MEMORIAS

DEFINICIÓN:

Bloque funcional cuyo objetivo es almacenar todo tipo de información (datos e instrucciones).

CARACTERÍSTICAS DESEABLES:

- Capacidad de almacenamiento.
- Mayor velocidad de acceso.
- Menos costo.
- Mayor seguridad.

JERARQUÍA:	Diccionario de términos:
+ CPU. + Registros. + Memoria caché. + Memoria principal. + Memoria secundaria.	– Tiempo de acceso. – Tiempo de ciclo. – Palabra de memoria. – Palabra de CPU. – Costo ($\frac{Costo}{Gigabytes}$). – Capacidad de memoria. – Wait State.

CLASIFICACIÓN DE MEMORIAS

Se pueden clasificar de diversas maneras.

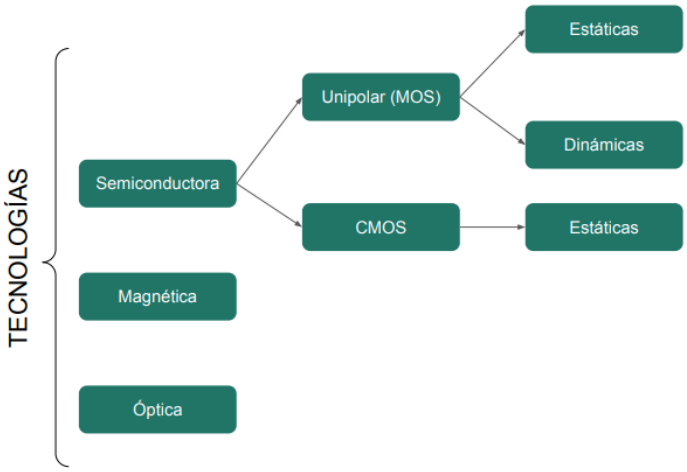
Según su función:

- ROM: Solo lectura.
- RWM: Lectura/Escritura. Se pueden leer y modificar
- RMM: Mayormente de lectura. Pueden ser escritas pero su función principal es de lectura.
- NVRW: No volátiles de lectura/escritura.

Según la vida útil de los datos que se almacenan en ella:

- Volátiles: Los datos se pierden cuando se interrumpe la alimentación eléctrica.
- No volátiles: Los datos NO se pierden cuando se interrumpe la alimentación eléctrica.

Según la tecnología de fabricación:



Según la forma de acceder a los datos:

- Accesible por dirección: Esto requiere conocer la posición física para acceder al dato. A su vez puede dividirse en:
    - Acceso aleatorio.
    - Acceso secuencial: Tiempo acceso al dato depende según la posición en la memoria.

- Accesible por contenido:
    - Se utiliza parte del dato para acceder al resto. (Memoria caché)

MEMORIAS DE ACCESO ALEATORIO:

SRAM	*DRAM*
+ *Almacenan c/bit en un Flip-Flop.* + *Requieren varios transistores para almacenar c/bit.* + *Retienen el valor almacenado mientras se mantenga la alimentación eléctrica.* + *Rápidas en lectura y escritura.*	+ Utilizan capacitores para almacenar un bit. + Un transistor por bit. + Se descargan incluso si tienen electricidad. + Tienen circuitos de refresco. + Utilizadas en memorias de alta capacidad de almacenamiento.

MEMORIAS DE SOLO LECTURA (READ ONLY MEMORY):

Las memorias de solo lectura, más conocidas como memorias ROM, solo pueden leerse, pero no grabarse por el usuario. La grabación o programación las realiza el fabricante.

Ventajas:

- ☺ En general son memorias de acceso rápido.
- ☺ No se pueden modificar.

Desventajas:

- ☹ Costo de fabricación altos.
- ☹ Tiempos de fabricación elevados.
- ☹ No se pueden cometer errores en su programación.

MEMORIAS DE SOLO LECTURA PROGRAMABLE (PROM):

Son memorias de solo lectura que pueden ser programados por el usuario. Se pueden clasificar como RMM. NO CONFUNDIR CON MEMORIAS ROM, NO SON LO MISMO.

Ejemplos de memorias PROM:

- PROM OTP.
- EPROM.

- EEPROM (E^2PROM).
- FLASH.

PROM OTP	*EPROM*
+ *Pueden ser grabadas por única vez.* + *Todos los bits en 0 antes de ser grabada.* + *Bits conectados por un fusible que al quemarse graba la memoria.* + *GRABACIÓN IRREVERSIBLE.*	+ Posee una “ventana” que al ingresar luz UV borra la memoria. + Debe removerse del circuito para ser borrada. + Dispositivo especial de grabado. + Vida útil limitada por cantidad de escrituras.

E^2PROM	*FLASH*
+ *Borrado a nivel de bit o bloque.* + *Borrado con electricidad.* + *Hardware incorporado para escribirla y borrarla.* + *Vida útil mayor a EPROM.* + *Más lentas en lectura que escritura.*	+ Lo mejor de las EPROM y las E^2PROM. + Borrado a nivel de bloque. + NOR FLASH: Mayor velocidad lectura, menor escritura. + NAND FLASH: Menor velocidad lectura, mayor escritura.

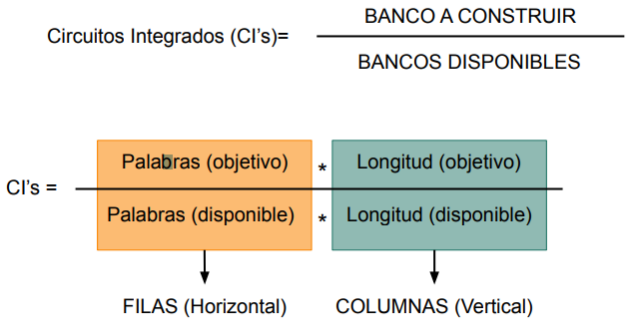
MEMORIA PRINCIPAL:

La memoria principal almacena palabras. Se la puede considerar como un vector de registros, al que accedemos indicando la dirección de cada posición dentro de ella.

¿Qué son los circuitos integrados?

- Bloques constructivos para diseño de memorias de mayor capacidad.

Fórmula:



MULTIPLEXADO CAS Y RAS:

- RAS: Utilizado para acceder a la fila de la memoria. Al activarse la señal RAS se habilita una fila específica de celdas de memoria.
- CAS: Selecciona la columna dentro de la fila PREVIAMENTE activada.
- Multiplexado: Primero se envía la dirección de la FILA (RAS), luego se envía la dirección de la COLUMNA (CAS). Permitiendo que se reduzcan la cantidad de pines en el chip de memoria. (PENSARLO COMO SI FUERA UNA MATRIZ).

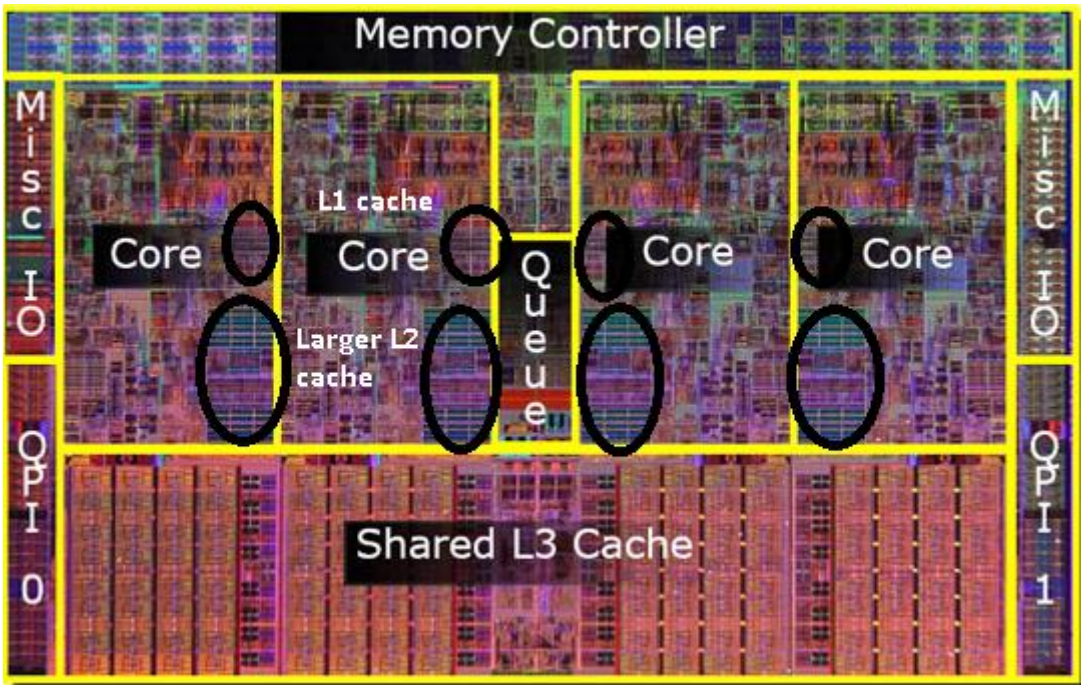
Eficiencia y Pipelining

¿Qué es el pipelining y cuáles son sus problemas y soluciones?

El pipelining es una técnica utilizada en la arquitectura de computadoras para mejorar el rendimiento del procesamiento de instrucciones. Permite que múltiples instrucciones sean superpuestas en su ejecución, de manera que diferentes instrucciones se lleven a cabo simultáneamente. El objetivo del pipelining es aumentar el rendimiento total del procesador y la eficiencia del uso de sus recursos.

El pipelining en procesadores presenta varios problemas que deben ser abordados para maximizar su efectividad. Uno de los principales problemas es las dependencias de datos, que ocurren cuando una instrucción depende del resultado de una instrucción anterior que aún no ha terminado su ejecución. Esto se puede solucionar mediante técnicas como el adelantamiento, que permite usar los resultados intermedios directamente sin esperar, la inyección de burbujas, que pausa el pipeline hasta que la dependencia se resuelva.

Otro problema significativo son las dependencias de control, que surgen con las instrucciones de salto o de rama. Estas dependencias se manejan comúnmente mediante la predicción de ramas, donde el procesador adivina la dirección de la rama y continúa la ejecución basada en esa predicción. También se puede utilizar la ejecución especulativa, que ejecuta ambas posibles rutas y descarta la incorrecta después de la evaluación.



MEMORIA CACHÉ

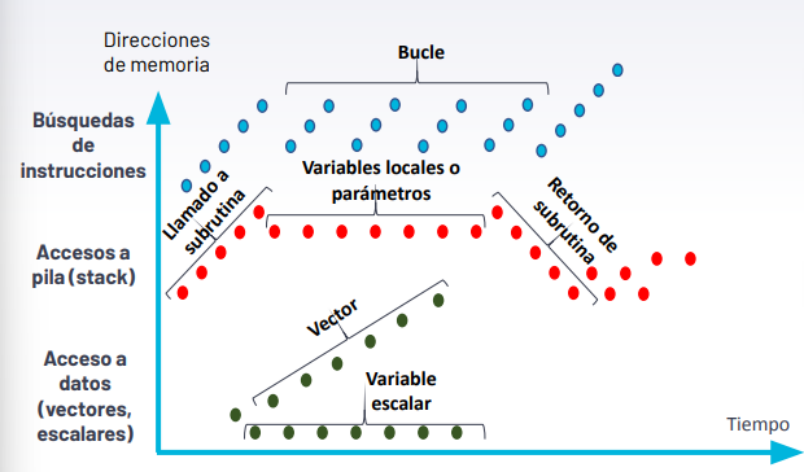
¿Qué hace una memoria caché?

La memoria caché (**MC**) es la que se encarga de agilizar las operaciones entre la CPU y la memoria principal (**MP**). Su ubicación lógicamente es entre los registros y la memoria principal.

La CPU NO sabe que existe la MC. De hecho, la CPU sigue accediendo a la MP mientras que la MC opera de manera transparente para hacer que estos procesos se hagan de manera más rápida de cómo sería si solo la CPU tuviese que comunicarse con la MP.

¿Por qué se crea la memoria caché?

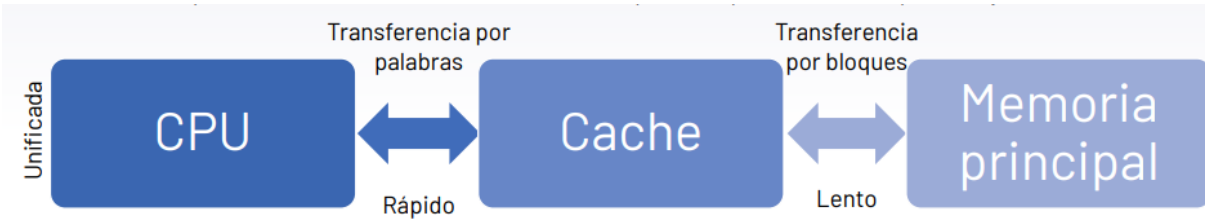
Básicamente en el año 1971 se publica en el IBM System Journal un estudio sobre cómo se repetían patrones comunes y predecibles cuando el CPU accedía a la MP.



Con esta información se elaboraron dos principios:

- **Principio de localidad temporal**: Cada vez que se accede a un dato o instrucción lo más probable es que se acceda de nuevo en un futuro cercano.
- **Principio de localidad espacial**: Cada vez que se accede a un dato o instrucción es probable que se accedan a datos o instrucciones cercanos en un futuro cercano.

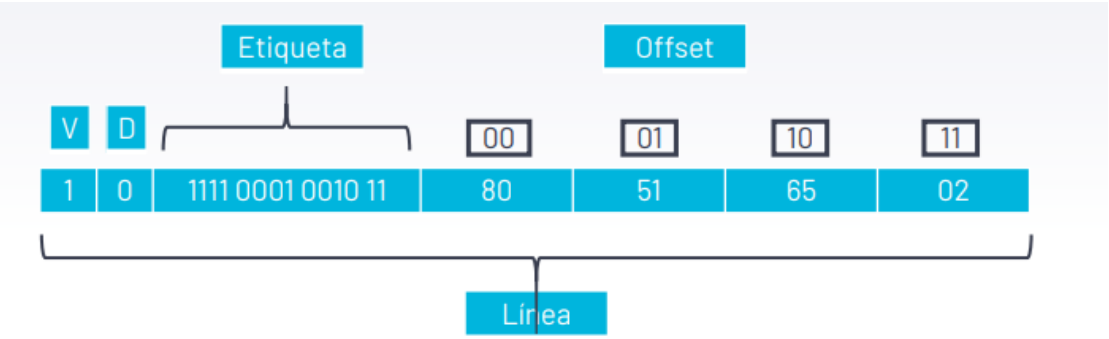
Esquema de funcionamiento de la MC:



Importante: Tener presente que si el ADMINISTRADOR DE LA MEMORIA CACHÉ (**MMU**) verifica que una dirección solicitada por la CPU está en la **MC** entonces se produce un **HIT** (éxito) y procede a enviar el dato solicitado, que se encuentra en la **MC**, a la CPU. Por el contrario, si no se encuentra lo solicitado por la CPU en la MC entonces se produce un **MISS** (falla) y procede a leer la **MP** y actualizar la **MC.**

Características de la MC:

+ Direccionamiento: POR CONTENIDO (Es decir, se extrae una parte del dato para acceder al resto).



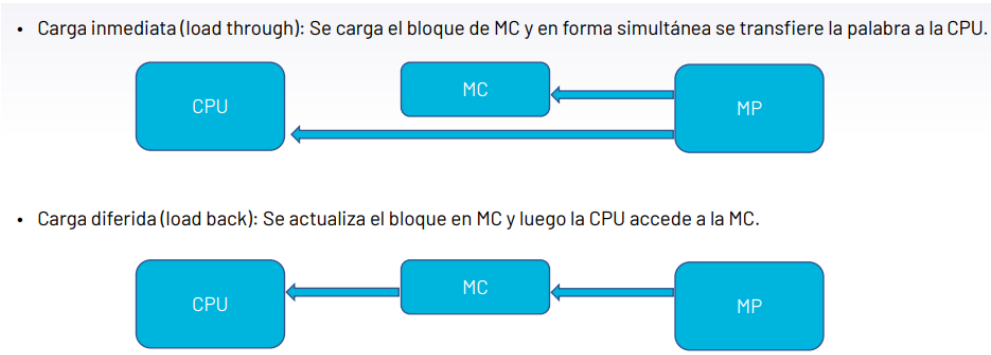
- **Etiqueta**: La etiqueta es aquello que se almacena en cada línea o bloque de la **MC**. Es la parte del dato que junto al **OFFSET** nos permite acceder a la palabra completa requerida.
- **OFFSET**: El offset no se guarda, se utiliza para seleccionar la palabra requerida.
- **V (bit de validez)**: Se usa para indicar si la línea contiene o no un bloque que pertenezca al proceso en ejecución.
- **D (dirty bit)**: Indica si la línea sufrió modificaciones y por lo tanto debe actualizar la **MP** antes de ser reemplazada. (Dirty = Modificado, Clean = No Modificado).

+ Políticas de carga y escritura:

Carga (**cuando** se produce una **FALLA**):

- Load through: Se carga el bloque de **MC** y en forma simultánea se transfiere la palabra a la **CPU**.

- Load back: Se actualiza el bloque en **MC** y luego la CPU accede a la **MC**.

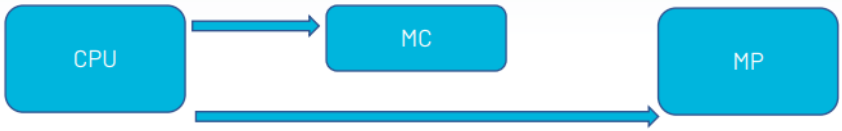


Escritura (**cuando** se produce un **ACIERTO**):

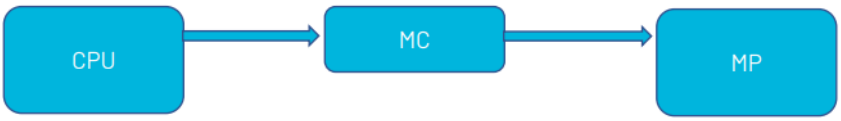
- Write through: La **CPU** escribe en la **MP** y en forma simultánea se actualiza la **MC**. (Segura pero mala performance).

- Write back: Se actualiza la **MC** y luego cuando se reemplaza la línea se actualiza **MP**. (Se utiliza un **dirty bit** para indicar que se modificó la línea).

- Escritura inmediata (write through): Se actualiza la MC y en forma simultánea se actualiza MP.
  - Segura (coherencia) y sencilla de implementar.
  - Peor performance, tiende a cuellos de botella y exceso de tráfico a memoria.
  - El procesador puede que deba demorarse (*stall*) durante la escritura. Se usan *write buffers* para reducir esto.

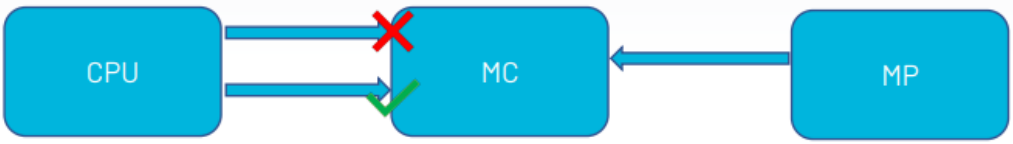


- Escritura diferida (write back): Se actualiza el bloque en MC; cuando se reemplaza la línea se actualiza MP
  - Se utiliza un bit de suciedad (**dirty bit**) para indicar que la línea fue modificada.
  - Si se producen lecturas/escrituras Mem <-> E/S o en entornos multiprocesador puede generar inconsistencias.



¿Qué sucede si **falla** la **escritura**?

- Write allocate: Se carga el bloque en **MC** y se actualiza.



- Write no allocate: Se actualiza en **MP** sin cargar en **MC**.

+ Organización:
  - Asignación asociativa (Fully asociative).

Etiqueta	Offset
$\log_2\left(\frac{MP}{Tam.\ línea}\right)$	$\log_2(Tam.\ línea)$

- Mapeo directo.
    - **Trashing**: Situación en la que se hacen referencias sucesivas a dos direcciones de memoria pertenecientes a bloques a los que les corresponde la misma línea de MC. Almacenar un vector por filas y accederlo por columnas es un típico ejemplo. Algunas implementaciones incluyen una "Victim caché" asociativa (muy pequeña) que retiene la línea recién reemplazada para reducir el impacto del thrashing.

Etiqueta	Línea	Offset
$\log_2\left(\frac{MP}{MC}\right)$	$\log_2(Cant.\ líneas)$ $\log_2\left(\frac{MC}{Tam.\ línea}\right)$	$\log_2(Tam.\ línea)$

- Asignación asociativa por conjuntos.

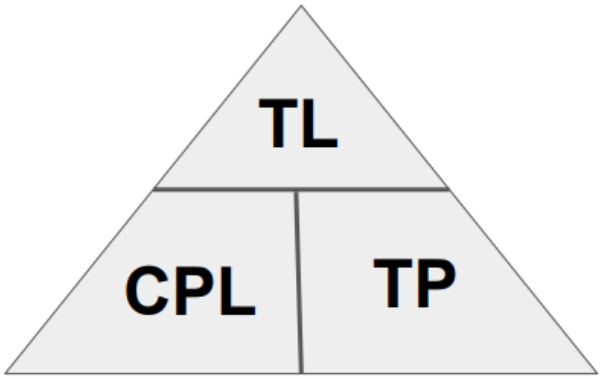
Etiqueta
$Bits\ total\ -\ Bits\ conjunto\ -\ bits\ offset$

Conjunto (Set)	Offset
$\log_2(Cant.\ conjuntos)$	$\log_2(Tam.\ línea)$

Otras fórmulas importantes:

$$Cantidad\ de\ bloques\ =\ \frac{tam.\ MP}{tam.\ línea}$$

$Longitud\ etiqueta\ =\ \log_2(cant.\ bloques) - \log_2(cant.\ líneas)$ (MAPEO DIRECTO)



Tamaño de línea [byte] = Tamaño de palabra [byte] *Cantidad de palabras por línea

Cantidad de palabras por línea = Tamaño de línea [byte] / Tamaño de palabra [byte]

Tamaño de palabra [byte] = Tamaño de línea [byte] / Cantidad de palabras por línea

+ Políticas de reemplazo (SOLO PARA ASOCIATIVA Y ASOCIATIVA POR CONJUNTOS):
    - LRU (least recently used) - Reemplaza el que no uso hace más tiempo.
    - LFU (least frecuently used) - Reemplaza el que menos veces se usó.
    - FIFO (first in first out) - Reemplaza el más viejo de todos.
    - RANDOM – Reemplaza cualquiera.
    - Pseudo-LRU. - Se asigna un bit uso por vía para cada línea. Cuando se accede a un conjunto, el bit de la línea accedida se activa. Si todos los bits están activos, se desactivan todos con excepción del último.

Debugging

Computadoras vs microcontroladores

- Computadora: una computadora de propósito general posee 3 bloques funcionales (**CPU, MEM, E/S**).
- Microcontrolador: un microcontrolador también tiene 3 bloques funcionales, pero estos se encuentran integrados en un **único chip**. Su memoria principal **NO** se puede expandir y suelen programarse en **Bare-Metal**, es decir, sin S.O.

Breakpoint HW SW

- Breakpoint HW: Utilizan registros físicos, los cuales son limitados, para almacenar direcciones de memoria donde se desea detener la ejecución del programa bajo ciertas condiciones.

- Breakpoint SW: **Los implementa el debugger**. Donde se halle un breakpoint, el debugger quita la instrucción del programa y la reemplaza por una instrucción de tipo **break** que detiene la ejecución del programa.

Cómo se trabaja en programación

- Al escribirse un programa en **ALTO NIVEL** (lenguajes como C, Java, C++, etc.) se hará uso de las bibliotecas de **ESPRESSIF**. Luego, pasará por un **COMPILADOR**, cuya función será generar un código de **BAJO NIVEL** (Assembler RISC-V) y se ensamblará usando un **ENSAMBLADOR**, el cuál generará un **CODIGO DE MAQUINA.**

Por último, el código de máquina lo vamos a transferir a la placa para que el **MICROCONTROLADOR** lo ejecute, utilizando protocolo **JTAG.**

Entrada / Salida

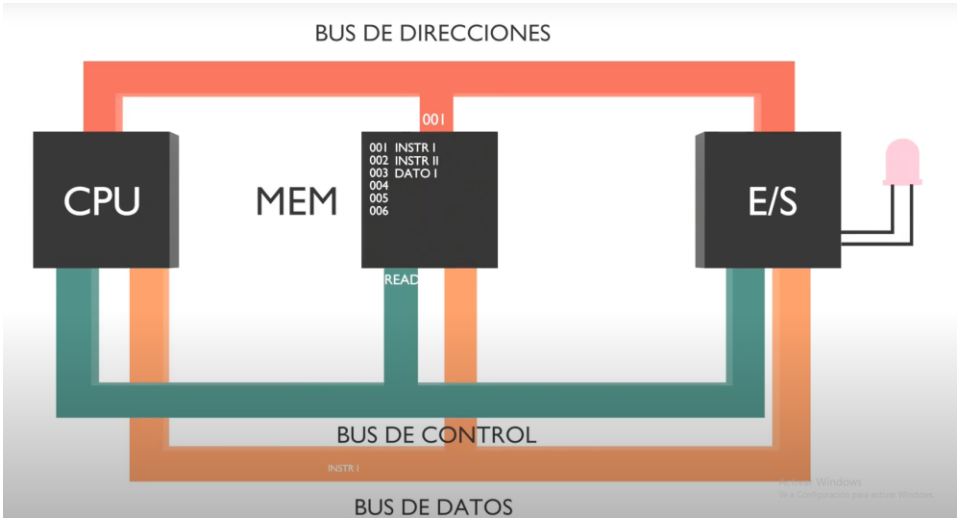
Interfaz

Es un **medio** o punto de conexión que **permite** la **comunicación entre** la **CPU** y los **dispositivos externos** o **periféricos**. Son fundamentales para el funcionamiento eficiente y efectivo entre los diferentes componentes del sistema.

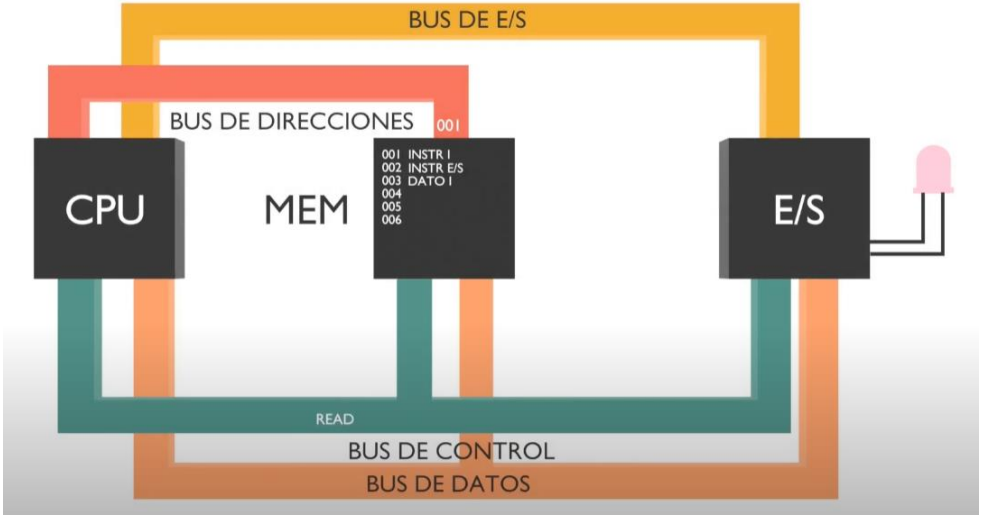
- Interfaz genérica: Controla cualquier dispositivo.
- Interfaz dedicada: Controla un dispositivo específico.

3 buses vs 4 buses

En la **arquitectura de 3 buses** se **reserva** un intervalo de **direcciones** en memoria principal (**MP**) **para** las interfaces **E/S**, permitiendo un acceso a un dispositivo externo como si estuviese en memoria. Permite instrucciones más sencillas, pero decodificación más compleja.



En cambio, en la **arquitectura de 4 buses** se crea un mapa apartado de la **MP**, conteniendo las direcciones **E/S**. Por lo tanto, requiere instrucciones específicas, pero decodificación más sencilla.



Administración de Entrada / Salida

- Polling: Técnica utilizada para mejorar la comunicación entre la CPU y los dispositivos periféricos. En esta técnica, la CPU consulta de forma regular cada dispositivo periférico para verificar si necesita atención o tiene datos listos para ser procesados.

- Interrupciones: Las interrupciones permiten que los dispositivos notifiquen a la CPU cuando necesitan atención. Permitiendo liberar a la CPU para realizar otras tareas hasta que ocurra una interrupción. Siempre termina la ejecución actual antes de atender la interrupción.

    - Tipos de interrupción:
        - IRQ (Interrupt Request): Señal asincrónica por el bus de control originada por la interfaz hacia la CPU.
        - NMI (Non-Maskable Interrupt): Mismo concepto que IRQ, pero la atención no se puede aplazar.
        - Línea individual: Cada interfaz tiene su propia línea directa al CPU, pero es costoso.
        - Línea compartida: Varias interfaces comparten una única línea. Para identificar dentro de esta línea se puede usar polling, solo que lo hace al momento que recibe una interrupción.

- Acceso Directo a Memoria (DMA): Es una técnica que permite almacenar un contenido del interfaz directo a memoria, sin tener que pasar por la CPU y un programa ASM. Se necesita una interfaz dedicada y es totalmente por hardware.

  DMA Controller (DMAC) es una interfaz que se programa que permite mover bloques entre MP y E/S. Toma el control de los buses y avisa al CPU cuando termina una interrupción.

  - Técnicas de DMA
    - Detención de la CPU: Transfiere un bloque en ráfagas. Mientras se hace una transferencia, la CPU permanece desconectada de los buses, hasta que no recibe una interrupción el DMAC controla los buses.
    - Robo de Ciclos: Aprovecha los ciclos de reloj que no utiliza la CPU para transmitir una o más palabras, aprovechando al 100% el bus.

Polling vs Interrupciones

Las interrupciones son más eficientes ya que solo actúan cuando es necesario, tiene baja latencia de respuesta, permite el manejo de prioridades, tiene una carga de trabajo baja debido a que solo actúa cuando es necesario, es eficiente en la gestión de recursos y tiene un mayor rendimiento global.

Excepciones, interrupciones y traps

+ Excepción: Condición inusual que se produce cuando termina de ejercutarse una instrucción. (Ocurre algo que no debería ocurrir). Ejemplo: Breakpoint SW, acceso no alineado. TIENEN PRIORIDAD ABSOLUTA.

  - Registros para excepciones
    - mepc: almacena el valor del Program Counter (**PC**) cuando se produjo la excepción o interrupción.

    - mcause: indica si se produjo una excepción (**bMS = 0**) o una interrupción (**bMS = 1**). El resto almacena un número de identificación.

        - mtval: almacena un valor necesario para resolver el problema.

        - mscratch: es un registro extra donde podemos guardar un valor que es importante para el trap, pero no es de uso general. Los registros de uso general NO se almacenan automáticamente.

+ Interrupción: Es un aviso de que ocurrió algo con un periférico y puede ser o no atendido. Puede ocurrir durante cualquier momento de la ejecución del programa.

    - Registros para interrupciones
        - mtvec: almacena la dirección del comienzo del vector de interrupciones. Los 2 bits menos significativos indican el modo de operacion.
            - Modo directo: toda excepción o interrupción se maneja por mtvec[0].

            - Modo vectorizado: excepciones se manejan por mtvec[0], pero las instrucciones se manejan por mtvec[mcause*4].

    Posiblemente importante: En cada posición del vector se almacena una instrucción. Cada instrucción debe saltar al trap correspondiente.

+ Trap: Es el hecho de capturar una excepción o interrupción, donde se transfiere el control de la CPU a una rutina que se va a encargar de manejar ese trap. (Se “atrapa”).

Niveles de privilegio (RISC-V)

1. Máquina.
2. Supervisor.
3. Usuario.

Tipos de interrupción

- Software: son generadas por código que escribe en el controlador programable de interrupciones (CLINT).
- Timer: generada cuando se produce un evento de timer que requiere atención.
- Externas: generadas por un dispositivo o periférico que interrumpe.

UART – SPI – I2C

UART

Es un componente que se utiliza para facilitar las comunicaciones serie de computadoras. Punto intermedio entre la CPU y microcontrolador. Transforma los bits paralelos del CPU a serie, y lo inverso para recibir. Permitiendo la comunicación con tan solo **un** cable. Se suele utilizar para comunicaciones a largas distancias, entre dispositivos separados, y debugging.

Composición de la comunicación asincrónica

Solo hay **UN** cable, no hay señal de clock, por esta razón se le dice señal asincrónica.

El **estado en reposo** esta **siempre** en **1**. Su funcionamiento se base en un **bit start** en 0, este bit indicará al receptor que se sincronice para comenzar la transmisión de datos. Luego, vienen los **bits de datos** (7 u 8 bits) junto con el **bit de paridad**, **ODD** o **EVEN (impar o par)** (opcional), y luego el bit de stop (mismo que reposo) que finaliza.

Oscilador de cristal

Un oscilador de cristal es un dispositivo electrónico que utiliza propiedades piezoeléctricas de un cristal, generalmente cuarzo, para generar una señal de oscilación con una frecuencia precisa.

Estos osciladores son esenciales en numerosos dispositivos electrónicos para proporcionar una base de tiempo estable y precisa.

SPI

Es un protocolo de comunicación seria para la transferencia de datos entre una computadora maestra (master) y computadoras esclavas (slaves). Consta de cuatro líneas unidireccionales.

→ SCLK (Serial Clock): Línea de reloj que sincroniza la transferencia, dado por el master.
→ MOSI (Master Output, Slave Input): Línea por donde el master envía los datos.

→ MISO (Master Input, Slave Output): Línea por donde el esclavo envía datos.
→ SS (Slave Select): Línea individual por dispositivo para establecer una comunicación directa entre el esclavo y maestro mienstras el resto está inactivo.

I2C

Es un protocolo de comunicación serial, permite conectar muchos dispositivos por medio de dos cables bidireccionales:

→ SDA: Línea de datos.
→ SCL: Clock se basa en el concepto de master/slave. El master sería el microcontrolador y los slaves serían los sensores, controladores, memorias, etc.

Este protrocolo utiliza un esquema de direccionamiento para identificar cada dispositivo slave en el bus. Cada dispositivo tiene una dirección única que puede ser de 7 bits o 10 bits.