROR ABSOLUTO, RELATIVO Y MAXIMO

- El **error absoluto** es la diferencia entre el valor representado y el valor que quiero representar $EA = |X - N_{aRepr}|$.
- El **error absoluto máximo** cumple $EA_{max} \leq res/2$.
- El **error relativo** lo calculo haciendo $ER = EA/N_{aRepr}$

ESTÁNDAR IEEE 754 SIMPLE Y DOBLE PRECISIÓN

El IEEE 754 es un estándar de aritmética en coma flotante. Este estándar especifica cómo deben representarse los números en coma flotante con simple precisión (32 bits) o doble precisión (64 bits), y también cómo deben realizarse las operaciones aritméticas con ellos.

Emplea mantisa fraccionaria, normalizada y en representación signo magnitud (M y S), sin almacenar el primer dígito, que es igual a 1. El exponente se representa en exceso, que en este caso no se toma como 2^{n-1} , sino como $2^{n-1}-1$. Además, la base implícita de este sistema es 2.

- **Simple Precisión:** el estándar IEEE-754 para la representación en simple precisión de números en coma flotante exige una cadena de 32 bits. El primer bit es el bit de signo (S), los siguientes 8 son los bits del exponente (E) y los restantes 23 son la mantisa (M). La **mantisa** es fraccionaria normalizada, con la coma después del primer bit que es siempre uno (1,) en M y S; y el **exponente** se representa en exceso.
- **Doble precisión:** el estándar IEEE-754 para la representación en doble precisión de números en coma flotante exige una cadena de 64 bits. El primer bit es el bit de signo (S), los siguientes 11 son los bits del exponente (E) y los restantes 52 son la mantisa (M).

| | SIMPLE PRECISIÓN | | | DOBLE PRECISIÓN | | |
|---------------------|------------------|-----------------------------|--------------|-----------------|-------------------------------|--------------|
| | S = 1 | EXPONENTE = 8 | MANTISA = 23 | S = 1 | EXPONENTE = 11 | MANTISA = 52 |
| Bits totales | | 32 | | | 64 | |
| Exponente en exceso | | 127 | | | 1023 | |
| Rango de exponente | | -126 +127 | | | -1022 +1023 | |
| Rango de números | | 2^{-126} $\sim 2^{128}$ | | | 2^{-1022} $\sim 2^{1024}$ | |

| EXPONENTE | MANTISA | SIGNO | VALOR |
|----------------------|------------|-------|---|
| $0 < E < 255$ [2047] | Cualquiera | 0 1 | $\pm 2^{E-127[1023]} \times 1.M$ |
| 255 [2047] | No nulo | 0 1 | NaN (not a number) |
| 255 [2047] | 0 | 0 1 | $\pm \infty$ |
| 0 | No nulo | 0 1 | $\pm 2^{-126[-1022]} \times 0.M$ (sin normalizar) |
| 0 | 0 | 0 1 | ± 0 |

[DEL STALLING] El Estándar IEEE 754 se desarrolló para facilitar la portabilidad de los programas de un procesador a otro y así poder alentar el desarrollo de programas numéricos sofisticados. Dichos formatos reducen la posibilidad de que el resultado final se vea deteriorado por un error de redondeo.

LÓGICA DIGITAL

Un circuito digital es en el que están presentes dos valores lógicos. Y las compuertas son dispositivos electrónicos que pueden realizar distintas funciones con estos dos valores lógicos. Las compuertas constituyen la base de hardware sobre la que se construyen todas las computadoras digitales. Para describir los circuitos que pueden construirse combinando compuertas, se requiere un nuevo tipo de álgebra, en donde las variables solo pueden ser 0 o 1: esta es el **álgebra booleana**.

Puesto que una función booleana de n variables tiene 2^n combinaciones de los valores de entrada, la función puede describirse totalmente con una tabla de 2^n renglones, donde cada uno indica un valor de la función para cada combinación distinta de las entradas. Dicha tabla se la conoce como **tabla de verdad**.

Identidades del álgebra booleana

| | | |
|---------------------|---|---|
| Identidad | $1 \cdot A = A$ | $0 + A = A$ |
| Nula | $0 \cdot A = 0$ | $1 + A = 1$ |
| Idempotencia | $A \cdot A = A$ | $A + A = A$ |
| Inversa | $A \cdot \bar{A} = 0$ | $A + \bar{A} = 1$ |
| Commutativa | $A \cdot B = B \cdot A$ | $A + B = B + A$ |
| Asociativa | $(A \cdot B) \cdot C = A \cdot (B \cdot C)$ | $(A + B) + C = A + (B + C)$ |
| Distributiva | $A + (B \cdot C) = (A + B) \cdot (A + C)$ | $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ |
| Absorción | $A \cdot (A + B) = A$ | $A + (A \cdot B) = A$ |
| De Morgan | $A \cdot \bar{B} = \bar{A} + \bar{B}$ | $\bar{A} + \bar{B} = \bar{A} \cdot \bar{B}$ |

Funciones lógicas

Toda función lógica puede representarse mediante una tabla de verdad y un diagrama de compuertas lógicas. Si quiero escribir la función lógica a partir de una tabla de verdad, debo tener en cuenta que deben haber tantos términos como unos en los resultados de la tabla, y las variables que valen 0 en la tabla aparecen negadas.

Por ejemplo para la siguiente tabla de verdad tenemos la función $\rightarrow F = \bar{A}B + A\bar{B}$

| A | B | | F |
|---|---|--|---|
| 0 | 0 | | 0 |
| 0 | 1 | | 1 |
| 1 | 0 | | 1 |
| 1 | 1 | | 0 |

Suma de Productos: Es posible inferir la fórmula lógica asociada a una función desconocida de la cual sólo se conoce la respuesta ante todas las combinaciones posibles de entradas. La función tendrá tantos términos como unos tenga el resultado de la tabla.

Circuitos integrados

Las compuertas no se fabrican ni se venden individualmente, sino en unidades llamadas **circuitos integrados**, también conocidos como IC o chips. Un IC es un trozo cuadrado de silicio de unos 5mm en el que se han depositado algunas compuertas. Estos suelen montarse en paquetes de plástico o cerámica rectangulares, cuyos lados largos cuentan con dos filas paralelas de

terminales conectadas a la entrada o salida de alguna compuerta del chip, a la alimentación eléctrica o a tierra.

Circuitos combinacionales o combinatorios

Un circuito combinacional es un conjunto de puertas lógicas interconectadas, cuya salida, en un momento dado, es función solamente de los valores de las entradas en ese instante. La aparición de un valor en las entradas viene seguido casi inmediatamente por la aparición de un valor en la salida, con un retardo propio de la puerta.

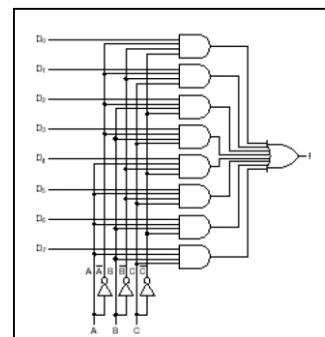
En general, un circuito combinacional consiste de n entradas binarias y m salidas binarias. Al igual que una compuerta lógica, un circuito combinacional puede definirse de tres formas:

- **Tabla de verdad:** Para cada una de las posibles combinaciones de las n señales de entrada, se enumera el valor binario de cada una de las m señales de salida.
- **Símbolo gráfico:** Describe la organización de las interconexiones entre puertas.
- **Ecuaciones booleanas:** Cada señal de salida se expresa como una función booleana de las señales de entrada (suma de productos).

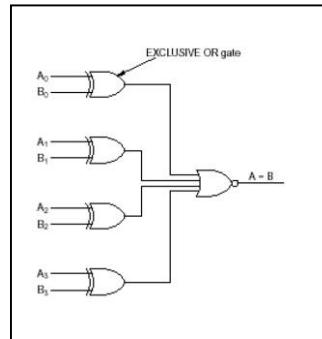
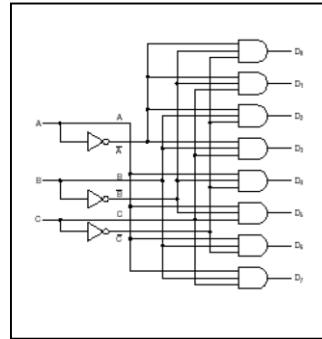
Los circuitos combinacionales implementan las funciones esenciales de una computadora digital. Sin embargo, ellos no proporcionan memoria, que es un elemento también esencial para el funcionamiento. Para estos fines, se utilizan circuitos lógicos digitales más complejos denominados circuitos secuenciales.

Los circuitos combinacionales cuentan con una o varias entradas y salidas. Los mismos responden a los valores lógicos en las entradas, y la o las salidas están determinadas exclusivamente por los valores de entrada en ese instante. Entonces, si cambia la entrada, cambia la salida. Además, los valores pasados de las entradas, no influyen en los valores de la salida. Por ejemplo, algunos circuitos combinacionales de uso frecuente son:

- **Multiplexores:** circuito con 2^n entradas de datos, 1 salida de datos y n entradas de control que seleccionan una de las entradas de datos. Estos pueden usarse para seleccionar una de varias entradas o para implementar una tabla de verdad. También como convertidor de datos paralelos a seriales.



- **Decodificadores:** estos son circuitos que aceptan un número de n bits como entrada y lo utiliza para seleccionar (poner en 1) una y sólo una de las dos líneas de salida. Por ejemplo, imagínate una memoria que consiste en 8 chips; un decodificador 3 a 8, tiene 3 entradas A, B, C para seleccionar uno de los 8 chips, puesto que, gracias a la combinación, sólo una línea de salida se pone en uno, sólo se podrá habilitar un chip a la vez.
- **Comparadores:** estos comparan dos palabras de entrada. En el ejemplo, el comparador acepta dos entradas cada una de 4 bits y produce 1 si son iguales y 0 si no.



Circuitos secuenciales

Los circuitos lógicos secuenciales se asocian al estudio de dispositivos de almacenamiento en general, en donde una de sus características principales es que sus salidas dependen de las entradas actuales, de entradas en tiempos anteriores y de una señal externa de reloj.

Las salidas dependen tanto de las entradas como del estado interno del circuito. Estos circuitos tienen la característica de almacenar valores lógicos internamente, y los mismos se almacenan, aunque las entradas lógicas no estén.

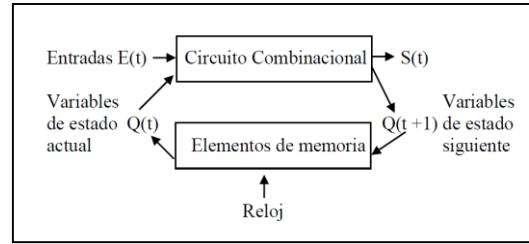
Conceptos Generales de Latches y flip-flops:

Los circuitos **biestables** son aquellos que poseen dos estados estables que se pueden mantener por tiempo indefinido, salvo que las entradas provoquen un cambio, lo que nos permite tener almacenado un dato en un dispositivo por el tiempo que se deseé. Según la manera en que las salidas respondan a las señales lógicas presentes en la entrada, los biestables se clasifican en **SR**, **J-K**, **D** y **T**.

Respecto del instante en que pueden cambiar dichas salidas, pueden ser:

- **ASINCRÓNICOS:** cuando en la entrada se establece una combinación, las salidas cambiarán.
- **SINCRÓNICOS:** la presencia de una entrada especial determina cuando cambian las salidas, acorde a las entradas.

Las salidas del circuito, además de ser función de las entradas son función de la información almacenada en elementos de memoria del circuito, en el momento que se producen las entradas. Están formados por un circuito combinacional y un bloque de elementos de memoria:



La señal del reloj indica a los elementos de memoria cuando deben cambiar su estado. Existen dos tipos de biestables muy importantes: el latch y el flip-flop. Estos circuitos están compuestos por compuertas lógicas y lazos de retroalimentación y son considerados los circuitos básicos que constituyen los sistemas digitales. El **latch** es un circuito biestable asincrónico, es decir que sus salidas cambian en la medida en que sus entradas cambien. El **flip-flop** es un dispositivo secuencial sincrónico que toma muestras de sus entradas y determina una salida sólo en los tiempos determinados por el reloj (CLK).

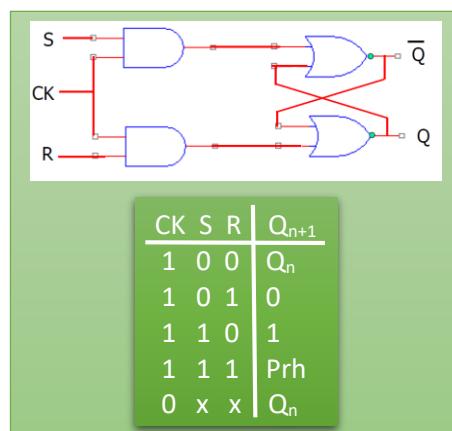
Latch SR: para crear una memoria de un bit necesitamos un circuito que de alguna manera “recuerde” los valores de entrada anteriores. Podemos construir un circuito así a partir de dos compuertas NOR. Este circuito tiene dos entradas S [set] para establecer el latch, es decir, ponerlo en 1, y R [reset] para restablecerlo (ponerlo en 0). También hay dos salidas, Q y \bar{Q} que son complementarias. A diferencia de los circuitos combinacionales, las salidas de un latch no están determinadas de forma única por las entradas vigentes.

Flip-Flop

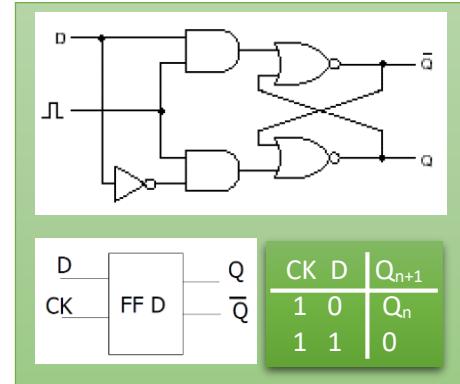
En muchos circuitos es necesario muestrear un valor que hay en una línea dada en un instante dado y almacenarlo. En esta variable, llamada Flip-Flop, la transición de estado no ocurre cuando el reloj es 1, sino durante la transición del reloj de 0 a 1 (flanco ascendente) o de 1 a 0 (flanco descendente).

¿Qué diferencia hay entre un Flip-Flop y un Latch? Un Flip-Flop se dispara por flanco, mientras que un Latch se dispara por nivel.

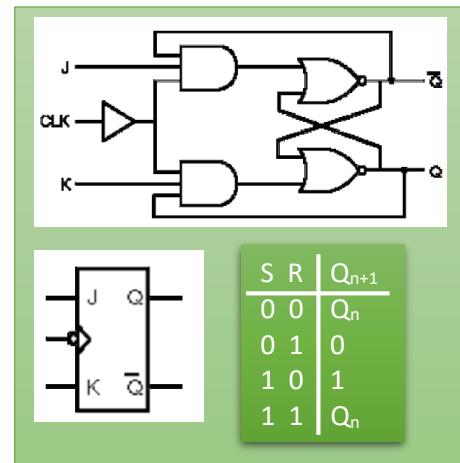
Flip-Flop SR: es un circuito biestable conformado por un detector de transición de impulsos que está encargado de detectar cuándo se tiene un flanco de subida o de bajada del reloj (CLK), dos compuertas AND y dos compuertas NOR. En estas compuertas NOR, una de las salidas está conectada a la entrada de la otra compuerta, logrando una retroalimentación.



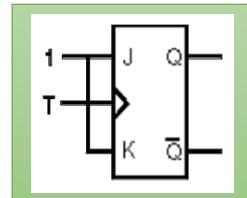
Flip-Flop D: en el Flip-Flop SR hay que aplicar dos entradas diferentes para cambiar el estado, el Flip-Flop D permite aplicar una sola entrada para cambiar la salida. El Flip-Flop D está compuesto por dos compuertas AND encargadas de enviar la señal de habilitación a dos compuertas NOR. La salida de una compuerta NOR se transforma en la entrada de la otra (retroalimentación). Hay una gran similitud con el Flip-Flop SR, sólo difieren en que este tiene una sola entrada de habilitación y en que la entrada de Reset es igual a la de Set negada.



Flip-Flop JK: El biestable SR presenta problemas cuando se activan simultáneamente las dos entradas S y R. Podemos diseñar un biestable similar que no presente estos problemas a partir de un biestable D. Este es el Flip-Flop JK. En este caso, para lograr un valor estable cuando se activan ambas entradas, se hace una retroalimentación de Q y \bar{Q} con las compuertas de la entrada.



Flip-Flop T: el Flip-Flop T [Toggle] mantiene su estado o lo cambia dependiendo de valor de T cada vez que se activa. La salida Q cambiará de 1 a 0 o de 0 a 1 en cada pulso de la entrada T. Se puede implementar utilizando un biestable JK.

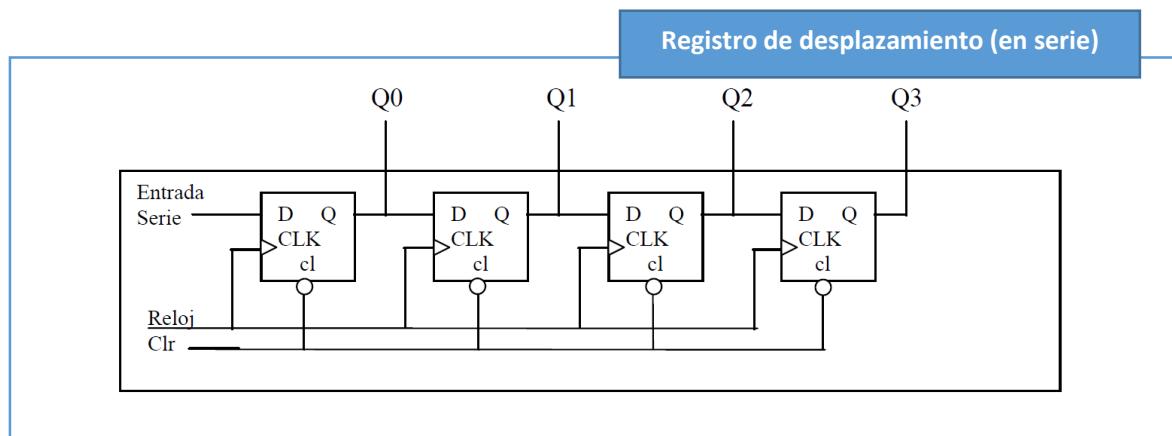
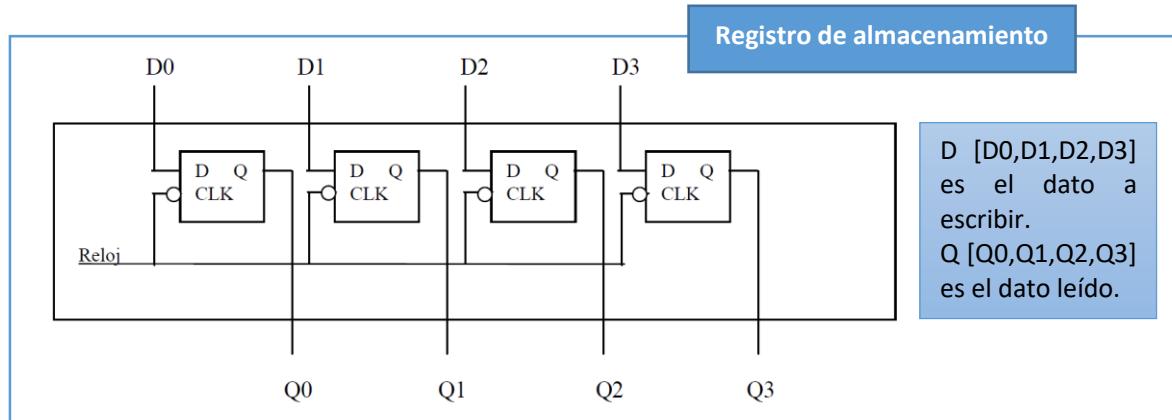


Relojes

En muchos circuitos secuenciales el orden en que ocurren los sucesos es crucial. A veces un suceso debe suceder a otro, u ocurrir dos sucesos en simultáneo. Para que los diseñadores puedan establecer las relaciones de temporización requeridas, muchos circuitos digitales emplean relojes que hacen posible la sincronización. Un reloj es un circuito que emite una serie de pulsaciones con una anchura de pulsación precisa y un intervalo preciso entre pulsaciones consecutivas. Es una señal de tiempo precisa que determina cuando se producen los eventos. El periodo entre los flancos correspondientes de dos pulsaciones consecutivas se denomina **tiempo de ciclo del reloj**.

Registros

Se forman a partir de biestables de tipo D conectados en cascada. Un registro con N biestables es capaz de guardar N bits. Estos son circuitos sincrónicos y todos los biestables están gobernados por la misma señal del reloj. Estos pueden ser de almacenamiento o de desplazamiento.



CICLO DE INSTRUCCIÓN

La computadora tiene como función ejecutar programas, los mismos están compuestos por instrucciones almacenadas en memoria. La CPU procesa estas instrucciones, trayéndolas desde memoria una por vez y luego cumpliendo cada operación ordenada.

Podemos descomponer el procesamiento de instrucciones en dos etapas:

- Etapa de búsqueda: se lee desde memoria y es común a todas las instrucciones.
- Etapa de ejecución: dependiendo de la instrucción puede implicar varias operaciones.

El procesamiento requerido para una sola instrucción se llama **CICLO DE INSTRUCCIÓN**, que consta de dos pasos: ciclo de búsqueda y ciclo de ejecución. La ejecución del programa se interrumpe sólo si la máquina se apaga, hay un error o hay una instrucción que interrumpe a la computadora.

Al principio de cada ciclo, la CPU busca una instrucción en memoria. En la CPU hay un registro llamado **Contador de Programa [PC]**, que tiene la dirección de la próxima instrucción a buscar. La CPU, después de buscar cada instrucción, incrementa el valor contenido en PC, así podrá buscar la siguiente instrucción en secuencia (de esta manera, el PC siempre va a estar apuntando a la próxima instrucción a ejecutar).

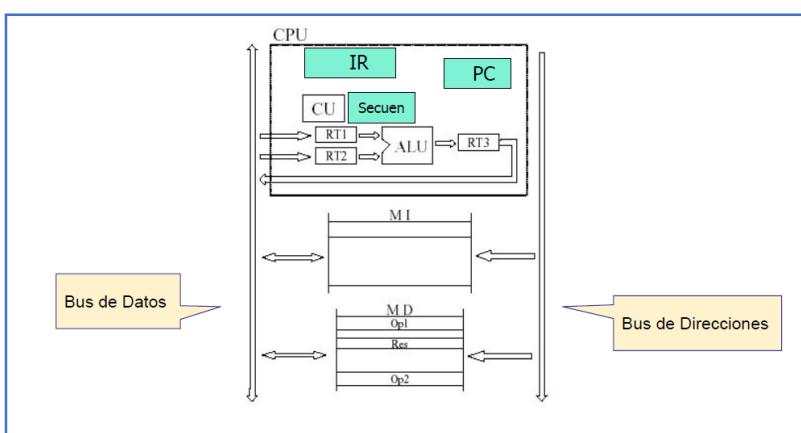
La instrucción buscada se carga dentro de un registro de la CPU, llamado **Registro de Instrucción [IR]**. Esta instrucción está en la forma de un código binario que especifica las acciones que tomará la CPU. La CPU interpreta cada instrucción y lleva a cabo las acciones requeridas. A esta interpretación se la llama “decodificar”. En general, dichas acciones caen en cuatro tipos:

- CPU-Memoria: los datos pueden transferirse entre memoria y CPU.
- CPU-E/S: los datos pueden transferirse entre el CPU y E/S.
- Procesamiento de datos: el CPU efectúa operaciones aritméticas o lógicas en los datos.
- Control: se altera la secuencia de ejecución de instrucciones (saltos).

Esquematización de búsqueda y ejecución en un ciclo de instrucción

El dibujo nos esquematiza una CPU y la memoria del sistema (falta el bloque de E/S).

Dentro del CPU esta la ALU, dos registros temporarios a la entrada de la ALU y uno a la salida



[RT1, RT2, RT3], luego los registros en celeste que son el Registro de Instrucciones [IR] y el Contador de Programa [PC]. Luego el otro bloque celeste son los secuenciadores que forman parte de la Unidad de Control [UC] que nos va a indicar en que paso estamos dentro de la ejecución de una instrucción. Luego la memoria dentro del esquema está separada en dos partes: la Memoria de Instrucciones [MI] y la Memoria de Datos [MD]. A la derecha e izquierda están los Buses: el Bus de Direcciones y el Bus de Datos (no está esquematizado el Bus de Control). Recordar que la CPU se conecta al Bus de Datos por medio del registro MBR; y se conecta al Bus de Direcciones por medio del registro MAR.

1. El primer paso en el ciclo de instrucción es la búsqueda de la instrucción. El contenido del PC se utiliza para direccionar la memoria y así poder hacer la lectura del código de la instrucción. La dirección viaja por el Bus de Direcciones desde la PC hasta la MI. Luego, a través del Bus de Datos va la información de la instrucción que vamos a ejecutar desde la CPU al IR.
2. El segundo paso es la decodificación de la instrucción, la información del código de operación que esta copiado en el IR, es decodificada por la UC para así saber lo que tiene que hacer a continuación. [ESTOS DOS PASOS SON COMUNES A TODAS LAS INSTRUCCIONES].

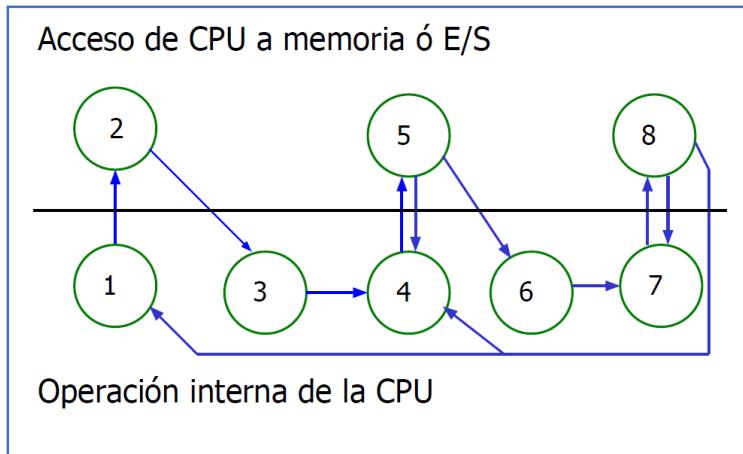
Supongamos que tenemos que hacer una operación aritmética con dos operandos que van a estar en memoria y luego de realizar la operación se guarda el resultado en memoria también.

3. Búsqueda del primer operando: parte de la información de la instrucción que acaba de leerse dice como calcular la dirección del primer operando. Entonces la CPU coloca la dirección del operando en el Bus de Direcciones hace un ciclo de acceso para la lectura de la memoria, y la memoria entrega el dato que viaja a través del Bus de Datos, ingresa a la CPU y llega al Registro Temporario 1 [RT1].
4. Búsqueda del segundo operando: se repite lo mismo que el paso anterior, pero con el segundo operando.
5. Operación aritmético-lógica realizada por la ALU, que luego coloca el resultado en el Registro Temporario 3 [RT3] que es el registro de salida de la ALU. (Si estuviera el registro de Flags, se actualizaría el mismo luego de hacer esta operación).
6. Almacenar el resultado: la instrucción nos dice dónde colocar el resultado, entonces la CPU coloca la dirección en el Bus de Direcciones y hace un ciclo de acceso para escritura de la memoria. Esta vez, es la CPU la que coloca el dato en el Bus de Datos que viaja por él y es copiado en la dirección de memoria que le corresponde ser guardado. Entonces el flujo en este caso es desde el RT3 hasta la Memoria de Datos [MD], haciendo la escritura del resultado.

7. Cálculo de la dirección de la próxima instrucción: se incrementa el Contador de Programa [PC], así este apuntará a la dirección de memoria siguiente a donde estaba la instrucción que acabamos de ejecutar. Y así se reinicia el ciclo para la próxima instrucción.

Diagrama de estados

Este diagrama muestra los distintos estados por los que pasa la CPU durante la ejecución de un ciclo de instrucción. No todos los ciclos de instrucción involucran



todos los estados, algunos estados pueden no darse y otros pueden llegar a repetirse.

Cada uno de estos círculos representan un estado dentro de un ciclo de instrucción, y las flechas nos marcan el flujo de los estados. Los estados por encima de la raya horizontal (2, 5 y 8) son estados que involucran manejo de los buses, es decir acceso de la CPU a la memoria o a E/S para leer o escribir información; y los estados que están por debajo de la raya horizontal (1, 3, 4, 6, 7) son operaciones internas de la CPU, es decir que no involucran el uso de los buses (sólo los buses internos de la CPU).

- **ESTADO 1: cálculo de la dirección de la instrucción.** Esto involucra utilizar el PC, incrementarlo para que apunte a la próxima instrucción a ejecutar y así poder acceder a ella. Una vez que tenemos la dirección de la instrucción pasamos al siguiente estado.
- **ESTADO 2: búsqueda de la instrucción.** Se hace un acceso al Bus de Direcciones con la dirección de la instrucción que se generó en el E1, se hace el acceso a la memoria para ir a buscar la instrucción. Esta instrucción se va a colocar en el IR (es decir la instrucción viaja de la memoria a la CPU y se guarda en el IR, mediante el Bus de Datos).
- **ESTADO 3: decodificación de la instrucción.** Se analiza la instrucción para determinar qué tipo de operación hay que hacer, que operandos y cuantos operandos se van a utilizar. Esto dependerá de que instrucción se trate. [ESTOS TRES ESTADOS SON COMUNES A TODAS LAS INSTRUCCIONES].
- **ESTADO 4: cálculo de la dirección del operando.** Si la instrucción implica la referencia de un operando en la memoria de E/S es acá donde se determina la dirección. Puede ser que la instrucción nos indique directamente donde está el operando, o puede que tengamos que realizar una cuenta antes para saber en dónde se encuentra dicho

operando. Este estado está asociado con los Modos de Direccionamiento. Además, si no necesitamos operando esta parte no sería necesaria.

- **ESTADO 5: búsqueda del operando.** Se realiza el ciclo de acceso a Memoria o a E/S para acceder a la información. Como involucra a la Memoria o E/S este estado está por encima de la línea horizontal, es decir, está involucrando el manejo de los buses. [Los estados 4 y 5 se pueden repetir, por ejemplo, si necesito buscar un segundo operando].
- **ESTADO 6: operación sobre los datos.** El o los operandos necesarios ya están en la CPU y ahora hay que realizar la operación sobre esos datos (suma, resta, multiplicación, etc.).
- **ESTADO 7: cálculo de la dirección del resultado.** Ya hicimos el cálculo en la CPU y ahora hay que calcular la dirección en la cual voy a guardar este resultado. La instrucción nos tiene que haber dado la información, de donde debo guardar el resultado. Una vez que tengo dicha dirección paso al último estado.
- **ESTADO 8: almacenamiento del resultado.** Se hace la operación de almacenar la información. En este estado vuelve a ser necesario el uso de los buses, pues debo enviar el resultado ya sea en Memoria o en E/S.

En algunas máquinas hay instrucciones que procesan arreglos de operandos (vectores). Entonces, luego de almacenar una componente del vector resultado [Estado 8] se procesa el siguiente componente del vector [Estado 4]. En este caso, la instrucción sólo se decodifica una vez y luego repite los estados del 4 al 8 para operar cada componente del vector.

EJEMPLO: supongamos que tenemos la siguiente instrucción **ADD A,B** en donde ADD es el código de operación; A es la referencia al primer operando, y referencia a un registro; y B es la dirección a la que hay que ir a buscar el segundo operando, que es una dirección de memoria. Esta instrucción tiene una determinada cantidad de bits, algunos van a estar asociados con el código de operación, otros bits para la referencia al registro, nos va a decir con cuál de los registros internos posibles vamos a trabajar, y otros bits estarán asociados a la dirección del segundo operando. El objetivo de esta instrucción es sumar esos dos operandos y guardar el resultado. Dado que acá no hay un tercer campo en el cuál se guardará el resultado, vamos a considerar implícita dicha información, entonces el resultado se guardará en la referencia al primer operando (es decir, se guardará en A).

- Primero vamos a buscar la instrucción en Memoria, para esto incrementamos el PC. La CPU busca la instrucción en memoria. Para esto copia el valor del PC al MAR y de ahí al Bus de Direcciones. La UC envía las señales necesarias para una operación de lectura. Se

pueden leer uno o más bytes. A través del Bus de Datos se trae la información que pasa por el MBR y llega al IR.

- Luego de buscar la instrucción, la CPU debe incrementar el PC para apuntar a lo que sigue, que puede ser un dato, una dirección (en el caso de que la instrucción actual involucre varias palabras) o la siguiente instrucción. (Puede que primero accedemos a memoria y luego incrementamos el PC para la próxima instrucción o al revés, primero incrementamos el PC y luego vamos a buscar la instrucción en la dirección de Memoria que nos indica el PC).
- Decodificamos la instrucción, para saber qué tipo de operación vamos a hacer (suma, resta, etc). En este momento la CPU no sólo se entera de la operación, sino también donde se encuentran los datos sobre los cuales operar. **La instrucción es AUTOCONTENIDA, en ella está toda la información que se necesita.** (que es lo que hay que hacer, con que operandos, donde están, donde colocar el resultado y donde está la próxima instrucción).
- Si es necesario buscar una constante en una dirección de memoria (en el caso de que se requiera voy a buscar una palabra, incremento el PC y luego voy a buscar otra palabra, es decir, debería apuntar más allá de la constante si esta se guarda en dos direcciones contiguas de memoria).
- Si es necesario, hay que calcular la dirección del operando. En el paso anterior la CPU ya determinó si tiene que ir a buscar un operando o constante a memoria, y si esta ocupa una celda o más bytes en memoria. Si hemos leído más palabras en la instrucción la CPU debe incrementar el PC en el valor adecuado de celdas.
- Buscar uno de los operandos desde memoria o registro.
- Buscar el otro operando desde registro.
- Realizar la suma
- Almacenar el resultado en A, que es la información implícita en esta instrucción.

FORMATO DE INSTRUCCIONES Y MODOS DE DIRECCIONAMIENTO

¿Cómo es el formato de una instrucción? ¿Cómo está guardada una instrucción en memoria? ¿Qué campos la componen? ¿Cómo se distribuyen los bits? Sabemos que todas las instrucciones son **AUTOCONTENIDAS**, esto significa que la instrucción da toda la información necesaria para realizar todas las acciones que se necesiten para llevar a cabo la instrucción, y para el encadenamiento con la próxima instrucción también.

Los elementos de una instrucción de máquina son:

- El **código de operación**, que es un código binario que el fabricante asigna para cada operación que se pueda llevar a cabo, y nos especifica la operación a realizar.
- La **referencia del operando fuente**, es una determinada cantidad de bits que nos van a permitir encontrar **dónde está el operando fuente**. La mayoría de las operaciones aritméticas o lógicas involucran dos operandos, entonces vamos a tener que poder especificar en la instrucción dónde está el operando 1 y el operando 2 (si existiese).
- La **referencia del operando resultado**, nos indica dónde colocar el resultado.
- La **referencia a la próxima instrucción**, para saber dónde está la próxima instrucción. Lo natural es tener un Contador de Programa [PC] que nos indique la dirección de la próxima instrucción a ejecutar, puesto es que lo normal es que una instrucción este a continuación de la otra en posiciones consecutivas de memoria (en modo secuencial). Pero esto no siempre es así, puesto que por ejemplo en el caso de haber instrucciones de control condicionales esto no se cumpla.



Los operando fuente y resultado pueden estar en tres lugares: pueden estar en Memoria, en una Registro de la CPU, o dentro de los circuitos de E/S.

Cada instrucción dentro de la CPU está representada por un cierto número de bits que nos van a indicar todo lo necesario referente a la instrucción (¿Qué hacer? ¿Con qué? ¿Dónde va el resultado? ¿Dónde está la próxima instrucción?). Este conjunto de bits que codifican la instrucción lo podemos considerar dividido en campos, donde cada campo nos da la información necesaria para cada una de esas cuestiones que tenemos que especificar. El esquema que nos dice como están distribuidos esos campos es lo que se conoce como **FORMATO DE INSTRUCCIÓN**. Vamos a tener formatos de instrucción distintos, con más o menos campos y hay que analizar cuantos bits se asignarán para cada campo. La totalidad de los bits de cada campo nos darán el largo de la instrucción.

En programación es muy incómodo estar trabajando en lenguaje de bits, entonces lo que se hace es usar una representación simbólica, por ejemplo, **ADD para la suma, MUL para la multiplicación, etc.** Estos se llaman **Nemáticos**. Del mismo modo, los operandos también

conviene representarlos con una representación simbólica, por ejemplo, MOV Reg1, Mem1 es una instrucción que copia lo contenido en la dirección de memoria Mem1 a un registro Reg1. Obviamente tiene que estar previamente definido que se entiende por Mem1 y Reg1. En este ejemplo tenemos bits asignados al código de operación, otros al operando fuente y otros para el operando resultado. ¿Qué se va a codificar en la memoria? La secuencia de n bits que forman dicha instrucción.

Ejemplo en lenguaje de alto nivel: $X := X + Y \rightarrow$ Esta instrucción suma los valores almacenados en las posiciones de memoria X e Y, para luego guarda el resultado en X. Esto puede implicar cargar registros, sumarlos y luego almacenar el resultado en memoria. Hay que tener en cuenta que una instrucción en alto nivel puede requerir varias instrucciones de máquina.

La diferencia entre el lenguaje de alto nivel y el lenguaje de máquina (bajo nivel) es que el lenguaje de alto nivel expresa operaciones en forma concisa usando variables, mientras que el lenguaje de máquina expresa las operaciones de forma básica involucrando movimiento de datos y uso de registros. Además, cualquier programa escrito en lenguaje de alto nivel se debe convertir en lenguaje de máquina para ser ejecutado, pues este lenguaje es cómodo para el programador, pero el procesador no puede ejecutar eso directamente, lo que puede ejecutar el procesador son las instrucciones de máquina utilizando los registros internos de la CPU. El conjunto de instrucciones de máquina debe ser capaz de expresar cualquiera de las instrucciones de un lenguaje de alto nivel.

Podemos categorizar las instrucciones de máquina en:

- **Procesamiento de datos:** instrucciones relacionadas con operaciones aritméticas y lógicas.
- **Almacenamiento de datos:** instrucciones relacionadas con transferencias dentro del sistema, entre la memoria y el CPU.
- **Instrucciones de E/S:** en algunos casos, la arquitectura requiere una salida específica para la transferencia de datos entre la computadora y los mecanismos externos.
- **Instrucciones de control:** manejan el flujo del programa, por ejemplo, instrucciones de salto o ramificación. También hay instrucciones de control que modifican registros especiales que afectan el comportamiento de la CPU, y esas instrucciones también entran dentro de esta categoría.

¿Qué número de direcciones necesito en una instrucción de máquina? En principio las direcciones que se necesitan son cuatro: dos direcciones para hacer referencia a los operandos fuente, una que hace referencia al operando resultado y la dirección de la próxima instrucción (y además necesitamos bits para indicar el Código de Operación, pero esto no es una dirección).

Este sería el caso de una máquina de 4 direcciones, aquí no hay Contador de Programa, la dirección de la próxima instrucción está contenido en la instrucción actual. Tenemos direcciones explícitas para operandos fuentes y operando resultado, y para la próxima instrucción. Estas son muy raras puesto que cada campo de dirección necesita tener bits para acomodar una dirección completa. Entonces, por ejemplo, si cada dirección ocupa 24 bits, necesito $24 \times 4 = 96$ bits por instrucción para indicar las direcciones.

| | | | | |
|----------------|----------------|----------|----------|----------------|
| Cód. Operación | Dir. Resultado | Dir. Op1 | Dir. Op2 | Dir. PróxInstr |
|----------------|----------------|----------|----------|----------------|

Ya sabemos que las instrucciones en un programa se ordenan en forma de secuencia entonces si sabemos que las mismas se encuentran en posiciones contiguas de memoria bastaría contar con un registro en el CPU llamado Contador de Programa, que vaya haciendo el encadenamiento de instrucción a instrucción, entonces de este modo cada vez que leemos una instrucción incrementamos el PC y de este modo siempre estará apuntando a la próxima instrucción. Entonces la dirección de la próxima instrucción estará implícita en el PC, y ahora necesitaría sólo 3 direcciones por cada instrucción de máquina. Esta es una máquina de 3 direcciones. Ahora se acorta la cantidad de bits que tengo por instrucción dedicado a direcciones, siguiendo el ejemplo anterior de 24 bits por dirección, necesito $24 \times 3 = 72$ bits por instrucción para indicar las direcciones en una máquina de tres direcciones.

| | | | |
|----------------|----------------|----------|----------|
| Cód. Operación | Dir. Resultado | Dir. Op1 | Dir. Op2 |
|----------------|----------------|----------|----------|

Luego tenemos las máquinas de 2 direcciones, ¿Cómo podemos reducir el tamaño de la instrucción en base a los bits dedicados a las direcciones? Podemos tener alguna otra información implícita. Esta información implícita sería la dirección en la cual se guardará el resultado, la misma va a ser la dirección del primer operando. [ADD AX,BX → de modo implícito está indicado que la suma entre el contenido del registro AX y el registro BX va a ser guardado en la dirección del primer operando, es decir dentro del registro AX.] En este caso y siguiendo la idea de 24 bits por dirección, estaríamos dedicando $24 \times 2 = 48$ bits a las referencias de direcciones. Tenemos menos elección para guardar el resultado porque este va a ser guardado en la dirección referenciada por el operando 1. La manera de compensar esta limitación es tener una operación que nos permita mover la información contenida en el lugar al que apunta esa dirección (por ejemplo, un MOV).

| | | |
|----------------|--------------|----------|
| Cód. Operación | Dir. Op1/Res | Dir. Op2 |
|----------------|--------------|----------|

Por último, está el caso de las máquinas de 1 dirección, tenemos que suponer implícita otra información más para así sacar otro campo dedicado a una dirección. Esta máquina dispone de un registro especial en la CPU llamado acumulador que es uno de los operandos de la operación y además el destino para el resultado (cumple el rol del operando 1 en la máquina de dos direcciones, sólo que en este caso es un registro de la CPU). Este operando lo vamos a ir a buscar en memoria y lo vamos a colocar en un registro interno acumulador, para no perder flexibilidad vamos a necesitar un código de operación para cargar el acumulador [LOAD] y otro para descargarlo [STORE].

Cód. Operación Dir. Op2

RESOLVER LA SIGUIENTE ECUACIÓN → A = (B + C) * D - E

| Máquina de 3 direcciones | Máquina de 2 direcciones | Máquina de 1 dirección |
|--|--|---|
| <p>ADD A, B, C 1ac MI / 3ac MD MUL A, A, D 1ac MI / 3ac MD SUB A, A, E 1ac MI / 3ac MD</p> <p>3 INSTRUCCIONES → EN TOTAL SON 3 ACCESOS A MI Y 9 ACCESOS A MD</p> | <p>MOV A, B 1ac MI / 2ac MD ADD A, C 1ac MI / 3ac MD MUL A, D 1ac MI / 3ac MD SUB A, E 1ac MI / 3ac MD</p> <p>4 INSTRUCCIONES → EN TOTAL SON 4 ACCESOS A MI Y 11 ACCESOS A MD [la 1er instrucción accede a MD dos veces, una para buscar los datos de B y otra para dejarlos en A; las otras acceden 3 veces, 2 para buscar operandos y luego una vez más a A para dejar el resultado]</p> | <p>LOAD B 1ac MI / 1ac MD ADD C 1ac MI / 1ac MD MUL D 1ac MI / 1ac MD SUB E 1ac MI / 1ac MD</p> <p>STORE A 1ac MI / 1ac MD</p> <p>5 INSTRUCCIONES → EN TOTAL SON 5 ACCESOS A MI Y 5 ACCESOS A MD</p> |

El conjunto de instrucciones es el medio que tiene el programador para poder controlar la CPU.

El fabricante tuvo que tener determinadas consideraciones:

- Que tipos de operaciones se pueden manejar, cuáles [si por ejemplo quiero multiplicar y no tengo una operación para eso, deberé implementar otro modo de hacerlo como por ejemplo sumas sucesivas] y cuántas [si yo tengo muchas operaciones voy a necesitar más bits para codificarlas]. Tenemos distintos tipos de operaciones posibles:
 - Instrucciones de transferencia de datos: MOV, LOAD, STORE
 - Instrucciones aritméticas: ADD, SUB, MUL, INC, DEC
 - Instrucciones lógicas: AND, NOT, XOR, OR
 - Instrucciones de conversión: por ejemplo, de binario a decimal.
 - Instrucciones para manejo de Entrada y Salida de datos: son para transferir datos entre la E/S y la CPU. Por ejemplo, IN, OUT

- Instrucciones de transferencia de control: por ejemplo, las de bifurcación (condicionales) o los saltos.
- Instrucciones de control de sistema: estas son las que maneja el SO
- Que tipos de datos vamos a trabajar, puesto que las instrucciones tienen que dar soporte a distintos tipos de datos. Los tipos de datos más importantes son direcciones, números (entero, punto fijo, etc), caracteres (ASCII, BCD) y datos lógicos.
- Cuál va a ser el formato de las instrucciones en esta arquitectura [longitud (cantidad de bits), número de direcciones, tamaños de cada campo].
- Uso de registros, cantidad que se pueden referenciar mediante las instrucciones y uso específico o general de los mismos.
- Modos de direccionamiento, esto es la manera que tenemos de especificar la dirección de un operando o de una instrucción.

Modos de direccionamiento

Como ya vimos, en una instrucción se necesitan bits para expresar el Código de Operación, que indica la operación que vamos a hacer y otra cantidad de bits para especificar de donde provienen los datos que vamos a utilizar en esta instrucción. ¿Cómo se puede reducir el tamaño de estas especificaciones? Hay dos métodos generales:

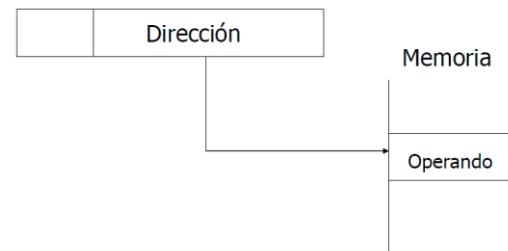
1. Si un operando va a usarse varias veces, puede colocarse en un registro dentro de la CPU, esto tiene como ventaja que el acceso será más rápido, puesto que todo lo que está colocado en registro del CPU se accede más rápidamente; y además voy a necesitar menos bits pues por ejemplo si tengo 32 registros dentro del CPU me alcanzan 5 bits para referenciar al operando contenido en el registro del CPU, pues $2^5 = 32$. De este modo reduzco el tamaño de las instrucciones.
2. Puedo especificar uno o más operandos en forma implícita, por ejemplo, en el caso de las máquinas de una dirección con el registro acumulador lo que hago es ahorrarme bits en la instrucción que me refieran las instrucciones de uno de los operando fuentes y el operando resultado.

Por lo general los modos de direccionamiento tienen como objetivo: disminuir la cantidad de bits dentro de la instrucción; lograr que existan direcciones que no precisan conocerse hasta la ejecución del programa (por ejemplo, no necesito saber la dirección del operando para escribir un programa, sólo con usar la representación simbólica del operando puedo escribir mi programa y después la CPU se encargará de decodificar las instrucciones); poder manejar los datos de manera más eficiente, por ejemplo, en las estructuras de datos de tipo arreglos.

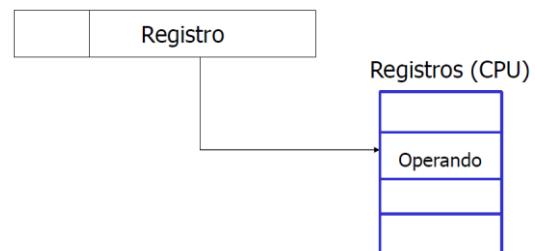
- **Modo de Direccionamiento Inmediato [MOV AL , 4]**: en este modo de direccionamiento el operando esta codificado en la instrucción, es decir, se obtiene automáticamente de la memoria al mismo tiempo que la instrucción. En este caso no hay que ir a hacer una referencia posterior a memoria, cuando nos llevamos la instrucción a la CPU, llevamos codificado dentro de la instrucción el operando. Este modo de direccionamiento se utiliza para definir constantes y también para inicializar variables, por ejemplo. La desventaja es que el tamaño del operando va a estar limitado al tamaño de ese campo de direccionamiento, entonces, si por ejemplo el campo de direccionamiento para ese operando es de 16 bits, tengo que tener en cuenta que el operando no puede ocupar más de 16 bits.



- **Modo de Direccionamiento Directo [ADD BX , NUM]**: en este caso se nos da dentro de la instrucción en el campo de dirección, directamente la dirección del operando, entonces usamos esa dirección para ir a buscar el operando a la Memoria. Por ejemplo, si el operando dice que es 11FFH entonces el CPU va a ir a buscar el operando (o la dirección a donde colocar el resultado) a la dirección de memoria 11FFH. Es un modo simple, pero de nuevo estaremos limitados por la cantidad de bits del campo de dirección de esta instrucción. Este modo de direccionamiento se usa para acceder a variables globales cuya dirección se conoce al momento de la compilación. En estos casos por lo general se va a utilizar una representación simbólica que indique a qué dirección ir a buscar el operando fuente o resultado (es decir, el nombre de la constante o variable).

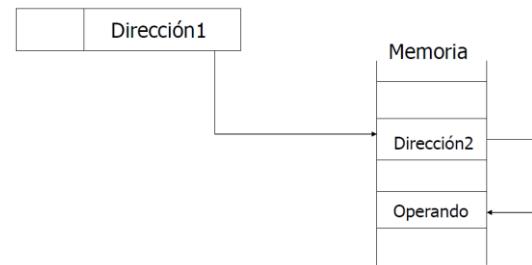


- **Modo de Direccionamiento por Registro [ADD AL , BL]**: en el campo de dirección tenemos una referencia a un registro interno del CPU. Con esa referencia vamos a buscar el dato, sólo que esta vez no tenemos que acceder a Memoria, sino a los registros de la CPU. Como los registros del CPU son pocos comparados con la Memoria, entonces este campo de dirección referenciando a un registro va a ocupar menos bits. Conceptualmente es igual al Modo de Direccionamiento Directo, sólo que se especifica un registro interno de la CPU en lugar de una posición de memoria. La referencia a registro ocupa menos bits que la especificación de la dirección y no



requiere acceso a memoria de datos. La desventaja es que los registros no son muchos, entonces es un recurso limitado, por lo tanto, hay que analizar bien como usarlos.

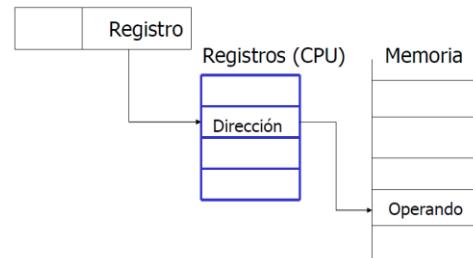
- **Modo de Direccionamiento Indirecto por Memoria:** en este caso no vamos directamente a la dirección de memoria que necesitamos, sino que tenemos que dar un paso intermedio antes. Es decir, en el campo de la instrucción, ya no nos dan la dirección efectiva del operando, sino que nos dan una dirección que tiene guardada la dirección efectiva del operando.



Entonces se lee la dirección y se accede a

Memoria, y ahí se hace un segundo acceso a Memoria para acceder al operando. ¿De qué nos sirve que en la instrucción este la dirección de la dirección del operando? Es para tratar de solucionar el problema del Modo de Direccionamiento Directo, y así, con una dirección de menos bits en la instrucción, se apunta a una dirección de más bits. La ventaja es que tendremos un espacio de direccionamiento mayor, pues el sitio en memoria al que esta apuntando el campo de mi instrucción puede tener más bits. Pero la desventaja es que voy a tener más accesos a memoria, entonces, este modo de direccionamiento será más lento que el modo de direccionamiento Directo. Otra ventaja es que no necesito conocer la dirección efectiva del operando en el momento en el que escribo mi programa.

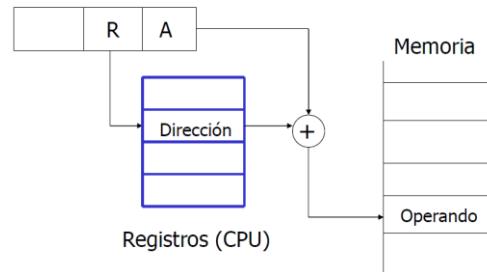
- **Modo de Direccionamiento Indirecto vía Registro [ADD AL, [BX]]:** en este caso, a través de un registro accedemos a una dirección de memoria. Es decir, el campo de la instrucción tiene la dirección a un registro de la CPU que guarda una dirección de memoria, en la cual tenemos guardada el operando que queremos usar, esta es la dirección efectiva del operando. La ventaja es que son muy pocos bits para especificar el registro. Tenemos un espacio de direccionamiento grande y se accede una sola vez a memoria, para ir a buscar el operando, ya que el paso indirecto se da adentro de la CPU entonces es más rápido.



En este caso el registro sería un registro puntero pues apunta a la dirección efectiva del operando.

- **Modo de Direccionamiento por Desplazamiento:** este modo implica especificar dos cosas, un registro involucrado y una cantidad que hay que sumársela al contenido del registro para encontrar la dirección efectiva del operando. Entonces la dirección efectiva

de aquello que queremos referenciar se calcula con el contenido de un registro más una cantidad binaria que se le suma. El registro puede estar implícito o no, dependiendo de que instrucción sea y del procesador. Si bien se están dimensionando dos conjuntos de bits, uno que nos referencia la dirección de un registro y otro que nos indica una cantidad que sería el desplazamiento, estamos especificando sólo una dirección. Este modo de direccionamiento combina las capacidades de los modos de direccionamiento indirecto y directo. Hay distintos Modos de Direccionamiento por Desplazamiento, los más comunes son:



- **MDD por Desplazamiento Relativo:** el registro que está involucrado en el direccionamiento está implícito y es el Contador de Programa. Entonces la dirección de la instrucción actual se suma al campo de dirección (que sería el desplazamiento) para producir la dirección efectiva. Este campo de dirección está dado en Ca2. Entonces nos permite referenciar posiciones relativas a donde estamos parados en el programa. Esto tiene mucha utilidad, por ejemplo, cuando estamos manejando instrucciones de salto, por lo general no saltamos a lugares muy alejados del que nos encontramos, entonces este campo que nos indica que tanto tenemos que saltar puede ser de pocos bits. Además, al ser el PC el registro implícito, esto nos permite construir código que se llama reubicable, es decir el Contador de Programa yo lo corro más arriba o más abajo en memoria, y toda la secuencia de instrucciones que conforman el programa, al cambiarse el PC no importa si es más arriba o más hacia abajo, se va a mantener y va a seguir referenciando la misma dirección.
- **MDD por Desplazamiento de Registro Base:** el registro referenciado contiene una dirección de memoria y el campo de dirección tiene un desplazamiento. Esto es muy útil para trabajar por ejemplo con la estructura arreglo. Supongamos que yo quiero trabajar siempre con un dato que está en la posición 4 de un arreglo, y yo quiero trabajar con distintos arreglos, pero siempre con el dato de la posición 4, entonces puedo ir cambiando la dirección del registro, pero el desplazamiento siempre será el mismo.
- **MDD por Desplazamiento Indexado:** acá también se direcciona la memoria con un registro más un desplazamiento. En esencia es igual desplazamiento por Registro Base, lo que cambia es que se interpreta por desplazamiento y por

dirección base. Acá el desplazamiento estaría dado por el contenido del registro, y la dirección base está dada por el campo A. Entonces la dirección base (desde donde comienzo) va a estar fija, y el desplazamiento que se lleve a cabo va a depender del contenido del registro. Este registro lo puedo ir variando por ejemplo en un lazo (iteración) y entonces así voy accediendo a distintos lugares de memoria en base a los distintos desplazamientos que se van dando. En el caso de un arreglo, en el campo de la instrucción especificamos la dirección del comienzo del arreglo y en el registro (que vamos a llamar registro índice) vamos a tener la información que nos indique como nos vamos a desplazar dentro de ese arreglo. Por esto mismo es que podemos decir que la Indexación proporciona un mecanismo eficiente para realizar operaciones iterativas. Algunas máquinas pueden realizar una auto-indexación, es decir, puede incrementar o decrementar este registro como parte de la instrucción.

- **Modo de Direccionamiento del Stack:** el Stack es un área de memoria que está administrada como una estructura de tipo Pila, en donde el último en entrar es el primero en salir, es decir, la información dentro de la estructura se va apilando. Entonces vamos a tener un registro puntero de Pila que apunta a una dirección. Cada vez que meto algo en la pila, al puntero se le cambia el valor para que apunte a la dirección en donde está guardada la última información ingresada. Este registro llamado Puntero de Pila [Stack Pointer] está implícito y se incrementa/decrementa automáticamente. Este puntero siempre va a apuntar al tope de la pila.

REGISTROS

Dentro de la CPU tenemos disponibles **registros que son visibles para el usuario**, es decir, para el programador, y otros que son **registros de control y estado**, que son utilizados por la UC para controlar las operaciones de la CPU y no son visibles por el programador.

¿A qué le llamamos registros visibles? Estos son los que pueden ser afectados por el programador directamente a través de las instrucciones. Si yo tengo una instrucción de suma sobre un registro AX con un operando inmediato, se puede decir que el registro AX es visible para mí porque yo puedo afectarlo mediante una instrucción. Ahora un registro temporal que este en la entrada de la ALU que es necesario para alojar temporalmente un dato hasta que se haga la operación, este registro yo puedo suponer que existe (ya que lo vi en la clase de Organización de computadoras) pero no tengo modo de que pueda manejarlo. Por ejemplo, el Registro de Instrucciones [IR] es un registro de control y estado porque no puedo controlar lo que está dentro de ese registro, aunque sepa para qué sirve y sepa que existe.

Registros visibles para el usuario: tenemos varios tipos

- **Registros de propósito general:** tienen múltiples funciones, en principio pueden ser usados para cualquier operando y ese operando puede utilizarse para cualquier código de operación. A veces pueden existir restricciones (por ejemplo, registros sólo usados para operandos en Punto Flotante). También se pueden utilizar para direccionamiento (por ejemplo, Direccionamiento Indirecto vía Registro).
- **Registros exclusivos para trabajar con datos.**
- **Registros para alojar direcciones:** estos pueden ser asignados para un Modo de Direccionamiento con Desplazamiento, por ejemplo, el registro índice para el MDD auto-indexado.
- **Registros de estado en donde están almacenados los códigos de condición.**

¿Conviene tener todos los registros para propósito general o conviene especializar su uso? La primera respuesta que tenemos es que, si todos los registros son de propósito general, esto va a afectar el tamaño de las instrucciones. Ahora si tengo registros especializados, puede ahorrar bits para los casos especializados en el que necesite menos bits, pero se limita la flexibilidad del programador.



La **cantidad** de registros que tenga en la CPU es importante pues:

- Afecta el tamaño de la instrucción (pues necesito más bits para el campo de dirección para el registro).
- Si tengo una mayor cantidad de registros, voy a precisar un mayor número de bits para poder codificarlos.

- Si tenemos pocos registros, significa que no voy a poder tener muchos datos dentro de la CPU; entonces voy a tener que ir permanentemente a Memoria a buscar mis datos.
- El número óptimo de registros suele ser entre 8 y 32 registros (direcciones de 3 y 5 bits).

En cuanto a la **longitud** de los registros:

- Los registros de direcciones deben ser capaces de almacenar la dirección más grande que podamos trabajar. Si, por ejemplo, las direcciones con las que va a trabajar nuestro procesador son de 32 bits, nuestros registros tienen que tener espacio suficiente para guardar estas direcciones de 32 bits.
- Si los registros guardan datos, la idea es que puedan guardar la mayoría de tipos de datos que trabaje ese procesador. En algunas máquinas se permite que un par de registros puedan estar almacenando partes de un mismo dato, es decir que estos registros estén de manera contigua y actúen como un solo registro, para poder trabajar un dato de doble longitud.

El CPU también cuenta con **bits de condición [Flags]**:

- Estos son establecidos por la CPU como resultado de operaciones.
- Pueden ser utilizados por las instrucciones de bifurcación condicional, por ejemplo, saltar si hubo Overflow.
- Generalmente no son alterados por el programador, sólo son utilizados para sus instrucciones condicionales. Igualmente hay instrucciones para alterar estos bits pero no es una práctica tan común.

Respecto a los registros de **control y estado**:

- Hay algunos de estos registros que son básicos para el funcionamiento de la CPU, como el Registro Contador de Programa **[PC]**, el Registro de Instrucción **[IR]**, el Registro Buffer de Memoria **[MBR]** que vincula la CPU con el Bus de Datos o el Registro de Dirección de Memoria **[MAR]** que vincula la CPU con el Bus de Direcciones. **Estos 4 registros son esenciales puesto que se emplean para el movimiento de datos entre la CPU y la Memoria.**

EJEMPLOS DE ORGANIZACIONES DE REGISTROS EN DOS PROCESADORES

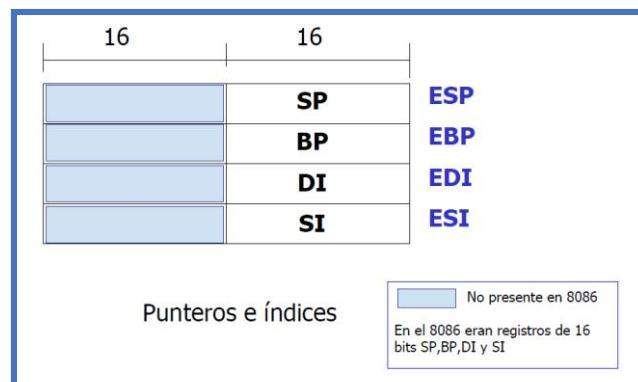
Intel - Línea de procesadores 8086

Fue el primer procesador de esta línea, si bien los siguientes modelos que salieron fueron agregando complejidades a su CPU; fue siempre conservando la posibilidad de funcionar en un modo compatible con los programas hechos para los primeros procesadores. En el modelo de

registros de esta línea, los registros en azul (de 32 bits) no estaban presentes en el 8086, aparecen más adelante. Los registros eran de 16 bits: AX, BX, CX, DX (sólo se muestran los registros visibles para el programador). Estos **4 registros son de uso general**. También puedo hablar, en lugar de AX, de AH y AL, que

son registros de 8 bits cada uno. El uso es indistinto y puedo mezclar registros de todo tipo. En el caso del MSX88, que simula el Intel 8086, no existen los registros de 32 bits (EAX, EBX, ECX, EDX), sólo los de 16 bits y 8 bits.

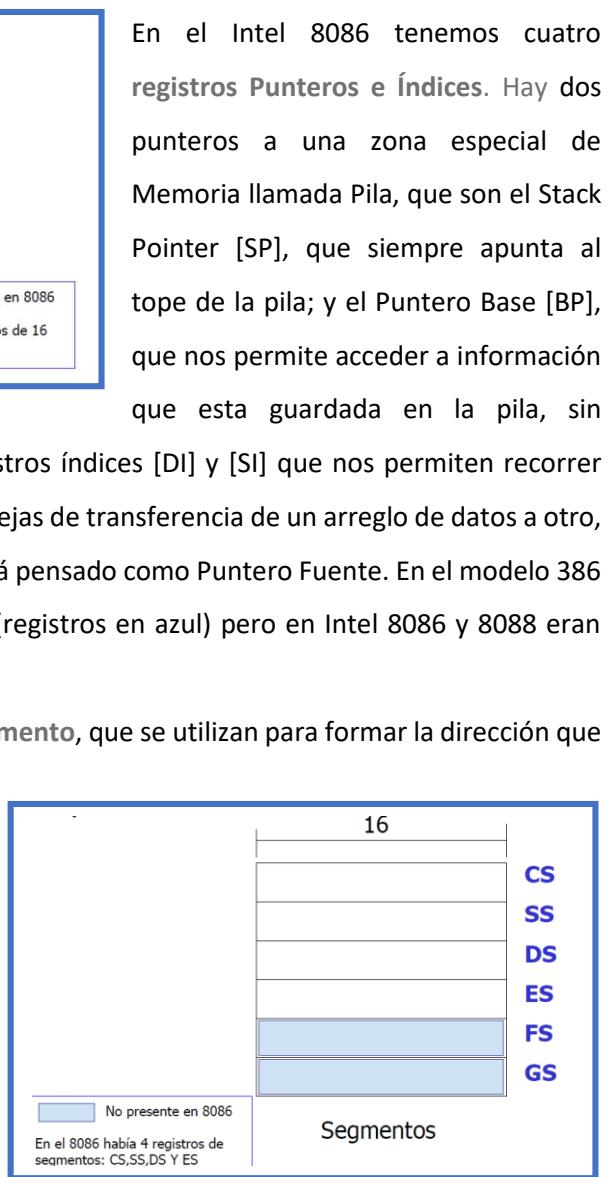
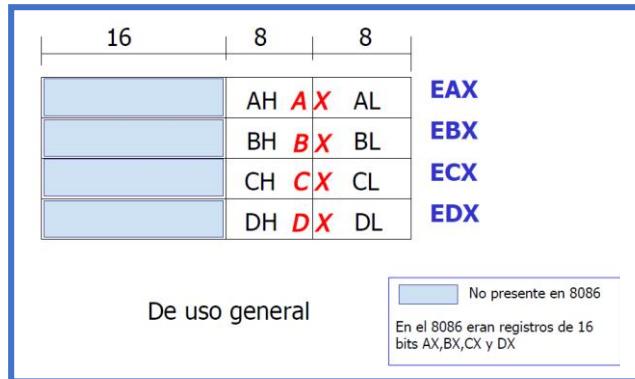
¿Qué significa que los registros se puedan ver como EAX, AX, o AH y AL? Es más versátil a la hora de trabajar con datos o direcciones que ocupan más o menos bits.



En el Intel 8086 tenemos cuatro **registros Punteros e Índices**. Hay dos punteros a una zona especial de Memoria llamada Pila, que son el Stack Pointer [SP], que siempre apunta al tope de la pila; y el Puntero Base [BP], que nos permite acceder a información que está guardada en la pila, sin necesidad de desapilar. Luego hay dos registros índices [DI] y [SI] que nos permiten recorrer arreglos. Por ejemplo, en operaciones complejas de transferencia de un arreglo de datos a otro, el DI funciona como Índice Destino y el SI está pensado como Puntero Fuente. En el modelo 386 en adelante, estos son registros de 32 bits (registros en azul) pero en Intel 8086 y 8088 eran registros de 16 bits.

Están también los llamados **Registros de Segmento**, que se utilizan para formar la dirección que sale al Bus de Direcciones. En el Intel 8086, había 4 registros de segmento, luego se agregaron dos más. Originalmente se usaban para completar la generación de la dirección. El bus de direcciones en el 8086 era de 20 bits, es decir, las direcciones eran de 20 bits, pero los registros eran de 16 bits.

¿Cómo se complementan los 20 bits de



direcciones? Lo que se hacía era sumar el contenido de un registro de segmento con el registro de direcciones que estábamos usando para el direccionamiento. Por ejemplo, si queríamos acceder a la pila se usaba el registro de Segmento de Stack [SS] y el Stack Pointer [SP]. Ahora, ¿Cómo logramos los 20 bits del Bus de Direcciones? Esta suma se hace desplazada, el registro de Segmento estaría aportando 20 bits donde los cuatro bits menos significativos siempre valen 0 y a eso se le suma el registro usado en el direccionamiento.



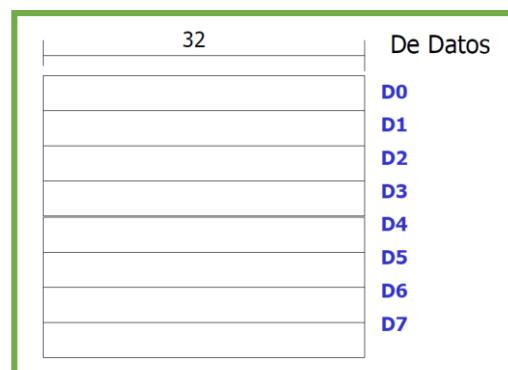
Luego tenemos el Registro Contador de Programa [IP] que en el entorno Intel se llama distinto, y el Registro de Flags. En Intel 8086 eran registros de 16 bits y luego después de 386 en adelante eran de 32 bits (en azul).

Los registros llamados de propósito general (AX, BX, CX, DX), si bien son de uso general, tienen funciones particulares cada uno de ellos.

- Registro **AX**: es el registro **Acumulador**, es el que se usa principalmente en las operaciones aritméticas.
- Registro **BX**: es el que funciona como **Puntero Base**, es el único que me sirve para el Direccionamiento Indirecto vía Registro [BX].
- Registro **CX**: tiene función específica de **Contador** para las instrucciones de iteración (lazos).
- Registro **DX**: es un registro **complementario del AX** en el caso de instrucciones para multiplicación y división.

Motorola - Línea de procesadores 68000

Esta es otra línea de procesadores que es contemporánea con el Intel 8086. Motorola 68000 es el primer integrante de esta familia, que tenía una determinada organización de registros que los nuevos modelos seguirán manteniendo, pero con cambios en los tamaños del Bus de Datos y Direcciones, mejoras en la velocidad del reloj, y demás. En este modelo hay 8 registros (D0 a D7) que son de propósito general para datos. No hay ninguno de estos registros que tenga una función especial y todos son de 32 bits.



Luego hay 9 registros (A0 a A7 + A7') que son de propósito general para direcciones. El registro

| De | A0 Direcciones |
|--------------------------------|----------------|
| | A1 |
| | A2 |
| | A3 |
| | A4 |
| | A5 |
| | A6 |
| | A7 |
| | A7' |
| Apuntador del stack usuario | |
| Apuntador del stack supervisor | |

A7 tiene dos versiones que son dos punteros de pila. Hay un puntero que es para el usuario (programador) y otro puntero de uso más privilegiado que es para supervisor. Sería para dos modos de trabajo, el de correr los programas normales y el de correr los programas de más nivel que serían los del SO. Luego, los restantes registros

de direcciones de A0 a A6, funcionan como punteros en donde se alojan las direcciones completas y da lo mismo usar cualquiera de ellos.



Finalmente tenemos el registro Contador de Programa [PC] y el registro de Estado.

Instrucciones del Intel 8086 más a detalle: mayormente las instrucciones aritméticas y lógicas precisan dos operando (no todas). Estos son los operandos destino y fuente. Entonces estaríamos trabajando con instrucciones de dos direcciones, en donde para cada uno de los operandos está especificado por los Modos de Direccionamiento vistos. Llamando:

| Cód. Instr. | Op. destino | Op. fuente |
|-------------|-------------|------------|
|-------------|-------------|------------|

- **[mem]** → especificación a una dirección de memoria.
- **[reg]** → registro de la CPU
- **[inm]** → dato inmediato

Las instrucciones de memoria pueden tener las siguientes formas:

1. Código de instrucción **mem,reg** → [MOV DATO,AX]
2. Código de instrucción **reg,mem** → [MOV AX,DATO]
3. Código de instrucción **reg,reg** → [MOV CX,AX]
4. Código de instrucción **reg,inm** → [ADD AX,4]
5. Código de instrucción **mem,inm** → [SUB DATO,2]

Hablamos de operando destino y operando fuente, con el operando destino siempre a la izquierda, porque si hay un movimiento de datos, siempre será de derecha a izquierda. Además si por ejemplo tenemos una suma, el resultado entre los dos operandos también se guarda en el operando destino.

En los casos de **modos de direccionamiento por registro**, cuando el procesador trabaja con esta instrucción, no hay que ir a buscar operandos a memoria pues ambos operandos están dentro

de la CPU, lo mismo para guardar el resultado, se guarda en un registro de la CPU. Por eso, esta instrucción es de ejecución rápida. **Ejemplo → [MOV CX,AX]**

En los **modos de direccionamiento inmediato via registro**, **Ejemplo → [ADD CX,4]** en este caso, al operando fuente se le suma el contenido del operando destino que es un dato inmediato. El direccionamiento inmediato sucede sólo en el operando fuente, puesto que el operando destino usa el direccionamiento via registro.

En los **modos de direccionamiento directo**, el operando fuente es una dirección de memoria en la cual se encuentra el dato con el que quiero realizar la operación. **Ejemplo → [ADD CX,[2003h]]**, entonces yo quiero sumarle al contenido de CX, el contenido que esta en la dirección de memoria 2003h. Como en este caso el registro CX es de 16 bits, y la dirección de memoria 2003h guarda un dato de 8 bits, se toma el dato guardado en la memoria 2003h más el dato guardado en la memoria que le continúa que es 2004h. Ahora si tengo **Ejemplo → [ADD CL,DIR]**, siendo DIR una variable declarada en una determinada dirección, (2003h DIR=4) en este caso, al ser un registro de 8 bits y la dirección guardar un dato en memoria de 8 bits también, lo único que sumo al registro CL es el dato contenido en la dirección de memoria que llamo con la variable DIR (que sería al 2003h).

En los **modos de direccionamiento indirecto vía registro**, le sumo (o la operación que indique la instrucción) al operando destino, el contenido que se encuentra en la dirección de memoria a la que apunta la variable BX. **Ejemplo → [ADD AL,[BX]]**, supongamos que en BX tengo guardada la dirección de memoria 1002h, entonces al registro AL le voy a sumar el dato que tengo guardado en la dirección de memoria 1002h. En este caso entonces hago un acceso a memoria para ir a buscar el operando fuente (accedo a la memoria 1002h). Si el operando destino, en lugar de ser de 8 bits fuera de 16 bits, entonces accedo a memoria 2 veces (a la dirección 1002h y a la dirección 1003h).

Luego esta el **modo de direccionamiento base + índice**, en donde tenemos dos registros índice que podemos usar: el SI y el DI, y como registro base podemos usar BX o BP, pudiendo hacer cualquiera de las cuatro combinaciones [BX+SI ; BX+DI ; BP+SI ; BP+DI]. **Ejemplo → [MOV CX,[BX+SI]]**, muevo al registro CX, el dato que esta en la dirección de memoria dada por la suma entre la base y el indice (BX+SI); y si **Ejemplo → [MOV [BX+SI],CX]** entonces muevo a la dirección de memoria (BX+SI) el dato que se encuentra en CX.

En el **modo de direccionamiento relativo por registro**, en la instrucción codificamos con que registro base queremos trabajar, y con que desplazamiento, **Ejemplo → [MOV AL,[BX+2]]**, queremos buscar el operando que esta apuntado por BX+2 y mover el dato contenido en esa dirección de memoria al registro AL **Ejemplo → [MOV [BX+2aH],AX]**, y en este caso, muevo el

contenido del registro AX a la dirección de memoria dada por BX+2aH (y a la dirección siguiente puesto que la dirección guarda sólo 8 bits y el registro AX guarda un dato de 16 bits).

Por último está el **modo de direccionamiento relativo base+índice**, en donde combino un registro base, un índice y además un desplazamiento. Ejemplo → [MOV [BX+DI+2aH],AX], enviamos a memoria el contenido de AX, y la dirección efectiva en memoria me la van a dar la suma del registro base BX más el registro índice DI más el desplazamiento 2aH, que va a estar también codificado en la instrucción (y el dato lo guardaré en esta posición y en la siguiente por ser AX un registro de 16 bits).

en conclusión, ¿Conviene que las instrucciones sean cortas o largas? Si los registros son de uso especial me permite acortar un poco los campos de especificación, y si todos los registros son de propósito general necesito más bits, entonces tener más flexibilidad a la hora de programar quizás me impacte al tener instrucciones más largas, y esto en cuanto a su codificación implica que la memoria va a estar organizada en palabras que quizás sean de un ancho mucho más corto que la instrucción, pues voy a necesitar varias palabras para codificar el formato dado de dicha instrucción. Entonces, cada vez que voy a buscar una instrucción a memoria, quizás requiera si la palabra es muy larga muchos accesos, y esto implica que la CPU tenga que esperar por estos datos más tiempo, puesto que la memoria es muy lenta comparada con la velocidad de trabajo del CPU. Entonces, esta flexibilidad me impacta pues termino trabajando con instrucciones de muchos más bits, entonces la flexibilidad se contrapone con el tiempo que requiero para ir a buscar esas instrucciones a memoria (ancho de banda: cantidad de datos que puedo sacar de la memoria por segundo). Entonces es una negociación, si las instrucciones son más cortas el proceso de ejecución de una instrucción es más rápido, pero por ahí requiero más instrucciones, pues las instrucciones que son más complejas y largas, suelen permitir hacer más cosas por instrucción.

SUBSISTEMA DE MEMORIA

Hay distintos niveles de memoria en un sistema de cómputo que establecen una jerarquía, en donde los distintos niveles utilizan distintas tecnologías para ser construidos pues tienen distintos tamaños, velocidad de trabajo, etc. Además, hay distintas características que permiten definir la memoria.

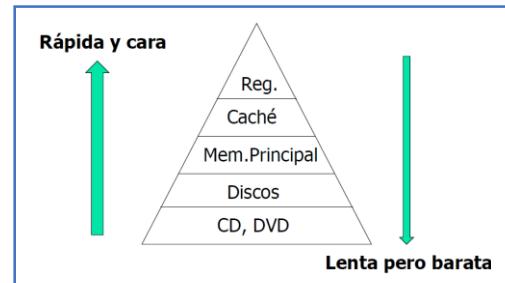
Desde el comienzo de los microprocesadores se viene cumpliendo una especie de regla, que dice que la velocidad del procesador se duplica cada 18 meses (sin variar el precio), esto significa que, para un determinado momento tecnológico y un determinado precio, los procesadores ejecutan una determinada cantidad de instrucciones por segundo, y lo esperable es que para dentro de un año y medio, los procesadores ejecuten el doble de instrucciones por un mismo precio. Y la otra regla es que por el mismo precio que pagamos una memoria de un tamaño determinado, en cuatro años vamos a tener una memoria de un tamaño cuatro veces mayor, mientras que la velocidad de memoria aumenta a razón de un 10% anual.

Como ya vimos anteriormente, cuando la CPU tiene que hacer un acceso a memoria, este es un proceso más lento de lo que es trabajar dentro del CPU. A medida que aumenta la brecha entre las velocidades del procesador y de la memoria, las distintas arquitecturas buscan tender un puente sobre esta brecha. Una computadora típica suele tener distintos tipos de memoria, desde una rápida y cara (en término de costo por bit) como el caso de los registros de la CPU, hasta una barata y lenta como el caso de los discos. Esto no quiere decir que un disco sea barato, sino que el costo por bit almacenado dentro de un disco es mucho más barato que el costo por bit almacenado dentro de un registro.

La idea es que la interacción entre los distintos niveles de memoria sea tal que se logre un funcionamiento equivalente a tener una computadora con una gran memoria y muy rápida, pero sin tener que pagar el costo de esto, sino pagar un costo más reducido, distribuyendo distintos niveles de memoria que funcionen con distintas velocidades y distinto costo por bit, pero trabajando en forma coordinada.

Jerarquía de memoria: es la forma en la que están organizados estos distintos niveles de memoria. En el tope de la jerarquía de memoria, están los registros de la CPU (memoria más rápida) mientras que en la base están las memorias secundarias como los discos magnéticos o los dispositivos de almacenamiento removibles como CD o DVD (memorias más lentas).

Mientras más rápida es la memoria, más caro el costo por bit, y mientras más barato sea el costo por bit, más lenta es a su vez la memoria.



Entre la Memoria Principal y la CPU hay otro tipo de memoria para salvar la brecha, esta es la **Memoria Cache**, hay distintos tipos y niveles de Cache (nivel 1, nivel 2 e incluso nivel 3).

A medida que ascendemos tenemos mayor rendimiento y más costo por bit. Además, cuando ascendemos, también aumenta la frecuencia de accesos a este tipo de memoria. Esto quiere decir que en la cima de la pirámide los registros están siendo continuamente utilizados, la Cache, es reiteradamente usada, con mucha más frecuencia que la Memoria Principal, pero no tanto como los registros, y así siguiendo, es decir: los datos que están más cercanos a la CPU, son los que van a ser accedidos más frecuentemente.

La memoria del computador va a estar distribuida en distintos niveles, y como el objetivo, en cuanto a performance, costo y tamaño, van a ser diferentes en los distintos niveles, las tecnologías utilizadas para su construcción e incluso los fundamentos físicos van a ser distintos.

Lo mismo pasa con la ubicación dentro del sistema, por ejemplo, los discos magnéticos no están

| Tipos de memoria | Tiempo de acceso | Tamaño típico |
|------------------|------------------|---------------|
| Registros | 1 ns | 1 KB |
| Caché | 5-20 ns | 1 MB |
| Mem. Principal | 60-80 ns | 1 GB |
| Discos | 10 ms | 160 GB |

accedidos a través de buses sino a través de dispositivos de E/S. el objetivo entonces será tener una buena capacidad de almacenamiento con un tiempo de acceso reducido.

Para que todo esto funcione, los mecanismos de pasaje de información de los distintos niveles tienen que funcionar correctamente, para que, desde de un punto de vista macro, todo funcione a una alta velocidad como si tuviera una memoria muy rápida, pero al costo por bit de una memoria lenta.

Las memorias tienen distintas **características** que nos permiten definirlas.

- **Duración de la información**

1. **Memorias volátiles:** estas son memorias que precisan estar alimentadas, es decir conectadas a la instalación eléctrica para mantener la información. Las mismas pueden ser leídas o escritas, pero una vez que cortamos la alimentación del sistema la información se pierde. Es el caso de la RAM.
2. **Memorias NO volátiles:** estas son aquellas que también pueden ser leídas o escritas, pero no precisan de alimentación eléctrica para mantener la información, si precisan alimentación eléctrica para poder modificarlas, verlas o escribirlas, pero una vez que desconectamos la alimentación del sistema, la información queda ahí tal cual la dejamos. Este es el caso de los discos.
3. **Memorias permanentes:** son aquellas que no pueden ser modificadas, al menos no en forma normal. Son memorias de lectura únicamente, por ejemplo, la ROM,

o la EPROM, en el caso de esta última, se puede borrar haciendo un ciclo eléctrico particular sobre la misma, pero en uso habitual no se puede borrar.

- **Modo de acceso**

1. **Acceso por palabra:** podes acceder de a una palabra por vez, es decir acceder por dirección y tomar una palabra. Por ejemplo, la Memoria Principal.
2. **Acceso por bloque:** tomar un bloque de información, por ejemplo, el caso de los discos o la Caché, estos no transfieren de a un byte por vez, sino que transfieren sectores o grupos de bytes

- **Velocidad:** está relacionada con cuanto tiempo tarda la memoria en darnos una información o en cuánto tiempo podemos escribirla.

1. **Memorias semiconductoras:** el parámetro clave es el **tiempo de acceso**, que es el tiempo que transcurre desde que se inicia la operación de lectura/escritura hasta que se obtiene o almacena el dato. Ahora, una vez que se lee o escribe el dato, no necesariamente se puede empezar otro ciclo de lectura, sino que muchas veces, es necesario esperar un poco más antes de empezar un nuevo ciclo, por esto es que se habla de **tiempo de ciclo** que es el tiempo mínimo que tiene que haber entre dos operaciones sucesivas sobre una memoria.

$$T_{\text{ciclo}} > T_{\text{acceso}}$$

2. **Memorias magnéticas:** en este tipo de memorias necesitamos movilizar partes mecánicas sobre la superficie del disco entonces se necesita tiempo para posicionar el cabezal en el lugar que va, que el disco gire lo suficiente para tener bajo el cabezal la información que se está buscando y recién ahí comienza a contar el tiempo de latencia o de lectura.
Tener en cuenta que la velocidad de transferencia se mide en bytes/segundos, y está asociada en el caso de los discos magnéticos a la velocidad de giro del disco.

$$T_{\text{acceso}} = T_{\text{posCab}} + T_{\text{latencia}}$$

- **Método de acceso**

1. **Acceso aleatorio:** es aquella en la que el tiempo de acceder a una ubicación es independiente de la secuencia de accesos que hice antes y es constante, esto significa que da lo mismo en qué lugar esté la información que busco pues el acceso no es secuencial. Por ejemplo, la Memoria Principal, accedo a un dato por medio de una dirección y voy directamente a ese dato sin pasar por los anteriores, entonces da lo mismo si el dato está ubicado en la primera posición de memoria o en la última.

2. **Acceso secuencial:** es otro tipo de memoria en la cual hay que hacer una secuencia específica para encontrar la información que estamos buscando, y nos obliga a dar una serie de pasos en orden. Un ejemplo son las unidades de cinta magnética, en donde la información se va almacenando por records, y para acceder a un record determinado tengo que pasar por todos los anteriores. Entonces, el tiempo de acceso va a ser variable dependiendo del dato al que quiero acceder.
3. **Acceso directo:** este acceso también es de tiempo variable. los bloques o registros individuales tienen una dirección única que se basa en la localización física. Este es el caso de los discos, en los cuales nosotros sabemos en qué circunferencia (pista) sobre ese disco está la información a la cual queremos acceder. Entonces es necesario movilizar el cabezal hasta la pista correspondiente y luego esperar que llegue hasta el cabezal el sector que se está buscando. Pero es directo porque directamente vamos a esa pista, no tenemos que pasar por todas las pistas anteriores leyendo la información de cada una, simplemente nos posicionamos en determinado lugar porque ahí está la información. Naturalmente, posicionarnos en determinada pista, puede requerir más o menos tiempo, dependiendo a que distancia está el cabezal de dicha pista.
4. **Acceso asociativo:** en este caso no se sabe en dónde está la información que estamos buscando, pero conocemos parte de la información que buscamos.
 Esto es, conocemos el valor de un campo de bits de la palabra que estamos buscando y lo que hacemos es recuperar toda la palabra, entonces el hardware de este tipo de memoria se hace en comparación con todas las palabras que componen esa memoria, comparando el campo de bits, que sería la clave de búsqueda, hasta que se da con la palabra que se está buscando. Dicho de otro modo, se accede conociendo parte del contenido de la palabra buscada, pero no por su dirección. La comparación se hace en todas las palabras a la vez. Este es el caso de la memoria Caché, y el tiempo de acceso sería invariable independientemente de la palabra que estuvieramos leyendo (como en el caso de las memorias de acceso aleatorio, sólo que acá no accedemos por dirección, sino por contenido).

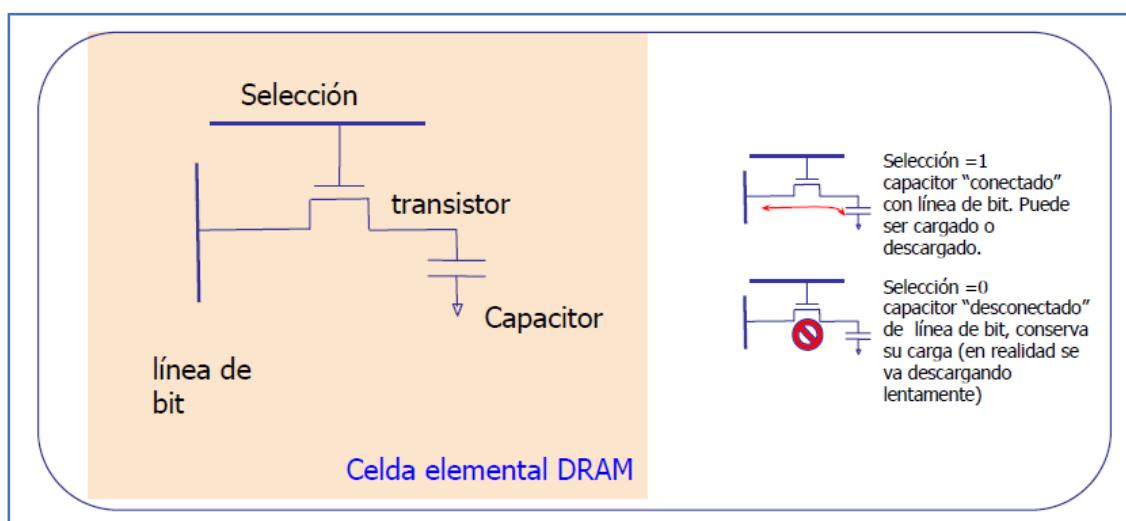
RAM – Memoria de Acceso Aleatorio (Memoria Principal)

Como dijimos, que la memoria sea de acceso aleatorio significa que se puede acceder a cualquier posición dentro de la memoria sin variar el tiempo de acceso a la misma, independientemente de la posición de acceso que sea, y esto es por medio de una dirección.

Hay dos tecnologías básicas: una es que cada celda de memoria sea basada en Flip-Flop, esto es lo que se usa en las memorias estáticas o SRAM; y el otro tipo la construcción de memoria está basada en transistores, y son lo que se llamas memorias dinámicas o DRAM, en estas, en lugar de tener un Flip-Flop, la información de si está almacenado un 1 o un 0 se da a partir de la carga que tenga un dispositivo eléctrico que se llama capacitor, que tiene la capacidad de almacenar o retener carga eléctrica.

El acceso a las ROM también es de tipo aleatorio (también es una RAM en el sentido de que es una memoria de acceso aleatorio, pero le decimos RAM específicamente a la Memoria Principal que es la memoria de lectura y escritura).

¿Cómo es una celda de memoria dinámica basada en transistores y capacitores?



En el dibujo está esquematizada la celda básica. Tiene un transistor y un capacitor. El capacitor es capaz de retener carga eléctrica, uno lo conecta a un circuito y lo puede cargar eléctricamente, después uno desconecta el capacitor del circuito de carga, pero este queda cargado. Entonces se considera que cuando el capacitor está cargado, almaceno un 1, y cuando esta descargado almaceno un 0. Para cargar o no el capacitor, se utiliza el transistor que es comandado por la línea de selección y que cuando la selección este en 1, hace que el transistor actúe como una llave que vincula a la línea de bit con el capacitor, entonces es posible enviarle carga al capacitor y también descargarlo. Cuando la línea de selección este inactiva, es decir este en 0, no está vinculado el capacitor con la línea de bit, por lo tanto, el capacitor conserva su carga.

Pero la carga del capacitor no dura eternamente, la va perdiendo a medida que pasa el tiempo, entonces para que este esquema funcione, es necesario que cada tanto el capacitor sea refrescado, es decir volverlo a cargar a tope, si es que estaba cargado. Entonces frecuentemente hay que hacer un proceso de lectura de ese capacitor, conectando a través de la línea de selección del transistor y desde la línea de bit censando si ese capacitor tiene un nivel de carga eléctrica mayor que un determinado nivel, si esto se cumple significa que el capacitor guardaba un 1, entonces se le repone la carga completa.

Entonces en las memorias dinámicas, además de las celdas elementales, vamos a necesitar circuitos que realicen esta operación de refresco de cada celda varias veces por segundo.

Haciendo una comparación entre las memorias dinámicas y estáticas, las memorias dinámicas almacenan más información en la misma superficie de circuito integrado que las estáticas, pues las celdas elementales de las memorias dinámicas (construidas a base de capacitores) son más chicas que las celdas elementales de la memoria estática (construidas a base de Flip-Flop). Pero la desventaja de las memorias dinámicas es que hay que refrescarlas porque los capacitores se van descargando, mientras que a las estáticas no, entonces son memorias más lentas que las estáticas justamente por las cuestiones constructivas, las memorias estáticas están comutando el estado biestable que es el Flip-Flop, mientras que las memorias dinámicas hay que ir chequeando y reponiendo la carga de un capacitor, inclusive el proceso de lectura es destructivo porque al acceder a un capacitor para ver si está cargado, al chequear esto un poco lo descargo. En conclusión, la memoria dinámica, al mismo costo de hacer un circuito integrado se pueden colocar más celdas dinámicas en un circuito integrado que en uno estático, por eso el costo por bit en la memoria dinámica es más bajo que en la memoria estática. Pero, por otro lado, las memorias estáticas son más rápidas. Entonces las memorias dinámicas son usadas principalmente en la memoria principal, mientras que las memorias estáticas, que son más rápidas, son usadas en la memoria Caché.

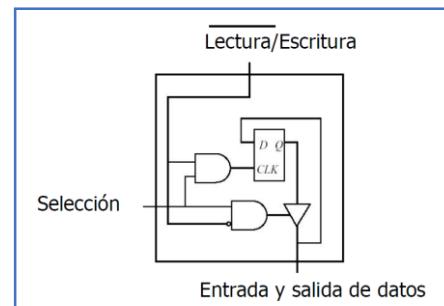
El elemento básico de una memoria semiconductora es la celda de memoria. Todas las celdas de memoria de semiconductor comparten tres propiedades:

- Todas tienen dos estados estables para representar al 0 y al 1. Haciendo una breve referencia a la Memoria Dinámica, es estable entre comillas porque el capacitor se va descargando, pero dado a que el circuito de refresco está permanentemente trabajando, decimos que es estable.
- Se puede escribir en ellas al menos una vez.
- Se pueden leer para conocer su estado.

Normalmente una celda tiene una línea de selección (que selecciona la celda de memoria), una de control (que indica lo que voy a hacer con esa celda, si lectura o escritura) y una de escritura/lectura de datos.

El proceso de lectura/escritura de la celda es: primero se selecciona la celda mediante la línea de selección, luego se le da la orden mediante la línea de control de lectura o escritura, y por último, mediante la línea de escritura/lectura recibimos el dato a ser leído o enviamos el dato a ser guardado.

En el siguiente esquema de una **celda de memoria** estática basada en **Flip-Flop**. Como elemento de memoria biestable tenemos un Flip-Flop D, en la línea de entrada y salida de datos tenemos un buffer de tres estados, que tiene en su salida la posibilidad de colocar 0 o 1 dependiendo lo que tenga en su entrada, pero además tiene un tercer estado que es como si estuviera desconectado.

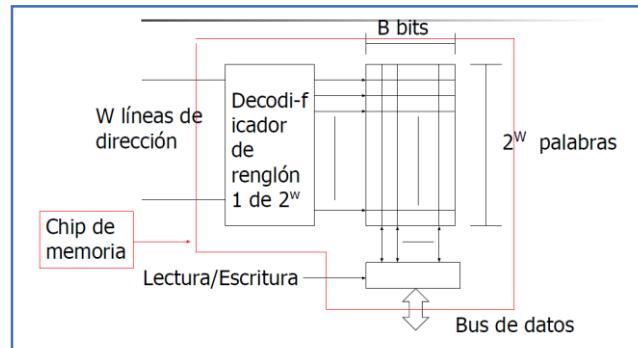


Luego a la izquierda tenemos la línea de selección, qué si vale 0, ambas compuertas AND están entregando 0, la compuerta que va hasta el reloj del Flip-Flop nos va a indicar que ese Flip-Flop está inactivo, porque no recibe señal, por lo tanto, lo que tenga en la entrada D no importa. Por otro lado, mientras la otra compuerta AND este entregando un 0, el buffer de 3 estados que maneja la salida de la información va a estar en estado abierto, desconectado del resto del sistema. Ahora si la línea de selección está en 1, en principio ambas compuertas están habilitadas, pero hay que ver qué pasa con la otra línea. La otra línea es la de lectura/escritura, qué si está en el estado de escritura, es decir la línea vale 1, la AND que está habilitada va a ser la de arriba, que está conectada con la línea de reloj, entonces la señal del reloj va a estar en 1 y el Flip-Flop va a almacenar el dato que entra por la línea D. Cuando, por el contrario, la señal de lectura/escritura está en 0, es decir en estado de lectura, la compuerta AND de arriba que conecta al reloj está inhabilitada y la compuerta AND de abajo está habilitada. Esta habilita al buffer para dejar pasar información de la línea Q hasta la línea de salida de la celda.

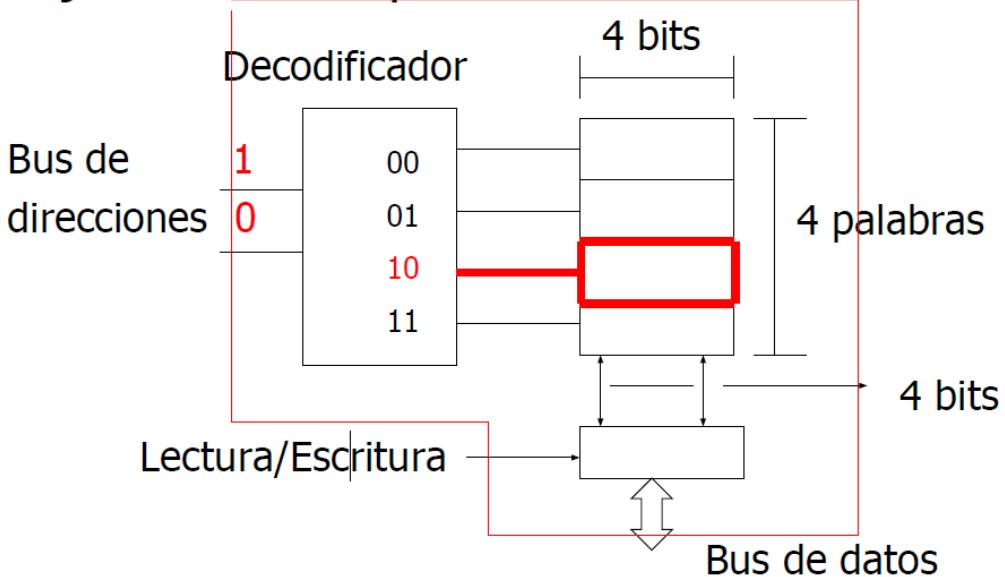
Una memoria de un bit la implementamos con un Flip-Flop y armamos registros sencillos de n bits juntando en paralelo n Flip-Flops. Para construir memorias más grandes se requiere una organización diferente en la cual sea posible direccionar palabras individuales.

Organización del CHIP: cada chip contiene un arreglo de celdas de memoria, y en las memorias de semiconductor se utilizan dos enfoques organizacionales diferentes, el 2D y el 2½D.

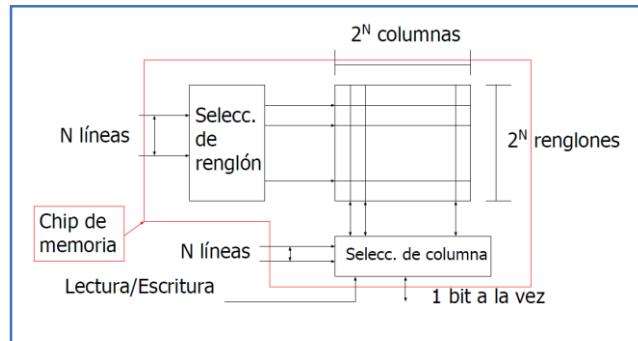
- **Organización 2D:** el arreglo está organizado en 2^W palabras de B bits cada una. Cada línea horizontal (una de 2^W) se conecta a cada posición de memoria, seleccionando un renglón. Las líneas verticales conectan cada bit a la salida. El decodificador que está en el chip tiene 2^W salidas para W entradas (bit del bus de direcciones).

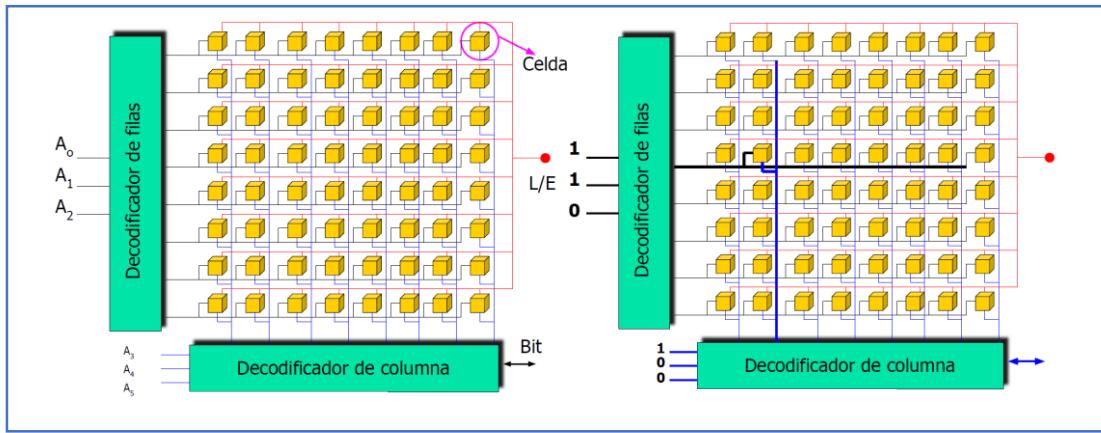


Ej. memoria 4 palabras de 4 bits



- **Organización 2½D:** el arreglo es cuadrado y funciona parecido que el arreglo 2D. La principal diferencia con el arreglo 2D es que los bits de una misma palabra están dispersos en distintos chips, y cada chip aporta un bit de cada palabra; además la dirección se divide en dos partes: una parte es para la selección de un renglón y la otra parte para la selección de una columna. Además, en este tipo de organización vamos a tener dos decodificadores.





En conclusión, en el 2D todos los bits de una palabra están en el mismo chip, y en el 2½D los bits de una misma palabra estarán en distintos chips. Ahora el arreglo de 2D, si tengo un número grande de palabras con pocos bits, va a ser muy largo y estrecho. Cada línea de selección de palabra tiene que tener un manejador y conectarse a un decodificador, entonces ocupan mucha superficie, y trae un montón de problemas de índole técnica y constructiva (por ejemplo, no va a ser lo mismo censar si un capacitor que está cerca está cargado, a uno que está un millón de líneas de distancia). Ahora en un arreglo cuadrado necesitamos menos bits para decodificar líneas y columnas, entonces van a ser dos decodificadores mucho más chicos, está todo mejor distribuido. Esto significa que el arreglo 2½D, al usar decodificación separada de filas y columnas, reduce la complejidad de los decodificadores.

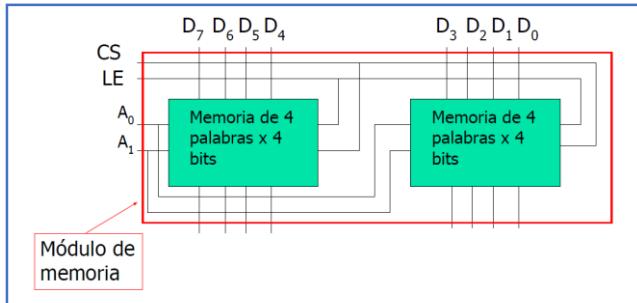
Además, respecto a la detección de errores, el arreglo 2D dificulta el uso eficaz de los circuitos correctores de error. En 2½D, al estar los bits dispersos en distintos chips, hay menos probabilidad de error, puesto que a lo sumo falla uno de los chips que aportan un bit a esta palabra.

Construcción en paralelo y en serie

Supongamos que queremos construir una determinada memoria a partir de módulos de chips que son más chicos que la cantidad de memoria que queremos construir: tengo módulos que tienen menos palabras que la cantidad de palabras que quiero en la memoria o tengo módulos cuyo ancho de palabra es más chico que el ancho de palabra que yo quisiera en la memoria.

¿Cómo se pueden conectar estos módulos más chicos para construir esa memoria resultante más grande?

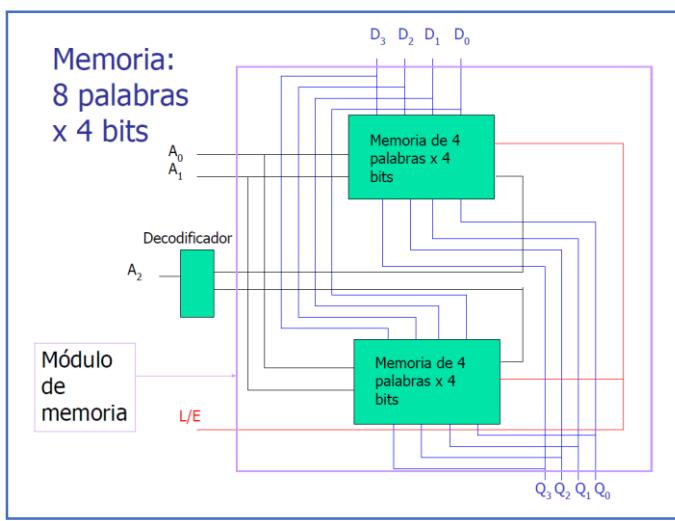
- **Primer caso:** cada memoria cubre el espacio de direccionamiento requerido, pero sólo cubre una parte de la palabra (cantidad de palabras que quiero, pero de menor cantidad de bits). Solución: usar varios módulos en PARALELO.



En este caso, el ancho de cada módulo es más chico que la palabra que buscamos, entonces la solución es utilizar varios módulos en paralelo, entonces cada módulo aporta una cierta cantidad de bits de cada palabra. En el

ejemplo, cada módulo me aporta 4 palabras de 4 bits, entonces si yo quiero 4 palabras de 8 bits, conecto dos módulos en paralelo y cada módulo me aportará 4 bits por palabra. En este caso, las líneas de lectura/escritura, las líneas de selección y las líneas de direcciones son comunes a ambos módulos, pues si quiero por ejemplo acceder a la primera palabra de 8 bits, debo acceder a la primera palabra de 4 bits que me aporta cada módulo cada módulo, ambas son direccionadas con la misma línea de direcciones, seleccionadas con la misma línea de selección, y voy a realizar la misma operación en ambas (sea lectura o escritura).

- **Segundo caso:** ahora la longitud de la palabra es la deseada, pero los módulos no tienen la capacidad deseada (tengo menos palabras de las que quiero, pero de la longitud correcta). En este caso, cubro un cierto rango de direcciones con módulos de memoria en SERIE. En este caso, a cada módulo se lo va a ver en direcciones distintas.



En el ejemplo, ahora queremos 8 palabras de 4 bits, pero nuestro módulo de memoria aporta 4 palabras de 4 bits, entonces la longitud de la palabra es correcta, pero tengo 4 palabras en lugar de 8. Acá vamos a necesitar una tercera línea de dirección, pues debemos direccionar 8 palabras (estas son 2^3 direcciones). Entonces las líneas

bajas de direcciones (A_0 y A_1) se comparten entre los dos módulos, y la línea alta (A_2) nos permite seleccionar entre un módulo y el otro. La línea de lectura/escritura es compartida, porque si bien llega a ambos módulos, sólo uno de ellos va a poder recibir la señal de dicha línea, según si está o no seleccionado por la línea de dirección A_2 .

¿Cómo son las tecnologías de las memorias dinámicas actuales?

Como vimos, la memoria dinámica de RAM básica, cuenta con celdas elementales de bits organizadas en arreglos $2 \times D$ y se accede a las celdas de bits para leerlas o escribirlas. Para

mejorar la performance, empezaron a hacerse agregados a estas memorias dinámicas básicas.

Surgió la memoria DRAM Enhanced que contiene una pequeña memoria estática dentro del chip de memoria que guarda la última línea leída (es decir, el último conjunto de bits leído).

¿Cuál es la finalidad de esto? Si en el futuro cercano, queremos acceder a un bit que está contiguo con ese podemos hacerlo sin tener que acceder al arreglo cuadrado.

Luego surgió la Caché DRAM que incluye una memoria SRAM más grande en la cual se pueden almacenar varias líneas, entonces permite anticipar el acceso de los futuros accesos, esto funciona puesto a que la CPU es bastante probable que haga accesos secuenciales.

La última es la Synchronous DRAM (SDRAM) o memoria dinámica sincrónica, que es la que se utiliza mayormente en las computadoras. Acá estaría sincronizada la entrega de datos con un reloj externo y el funcionamiento es que se presenta una dirección a la RAM, la CPU se libera (no se queda esperando el dato) y unos ciclos de reloj después va a buscar ese dato. Tiene un modo de trabajo que se llama Burst que le permite a la SDRAM enviar datos en forma de bloques,

CACHE Y MEMORIA PRINCIPAL

Como ya se dijo anteriormente, hay una gran brecha entre la velocidad de la CPU y la de la Memoria Principal. Esto se debe a que, si bien, a medida que evoluciona la tecnología cada vez se pueden colocar más circuitos dentro de un chip y hay más recursos electrónicos para hacer cosas, la gran diferencia está en cómo se usan esos circuitos, por un lado, los diseñadores de CPU hacen organizaciones cada vez más complejas logrando máquinas cada vez más veloces, por otro lado, los diseñadores de memoria usan esa mayor disponibilidad de recursos electrónicos para aumentar la capacidad disponible de memoria. Esto hace que la brecha de velocidad sea cada vez más grande.

Esta diferencia de velocidades implica que después que la CPU emite una solicitud de lectura a la memoria (por el bus de direcciones y el bus de control), pasan muchos ciclos de reloj antes de que reciba la palabra que necesita por el bus de datos.

En todos los ciclos de instrucción, la CPU accede al menos una vez a memoria para buscar la instrucción (acceso a memoria de instrucción), y muchas veces tiene que también acceder a memoria de datos para ir a buscar operandos o escribir resultados. Entonces, la velocidad a la cual la CPU ejecuta instrucciones está limitada por el tiempo del ciclo de memoria. Si la memoria tiene un ciclo lento la CPU se va a retrasar.

Pero el problema no es tecnológico, sino que es económico, es decir, se podrían construir memorias tan rápidas como la CPU, pero para obtener la máxima velocidad tiene que estar dentro del chip de la CPU. El tema es que cuando salimos a los buses, ahí se vuelve un acceso lento.

La idea es entonces combinar una memoria que sea muy rápida y más pequeña, con otra memoria que sea más lenta, pero grande (jerarquías de memoria). De modo que la CPU pueda estar funcionando con esa memoria rápida y disponer de un espacio de almacenamiento grande. Esto mismo, y el uso de la memoria Cache, es posible gracias a que los programas presentan dos principios o propiedades:

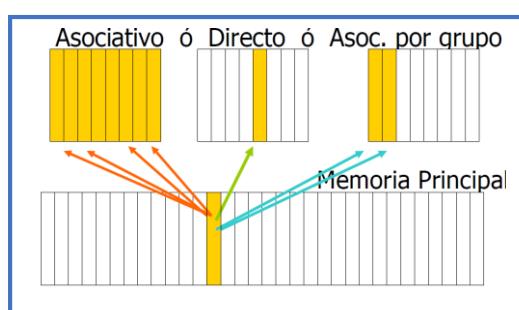
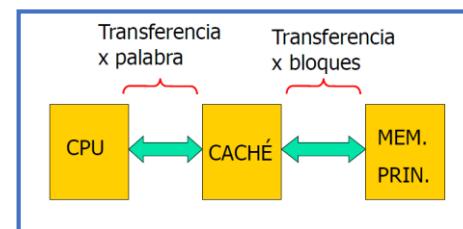
- **Principio de Localidad Espacial de Referencia:** este dice que cuando se accede a una palabra de memoria, es muy probable que el o los próximos accesos sean en la vecindad de la palabra a la que acabamos de acceder. Este principio se basa en:
 1. **El código se ejecuta de manera secuencial:** las instrucciones están en memoria una debajo de la otra en posiciones contiguas de memoria y lo más frecuente es ejecutar la instrucción que sigue a la anterior en la posición de memoria.
 2. Otra razón en la que se basa dicho principio es que generalmente **cuando programamos, declaramos variables relacionadas muy juntas.**

3. Otra justificación para este principio es el **acceso a estructuras de datos tipo matriz o pila**. Por ejemplo, es normal que cuando accedamos a un elemento de un vector, luego accedamos al siguiente muchas veces de modo ordenado (recorrido de un vector).
- **Principio de Localidad Temporal de Referencia:** enuncia que si accedemos a una posición de memoria es muy probable que en un lapso de tiempo corto tengamos que volver a acceder a esa misma palabra. Este principio se sustancia en:
 1. Es muy frecuente tener que hacer **lazos o bucles** mientras programamos, y esto nos lleva a tener que pasar por el mismo lugar y acceder a las mismas instrucciones puesto que estamos por ejemplo iterando, o dentro de un bucle.
 2. El **uso de las subrutinas** también sustenta este principio, pues estamos repitiendo código al usar varias veces una subrutina, y por lo general, es muy probable que esto suceda, ya que si un conjunto de instrucciones lo definimos como una subrutina es porque vamos a tener que accederlo varias veces y queremos evitar repetir código.
 3. Este principio también se basa en el **uso de las pilas**, puesto que, si apilé en la pila un dato, en algún momento lo voy a querer recuperar.

La idea general es que cuando se hace referencia a una palabra, ella y alguna de las vecinas se traen de la memoria grande y lenta a la Memoria Cache, para que sea muy probable que, en el siguiente acceso, la palabra buscada ya se encuentre en la Cache.



La transferencia entre la CPU y la Cache será por palabra, ahora el acceso y transferencia entre la Cache y la Memoria Principal va a ser por bloques, ya que acá no sólo preciso transferir la palabra que voy a necesitar, sino también el bloque que lo secunda. Los tamaños de los bloques que transfiero entre la Memoria Principal y la Cache son variables (8, 16, 64 palabras $[2^n]$).



Mapeo de la Memoria

El dibujo de abajo trata de representar una Memoria Principal dividida en bloques, con una cierta cantidad de palabras cada bloque. La Memoria Cache, es una memoria más chica, dividida en la misma cantidad de bloques que la Memoria principal. La idea es que, cada vez que necesitemos una palabra de la Memoria

Principal, ese bloque lo copiamos en la Cache para tenerlo disponible para futuros accesos. Hay distintos tipos de mapeo:

- **Mapeo asociativo:** en una Cache funcionando con mapeo asociativo, cualquier bloque de la Memoria Principal puede llegar a copiarse en cualquiera de los bloques de la Memoria Cache. Entonces, basta con que tenga un lugar libre en la Cache para copiar el bloque y lo voy a tener ahí hasta que ya no lo precise o tenga que reemplazarlo por otro.

¿Cómo es el proceso de búsqueda de las direcciones? La CPU en el próximo acceso busca una palabra y lo que tendría que hacer el hardware asociado con esta Memoria Cache es verificar si esa palabra la tiene copiada ya en la Cache, y si es así el acceso se hace rápido (a velocidad de la Cache). Si la palabra no está copiada, ahí tendremos que ir a buscar un nuevo bloque a Memoria Principal. ¿Cómo hace el hardware para darse cuenta si tiene o no copiada la palabra? Cuando la CPU emite la dirección (que es una dirección de Memoria Principal) permitiría con esa dirección ubicar cualquier palabra de cualquier bloque. Podemos suponer que la dirección está dividida en unos bits que están referenciando al número de bloque y una dirección dentro del bloque. Para cada bloque de la Memoria Cache vamos a tener una etiqueta asociada (registro) en el cual está anotado a qué número de bloque de la Memoria Principal corresponde el bloque de Cache en cuestión.

- **Mapeo directo:** en este caso, cada bloque de la Memoria Principal, sólo puede estar mapeado por un solo lugar de la Cache y la forma de asignar es dividir el número de bloque de la Memoria Principal por la cantidad de bloques de la Cache, y el lugar para guardar en la Cache me lo dará el resto de hacer esa división en módulo. Luego si por ejemplo tenemos 8 bloques de Cache y 32 bloques de Memoria Principal, y quiero saber en qué lugar de la Cache voy a guardar el bloque 10 de Memoria Principal, hago la cuenta: $10/8 = 1 * 8 + 2 \rightarrow$ Cómo 2 es el resto, guardaré el bloque 10 de Memoria Principal en el bloque 2 de Cache. Entonces el circuito de verificación para saber si tenemos o no copiado el bloque en la Cache, es mucho más simple. El

mapeo directo tiene una dificultad, que ocurre cuando las direcciones que está pidiendo el CPU pertenecen a dos bloques distintos que deberían estar mapeados en el mismo bloque de Cache. Supongamos que estamos accediendo al bloque 0 y al bloque 8 de la Memoria Principal (siguiendo el ejemplo), ahí tendríamos un conflicto porque ambos bloques se deberían estar mapeando en el bloque 0 de la Cache, entonces en sucesivos accesos tendríamos que estar comutando de bloques.

$$X / Y = C * Y + R$$

X = N° de bloque de Memoria Principal (el que quiero guardar).
Y = cantidad de bloques en Cache.
R = N° de bloque en cache en donde guardo X

- **Mapeo asociativo por grupo:** este tipo de mapeo es una mezcla de los dos mapeos anteriores. En este caso, lo que tenemos son múltiples alternativas para el acceso directo, por ejemplo, en el caso de la filmina estamos suponiendo que tenemos dos espacios en Cache para guardar un bloque de Memoria. En este caso estamos dividiendo la memoria Principal en módulo de la cantidad de grupos que tenemos en Cache. En el ejemplo, son 4 grupos de 2 bloques, entonces sí quiero saber en dónde puede ser guardado mi bloque de Memoria Principal, debo trabajar en módulo 4 y mirar el resto. Entonces siguiendo el ejemplo de la filmina, el bloque 12 de Memoria Principal irá en $12/4 = 3*4 + 0 \rightarrow$ grupo 0 \rightarrow bloques 0 y 1 de la Memoria Cache. Entonces, si la CPU pretende acceder a este bloque, y quiere verificar si está copiado en Cache, sólo debe verificar el bloque 0 y 1 de Cache, si no está ahí lo tiene que traer de Memoria Principal.

La eficiencia o efectividad que vamos a tener con la Cache va a depender de la frecuencia de aciertos, esto quiere decir, la cantidad de veces que en la Cache tenemos copiado lo que estamos buscando. Entonces se dice que hay un **acuerdo en la Cache, cuando los datos que necesita el CPU ya están en la Cache, y en este caso la CPU accede a los datos a la velocidad de la Cache.** Por el contrario, un **fallo en la Cache** ocurre cuando la CPU busca información que no está disponible en la Cache, es decir, no está en Cache copiado el bloque en el cual se encuentra la palabra que la CPU precisa. En este caso, habrá que ir a buscar el dato a la Memoria Principal y transferir el nuevo bloque a Cache, ¿Dónde se coloca el bloque? Eso dependerá del tipo de mapeo con el que trabaja la Cache. ¿Cómo sabe la CPU cual va a ser el bloque a reemplazar por este bloque nuevo? En el mapeo directo, como sólo puede estar en un solo lugar se guarda en ese lugar y listo; en el mapeo asociativo por grupo ¿Cuál supongo que no voy a usar más de los que tenía? Esto es más sencillo porque podría llegar a tener anotado con un bit cual es el último que usé, y reemplazo el bloque al que hace más tiempo que accedí. Ahora en el mapeo asociativo es más complicado, porque cuando traigo un nuevo bloque de Memoria Principal lo puedo poner en cualquier lado. Hay distintos algoritmos que se pueden implementar para que esto se decida rápido, una es ir computando y ver cuál es el bloque que hace más tiempo que no se usa, otro puede ser el agregado más antiguo.

Otro tema a considerar es el tamaño de los bloques: si los bloques son muy chicos significa que no llegaría a explotar completamente el principio de localidad espacial, si son demasiado grandes, tendría que dividir mi Cache en menos bloques entonces esto me limitaría en flexibilidad. Entonces hay que negociar entre estas dos cosas para optimizar al máximo.

Puede haber varios **niveles de Memoria Cache**, pues a la hora de agregar más recursos, nos conviene más agregar una Cache adicional, en lugar de agrandar la Cache que ya tenemos. Si por ejemplo nuestro primer nivel de Cache tiene un porcentaje de aciertos del 90%, nos conviene más mejorar el 90% de aciertos usando el mismo razonamiento en una Cache adicional (segundo nivel de Cache), en lugar de preocuparnos por mejorar el 10% de fallos.

MEMORIA EXTERNA: DISCOS MAGNÉTICOS

Dentro de lo que es Memoria Externa, primero veremos **Discos Magnéticos** (Discos Duros). Estos están formados por platos que son superficies rígidas y circulares que rotan alrededor de su centro, que originalmente eran de aluminio, actualmente se utiliza también vidrio, y están recubiertos de partículas de hierro que es un material magnetizable, es decir, que si lo sometemos a un campo magnético las partículas quedan con un magnetismo remanente (como si fueran pequeños imanes). En cuanto a la implementación del vidrio para los Discos Magnéticos, esto se debe a distintas propiedades físicas, como, por ejemplo, el hecho de que el vidrio es menos sensible a las temperaturas entonces se dilata menos que el aluminio, o que su superficie es más uniforme, con menos impurezas, lo cual es más ventajoso.

Principios físicos de los discos magnéticos

Imaginen el disco rotando, y por sobre el disco hay un cabezal que genera un campo magnético variable en la medida que va girando el disco, provocando así que se valla magnetizando el disco. Dependiendo como sea el campo magnético que genera ese cabezal las partículas quedarán magnetizadas en una dirección u otra (polo positivo o negativo de un imán). Esos cambios en la forma en que fueron magnetizados se hacen en función de la información que pretendemos guardar en ese disco. ¿Cómo se lee la información luego? En una segunda pasada cuando yo quiero leer la información, el cabezal ya no va a estar generando campo magnético sino mirando la orientación del material magnetizado que tiene debajo, y cuando este material esté pasando por debajo del cabezal va a estar generando en el cabezal un campo magnético generado por las partículas magnetizadas que pasan por debajo del mismo, generando un campo magnético variable al cabezal. Así el cabezal puede detectar como están orientados esas partículas imantadas que se generaron en la superficie del disco magnetizable. De esa manera recupera la información original, porque originalmente el disco se magnetizo en función de una secuencia de 0 y 1.

Estas pequeñas áreas del disco son magnetizadas en diferentes direcciones por un **transductor**. Este es un mecanismo que permite convertir un tipo de energía en otra (por ejemplo, energía eléctrica en energía magnética) entonces, a partir de una secuencia de 0 y 1 se genera una serie de pulsos eléctricos en el cabezal que son transformados en campo magnético variable y

generan las magnetizaciones en el disco que está pasando por debajo de ese cabezal y viceversa: cuando se quiere leer información del disco, este genera pulsos eléctricos en el cabezal que está pasando sobre él, que son convertidos en 0 y 1. Resumiendo, la lectura y escritura del disco es a través de la cabeza transductora (cabezal), además durante la lectura o escritura, el cabezal está quieto y es el plato el que gira debajo. Por último, el almacenamiento de 0 y 1 es por medio de la magnetización de pequeñas áreas del material del plato.

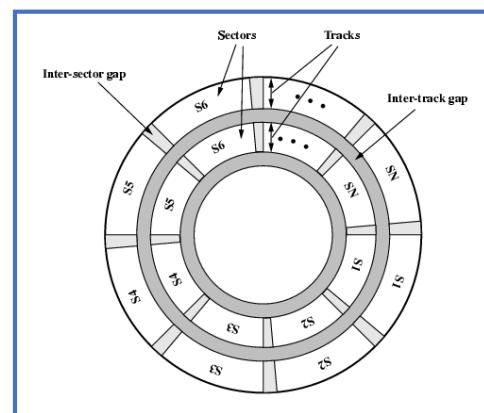
Ahora como el disco gira mientras el cabezal está quieto, esto significa que el cabezal está parado sobre una de las circunferencias o pistas del plato. Si yo quiero cambiar de circunferencia, lo que tengo que hacer es mover el cabezal hacia adentro o hacia afuera de donde estaba.

La información se guarda en estas circunferencias concéntricas llamadas pistas. Entre pista y pista hay un pequeño espacio (gaps) para evitar que la magnetización de una pista influya en la magnetización de la otra. Cuanto menos separada está una pista de la otra, significa que en el mismo disco puedo poner más pistas.

La cantidad de bits que se escriben por pistas es la misma para todas las pistas del disco, esto significa que las pistas que están más afuera del disco (que son más largas pues el perímetro de esa circunferencia será mayor). Entonces, dado que la cantidad de bits es la misma, significa que cada bit ocupa más superficie magnetizada en las pistas externas respecto a las internas.

La velocidad de giro del disco es constante en cuanto a velocidad angular (da la misma cantidad de vueltas en una determinada cantidad de tiempo). Pero, frente al cabezal va a pasar más rápido una pista externa comparada con una interna, pues las pistas internas son más cortas respecto a las externas.

La información de las pistas está dividida en sectores: en cada sector se guarda un cierto bloque de datos y el mínimo tamaño de información que se maneja en el disco es el sector. También se trabaja con conjuntos de sectores y eso se lo llama (clúster).



Formato de datos típico de un sector

Además de los bytes de datos con la información que guardamos en el sector, hay bytes adicionales que permiten identificar el sector y también corregir errores en el proceso de lectura, estos bytes adicionales son los que completan, dan formato y supervisan la lectura de los datos.

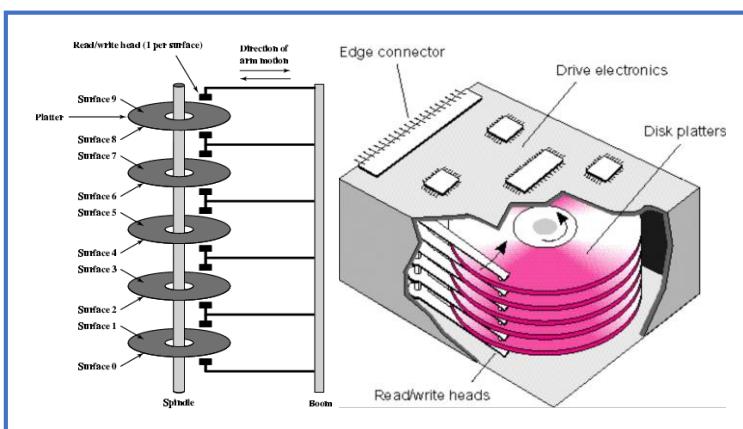
| Un sector | | |
|------------|-----------|---------------------|
| Encabezado | Datos | Código para errores |
| 10 bytes | 512 bytes | 12 bytes |

Hay un encabezado dentro del sector [10 bytes] con información que permite identificar de qué sector se trata, y además permite

hacer la sincronización de la lectura de ese sector. Luego hay un sector de **datos** [512 bytes], en ese lugar estarán los datos que queremos leer o escribir y por lo general el tamaño se expresa como potencia de 2. Y por último hay un **código de corrección de errores** [12 bytes], que permite detectar y corregir una cierta cantidad de errores que puedan producirse entre la escritura y la lectura.

Estructura de un disco

Puede haber múltiples platos en una unidad de disco rígido en donde tenemos una cabeza por cada cara, y todas las cabezas se mueven en forma solidaria, es decir, todas están apuntado al mismo número de Track al mismo momento (pero Tracks de distintas caras). Las pistas alineadas en cada plato forman cilindros y el hecho de que los datos sean almacenados por estos cilindros hace que se reduzca el movimiento de las cabezas y aumente la velocidad de respuesta.



El conjunto de las distintas pistas homólogas de cada disco conforma lo que se llama un **cilindro**. Entonces hablo de Cilindro 0 cuando todos los cabezales están posicionados en las pistas 0, por eso es que decimos que los datos están almacenados por cilindros.

Esto es, se guarda información en una pista, y si se me termina esa pista, en lugar de pasar a la pista que sigue dentro de la misma cara, cambio de cara o de disco, pero continuo en el mismo número de pista guardando datos (almacenamiento de datos por cilindro). De esa manera no tengo que estar moviendo las cabezas tan a menudo, ya que agoto toda la capacidad del cilindro en el cual estoy posicionado antes de pasar al cilindro que sigue.

Velocidad de giro de un disco

Como dijimos, el disco rota a una velocidad angular constante, esto significa que un bit más cercano al centro va a girar más lento respecto a un bit del borde del disco. Por ende, los sectores van a ocupar distintos espacios en las distintas pistas: un sector de las pistas externas va a ocupar más espacio que un sector de las pistas internas. Para la lectura y escritura lo que hay que hacer es mover el cabezal hasta una pista y esperar que gire hasta que el sector que necesitamos pase por debajo del cabezal, de este modo las partículas magnetizadas presentes en ese sector

generen las variaciones eléctricas en el cabezal permitiendo que así se extraigan los datos guardados en ese sector.

Los tiempos asociados al acceso al disco implican:

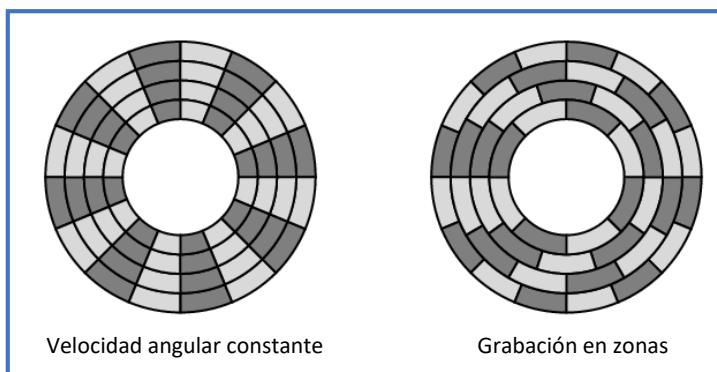
- **Tiempo de búsqueda [Seek]**: implica mover el cabezal hasta el cilindro correcto.
- **Tiempo de latencia**: es por rotación e implica esperar hasta que el sector que yo estoy tratando de leer o escribir pase justo por debajo del cabezal (en promedio tengo que esperar media vuelta, pero puede pasar que no espero nada o que tenga que dar la vuelta completa porque justo había pasado).
- **Tiempo de acceso**: este es el tiempo de búsqueda más el tiempo de latencia por rotación.
- **Tiempo total**: es el tiempo de acceso más el tiempo que tardo en transferir los datos.

Algo a tener en cuenta es que la velocidad de rotación del disco, nos va a determinar tanto el tiempo de latencia, como el tiempo de acceso y consecuentemente el tiempo total, puesto que van a depender del tiempo de rotación del disco, que depende a su vez de la velocidad de rotación que tiene.

Capacidad del disco

$$\text{Capacidad} = \frac{\text{bytes}}{\text{sector}} \times \frac{\text{sectores}}{\text{pista}} \times \frac{\text{pistas}}{\text{superficie}} \times \# \text{ de superficies}$$

La capacidad de un disco se puede sacar mediante el siguiente cálculo. De todos modos, hoy en día se usan las distintas pistas agrupadas en zonas de tal manera que se trabaje con distintas cantidades de bits por pista (ya no mantenemos la misma cantidad de información por pista).



Esto requiere de una electrónica más compleja.

Disco típico: tiene velocidad angular constante. Misma cantidad de sectores para cada pista: los sectores de afuera ocupan más espacio que los cercanos al centro, pero el

material de los sectores de afuera podría estar aprovechándolo para codificar más bits.

Grabación por zonas: en cada zona tengo una velocidad angular constante, pero esa velocidad no es la misma para todas las zonas. Los sectores ocupan la misma cantidad de espacio en todas las zonas. Se aprovecha mejor la superficie magnetizable de todo el disco y aumenta su capacidad. Pistas externas con más sectores que las pistas internas.

Formato del disco

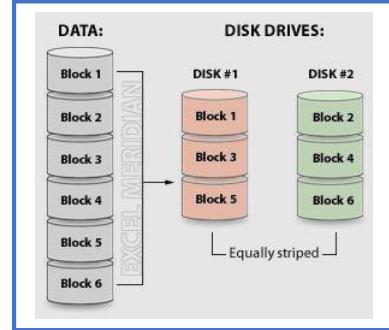
Para lograr colocar información en el disco y administrarla es necesario realizar el formato del disco, esto es: generar los encabezados de los sectores, los campos de corrección de errores, y así, de esa manera se puede ubicar un sector y se puede guardar o leer información del mismo. Normalmente, ese formato es un formato por software, porque se está grabando la información, no hay necesidad de marcas físicas en el disco (aunque también existe el formato por hardware que implementa justamente esto).

PERIFÉRICOS

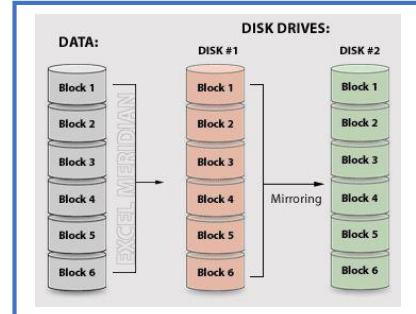
RAID: significa “Arreglo Redundante de Discos Independientes”, la finalidad de los RAID es tener un conjunto de discos físicos que son vistos por parte del SO como si fuera una sola unidad lógica.

La información del disco lógico va a estar distribuida en los distintos discos físicos trabajando en paralelo, con lo cual vamos a tener mayor capacidad de almacenamiento que si tuviéramos un solo disco físico, y también mejores prestaciones en cuanto a velocidad de acceso y protección de datos frente a roturas de discos. Hay distintas formas de organizar un conjunto RAID, estas formas se llaman niveles, aunque no indican una jerarquía entre ellas.

- **RAID 0:** mínimamente necesitamos dos discos para un RAID 0, pero pueden ser más. La finalidad es aumentar la capacidad de almacenamiento. Pero no es necesariamente sumar la capacidad total de los discos, sino tener en cuenta el disco con menor capacidad. Por ejemplo, si tenemos 4 discos de 40Gb la capacidad será de $4 \times 40\text{Gb} = 160\text{Gb}$; ahora si tenemos 4 discos, 1 de 40Gb y los otros 3 de 60Gb, la capacidad también será de 160Gb, pues de los discos de 60Gb yo en realidad sólo podría aprovechar 40 Gb, puesto que la información se tiene que ir distribuyendo de manera igual en los 4 discos. Bloques consecutivos van en discos diferentes: la información se distribuye de esta manera puesto que, si el SO necesita leer varios bloques de manera consecutiva, se pueden hacer ambas lecturas en simultáneo, y esto se traduce como más velocidad, porque entonces no se suman los tiempos de acceso si no están corriendo en paralelo. En este tipo de RAID no estamos protegiendo datos, no tenemos más protección de datos de la que tenemos con un disco simple. De hecho, si tengo varios discos, mientras más tengo, la probabilidad de que alguno me falle es más alta. ¿En qué caso se aplica este uso de RAID? En los casos en los cuales no importa el tema de la confiabilidad en cuanto a roturas, y se requieren grandes prestaciones en cuanto a velocidad de acceso.

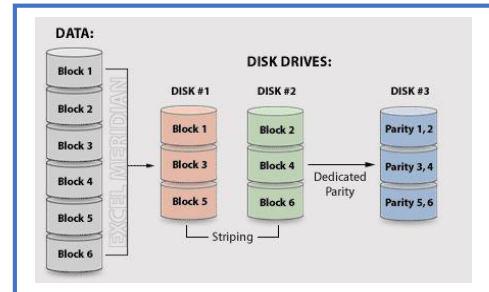


- **RAID 1:** con este tipo de RAID no se busca aumentar la capacidad de almacenamiento, de hecho no se aumenta, lo que se busca es aumentar la confiabilidad. En este caso tenemos como mínimo dos discos que trabajan en espejo, es decir, uno es una copia del otro. Entonces acá la confiabilidad aumenta mucho, pues, aunque sigue

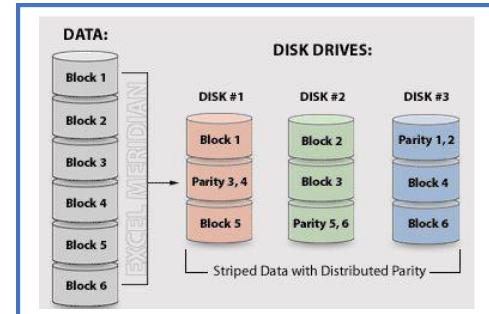


siendo igual de probable que alguno de los dos discos se rompa, es muy baja la probabilidad de que se rompan los dos discos al mismo tiempo. Entonces si se me rompe un disco el otro aún conserva la información, así que no pierdo datos, sólo debo reponer el que se rompió y volver a clonar la información que tenía. En cuanto a la capacidad no aumenta: si tengo 2 discos de 40Gb, la capacidad será de 40Gb, si tengo un disco de 40Gb y otro de 60Gb, la capacidad va a ser de 40Gb pues me guio por el disco más chico. En cuanto a la performance, para la escritura no ganamos nada, pues cada vez que tenemos que actualizar un bloque hay que hacerlo en los dos discos a la vez, ahora en la lectura si se pueden hacer lecturas por separado, trabajando en paralelo.

- RAID 4: aquí se agrega un disco para almacenar paridad, entonces mínimamente necesitamos tres discos: en disco 1 y disco 2 se distribuye la información como si fuera un RAID 0, en forma alternada, y el disco 3 se utiliza para almacenar paridad de otros discos (si tuviéramos 4 disco el disco 4 sería el que tiene la paridad de los primeros tres discos). La paridad es un NOR exclusivo de los bits y nos permite detectar si hay errores simples en alguna palabra que se hubiese leído en alguno de los otros discos. Y esto nos permite reconstruir la información en el caso de errores. En cuanto a las prestaciones, para la lectura es como un RAID 0, podemos hacer lecturas en varios discos simultáneamente sin ningún problema, con las escrituras se complica un poco porque tenemos un solo disco que almacena la paridad, entonces si por ejemplo tenemos que escribir en el disco 1, hay que modificar no sólo dentro del disco 1 sino también en el disco en el cual almacenamos la paridad. En cuanto a almacenamiento, tengo que tener en cuenta que uno de mis discos está destinado a la paridad, entonces si tenemos 3 discos de 80Gb, de almacenamiento voy a tener $2 \times 80\text{Gb} = 160\text{Gb}$ (pues el tercer disco no es para almacenamiento de datos, sino destinado a paridad).

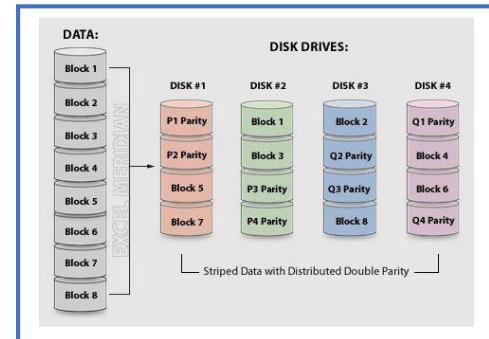


- RAID 5: es similar al RAID 4 en cuanto a lo que se busca que es aumentar capacidad, y seguridad a los datos, pero distribuye la paridad a lo largo de los distintos discos, es decir, no hay un disco exclusivo para paridad. En el arreglo se gasta el equivalente a uno de los discos para paridad, pero al estar la paridad distribuida mejora la performance. En ese sentido es mejor que el RAID 4, pero



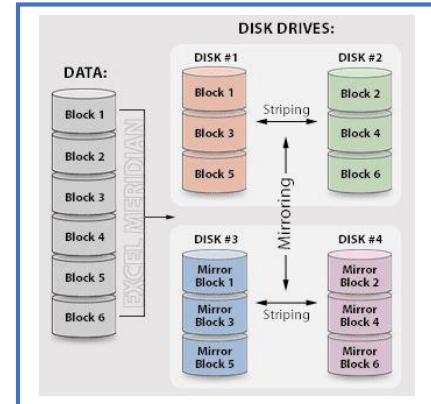
se vuelve más complicado de resolver cuando uno de los discos se rompe, puesto que es más complicado reconstruir el disco que se rompió teniendo la paridad distribuida. El RAID 5 es uno de los esquemas más usados.

- **RAID 6:** en este caso hay dos discos destinados a paridad distribuida, entonces aumenta la confiabilidad puesto que en este tipo de arreglo pueden llegar a romperse dos discos y aun así es recuperable la información del arreglo, en cuanto a capacidad, si tuviera 4 discos de 80Gb en realidad tengo destinado



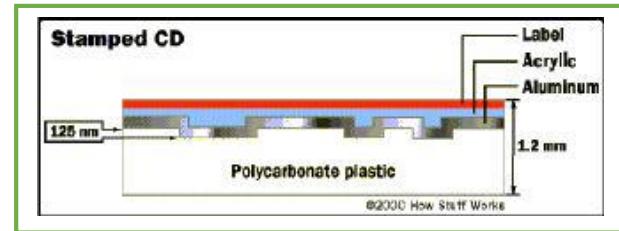
a capacidad 160Gb, ya que los otros dos discos están destinados a paridad.

- **RAID 0-1:** es una mezcla del RAID 0 y el RAID 1, necesitamos mínimamente 4 discos, en este esquema tenemos dos esquemas de nivel 0 espejados. En el ejemplo tenemos arriba disco 1 y disco 2 que están funcionando como un RAID 0, alternando la información en bloques, y hacia abajo tenemos espejado el mismo esquema. Entonces si tuviésemos 4 discos, en tamaño de almacenamiento tendríamos dos, puesto que la información está duplicada. Este esquema tiene altas prestaciones (las de las RAID 0 y 1), y mucha confiabilidad porque tenemos el esquema espejado, pero también tenemos mucho desperdicio porque la mitad del espacio la estamos gastando para el espejado.



ALMACENAMIENTO ÓPTICO: primeramente, hablaremos sobre el **CD-ROM**, que está basado sobre el CD de audio que es lo que se desarrolló originalmente para la industria discográfica. El CD-ROM es un disco de plástico transparente, en el cual se le graba la información grabando en relieve en espiral, en una de sus caras, compuesta por una sucesión de "pozos" llamados PITS. En una de sus caras se moldea un espiral, y la presencia o no presencia de PITS codifica 1 y 0. Esa espiral es una espiral muy fina, y tiene muchas vueltas en la superficie del disco (se trabaja en una escala de micrones). Para leer la información hay que recorrer el espiral detectando la presencia o no de los PITS, y la lectura se hace desde la cara que no fue moldeada (la cara plana) mediante un láser. Las diferencias de relieve (PITS0) a lo largo de la espiral, se detectan gracias a la luz reflejada, y para que la luz del láser se refleje, es necesario que a la cara moldeada se la recubra con un material muy reflectivo, de manera que refleje la luz del láser.

Pensado como CD de audio, la velocidad a la que se lee la espiral es lineal y constante, esto significa que tenemos que ir cambiando la velocidad angular, en la medida en que nos vamos alejando del centro de la espiral (más al centro, velocidad angular mayor). La velocidad lineal es de 1,2 m/s mientras que la velocidad angular varía entre 200 y 530 rpm. Si extendiéramos la pista tendría 5,6 km de largo (estos son 77 minutos aproximadamente por eso se podía grabar un CD de audio). La información está guardada en sectores (al igual que los discos magnéticos, sólo qué con otro formato, otra manera de codificar y otro tamaño de sector), y se pueden guardar 75 sectores por segundo.



Como CD de audio, se reproduce a 1,2 m/s, pero cuando usamos los CD para almacenamiento de datos podemos recorrer el CD a más velocidad, estas velocidades se expresan como múltiplo [24X o 48X] que representan la velocidad máxima que puede alcanzar la lectura y sería, por ejemplo en el caso de 24X → $24 \times 1,2 \text{ m/s}$ la velocidad máxima.

Formato de un sector de CD-ROM: cada sector ocupa 2352 bytes. Tienen un encabezamiento formado por 12 bytes de sincronismo y 4 bytes de identificación del sector (minuto, segundo, número del sector dentro del segundo, modo en el cual está codificado ese sector). Luego está el campo de datos de 2048 bytes, y por último hay 288 bytes destinados para corrección de errores, y si no hay errores destinado también como campo de datos (esto me lo va a indicar el modo dentro de la identificación del sector).

La capacidad de un CD-ROM está dada por la siguiente fórmula:

$$2 \frac{\text{KB}}{\text{sec}} \times 75 \frac{\text{sec}}{\text{seg}} \times 60 \frac{\text{seg}}{\text{min}} \times 74 \text{ min} = 666000 \text{ KB} \approx 650 \text{ MB}$$

Para el momento en que se comenzó a usar el CD-ROM como método de distribución de Software, fue muy novedoso, pues guardaba muchos más datos que los discos magnéticos flexibles (disquetes) y al mismo costo aproximadamente, por lo tanto, los costos de distribución de Software bajaron muchísimo, y su vez los costos de las unidades de lectura eran bajos, esto gracias a la industria musical.

El acceso al CD-ROM es un poco difícil respecto a la electrónica, pues para leer la información hay que ubicarse en la rama de la espiral donde se supone que se está encuentra la información que se está buscando, entonces para no recorrer de punta a punta el espiral, tiene que hacerse una aproximación. También hay que establecer la velocidad de giro del disco que cambiará

dependiendo que tan cerca estemos del centro, puesto que la velocidad angular varía. De todos modos, hoy en día el tamaño de almacenamiento del CD-ROM no es mucho.

Otros tipos de discos ópticos son el **CD-Recordable** o Grabable, que se compone de un disco de material plástico, que como no tiene nada grabado no tenemos la espiral moldeada, es de policarbonato, pero en la cara de arriba se deposita una capa de un material químico transparente y por encima de ese material una capa de aluminio reflectiva. La información es que este CD se graba con el mismo láser de lectura, pero funcionando a una potencia mayor que quema el compuesto químico y el mismo se vuelve opaco, así se genera un patrón que se interpreta como 0 y 1.

Luego está el **CD-RW** o Re-Grabable, que es un poco más complejo. Está hecho con un compuesto de cambio de fase que puede estar en dos estados: cristalino o amorfó. En el estado cristalino ese compuesto es transparente, y en el estado amorfó es opaco. Entonces en un estado deja pasar la luz y en el otro no, y para lograr un estado u otro lo que se hace es controlar el calentamiento del material. Para esto se necesitan dos potencias de láser distintas, más otra potencia más baja para la lectura del CD.

Con la llegada de **DVD**, y cuando se empiezan a utilizar estos para distribución de datos, ya estaba muy establecido como formato de distribución para el entretenimiento, es decir que ya se producía en grandes cantidades. El DVD fue concebido como una extensión del concepto de CD, en el sentido de que también es un disco de policarbonato donde también se graba una espiral por relieve, y también está grabada la información por formato de PITS. Como se trabaja en una escala de nanómetros, todas las distancias respecto al CD-ROM (que trabajaba en micrones) se achica. Para poder detectar estas dimensiones más pequeñas es necesario cambiar la luz del láser. Si bien se sigue utilizando la misma sigla, el DVD pensado para la industria del entretenimiento se llamaba “Digital Video Disk” y sólo se utilizaba para films, mientras que el DVD como dispositivo para computadoras es “Digital Versatile Disk”, y puede leer disco de computadoras o disco de videos.

El corte en un disco DVD es bastante distinto al del CD. Si bien tiene la misma altura que un CD, en realidad son dos discos pegados al medio, y habían de distintos tipos:

- **Simple lado – Simple capa:** con una sola capa de datos, esto es en relieve moldeado el disco de abajo, sobre eso se le coloca la placa de aluminio, y sobre eso otra placa de policarbonato del mismo espesor, pero sin relieve. Se lo lee del lado de abajo sólo que la capa que se lee está más cerca del láser respecto a lo que eran los CD. Guardan 4,7Gb.

- **Simple lado – Doble capa:** en el DVD se puede generar una segunda capa en la cual se hace una capa semi-reflectiva (que no llega a almacenar lo mismo que la capa reflectiva), entonces, controlando el enfoque del láser se puede hacer que este cense la capa intermedia o la capa final que es la que tiene el aluminio.
- **Doble lado – Doble capa:** en este caso, si bien el alto es el mismo, se genera una segunda capa semi-reflectiva, pero además el DVD se puede leer de ambos lados del disco.

Simple lado, simple capa (4,7GB)



Simple lado, doble capa (8,5GB)



Doble lado, doble capa (17GB)



Luego del DVD aparece el **Blu-Ray**, que es un disco también de 12 cm de policarbonato, donde también se graba en forma de espiral, pero en una escala menor al DVD, y para medir esas longitudes más chicas, es necesario volver a cambiar la longitud de onda del láser con el cual se puede leer la secuencia de 1 y 0 grabadas en el disco. En este caso se pasa a un láser de luz azul (por eso el nombre). En el caso del Blu-Ray, en el caso de un disco simple lado – simple capa, tiene una capacidad de 25Gb (frente al DVD de las mismas características que tenía una capacidad de 4,7Gb). Además, en el Blu-Ray está más cerca la capa de relieve respecto del borde en donde está el láser, puesto que hay que enfocarlo desde menos distancia. Este disco también se pensó para la industria del entretenimiento, pero en este caso guardaba videos con una definición ya mayor que con el formato DVD.

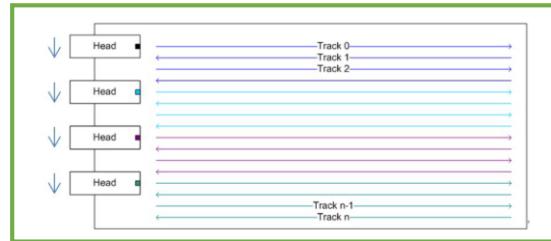
CINTAS MAGNÉTICAS: este es otro modo de almacenamiento para memoria externa, también fuera de línea. En cuanto al principio físico, la forma de grabar en las cintas magnéticas es como el de los discos magnéticos: se tiene un material base, sobre el cual se deposita un material que es magnetizable y magnetizando las sucesivas porciones. Entonces de una línea de material se puede recuperar la información, mirando las variaciones de magnetización.

Básicamente, lo mismo que teníamos en un Track de un disco magnético, en una circunferencia, acá lo vamos a tener en forma lineal a lo largo de una cinta magnética, tendremos un Track lineal que va desde una punta hasta la otra.

La cinta magnética es un medio lento, esto se debe a que, para leer algo que tengo en la cinta, tengo que arrancar desde el principio de la cinta hasta llegar al dato que quiero. Es muy económica, tiene el menor costo por byte en almacenamiento, pero las unidades de lectura para las cintas magnéticas son caras. ¿Para qué se las usa? Para BackUp y archivo.

Respecto a cómo se guarda la información en las cintas magnéticas, la información se guarda en Tracks paralelos en toda la cinta, es decir, la cinta no es un solo Track, sino que son varios Tracks que van de una punta hasta la otra de la cinta. Esta se encuentra enrollada en un carrete y para poder reproducirse o leerse, es necesario ir enrollando esa cinta en otro carrete.

Para trabajar con varios Tracks, paralelos a lo largo de la cinta hay varias cabezas de lectura o escritura que trabajan cada una leyendo o escribiendo sobre un Track, de tal manera que mientras se recorre la cinta, se pueden estar leyendo o escribiendo varios Tracks a la vez. Para aprovechar toda la cinta lo que se hace es trabajar en forma de serpentina: se graba o lee un Track por cada cabeza y cuando se llega al otro extremo de la cinta se vuelve leyendo o escribiendo sobre otros Tracks en sentido contrario.

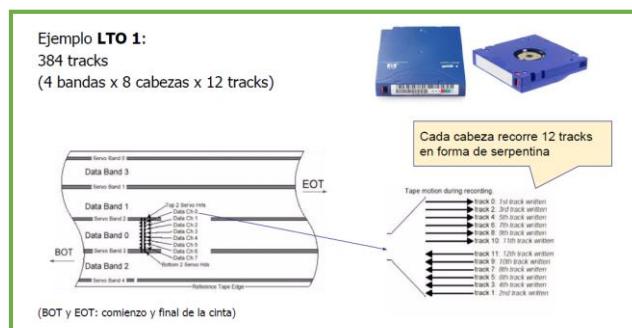


contrario. Para trabajar con esos distintos Tracks, al llegar a cada uno de los extremos, el conjunto de cabezas va descendiendo para pasar a otro conjunto de Tracks.

A lo largo del tiempo se han desarrollado, por distintos fabricantes, varias tecnologías y familias de productos diferentes de cada cinta magnética (cada uno con su tipo de cartucho, formato de grabación, tamaño de cinta, etc). En el ambiente de la tecnología, lo que pasa es que muchas tecnologías van cayendo en desuso puesto que aparecen nuevas tecnologías que las mejoran en gran medida.

Una de las tecnologías más difundidas actualmente es la llamada LTO (Linear Tape Open) que fue definida a partir de un estándar abierto por parte de IBM, HP y Seagate a fines de los 90. Estas tres compañías propusieron un estándar para construcción de cintas magnéticas que es el LTO, y es estándar abierto puesto que las especificaciones están completamente disponibles para que otros fabricantes puedan fabricarlas siempre y cuando cumplan todo el estándar.

En el LTO los Tracks que también se ubican a lo largo de la cinta, están agrupados en 4 bandas, es decir que hay 4 zonas en esa cinta en donde se desarrollan los Tracks. La información se irá grabando por banda: primero se graban todos los Tracks de la banda 0, luego la banda 1 y así siguiendo. Para pasar de banda a banda lo que hay que hacer es mover el conjunto de cabezas de escritura lateralmente (un movimiento mayor que el necesario dentro de la banda para realizar la serpentina de Tracks).



Además de unidades individuales para grabar las cintas, en estos sistemas hay equipos llamados **librerías**, que son capaces de tener varias cintas disponibles dentro de la unidad, como una estantería automática, en donde hay un mecanismo selector que lo que hace es, dependiendo de la cinta que se necesita leer, la toma y la coloca en la unidad de lectura. Entonces uno puede tener muchas unidades de cinta en el mismo equipo y una sola unidad de lectura que esta siendo alimentada del cartucho de cinta que se necesita leer.

En cuanto a los **tiempos de acceso**: desde que el cartucho se inserta en la unidad de lectura hasta que la cinta es alimentada dentro de la unidad de lectura y esta lista como para empezar a funcionar, pasan unos 15 segundos aproximadamente, y para llegar a la mitad de la cinta se tarda más o menos un minuto, esto es mucho tiempo comparado a los tiempos de acceso a un disco, pero como las cintas estan pensadas para BackUp y archivo, no nos importa, además es lento para acceder a un punto pero la velocidad de transferencia es alta, transfiere datos a la velocidad de un disco rígido (300Mb/seg).

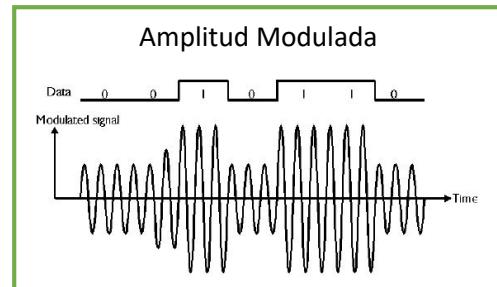
MODEM: la palabra Modem es un acrónimo que significa **M**Odulador – **D**EModulador. Para mandar información tenemos dos estados, en 0 y 1 que mandamos mediante niveles de tensión, ahora si yo quiero mandar información a través de un cable y el cable es muy largo, ya no será efectivo mandar 0 y 1 como dos niveles de tensión a muy larga distancia, por cuestiones eléctricas. Entonces será necesario generar una señal que pueda viajar a través de mayor distancia en el cable. Este es el caso de la comunicación de datos mediante una línea telefónica. El MODEM lo que va a hacer es adaptar una señal de 0 y 1 que manejamos dentro de nuestra computadora para poder transferirla a través de una línea telefónica. Cuando transferimos la información a través de la línea telefónica, vamos a tener determinada una tasa de **b**its **p**or **s**econdo **[bps]** que es la cantidad de bits por segundo que vamos a poder enviar mediante la señal. También tenemos la **t**asa de **B**audio que es la cantidad de cambios en la señal por segundo. Lo máximo que admite el sistema telefónico son 2400 baudios.

El tipo de señal tiene que ser una señal variable con una frecuencia entre 50 y 3500 Hz. Si la frecuencia que enviada está entre 50 Hz y 3500 Hz, viaja bien desde un teléfono hasta el otro. Si desde un punto de comunicación transmito una frecuencia constante, llega hasta el otro lado porque sería un tono de audio fijo, pero no puedo mandar información, puesto que para que en el destino llegue algún tipo de información lo que tengo que hacer es variar la señal que estoy mandando; le puedo variar la intensidad o el tono, por ejemplo, y así señalizar cosas. Esto sería **modular**: alterar una señal para que viaje desde un punto hasta otro con un tono, que yo voy a ir modulando de formas distintas o combinadas, de tal modo que del otro lado llegue bien una señal, y sabiendo cual fue la modulación que se usó para codificar la información, poder

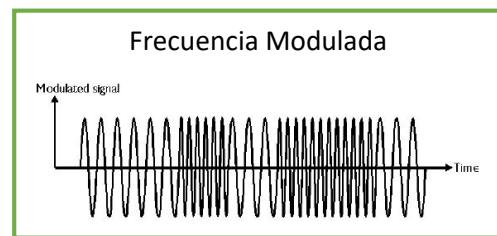
decodificarla. Entonces el MODEM justamente lo que hace es modular la información, es decir, a partir de una secuencia de 0 y 1 genera una modulación sobre una secuencia portadora, y del lado de la recepción tengo otro MODEM trabajando como un demodulador, que recibe una señal que está modulada, y recupera los bits que se usaron para modular.

Hay tres formas de modular:

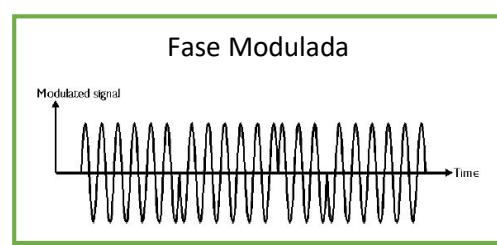
- **Modulación de Amplitud:** tenemos una secuencia de bits de 0 y 1 y correlacionado con esos 0 y 1 yo puedo ir modulando esa frecuencia llamada frecuencia de portadora. Como sube y baja la intensidad en esa frecuencia va a determinar la señal de 1 y 0 que quiero enviar mediante modulación de amplitud.



- **Modulación de Frecuencia:** el tono de menos frecuencia es el más grave y el tono de más frecuencia es el agudo. En este caso determino la señal de 0 y 1 que quiero enviar en base a la frecuencia de la señal.



- **Modulación en Fase:** tenemos la sinusoides que cada tanto salta y muestra una discontinuidad, saltándose un ciclo o un ciclo y medio, etc. Entonces cambiando la fase de esa señal se codifican los 0 y 1.



Podemos mandar varios bits por cada baudio. En los ejemplos anteriores tenemos una frecuencia para el cero y otra para el 1, ¿Qué pasa si tenemos más cantidad de frecuencias diferentes? Por ejemplo, si tengo 4 frecuencias, cada una de esas frecuencias codifica 2 bits, es decir, cada una de estas frecuencias me van a representar 00, 01, 10, 11. ¿Cuántas veces por segundo puedo cambiar de frecuencia o de amplitud? 2400 veces por segundo.

Notemos que los límites del sistema telefónico se dan en dos aspectos: las frecuencias que pueden pasar, que van de 50 a 3500 Hz, y la cantidad de veces que puedo cambiar la frecuencia o amplitud por segundo, que es 2400 veces como máximo. Si tuviese 8 niveles, transmito 3 bits por cada baudio y así siguiendo. Entonces se cumple que → $tasa \text{ bps} = tasa \text{ baudio} \times \log_2(n)$.

Además de la capacidad de modular y demodular, los MODEMS tienen la capacidad de hacer otro tipo de funciones, como, por ejemplo, la del discado (comunicarse con otro MODEM generando señales o recibir la señal de un MODEM que lo está llamando). La computadora tiene que poder controlar el discado, esto lo hace a través de un comando que le da al MODEM para que este disque. Una vez que se comunicó con el otro MODEM, se establece entre ambos una tasa de bit **[bit rate]** a la cual van a hacer la comunicación. Estos son los “Smart MODEMS” o también llamados “Hayes Compatible”. El máximo bit rate que han logrado esta línea de MODEMS por línea telefónica es de 57600 bps (56K).

Luego salieron otros llamados **MODEMS de banda ancha**, que esos utilizan otra tecnología, y necesitan otra línea, tienen que ser líneas digitalizables.

Respecto a la comunicación que hay entre la computadora y el MODEM, esta es una comunicación de 0 y 1, tenemos un cable para transmitir datos y un cable para recibir datos. Para que esta comunicación se pueda dar, es necesario tener ajustado el reloj puesto que tienen que saber cuándo empieza y cuando termina cada bit. Entonces además de transmitir los datos por una línea de datos, debo transmitir una línea de reloj, eso implicaría hacer una comunicación sincrónica. Pero se utiliza una comunicación asincrónica, entonces ¿Cómo sabe el MODEM cuando empieza un 0 y cuando empieza un 1 que le está transmitiendo la computadora? Para eso se utiliza un protocolo de comunicación asincrónica que establece lo siguiente: la línea siempre está en estado de reposo enviando un 1, cuando está por enviar un byte, pasa por un tiempo de bit esperado a 0, esto se llama marca de arranque. Luego se mandan los bits de datos (aproximadamente un byte), luego un bit de paridad y luego el bit de stop, para volver a comenzar el ciclo con otro bit de arranque. Esto se hace para evitar diferencias muy grandes entre los tiempos del MODEM y la computadora, hay que ir re-sincronizando cada aproximadamente un byte de envío de datos.

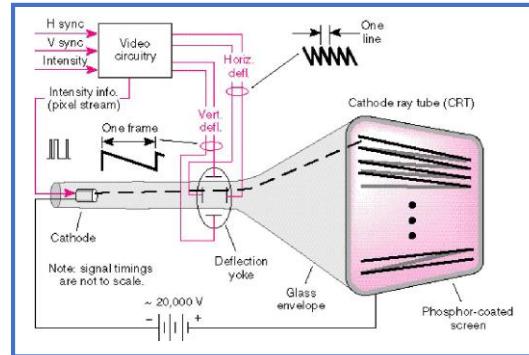
DISPOSITIVOS DE ENTRADA DE DATOS: es necesario tener la capacidad de poder dialogar a nivel usuario con nuestro equipo de cómputo, es decir poder ingresar datos al sistema, por ejemplo, a través del teclado o del mouse.

Las tasas de entrada de datos a través de teclado y mouse son muy lentas respecto a lo que es la computadora. Desde el teclado está previsto que a lo sumo se ingresen 10 caracteres de 8 bits por segundo, es decir 80 bits por segundo (es una tasa de entrada muy baja). Con el mouse es parecido, el ser humano no es capaz de clickear más rápido de lo que ingresa caracteres, es decir, 10 clics por segundo. Si es más rápido en cuanto a reportar los cambios de posición del puntero del mouse (se estima un cambio en los bits de la posición X e Y por milisegundo).

El desafío del diseño de dispositivos de entrada de datos manual es reducir el número de partes móviles, puesto que estas suelen ser las que generan problemas, se traban, ensucian, etc.

DISPOSITIVOS DE SALIDA DE DATOS

Primero vamos a hablar de los **monitores de video**, específicamente sobre los monitores tipo SRT. A la derecha hay un esquema de un **tubo de rayos catódico** o tubo de rayos de electrones. El lado de la pantalla (a la derecha) es lo que vemos como usuarios, está recubierta con un material pasado en fósforo que cuando recibe el impacto



de una corriente de electrones, brilla generando el punto luminoso que nosotros vemos. El haz de electrones sale desde la punta del tubo (donde dice cátodo), lo que ahí hay es una emisión de una corriente de electrones y una tensión muy alta entre el cátodo y la pantalla (20000V). esta tensión hace que los electrones que se emiten en el cátodo sean muy atraídos por la pantalla e impacten a alta velocidad. Este impacto de la corriente electrónica es lo que genera un punto en la pantalla. Pero hasta acá, lo único que tenemos es un punto en el centro de la pantalla.

Si miramos en el medio del esquema, hay unas placas que se llaman dispositivos de desplazamiento de deflexión, que lo que hacen es generar tensiones eléctricas de tal manera que atraen al flujo de electrones que está viajando desde el cátodo a la pantalla. Estas tensiones no son tan altas, entonces no sirven para frenar a los electrones, pero si para desviarlos. Si la placa que está arriba es positiva respecto de la que está abajo, los electrones tienden a ser atraídos un poquito hacia arriba, entonces de este modo, los electrones en lugar de impactar en la pantalla justo en el centro, se desviarán y pegarán el algún punto más arriba. Si la deflexión está cambiada, (la placa negativa arriba y la positiva abajo) impactarán en algún punto abajo del centro. Lo mismo sucede con las placas laterales. En definitiva, lo que se hace con esas placas es la deflexión horizontal y vertical. Las señales eléctricas son como un diente de sierra, en donde tiene más frecuencia el horizontal que el vertical, pero ambas son de la misma naturaleza, tienen un pico máximo y van bajando hasta un pico mínimo de una forma rectilínea con una cierta pendiente, y cuando se llega al pico mínimo se vuelve al pico máximo de un salto. ¿Cómo se ve eso en los electrones que van impactando en la pantalla? Cuando se está en los dos picos superiores, los electrones impactan sobre el extremo superior izquierdo. Cuando el diente de sierra superior se va desplazando hacia abajo, los electrones comienzan a impactar cada vez más hacia la derecha y luego cuando llega hacia su punto mínimo, es cuando está en la extrema

derecha. Luego de eso viaja rápidamente hacia su punto máximo de nuevo, mientras, paralelamente, el desplazamiento vertical va descendiendo y desplazando el haz luminoso hacia abajo. El movimiento de arriba hacia abajo es más lento que el barrido de izquierda a derecha. Este movimiento que arranca desde arriba y va bajando se llama **raster**. Para que este circuito de video funcione, necesita entradas que son de sincronismo horizontal y sincronismo vertical, estas son relojes que pasan de 0 a 1 e indican cuando comenzar el movimiento de deflexión horizontal y vertical. Además, tenemos la línea de intensidad que regula la intensidad con que se emiten los electrones desde el cátodo: si puedo controlar los niveles de intensidad puedo controlar el tener un punto luminoso o no tener un punto. A pesar de que es un solo punto luminoso el que tengo desplazándose en la pantalla, para el ojo humano es como si fuera una imagen fija simplemente porque no podemos captar los movimientos tan rápidos del punto. Esto se llama persistencia de las imágenes en la retina.

Si este punto luminoso lo vamos modulando vamos a tener zonas en la pantalla que repetidamente están siendo iluminadas, y otras que repetidamente no están siendo iluminadas, es decir segmentos que repetidamente están prendidos, y otros que están apagados.

En resumen:

- En estos casos, los monitores pueden ser de color blanco y negro.
- La imagen va a ser trazada en pantalla de a una línea por vez (esto sería el modo raster que explicamos más arriba).
- Los pixeles se marcan con el haz de electrones, y dicho haz se desvía horizontal o verticalmente por las placas de deflexión.
- Se muestran entre 50 y 60 cuadros completos por segundo, esto es, la cantidad de veces que se hace el barrido vertical, trabajando a esta velocidad, mi ojo no se da cuenta de que es sólo un punto luminoso viajando en pantalla.
- En cuanto a volumen de información, supongamos que la resolución vertical fuera de 500 líneas y la resolución horizontal de 700 puntos por línea, si esto lo hicieramos a 60 cuadros por segundo, entonces la señal de intensidad de prender o apagar el haz de electrones nos indica que estamos definiendo $60 \times 500 \times 700 = 21$ millones bits por segundo para poder crear una imagen que va a estar en blanco y negro.

En cuanto a la diferencia entre los monitores a color o los monitores en blanco y negro. En el caso de monitores en blanco y negro, si yo quisiera tener un monitor que me muestre imágenes con distintos niveles de intensidad, necesitaría que la señal no fuera sólo 0 y 1, sino que también se puedan codificar niveles de intensidad. Entonces si, por ejemplo, tengo 256 niveles de intensidad de punto luminoso necesito 8 bits para poder codificarlos [$2^8=256$].

Ahora en el caso de monitores a color, tenemos que tener en cuenta que podemos recrear cualquier color a partir de tres: rojo, azul y verde, de acá sale la forma de codificar color llamada RGB (red, green, blue). Entonces ¿Cómo generaría cualquier color en un monitor? Cada pixel está formado en pantalla por franjas de 3 colores, y a la vez vamos a tener 3 cañones de electrones. Hay un cañón que sólo enfoca a las partes rojas de cada pixel, otro a las partes azules y otro al verde. Entonces cuando se hace la deflexión horizontal y vertical los tres cañones son movidos juntos. Si además jugamos con la intensidad de esos haces de luz, teniendo 256 intensidades para cada uno (rojo, verde y azul), puedo combinar 2^{24} colores diferentes, esto es lo que se conoce como **True Color** que es lo máximo que distingue el ojo, y para generarla preciso 3 bytes (24 bits).

En cuanto a los datos y como guardamos la información que nos represente lo que queremos mostrar en pantalla (memoria de visualización), tenemos dos esquemas básicos: el **modo terminal** y el **mapeado en memoria**.

- **Modo terminal:** es un monitor de video apuntado a mostrar caracteres: mostrar sólo texto sin variar tipo ni tamaño de letra, tenemos la memoria de visualización y el teclado armados juntos en el mismo dispositivo y generando lo que se llama terminal de datos. Si, por ejemplo, yo quiero que en pantalla se muestre "hola" tengo que mandar la información para que muestre cada uno de estos caracteres, pero no tengo que decirle como es cada carácter.
- **Mapeado en memoria:** el monitor de video con memoria de visualización va a estar mapeado en memoria, es decir, va a ser accesible a través de los buses del sistema de cómputo. De manera que se pueda escribir rápidamente con alta tasa de transferencia de datos a la memoria de visualización. Eso va a estar orientado a trabajar en modo gráfico, voy a decidir que mostrar a nivel pixel, además me va a interesar trabajar con secuencia de imágenes que transmitan movimiento (sucesión de imágenes que muestran leves cambios cuadro a cuadro y cambian con una determinada tasa de imágenes por segundo, para que así mi ojo perciba el movimiento). Para esto necesito un ancho de banda grande, y por eso necesito tener conectados los buses.

Monitores alfanuméricos: en este caso, en la memoria sólo vamos a almacenar los códigos de carácter. Vamos a suponer la pantalla dividida en tantas filas por tantas columnas de caracteres. Ahora cada carácter va a implicar varios pixeles, pero eso lo va a definir la misma lógica del circuito de video, pero sólo le tuvimos que indicar que carácter ASCII es el que queremos mostrar. Entonces los códigos de carácter se convierten en pixeles gracias a una ROM de caracteres. Por

cada carácter que nosotros escribamos se va a generar la información de los distintos pixeles, esto es, pixeles sucesivos en varias líneas sucesivas.

A la derecha se ve un ejemplo de una pantalla alfanumérica con un texto determinado puesto en pantalla. En la memoria se van a almacenar sólo los códigos de carácter, vemos el detalle de lo que es la memoria de visualización, esta es una serie de bytes en donde con cada byte estoy definiendo cual es la serie ASCII que quiero representar. Voy a tener una secuencia de bytes que van a representar todos los caracteres de la fila 0, otros para la fila 1 y así siguiendo hasta la última fila. Si por ejemplo tenemos 80 columnas por 64 filas de caracteres, para cada fila n van a ser 80 bytes. El circuito de video lo que va a hacer es ir leyendo esta visualización y generando los pixeles, para eso vamos a usar una ROM de caracteres. Esta es una memoria de sólo lectura que ya fue grabada por el fabricante, que en distintas posiciones tiene una palabra binaria almacenada. Lo que vamos a tener en sucesivas posiciones de memoria es la codificación de los pixeles de cada línea de carácter.

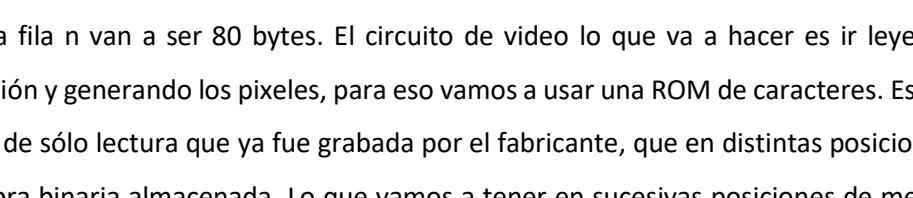


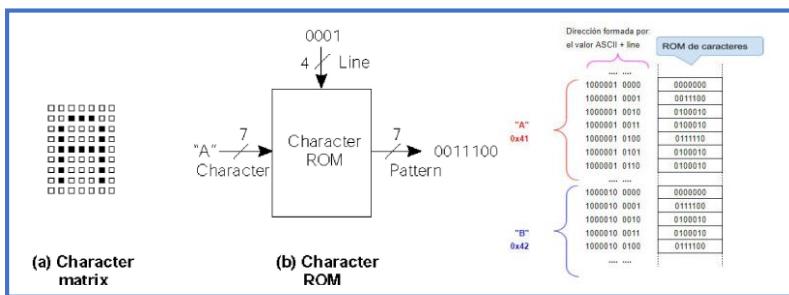
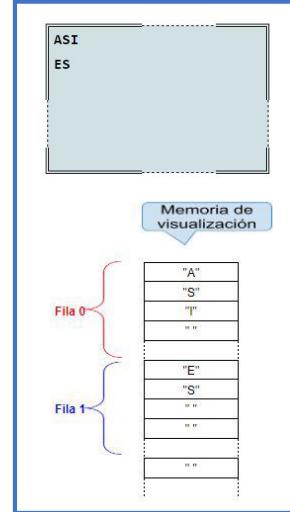
Diagrama de memoria de visualización:

Memoria de visualización

| |
|-----|
| "A" |
| "S" |
| "T" |
| " " |
| " " |
| "E" |
| "S" |
| " " |
| " " |
| " " |

Fila 0

Fila 1



La salida de la ROM de caracteres va a ir a un registro de desplazamiento, y va a haber un reloj que cada vez que cambia de estado indicará la salida por ese registro de desplazamiento de los bits de secuencia y así se genera una secuencia de bits que representan a esos bits que nos permiten dibujar una línea de la letra A.

Monitores gráficos (bit mapped): en este caso, cada pixel es representado por bits en memoria. En el caso de los visualizadores blanco y negro se puede usar un bit por pixel, mientras que en la gama de grises o en color se necesitarán varios bits por pixel.

IMPRESORAS

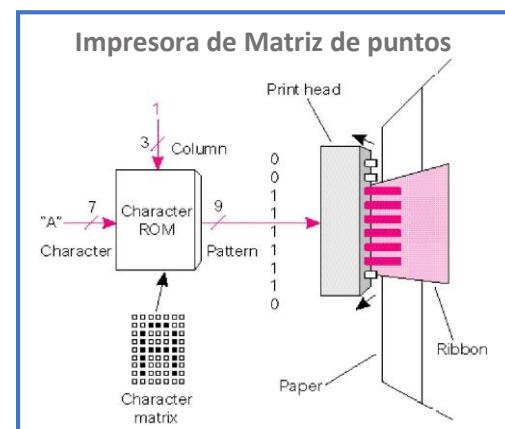
Las impresoras son periféricos que escriben la información de salida sobre papel. Su comportamiento inicialmente era muy similar al de las máquinas de escribir, pero hoy día son mucho más sofisticadas.

Las impresoras son, junto a las pantallas, los dispositivos más utilizados para poder ver en forma directamente inteligible para el hombre los resultados de un programa de computadora. Las impresoras tienen dos partes diferenciadas: la parte mecánica y la parte electrónica. Aquí la parte mecánica, además de encargarse de seleccionar el carácter a partir del código de E/S correspondiente, debe dedicarse a la alimentación y arrastre del papel.

Las impresoras tradicionalmente utilizaban papel continuo, en cuyos márgenes existen unos orificios. En este caso, el arrastre se efectúa por un tractor que dispone de unos dientes metálicos que encajan en los orificios laterales del papel. En la actualidad existen también impresoras que no necesitan papel continuo, efectuándose el arrastre por fricción o presión, como en el caso de las máquinas de escribir o en las fotocopiadoras convencionales.

Impresoras de impacto: son aquellas en las cuales hay un elemento físico que se mueve y golpea una cinta entintada que se apoya sobre el papel y con eso transfiere la información. Una de estas impresoras de impacto son las que se llaman de **Carácter Formado**, que tiene un sello o relieve con cada carácter entonces al golpear sobre el papel transfiere la imagen de ese carácter. Estas impresoras ya no se usan, pero hubo un formato de las mismas muy famoso conocido como formato Margarita, llamada así porque había una especie de rueda con rayos o pétalos que rota, donde en cada extremo de esos pétalos hay un carácter en relieve. Si queríamos cambiar el tipo de letra teníamos que cambiar la margarita, por eso se llama de carácter formado. Otro formato es el formato de cinta, en donde los caracteres en relieve están en una cinta metálica que se usa junto con una serie de martillos que van golpeando los caracteres y transfiriéndolos al papel.

Otro tipo de impresora de impacto es la **Impresora de Matriz de Puntos**, en este caso no tenemos carácter formado, sino que tenemos la posibilidad de generar una línea de puntos, y con mediante estas líneas verticales de puntos se irá armando la imagen del carácter. Hay una serie de punzones manejados por solenoides en línea (9 punzones), cada punzón golpea una cinta entintada y marca un papel, y de este modo se irá armando el carácter (dependiendo de lo que quiero imprimir se activan 1 o más punzones por vez).



Este tipo de impresora imprime una columna por vez, y puede usar una ROM generadora de caracteres. En este caso, la ROM se lee en paralelo por columna, en vez de ser en serie y por fila como es el caso anterior. En cuanto a definición, no es muy buena pues los caracteres están formados por estos puntos que son observables a simple vista. Son un poco ruidosas y lentas.

La ventaja que tienen es que pueden trabajar en ambientes más sucios, como talleres, y también la capacidad que tienen de trabajar sobre múltiples copias, es decir de la misma impresión puedo sacar varias copias (por ejemplo, usando carbónico).

Impresoras Láser: está orientada a página completa. Esta impresora recibe la información en forma gráfica, con una definición a nivel de pixeles de lo que se quiere imprimir. Tienen una alta definición, (hablamos de puntos por pulgada como la cantidad de puntos que la impresora puede hacer en una pulgada). La impresora láser hace un uso recurrente de la electricidad estática. Tenemos un tambor recubierto en un material fotosensible, mientras este tambor va rotando hay un electrodo que lo carga con carga eléctrica, y así se genera una carga eléctrica sobre todo ese tambor a medida que va rotando. Este tambor, al ser fotosensible, se vuelve conductor en las zonas en las cuales recibe luz. El tambor es iluminado con un láser que va generando puntos en los cuales lo vuelve conductor, y la carga eléctrica de ese lugar se pierde. Entonces, el tambor, luego de pasar por el láser, tiene zonas cargadas con carga eléctrica, y zonas que no (que son las que tocó el láser). El tambor sigue girando, ya con puntos cargados y puntos descargados, y pasa frente a un rodillo que tiene depositado tóner, un material plástico con un pigmento muy finito que se adhiere a las zonas cargadas. El tambor sigue girando hasta que llega a frente al papel, y debajo del papel hay un rodillo que está girando con una carga eléctrica más fuerte que la que tiene el tambor, por lo tanto, atrae las partículas de tóner contra el papel. De esa manera, los puntos que estaban cargados con tóner en el tambor se transfieren al papel, y de este modo, la imagen que se desarrolló en el tambor, se transfiere al papel. Todo sigue girando, y el papel se sigue desplazando con el tóner suelto en el lugar en el que tiene que ir. Para que el pigmento se adhiera, el papel pasa por unos rodillos llamados fusores que calientan el papel y el tóner para fundirlo. De esta manera el pigmento queda adherido al papel y el papel queda impreso.

Respecto al láser, este hace un recorrido de izquierda a derecha, descargando algunas zonas, para decidir a qué zonas se le va a adherir el tóner y a cuáles no. Para hacer ese barrido bastaría con tener un emisor láser y un espejo que se mueva de izquierda a derecha para reflejar el haz del láser. Además de esto, lo que se hace es modular el láser, prendiendo y apagándolo.

¿Cómo se logra tener una sensación de niveles de grises? Variando la cantidad de puntos del tóner negro en una determinada zona.

Impresoras Ink-Jet (Chorro de tinta): en este caso tenemos un cabezal que es una columna de puntos (similar a una impresora de matriz de puntos), pero acá no hay impacto, sino que hay gotas de tinta muy pequeñas, que del cabezal son descargadas en el papel. Entonces, mientras

la cabeza de impresión se mueve sobre el papel lo que se hace es lanzar “chorros” de tinta (estas pequeñas gotas) desde un montón de boquillas (300 aprox.). La impresora controla gota a gota si se va o no a lanzar en el papel, cada boquilla recibe la orden de enviar o no la gota, mientras se va desplazando lateralmente la línea de impresión. En este caso, las impresoras pueden imprimir en negro o en color. Para generar distintos colores en la impresión (vale también para las láser, sólo que estas no son tan comunes en color porque son muy caras), se usa una paleta de colores llamada sustractiva, en donde mientras más colores mezclo, más negra queda la tinta, y si no pongo ningún color, me queda el papel blanco (al revés que el RGB en los monitores). Los colores básicos en la impresión son Cian, Magenta y Amarillo (CMYK).

Las impresoras Ink-Jet son más económicas que las láser, pero son más caras a nivel insumos entonces a la larga, si tenemos que imprimir mucho, nos terminan conviniendo las láser porque son más baratas por copia. Ahora las Ink-Jet a color son mucho más económicas que las impresoras láser a color, por eso las impresoras láser a color no se ven mucho en el mercado.

En cuanto a tecnologías Ink-Jet hay dos: una que se llama **de Burbuja Térmica**, las HP y las Canon utilizan esta tecnología, y la otra forma es la **Piezoeléctrica**, que utiliza Epson. En ambos casos tenemos un cabezal con una columna de boquillas donde cada una de ellas tiene un transductor que genera la gotita. Además, en ambos sistemas se controla gota a gota si se va a inyectar o no sobre el papel.

En el primer caso, tenemos en cada boquilla un calefactor microscópico que se calienta y calienta la tinta, generando una burbuja en la tinta que genera presión y hace que una gota de tinta salga de la boquilla. Una vez que sucedió eso el calefactor ya se enfrió. Se enfriá la tinta de nuevo, y como se perdió una burbuja de tinta se genera una presión negativa en el receptáculo que lo que hace es que chupe una gota de tinta del depósito de tinta para reponer la tinta que perdió y así estar llena nuevamente.

En el segundo caso, formando parte de cada boquilla hay un cristal especial que se llama piezoeléctrico que cuando recibe una tensión eléctrica se deforma generando una presión en el receptáculo de tinta de esa boquilla, y gracias a esto se logra que se inyecte una gota de tinta hacia el papel. Luego se quita la tensión que generaba la deformación del piezoeléctrico y como falta una gota, se genera una presión negativa que hace subir una reposición de gota del depósito de tinta.

Ambas tecnologías compiten y son buenas, y hay toda una gama de calidades en torno a ellos.