Organización del Computador I

Gianfranco Zamboni

26 de octubre de 2019

Índice

1. Arquitectura de Von Neumann

El siguiente diagrama muestra como está conformada una computadora básica que usa este tipo de sistemas:

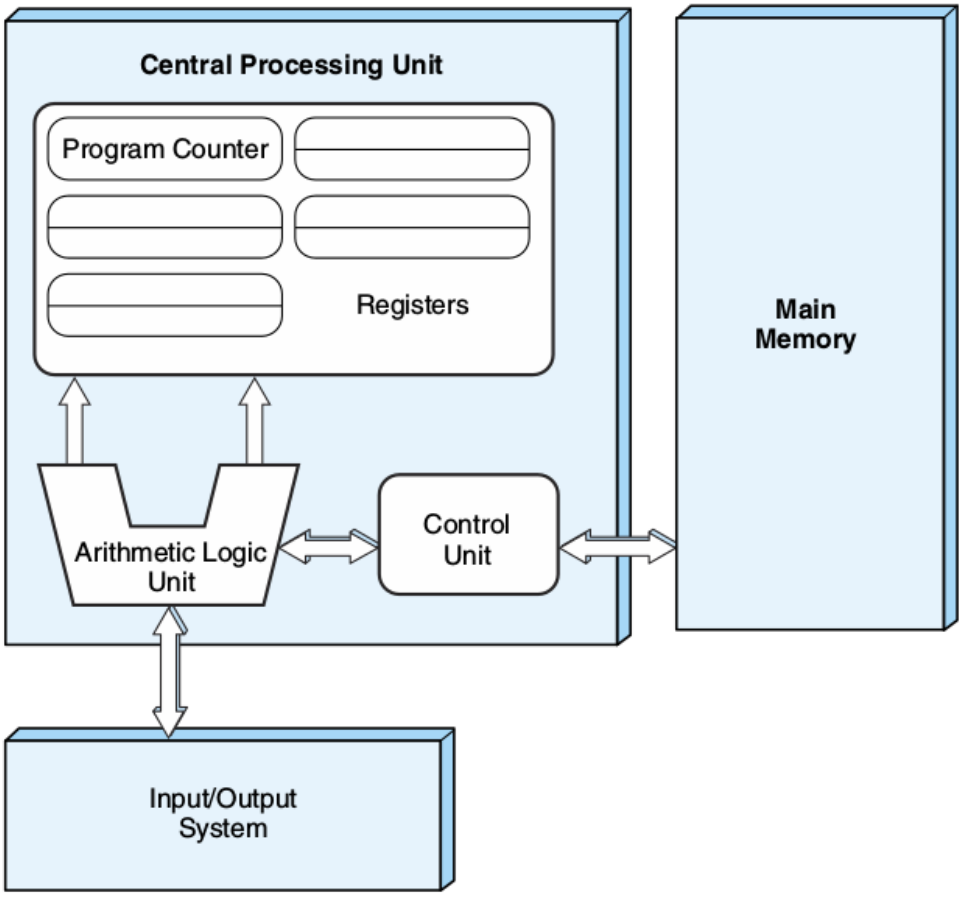


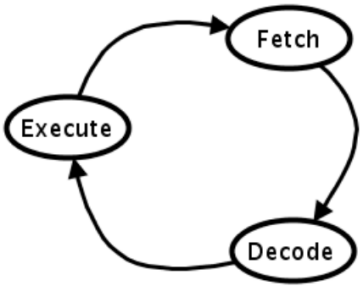
FIGURE 1.4 **The von Neumann Architecture**

Se puede ver que consta de 3 componentes principales:

- Una **memoria principal** (main memory) en donde se guardan los programas (secuencia de instrucciones) que controlan las operaciones realizadas por la computadora.
- Un **sistema de entrada/salida**
- Una **unidad central de procesamiento** (Central Processing Unit, CPU) que contiene una **ALU** (Arithmetic Logic Unit) para realizar operaciones aritméticas y lógicas, una **unidad de control** (Control Unit, CU) que comunica la ALU con la memoria principal y un conjunto de registros entre los cuales se encuentra el **Program Counter** (PC) que indica donde se encuentra la próxima instrucción que debe ser ejecutada.

Este tipo de arquitectura es capaz de llevar a cabo el procesamiento de instrucciones de manera secuencial y uno de sus grandes problemas, conocido como *cuello de botella de Von Neumann*, es que no puede realizar operaciones de datos y una extracción de instrucción simultanea debido a que la unidad de control y la memoria principal están conectados por un único camino que solo permite usar dichos componentes para una extracción de instrucción o para una operación de datos, pero no ambas a la vez.

1.1. Ciclo de ejecución de Von Neumman



Arriba, se muestra el ciclo de ejecución de Von Neumann, una iteración de dicho ciclo está caracterizada por 3 etapas:

- **Fetch**: La unidad de control busca, en memoria, la próxima instrucción a ser ejecutada usando la información del program counter.
- **Decode**: La unidad de control decodifica la instrucción para que sea entendible por la ALU y, si es necesario, busca en la memoria los datos necesitados para la correcta ejecución de la misma.
- **Execute**: La ALU ejecuta la instrucción decodificada y guarda los resultados en los registros o en la memoria.

2. Entrada/Salida

Se define un subsistema de I/O como un conjunto de componente que mueven datos entre dispositivos externos y un host compuesto por una CPU y un memoria principal.

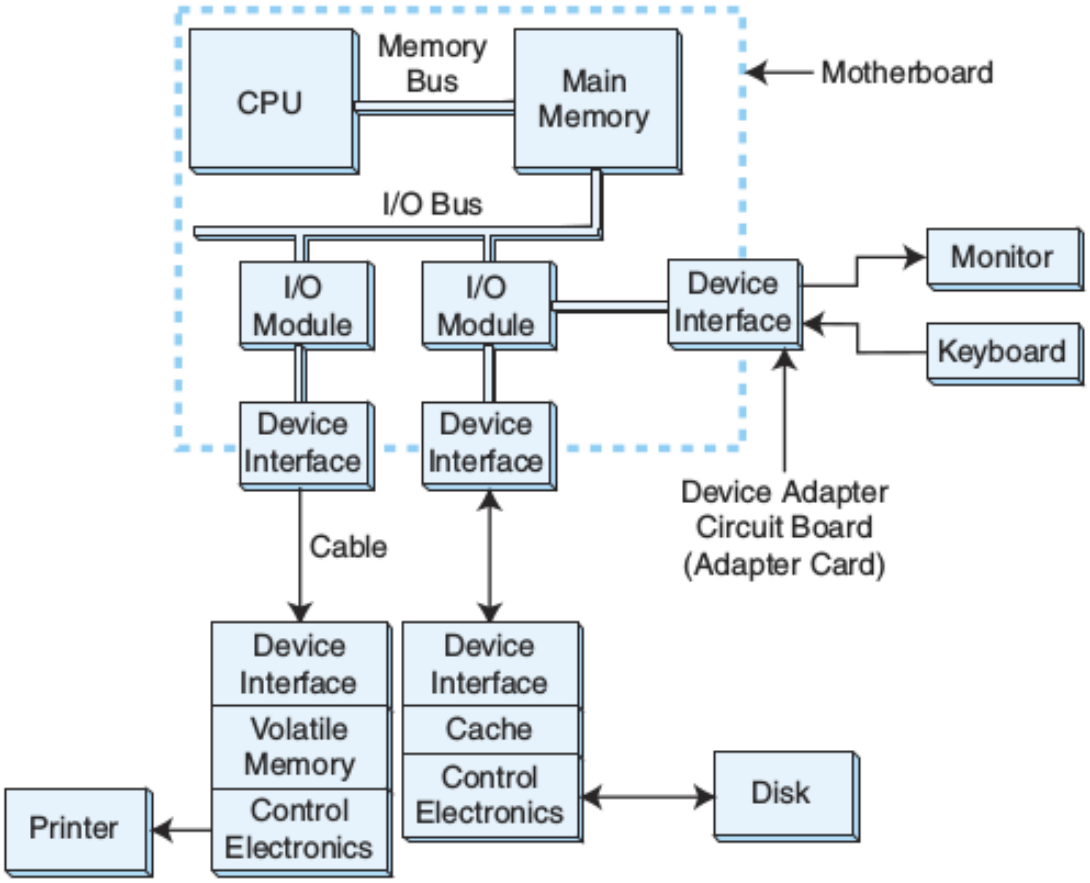


FIGURE 7.1 A Model I/O Configuration

Por lo general, un sistema de I/O esta compuesto por:

- Bloques de memoria principal dedicadas a funciones de entrada/salida
- Buses por los que se mueven distintos tipos de datos desde la memoria hacia los dispositivos y viceversa.
- Modulos de control, tanto en el host, como en los dispositivos que aseguran la correcta ejecución de las tareas de I/O.
- Modulos I/O que indican hacia dónde y desde dónde debe llegar la información.

Además, se necesita un mecanismo para controlar el funcionamiento de los distintos dispositivos y prevenir, de esta forma, los problemas que podría ocasionar, por ejemplo, la escritura en memoria de dos o más de ellos a la vez.

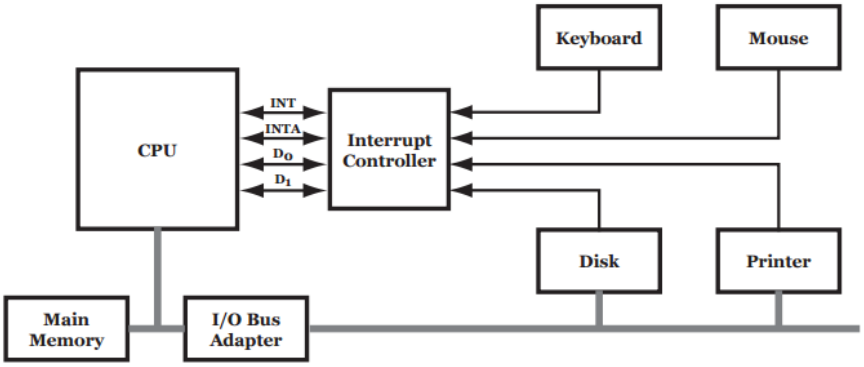
2.1. Polling

Se asigna un registro de usa exclusivo a cada dispositivo. La CPU monitorea constantemente cada uno de estos registros esperando a que algunos de los dispositivos escriba en su registro el estado *data ready* indicando, de esta manera, que desea realizar una operación de entrada/salida.

Uno de las grandes desventajas de este método es que, mientras no haya ningún request para hacer operaciones de I/O, la CPU se mantiene en el loop constante descripto mas arriba, prohibiéndole realizar cualquier otro tipo de tareas.

2.2. Interrupciones

Se agrega, al registro de flags del CPU, un flag de interrupción que indica cuando un dispositivo necesita realizar una operación de I/O. Por lo general para implementar un sistemas con interrupciones se usa un controlador de interrupciones (IC):



Los dispositivos se conectan al IC y cuando necesitan realizar alguna operación mandan una señal al mismo. Este, recibe la señal y manda la señal INT al CPU, activando su flag de interrupciones. El CPU termina de ejecutar la instrucción en curso y verifica si hay interrupciones pendientes. Si las hay, notifica al IC con la señal INTA indicando que está lista para atender el request y, entonces, el IC le envía el id del dispositivo que pidió el request.

Una vez hecho esto, el CPU guarda el contexto del programa que se está ejecutando, deshabilita las interrupciones y ejecuta la rutina de atención de interrupciones correspondientes. Por último, la CPU restaura el contexto del programa, vuelve a habilitar las interrupciones y vuelve a ejecutar el programa desde donde lo habia dejado.

2.3. Direct Memory Access (DMA)

Para mejorar el rendimiento del CPU durante las operaciones de I/O que implican transferencia de datos entre la memoria principal y algún dispositivo particular, se agrega un DMA Controller. Cuando este dispositivo se encuentra presente y se debe realizar una operación de este tipo, el CPU envia al DMA donde se encuentran los bytes que deben ser copiados, la cantidad que debe copiar y a dónde deben ser copiados.

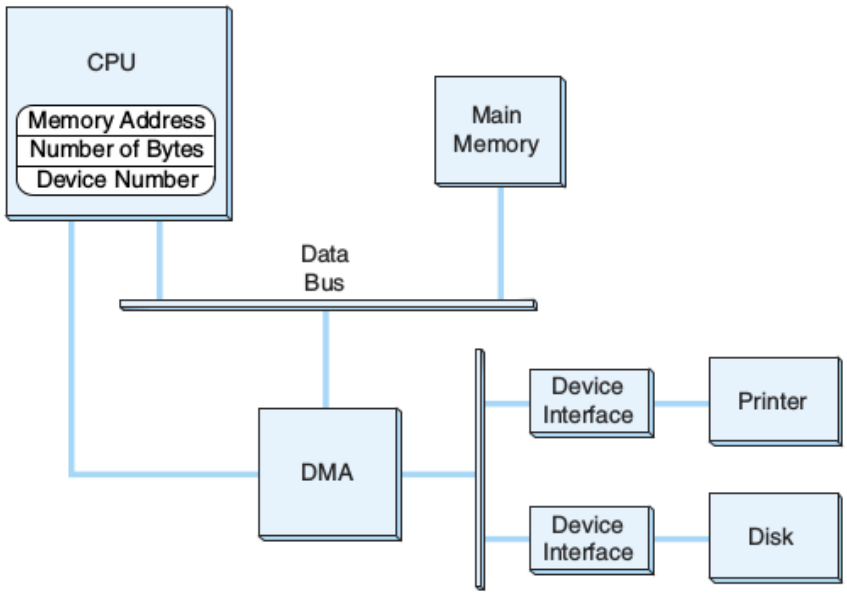


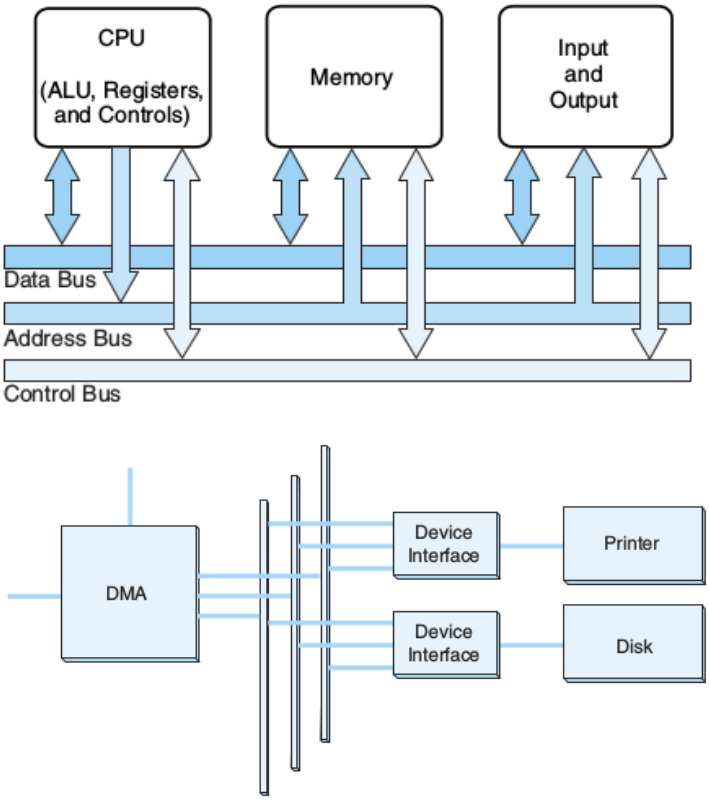
FIGURE 7.2 An Example DMA Configuration

Una vez que la transferencia es completada, el DMA manda una señal de interrupción al CPU indicando la finalización de la misma.

Se puede ver que el DMA y el CPU comparten el mismo bus de memoria, por lo que sólo uno de ellos puede realizar operaciones de lectura/escritura sobre la memoria principal. Por lo general, las operaciones de I/O tienen mayor prioridad que las del CPU debido a que una larga espera inactiva del dispositivo podría llevar a un timeout implicando la cancelación de la operación.

2.4. I/O Bus

Además del DMA, se puede resolver el problema del bus agregando buses dedicados. Es decir un bus para datos, otro para direcciones y otro de control:



El bus de control es necesario para enviar y recibir señales que indiquen el status de los dispositivos que intervienen en la comunicación, ya que se debe tomar en cuenta que estos funcionan a distintas velocidades y no es posible realizar operaciones sincrónicas (que dependan de un único reloj compartido para todos los dispositivos).

2.5. Cuadro comparativo

Método I/O	Complejidad hardware	Complejidad software	Velocidad
Polling	+	+ + +	+
Interrupciones	++	++	++
DMA	+ + +	+	+ + +

2.6. Channel I/O

Los métodos vistos hasta el momento transfieren data de a bytes o en pequeños bloques y en el caso de falla se le comunica a la CPU, la cual debe determinar los pasos a realizar a continuación. Estos funcionan bien en sistemas pequeños pero no escalan bien a grandes sistemas multiusuarios. Para estos casos se suele usar una configuración **Channel I/O**.

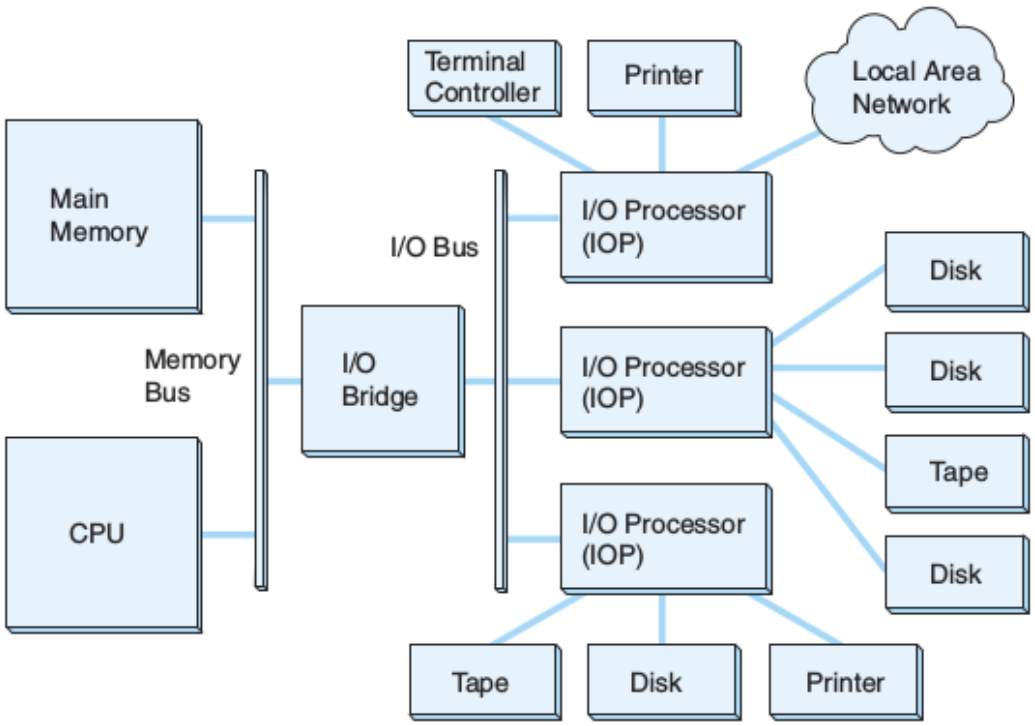


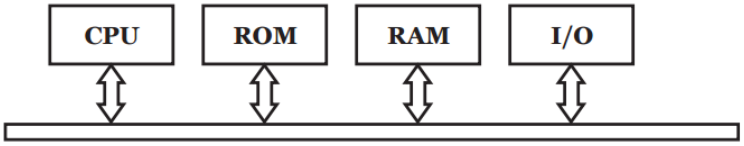
FIGURE 7.3 A Channel I/O Configuration

Cada canal de I/O es controlado por un **I/O Processor (IOP)** que ejecutan programas especificos de transferencia de datos y contienen distintos comandos de control especializados para los dispositivos conectados.

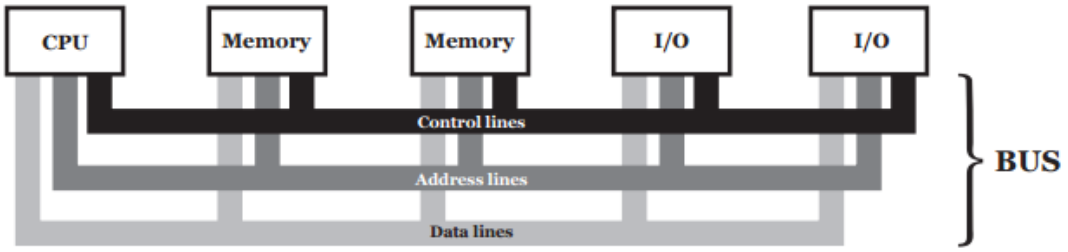
En este método, cuando una operación es requerida el IOP manda un señal de interrupción a la CPU. El CPU busca el programa que dicho IOP debe ejecutar y le envia la dirección en la que debe empezar a ejecutar. El IOP ejecuta la rutina de atención y finalmente manda una nueva interrupción a la CPU para indicar que finalizó su tarea.

3. Buses

Un bus es un camino de comunicación entre dos o mas dispositivos. Una característica clave es que se trata de un medio de transmisión compartido. Si dos o más dispositivos transmiten durante el mismo período de tiempo, sus señales puedes solaparse y distorsionarse.



Las computadoras poseen diferentes tipos de buses que proporcionan comunicación entre sus componentes dentro de la jerarquía del sistema. El bus que conecta los componentes principales del computador se denomina **Bus del sistema**. Dicho bus está compuesto por entre 50 y 100 líneas que pueden ser clasificadas en 3 tipos: líneas de datos, de direcciones y de control.

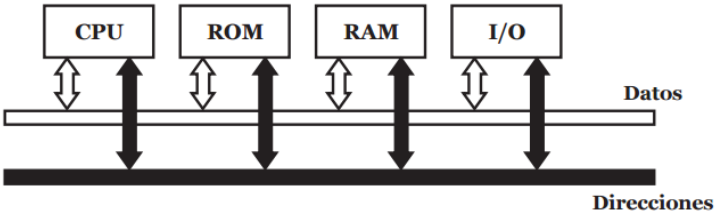


Las líneas de datos y las de direcciones sirven para transmitir datos y direcciones, respectivamente. Las líneas de control se utilizan para transmitir tanto ordenes como información de temporización entre los módulos del sistema.

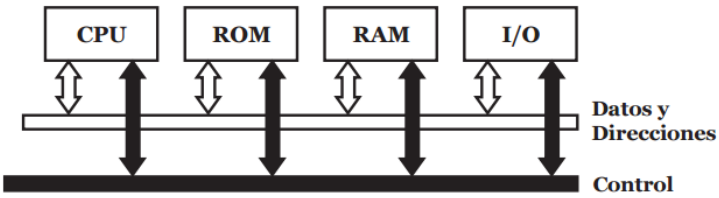
3.1. Tipos de buses

3.1.1. Tipos de línea

Pueden ser de dos tipos: Dedicadas ó Multiplexadas. Las líneas dedicadas están permanentemente asignadas a una función (Dedicación funcional) o a un subconjunto físico de componentes del computador (Dedicación física).



Las líneas multiplexadas varían su propositos a lo largo del tiempo



Aquí, la dirección se sitúa en el bus y se activa la línea de Dirección Válida. En ese momento, cada módulo dispone de un período de tiempo para copiar la dirección y determinar si es el módulo direccionado. Después, la dirección se quita del bus y las mismas conexiones se utilizan para la subsecuente transferencia de lectura o escritura de datos. Con este método se usan menos líneas por lo que se reducen espacio y costo. Sin embargo se necesita una circuitería mas compleja en cada modulo por lo que disminuye el rendimiento de la computadora.

3.1.2. Ancho de bus

La anchura del bus es la cantidad de líneas que tiene el mismo. Cuanto más ancho sea el bus de datos, es menor la cantidad de accesos a memoria ya que es mayor la cantidad de datos que se pueden transmitir simultáneamente.

La anchura del bus de direcciones afecta a la capacidad del sistema. Cuanto más ancho sea, mayor es el rango de posiciones a las que se puede hacer referencia.

3.1.3. Temporización

Los buses utilizan temporización síncrona o asíncrona para controlar la duración de los eventos. Con temporización síncrona, la presencia de un evento en el bus está determinada por un reloj. El bus incluye una línea *clk* a través de la cual transmite una secuencia en la que se alternan intervalos de igual duración de unos y ceros. Todos los eventos comienzan al principio de uno de estos ciclos (Un intervalo de unos más un intervalo de ceros).

Con la temporización asíncrona, la presencia de un evento en el bus es consecuencia de un evento previo. Este tipo de temporizadores es más difícil de implementar pero mejora el rendimiento de sistema cuando hay dispositivos con una gran diferencia de velocidades. La tempori-

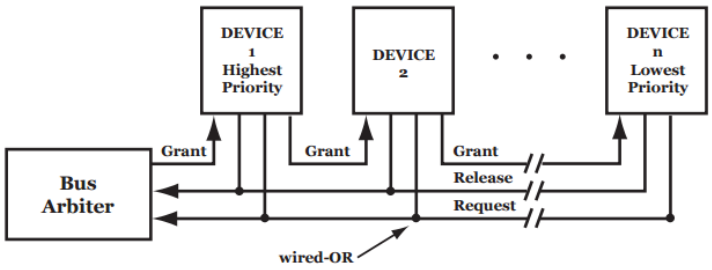
zación síncrona, es mas fácil de implementar, pero las velocidades del reloj se deben adaptar a la velocidad del dispositivo más lento del sistema.

En un bus dedicado sincrónico, las transferencias ocupan un solo ciclo de reloj. En un multiplexado, toma 2 o más pues en el primer ciclo se manda la dirección a la cual se quiere escribir o de la cual se quiere leer y en los próximos ciclos se realiza la transferencia de datos.

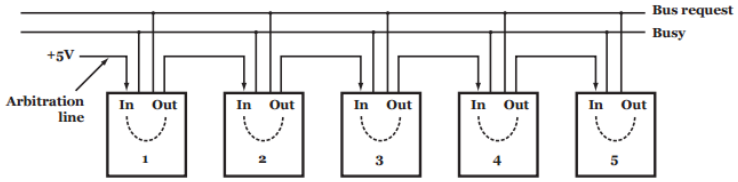
3.1.4. Arbitraje

En todos los sistemas, más de un modulo puede necesitar el control del bus. Puesto que en un instante dado solo una unidad puede usar el bus, se requiere algún método de arbitraje. Este arbitraje puede ser centralizado o distribuido.

En un esquema centralizado, un dispositivo denominado **Bus Arbiter** es el responsable de asignar tiempo al bus



En un esquema dsitribuido no existe dicho controlado. En su lugar, los módulos actuan conjuntamente para compartir el bus.



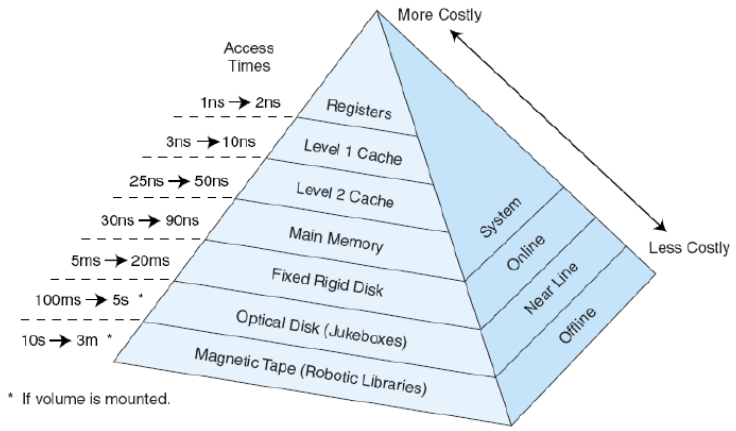
4. Memoria

Las memorias son los dispositivos utilizados para almacenar información y su rendimiento se mide según su:

- Capacidad de almacenamiento
- Tiempo de acceso
- Velocidad de transferencia
- Consumo de energía
- Tamaño físico
- Costo total y costo por unidad de almacenamiento.

Como regla general, mientras más rápida sea la memoria, más alto es el costo por unidad de almacenamiento. Por lo que un computador podría llegar a ser muy caro si solo se usa memoria rápida o muy ineficiente si solo se usa memoria barata.

Para resolver este problema, y lograr un balance costo-beneficio adecuado, los computadores actuales usan un sistema jerárquico de memorias con diferentes velocidades y capacidades de almacenamiento clasificando a las memorias segun su "distancia" al CPU. Está jerarquía se muestra abajo en forma de pirámide.



Mientras más arriba de la pirámide, la memoria es más costosa, más rápida y más cercana al CPU se encuentra, pero es usada en menor cantidad. Cuando el CPU necesita información, la busca, primero, en el nivel más alto de la piramide. Sino lo encuentra, lo busca en el siguiente nivel.

Cuando el procesador encuentra los datos que necesita, copia el bloque de memoria que contiene dicho datos en un nivel de memoria superior para tener un acceso más rápido hacia el mismo. Esto se debe a que se busca acceder a la mayor cantidad de información posible de la

manera más rápida y es muy probable que si se accedió a un dato de dicho bloque, se acceda a alguno de los datos subsecuentes.

Los distintos tipos de memorias usadas son:

- **RAM (Random Access Memory)**: Memorias de lectura/escritura usada para almacenar programas y datos necesarios para la ejecución de un programa. Éstas memorias son volátiles, es decir que, cuando se les corta el suministro de energía, pierden toda la información almacenada en ellas. Hay dos tipos de memorias RAM: Dinámicas (**DRAM**) y Estáticas (**SRAM**). Las primeras requieren una recarga de energía cada pocos milisegundos para mantener la información, las SRAM mantienen la información mientras se le suministre energía, son más rápidas pero, también, más caras que las DRAM.

  Por lo general, las caché son bancos de memorias SRAM, mientras que la memoria principal es una DRAM.

- **ROM (Read-Only Memory)**: Memorias no volátiles que no pueden ser modificadas y son usadas en pequeñas cantidades para guardar información crítica del sistema, como pueden ser, las instrucciones de inicio. Este tipo de memoria tiene sus variantes que son las **PROM**,**EPROM**,**EEPROM** y las memorias **Flash** que son memorias ROM con las ventajas de que, a través de ciertos métodos, pueden ser borradas y reusadas.

5. Caché

La memoria caché es un banco de memoria SRAM de muy alta velocidad que contiene una copia de los datos e instrucciones que están en la memoria principal. Su propósito es acelerar el acceso a estos ítem y así evitar *wait states*.

Su tamaño debe ser lo suficientemente grande como para que el procesador pueda resolver la mayor cantidad posible de búsquedas de código y datos y lo suficientemente chica para no afectar el rendimiento ni el consumo de energía del sistema.

Cuando se realiza una búsqueda en la caché hay dos posibles resultados:

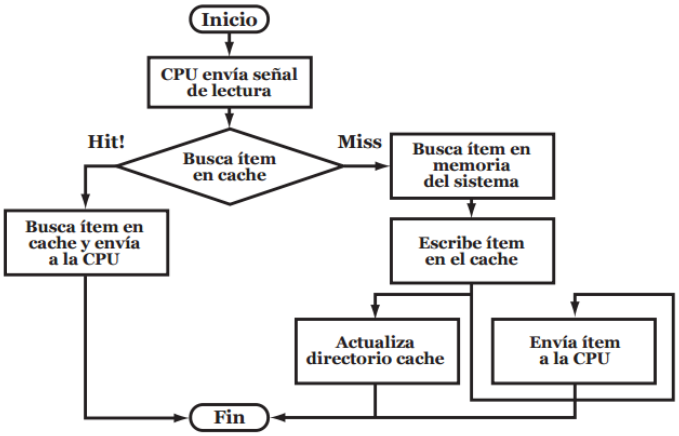
- **Hit**: El ítem se encuentra en la caché
- **Miss**: El ítem no se encontró en la caché y tiene que ser buscado en la memoria principal.

Se define **HitRate** como la cantidad de hits sobre la cantidad de accesos a la caché:

$$hitRate = \frac{\#Hits}{\#Accesos\ a\ cache}$$

Y se considera que mientras mas alto sea el HitRate, mejor performance tiene la caché.

El siguiente diagrama muestra como se realizaria una operación de lectura de datos:



Para implementar este funcionamiento se necesita de un dispositivo especial llamado controlador de caché.

Al momento de copiar datos en la caché, el controlador actúa de acuerdo a dos principios que surgen de analizar el comportamiento de los algoritmos de software usados habitualmente:

- **Principio de vecindad temporal**: Si un item es referenciado, la probabilidad de ser referenciado en el futuro inmediato es alta.
- **Principio de vecindad espacial**: Si un item es referenciado, es altamente probable que se referencien los vecinos a éste.

5.1. Estructura de la caché

Las cachés están divididas en líneas, que son el elemento mínimo de palabra de datos dentro de la caché. El tamaño de las líneas corresponde a un múltiplo de la palabra de datos de memoria. Esto se debe a que cuando se direcciona un ítem en memoria, se requerirá de los ítem que lo rodean.

Para guardar estos datos, el controlador de la caché debe saber en que lugar ubicarlo para obtener un acceso rápido. Para esto, cuando el procesador le pasa una dirección de memoria, éste agrupa sus bits en tres conjuntos:

- **Tag**: Identifica el número del bloque al que pertenece la línea en la memoria principal.
- **Línea o Set**: Indica en qué posición de la caché se ubica la línea.
- **Índice**: Indica en qué posición de la línea se encuentra el dato pedido.

Una vez conseguidos estos conjuntos, el controlador ubica en la caché dicha línea respetando ciertas politicas de decisión. Algunas de estas políticas son:

- **Mapeo Directo**: La línea se ubica en el número de línea indicado por *set*
- **Asociativa de 2 vías**: En este caso, la caché es divida en 2 y se toma a cada mitad como una caché por separado. En este caso, *set* indica a que mitad de la cache le corresponde almacenar la línea. Y la línea es almacenada en cualquier lugar disponible de esa mitad.

En el caso de que se use una caché asociativa, además, se debe elegir alguna política de desalojo de bloques, es decir, si la caché esta llena y se produce un miss, entonces habrá que remplazar alguna linea guardada por la nueva que se pide a la memoria principal. Estas políticas pueden ser:

- **LRU** (Least Recently Used): Se borra el bloque que hace más tiempo que no es usado
- **LFU** (Least Frequently Used): Se borra el bloque que menos haya sido usado.
- **FIFO** (First In First Out): Se borra el bloque que hace más tiempo que está en la cache.

5.2. Impacto del caché miss

En los computadores actuales, el pipeline es un mecanismo que permite superponer, en el tiempo, la ejecución de varias instrucciones a la vez. Si la búsqueda de una instrucción o un operando falla, entonces el procesador debe recurrir a la memoria principal. La demora en este acceso hace que el pipeline se atasque y se ocupen varios ciclos de reloj para que vuelva a tomar su ritmo de ejecución normal.

5.3. Escritura de datos

Llegado el momento, el procesador querrá modificar algún dato de la memoria, si esto sucede se debe asegurar que este cambio se vea reflejado tanto en la caché como en la memoria principal. Para lograr esto se puede usar algunos de los siguientes mecanismos:

- **Write Through**: El procesador escribe la DRAM y el controlador caché refresca la caché con el dato actualizado.
- **Write Through Buffered**: El procesador actualiza la caché y el controlador caché, luego, actualiza la DRAM mientras el procesador sigue ejecutando instrucciones y usando datos de la memoria caché
- **Copy Back**: Se marcan las líneas de la memoria caché cuando el procesador escribe en ellas. Luego, en el momento de eliminar esas líneas del caché, el controlador deberá actualizar la copia en la DRAM.

En todos los casos, si se produce un miss mientras el controlador caché esta actualizando las copias, el procesador debe esperar a que se termine este proceso.