ARQUITECTURA DE COMPUTADORAS

Repaso OC

1. 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.

2. Arquitectura Von Neumann.

Concepto de Arquitectura.

La arquitectura de computadoras se refiere a los atributos de un sistema que son visibles a un programador. Entre los ejemplos de atributos se encuentran el conjunto de instrucciones, mecanismos de E/S y técnicas para direccionamiento de memoria.

Modelo de Von Neumann.

Virtualmente todos los diseños de computadoras contemporáneas están basados en los conceptos desarrollados por John Von Neumann. En este caso, la Unidad Central de Procesamiento (CPU) está constituida por la Unidad de Control (UC) y la Unidad Aritmético-Lógica (ALU). Los datos y las instrucciones deben introducirse en el sistema y los resultados se proporcionarán mediante componentes de E/S. Además, se necesita la Memoria Principal para almacenar temporalmente datos e instrucciones. 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.

Arquitectura Von Newmann a) ¿Qué elementos componen una máquina con arquitectura Von Newmann? Describir la función de cada uno. [EXAMEN 12/24]

El modelo de Von Newmann está compuesto de los siguientes elementos:

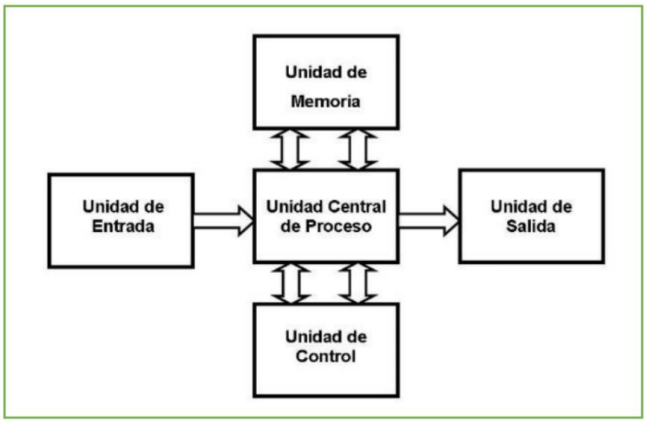
CPU (Unidad central de procesamiento): La CPU es el cerebro de la computadora. Se encarga de ejecutar las instrucciones de los programas, realizar operaciones, controlar el flujo de datos. La CPU consta de dos elementos principales:

- Unidad Aritmético-Lógica: es la que procesa lo datos.
- Unidad de Control: es la que dirige la operación.

Memoria principal: En donde se almacenan datos e instrucciones que la CPU necesita para ejecutar los programas.

Dispositivos de E/S: Permiten a la computadora interactuar con el mundo exterior. Los dispositivos de Entrada (teclado, mouse) permiten al usuario introducir datos en la computadora. Los dispositivos de Salida (monitor, impresora), muestran los resultados del procesamiento al usuario.

Bus del sistema: Es un canal de comunicación que conecta la CPU con la memoria y los dispositivos de E/S. Puede transmitir datos (Bus de Datos), instrucciones (Bus de Instrucciones) o direcciones (Bus de Direcciones).



Componentes de una Computadora

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 esta 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 que 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 que 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 esta 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 donde 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".

3. Repertorio de instrucciones.

Es el conjunto completo de instrucciones que se realizan en una CPU (código máquina, binario). Este es representado simbólicamente por un conjunto de códigos de ensamblaje de operaciones (ADD, SUB, LOAD) y de operandos (ejemplo ADD BX, AUX -> sumar contenidos del registro BX y dirección AUX, y guardar el resultado en BX).

Elementos de una Instrucción

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 donde 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 donde está el operando 1 y el operando 2 (si existiese).
- La referencia del operando resultado, nos indica donde 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.

Otras cosas a tener en cuenta

• Mientras más direcciones por instrucción, las instrucciones serán más complejas y habrá menos instrucciones por programa.

• Mientras menos direcciones por instrucción, las instrucciones serán menos complejas y tendré más instrucciones por programa. Además, la captación/ejecución de las instrucciones será más rápida.

• Registros vs Direcciones → Las operaciones entre los registros son más rápidas (pero la cantidad de registros es menor).

4. Ciclo de Instrucción. (→ Retomar en Interrupciones)

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:

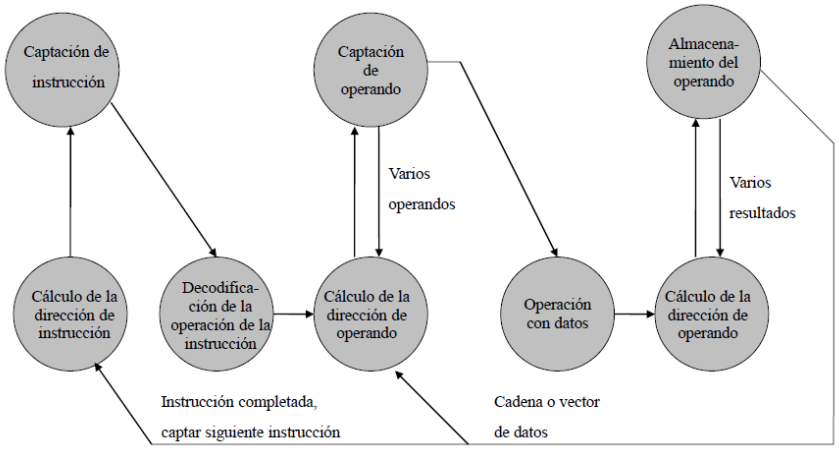
- Ciclo de Captación: la dirección de la instrucción que se debe captar se encuentra en el PC (Contador de Programa). La UC capta la instrucción desde la memoria y esta va al Registro de Instrucción IR. Se incrementa el PC (a no ser que se indique lo contrario). La UC interpreta la instrucción captada y debe llevar a cabo la acción requerida.
- Ciclo de Ejecución: acciones posibles: Transferencia de Datos (CPU-Memoria, CPU-E/S), Procesamiento de Datos (operación Aritmético-Lógica), Control (alteración en la secuencia de ejecución, como saltos).

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 (Tipos de instrucciones)

- [Almacenamiento de Datos] CPU - Memoria: los datos pueden ser transferidos desde la UCP a la memoria o desde la memoria a la CPU.
- [Transferencia de Datos] 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] Instrucciones aritmético-lógicas: la CPU puede realizar alguna operación aritmética o lógica sobre los datos.
- [Control] Instrucciones de Testeo y Flujo de Programa: 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.

Diagrama de Estados del Ciclo de Instrucción



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.

Tipos de operaciones

- Transferencia de datos: especificar operando fuente, operando resultado, tamaño de los datos a ser transferidos y MD.
- Aritméticas: operaciones básicas ADD, SUB, MUL, DIV, con números enteros con o sin signo, y punto flotante. Pueden incluirse otras operaciones (DEC, INC, NEG, ABS).
- Lógicas: operaciones que manipulan bits individualmente. Booleanas (AND, OR, XOR, NOT), operaciones de rotación (Rotate left/right).
- Conversión: operaciones para cambiar formato de datos (convertir de binario a decimal).
- E/S: pocas instrucciones, pero de acciones específicas. (IN, OUT, MOVE para movimiento de datos). Se pueden realizar a través de un controlador aparte DMA (Direct Memory Access).
- Control del Sistema.
- Control de Flujo: modifican el valor contenido en el registro PC. Saltos y llamadas a subrutinas (JZ, JNZ, JMP, CALL).

Instrucciones RISC Y CISC:

RISC: se basa en un conjunto reducido de instrucciones, simples y rápidas de ejecutar. Cada instrucción está diseñada para realizar una tarea sencilla en un solo ciclo de reloj.

CISC: utiliza un conjunto de instrucciones más complejo, con instrucciones que pueden realizar múltiples operaciones en un solo ciclo, lo que a veces puede hacer que la ejecución sea más lenta.

- Eficiencia en la ejecución:

En RISC, debido a la simplicidad de las instrucciones, cada instrucción tiende a ejecutarse en un solo ciclo de reloj. Esto generalmente resulta en un mayor rendimiento, especialmente en tareas que requieren operaciones repetitivas simples. CISC tiene instrucciones más complejas que pueden requerir varios ciclos de reloj para completarse. Sin embargo, algunas de estas instrucciones pueden realizar tareas complejas que en RISC requerirían varias instrucciones, lo que puede ser beneficioso en ciertas situaciones.

- Tamaño de las instrucciones:

Las instrucciones en un procesador RISC suelen tener un tamaño fijo, lo que simplifica el diseño del procesador y puede hacer que la decodificación de instrucciones sea más eficiente. Las instrucciones en un procesador CISC pueden tener tamaños variables, lo que puede hacer que el proceso de decodificación sea más complejo.

- Uso de Memoria:

RISC tiende a necesitar más instrucciones para realizar tareas complejas, pero cada instrucción es más eficiente en cuanto a tiempo de ejecución. Las instrucciones más complejas en CISC pueden reducir el número total de instrucciones necesarias, pero las instrucciones más largas y complejas podrían implicar un mayor uso de memoria.

5. Subrutinas

Las subrutinas han sido una innovación clave en el desarrollo de lenguajes de programación, ya que permiten estructurar el código de manera más organizada y eficiente. Una subrutina es un **bloque de código auto-contenido**, lo que significa que puede ejecutarse de manera independiente y cumplir con una función específica dentro de un programa. Esto permite a los programadores escribir segmentos de código reutilizables que pueden invocarse desde cualquier punto del programa, lo que facilita la gestión y mantenimiento del software.

Para invocar una subrutina, se utiliza la instrucción CALL (vonSim), que indica al programa que debe transferir el control de ejecución a la subrutina correspondiente.

Una de las principales ventajas de las subrutinas es que proporcionan **economía de código**, puesto que un mismo bloque de instrucciones puede ser reutilizado en diferentes partes del programa sin necesidad de reescribirlo, lo que reduce el tamaño total del código. Además, permiten **modularidad**, ya que el programa se divide en unidades pequeñas y manejables, lo que facilita su comprensión y modificación.

El uso de subrutinas también implica el **pasaje de argumentos o parámetros**. Estos parámetros permiten que la subrutina reciba información externa para realizar su tarea. Los parámetros pueden ser pasados de dos maneras:

- **Por valor**: Se pasa una copia de la variable, lo que significa que cualquier cambio realizado dentro de la subrutina no afectará el valor original.
- **Por referencia**: Se pasa la dirección de la variable, lo que permite que los cambios realizados dentro de la subrutina afecten directamente a la variable original.

a) ¿Qué métodos para pasaje de argumentos podemos utilizar en una computadora? [EXAMEN 03/24 – 08/24]

El pasaje de argumentos a Subrutinas (procedimientos o funciones) puede ser:

- Vía registros: el número de registros es la principal limitación (es importante documentar que registros se usan).
- Vía memoria: se usa un área de memoria definida (RAM), pero es difícil de estandarizar.
- Vía Pila: es el método más usado, el verdadero “pasaje de parámetros” y es independiente de memoria y registros. Hay que comprender bien el funcionamiento de la pila porque esta es usada tanto por el usuario como por el sistema (PUSH, POP / Tener en cuenta a donde apunta el puntero de la pila y cómo crece esta).

¿Qué es una pila? Describir el comportamiento con anidamiento de múltiples procedimientos/funciones utilizándola.

La pila o Stack es una estructura de datos que sigue el principio LIFO y se utiliza para gestionar la ejecución de programas y el uso de memoria de manera eficiente. Es un bloque de direcciones de memoria reservado para esta funcionalidad. Consta de dos operaciones básicas:

- PUSH → Agregar (apilar) un elemento en la parte superior de la pila.
- POP → Retirar (desapilar) el elemento superior de la pila.

Cuando anidamos múltiples procedimientos o funciones utilizando una pila, se está agregando llamadas a funciones dentro de otras funciones. Cada vez que se llama a una nueva función, se agrega un nuevo "marco de pila", que contiene variables locales, parámetros, la dirección de retorno, y cualquier otra información necesaria para la ejecución de esa función.

A medida que las funciones anidadas se completan, sus marcos de pila se retiran de la pila, y el control regresa a la función que las llamó.

Se debe:

a) Salvar el estado del BP (Puntero Base).
b) Salvar el estado de SP (Stack Pointer o Puntero de Pila).
c) Reservar espacio para datos locales (opcional).
d) Salvar valores de otros registros (opcional).
e) Acceder a parámetros.

Luego, para retornar:

- Los registros salvados deben ser descargados en orden inverso.
- Si se reservó espacio para variables locales, se debe reponer SP con el valor de BP que no cambió durante el procedimiento.
- Reponer BP.
- Volver al programa que llamó al procedimiento con RET.

ANEXO – Máquina de N-direcciones

Máquina de 4 direcciones, aquí no hay Contador de Programa, la dirección de la próxima instrucción está contenido en la instrucción actual. Tenemos direcciones explicitas 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.

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.

Luego tenemos las **máquinas de 2 direcciones**. Para reducir el tamaño de la instrucción, tendremos como información implícita la dirección en la cual se guardará el resultado, la misma va a ser la dirección del primer operando. [ADD AX, BX] 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).

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].

Entrada/Salida

1. Problemas de Entrada/Salida y Módulos de E/S

Los problemas de E/S surgen debido a la gran variedad de periféricos disponibles y los diferentes métodos de operación que utilizan. Los periféricos pueden transmitir diferentes cantidades de datos a diferentes velocidades, utilizando distintos formatos de datos y tamaños de palabra. Sin embargo, todos los dispositivos de E/S son más lentos que la CPU y la memoria RAM, lo que requiere la implementación de módulos de E/S con "inteligencia" adicional para manejar estas discrepancias.

a) ¿Cómo es la estructura de un módulo de E/S? [EXAMEN 11/23 – 11/24]

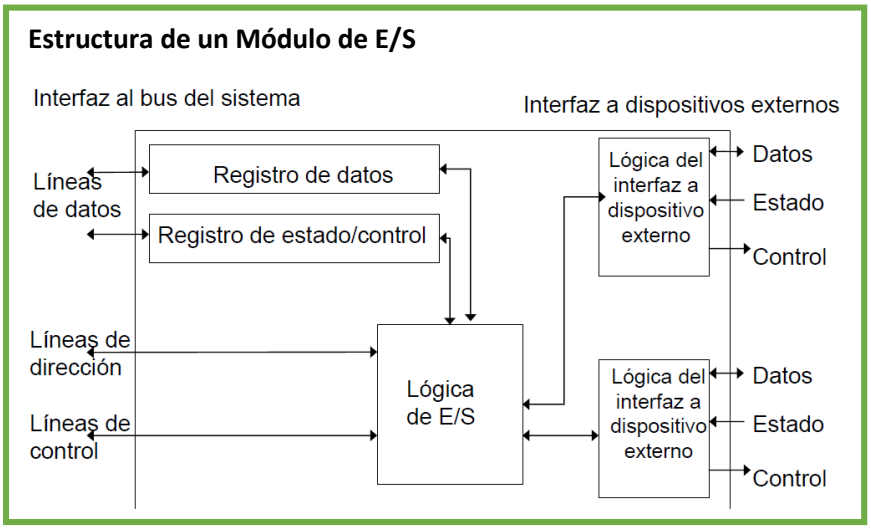
b) Esquematice y describa la estructura interna de un módulo de E/S. [EXAMEN 10/22]

2. Módulo de E/S

El módulo de E/S actúa como interfaz entre el procesador, la memoria (a través del bus) y los periféricos. Estos módulos pueden manejar uno o varios dispositivos simultáneamente, controlando la transferencia de datos y gestionando la comunicación entre los periféricos y la CPU.

Estructura de un módulo de E/S: El módulo se conecta al resto del computador a través de un conjunto de líneas (por ejemplo, líneas del bus del sistema). Los datos que se transfieren a y desde el módulo se almacenan temporalmente en uno o más registros de datos. Además, puede haber uno o más registros de estado que proporcionan información del estado presente. Un registro de estado también puede funcionar como un registro de control, para recibir la información de control del procesador. La lógica que hay en el módulo interactúa con el procesador a través de una serie de líneas de control. Éstas son las que utiliza el procesador para proporcionar órdenes al módulo de E/S. El módulo también debe ser capaz de reconocer y generar las direcciones asociadas a los dispositivos que controla. Cada módulo de E/S tiene una dirección única o, si controla más de un dispositivo externo, un conjunto único de direcciones. Por último, el módulo de E/S posee la lógica específica para la interfaz con cada uno de los dispositivos que controla.

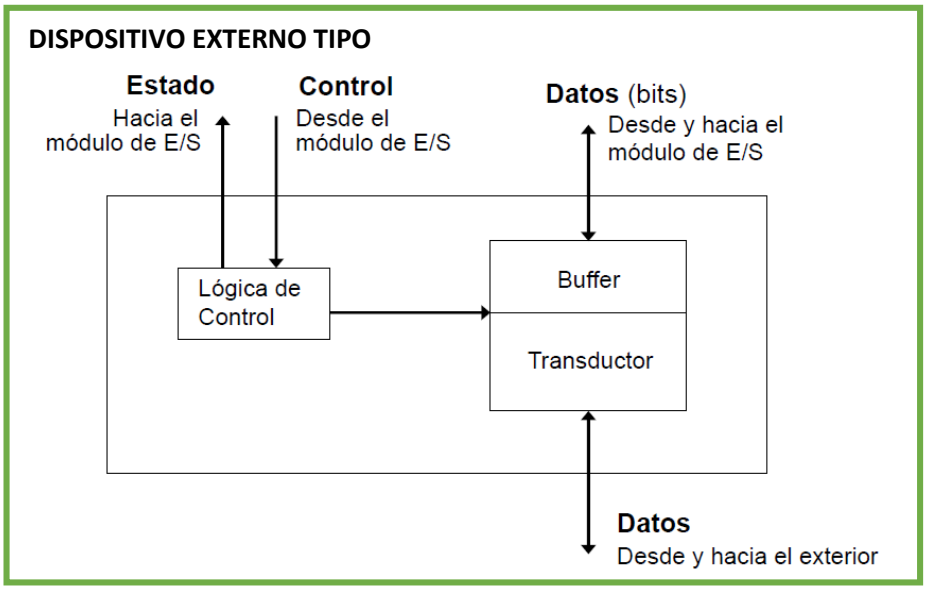
Estructura de un Módulo de E/S



3. Dispositivos Externos

Los dispositivos de E/S abarcan una amplia gama de categorías, entre las que se incluyen:

- **E/S básicos**: como monitores, pantallas, mouse y teclados.
- **Almacenamiento**: incluye discos duros, CD, DVD, entre otros.
- **Impresión**: impresoras y escáneres.
- **Comunicación con dispositivos remotos**: módems, accesos e interfaces de red.
- **Multimedia**: micrófonos y parlantes.
- **Automatización y control**: sensores, alarmas y sistemas de adquisición de datos.



4. Características de un Puerto de E/S

El puerto de E/S sirve como interfaz entre el periférico y el módulo de E/S, y se caracteriza por:

- **Señales de Control**: Determinan qué función debe realizar el módulo (por ejemplo, lectura o escritura).
- **Señales de Estado**: Indican si el periférico está listo o no (por ejemplo, "READY" o "NOT READY").
- **Control lógico**: Maneja el direccionamiento de los datos.
- **Transductor**: Convierte los datos de un formato a otro.
- **Buffer**: Se usa para la adaptación de datos (por ejemplo, en bloques de 1, 8 o 16 bits).

5. Funciones de un Módulo de E/S: los módulos de E/S cumplen varias funciones clave:

- Controlan y temporizan la operación de uno o más dispositivos.
- Interpretan las órdenes de la CPU y las transmiten a los periféricos.
- Gestionan la comunicación entre la CPU y la memoria.
- Controlan la transferencia de datos entre la CPU y los periféricos, adaptando formatos y velocidades.
- Informan a la CPU sobre el estado del periférico.
- Almacenan temporalmente los datos en buffers.
- Detectan y gestionan errores en la transferencia.

6. Capacidades de un Módulo de E/S

Los módulos de E/S están diseñados para ocultar las propiedades complejas de los dispositivos a la CPU, facilitando la programación. Además, pueden manejar uno o varios dispositivos y ofrecer diferentes niveles de control sobre ellos. En sistemas de alto rendimiento como mainframes, se emplean **canales de E/S** o **procesadores de E/S**, que se encargan de una parte significativa del procesamiento. En microcomputadoras, los módulos suelen ser **controladores de E/S**, que manejan tareas más simples.

La operación de E/S requiere varios elementos:

b) Describa los posibles modos de ubicación de los módulos de E/S. [EXAMEN 10/22]

(1) Direccionamiento: indica cómo se localizan los dispositivos de E/S en el sistema de memoria. existen dos enfoques principales:

- E/S Mapeada en Memoria: Los dispositivos de E/S y la memoria comparten un único espacio de direcciones, lo que permite acceder a los dispositivos de manera similar a la memoria. Tendremos una cierta cantidad de direcciones de memoria asignada a las operaciones de E/S, por lo tanto, no habrá necesidad de manejar instrucciones distintas.
- E/S Aislada: Los dispositivos de E/S tienen un espacio de direcciones separado de la memoria, es decir, por un lado, tendremos direcciones de memoria principal y por otro, direcciones asociadas a las E/S. Requiere líneas de E/S específicas y órdenes dedicadas para la comunicación (conjunto limitado de instrucciones).

(2) Transferencia de Información: incluye operaciones de lectura y escritura.

Describa las posibles técnicas que puede utilizar una CPU para realizar operaciones de E/S (Gestión de Transferencia).

(3) Gestión de la Transferencia: implica sincronización y control de datos en la transferencia. Existen varias técnicas para gestionar la transferencia de datos entre la CPU y los periféricos:

- **E/S programada con espera de respuesta**: Se produce un intercambio de datos entre la CPU y el módulo. La CPU tiene control directo sobre la operación y debe esperar a que el módulo termine, permaneciendo ociosa.
- **E/S con interrupciones**: La CPU no debe esperar a que la tarea finalice, puede seguir procesando, y el módulo de E/S le envía un pedido de interrupción a la CPU cuando está listo. En ese momento, el procesador ejecuta la transferencia de datos y continúa con el procesamiento previo.
- **E/S con acceso directo a memoria (DMA)**: La transferencia de datos se realiza entre un periférico y la memoria, sin intervención de la CPU, aumentando así el rendimiento de la aplicación. El controlador de DMA (DMAC) actúa como maestro de bus para transferir datos a y desde memoria a través del bus del sistema.

b) Describa el funcionamiento de la gestión de E/S programada con espera de respuesta. [EXAMEN 03/24 – 08/24 – 12/24]

E/S Programada con Espera de Respuesta: La CPU controla directamente la operación de E/S, verificando periódicamente el estado del dispositivo. En este caso, la CPU mantiene el control directo sobre la operación de E/S. La CPU envía comandos de lectura o escritura al módulo de E/S y espera que el módulo complete la operación antes de continuar. Sin embargo, este enfoque es ineficiente, ya que la CPU permanece inactiva durante la espera, lo que no es deseable.

- La CPU solicita la operación de E/S al módulo.
- El módulo de E/S realiza la operación.
- El módulo de E/S activa los bits de estado del dispositivo direccionado y espera.
- La CPU comprueba periódicamente el estado de esos bits, hasta detectar que la operación fue completada.
- En caso contrario, la CPU espera y vuelve a comprobar más tarde.

La ventaja es que la implementación es simple porque no requiere hardware adicional (como un controlador de DMA), pero es ineficiente, puesto que la CPU está constantemente comprobando el estado del dispositivo y no puede realizar otras tareas útiles mientras espera, lo que lleva a un uso ineficiente de los ciclos de CPU.

En resumen, en la gestión de E/S programada con espera de respuesta, la CPU está directamente involucrada en todas las operaciones de E/S, esperando que cada una se complete antes de continuar, lo que puede resultar en un uso ineficiente de los recursos del sistema.

Describa el funcionamiento de la gestión de E/S con interrupciones.

E/S con Interrupciones: La CPU no espera que el dispositivo termine su tarea; el dispositivo envía una interrupción a la CPU cuando está listo para continuar. En la E/S con interrupciones, la CPU no tiene que esperar la finalización de la tarea de E/S y puede seguir procesando otras tareas. El módulo de E/S notifica a la CPU mediante una interrupción cuando está listo para transferir datos, lo que permite un uso más eficiente de la CPU.

- La CPU envía una orden de lectura (READ). El módulo de E/S obtiene los datos del periférico mientras que la CPU realiza otro trabajo.
- La CPU chequea si hay pedidos de interrupciones pendientes al final de cada ciclo de instrucción (el módulo de E/S emite un pedido de interrupción a la CPU).
- La CPU detecta el pedido, guarda el contexto, interrumpe el proceso y realiza la gestión de la interrupción.
- La CPU solicita los datos (el módulo de E/S transfiere los datos).

b) Describa las características funcionales del acceso directo a memoria – DMA (etapas de transferencia). [EXAMEN 11/23 – 11/24]

E/S con Acceso Directo a Memoria (DMA): Permite que los periféricos accedan directamente a la memoria, liberando a la CPU de la tarea de transferencia de datos. El controlador de DMA [DMAC] es un dispositivo capaz de controlar una transferencia de datos entre un periférico y la memoria sin intervención de la CPU. Este debe actuar como maestro del bus durante la transferencia DMA y debe ser capaz de:

- Solicitar el uso del bus mediante las señales y la lógica de arbitraje necesarias.
- Especificar la dirección de memoria sobre la que se realiza la transferencia.
- Generar las señales de control del bus (tipo de operación L/E, señales de sincronización de la transferencia).

Etapas de una transferencia DMA: la inicialización de la transferencia de datos en un sistema que utiliza DMA (Acceso Directo a Memoria) involucra varios pasos clave en los que la CPU establece los parámetros necesarios para la transferencia entre el periférico y la memoria.

- **Inicialización de la transferencia**: primero, la CPU debe enviar al interfaz del periférico y al controlador DMA (DMAC) los detalles sobre la transferencia que se va a realizar. Esto incluye varios parámetros esenciales:

1. Inicialización del interfaz (Bus master: CPU - Bus slave: Interfaz): La CPU configura el interfaz del periférico proporcionando información como:
   - El número de bytes a transferir.
   - El tipo de transferencia que se realizará, ya sea una lectura o escritura.
   - Cualquier otra información de control adicional, como la pista, el sector o cualquier parámetro específico necesario para el periférico.
2. Inicialización del controlador DMA (Bus master: CPU - Bus slave: DMAC): En este paso, la CPU configura el controlador DMA, proporcionando parámetros adicionales que son necesarios para realizar la transferencia directa de datos entre el periférico y la memoria. Estos parámetros incluyen:
   - El número de bytes o palabras que se van a transferir.
   - El tipo de transferencia (lectura o escritura).
   - La dirección de memoria inicial para la transferencia de datos, indicando dónde en la memoria deben almacenarse los datos transferidos.
   - El número de canal a utilizar en caso de que el sistema tenga múltiples canales DMA.

   Una vez que la CPU ha enviado toda la información necesaria al interfaz del periférico y al controlador DMA, puede retomar sus tareas normales y no tiene que intervenir más en el proceso de transferencia. El controlador DMA se encargará de gestionar la transferencia de datos entre el periférico y la memoria, liberando a la CPU de esa carga y permitiéndole continuar con otros procesos sin interrupciones. Este enfoque mejora la eficiencia general del sistema, especialmente cuando se transfieren grandes cantidades de datos, ya que la CPU no está involucrada en cada paso del proceso de E/S.

- **Realización de la transferencia**: cuando el periférico está listo para realizar la transferencia se lo indica al DMAC. El DMAC pide el control del bus y se realiza la transferencia entre el periférico y la memoria [Bus Master: DMAC + periférico – Bus Slave: Memoria]. Después de la transferencia de cada palabra se actualizan los registros DMAC (número de bytes o palabras a transferir, dirección de memoria).
- **Finalización de la transferencia**: el DMAC libera el bus y devuelve el control a la CPU. Además, el DMAC suele activar una señal de interrupción para indicar a la CPU la finalización de la operación de E/S solicitada.

Problemas que puede haber: el acceso directo a memoria (DMA) es una técnica que permite a los dispositivos periféricos transferir datos directamente a la memoria sin la intervención constante de la CPU. Aunque esto mejora el rendimiento al reducir la carga de trabajo de la CPU, también puede generar ciertos problemas en el sistema, especialmente cuando el DMAC (Controlador de Acceso Directo a Memoria) utiliza intensivamente el bus de memoria. Uno de los principales problemas es que, si el bus está ocupado por el DMAC durante una transferencia, la CPU no puede acceder a la memoria para leer instrucciones o datos, lo que puede afectar negativamente el rendimiento.

Este problema se mitiga en sistemas con memoria caché, ya que la CPU generalmente lee las instrucciones desde la caché en lugar de acceder directamente a la memoria principal, lo que reduce el uso del bus. De este

modo, el DMAC puede aprovechar estos intervalos en los que la CPU no está utilizando el bus para realizar sus transferencias. En sistemas sin memoria caché, el procesador no utiliza el bus durante todas las fases de ejecución de una instrucción, por lo que el DMAC puede aprovechar las fases en las que la CPU no está utilizando el bus para realizar las transferencias.

Desarrolle cómo es el funcionamiento del DMA y los usos que de él se hacen.

El acceso directo a memoria (DMA) es una técnica que permite a los dispositivos periféricos transferir datos directamente a la memoria sin la intervención constante de la CPU.

Existen dos tipos principales de transferencias DMA: por ráfagas (burst) y por robo de ciclo (cycle-stealing). En el modo ráfaga, el DMAC solicita el control del bus a la CPU y, una vez que la CPU concede el bus, el DMAC no lo libera hasta que haya completado la transferencia del bloque completo de datos. Esto permite una transferencia más rápida, pero durante ese tiempo la CPU no puede utilizar el bus, lo que podría afectar el rendimiento del sistema.

Por otro lado, en el modo de robo de ciclo, el DMAC solicita el control del bus y, una vez que la CPU lo concede, realiza la transferencia de una sola palabra de datos antes de liberar el bus. El DMAC repetirá esta solicitud tantas veces como sea necesario para completar la transferencia del bloque de datos. Este modo no degrada tanto el rendimiento del sistema, pero la transferencia completa lleva más tiempo en comparación con el modo ráfaga. Además, este tipo de transferencia no es visto como una interrupción por parte de la CPU, por lo que no es necesario guardar el contexto del procesador. Es especialmente eficiente para transferencias de entrada/salida de múltiples palabras.

¿La coherencia de datos de un sistema jerárquico de memoria se ve afectado por el uso de DMA?

La coherencia de datos puede verse afectada por el uso de DMA, puesto que, si se realizan operaciones sobre un dato, éste se actualizará en caché y si no se vacía en memoria antes de que un dispositivo intente accederlo, se podría estar utilizando un valor erró neo, y viceversa.

Canales de Entrada/Salida: a medida que los dispositivos de entrada/salida (E/S) se vuelven más sofisticados, como es el caso de las tarjetas gráficas 3D, la evolución de los canales de E/S ha sido significativa. En los primeros sistemas, la CPU controlaba directamente los periféricos, pero luego se introdujeron módulos de E/S y controladores para facilitar la gestión de estos dispositivos. Posteriormente, se incluyó la capacidad de acceso directo a memoria (DMA), lo que permitió que los módulos de E/S transfirieran datos sin la intervención de la CPU. Los módulos de E/S modernos, además de tener su propio procesador, pueden incluir una memoria local, convirtiéndose en sistemas autónomos capaces de realizar tareas complejas.

- **Evolución:**

  (1) La CPU controla directamente los periféricos.

  (2) Se agrega un módulo de E/S o controlador.

  (3) Igual a 2 + llamado de interrupción.

  (4) El módulo de E/S provee el acceso directo a memoria (DMA).

  (5) El módulo de E/S tiene su propio procesador con su pequeño conjunto de instrucciones.

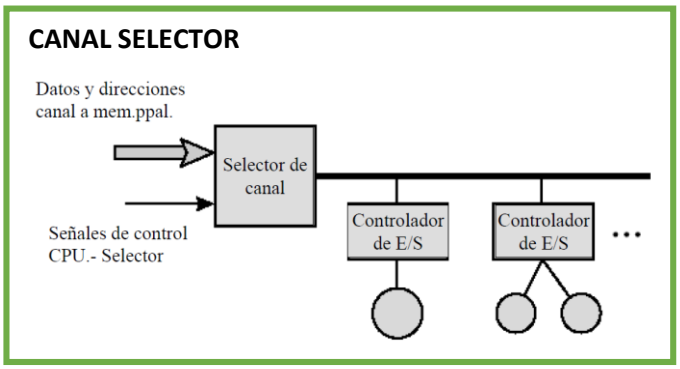
  (6) El módulo tiene además su memoria local, o sea se convierte en una computadora en si mismo.

  Los canales de E/S, que extienden el concepto de DMA, tienen la capacidad de ejecutar instrucciones de E/S de manera independiente de la CPU, lo que mejora la eficiencia. La CPU simplemente inicia la transferencia de

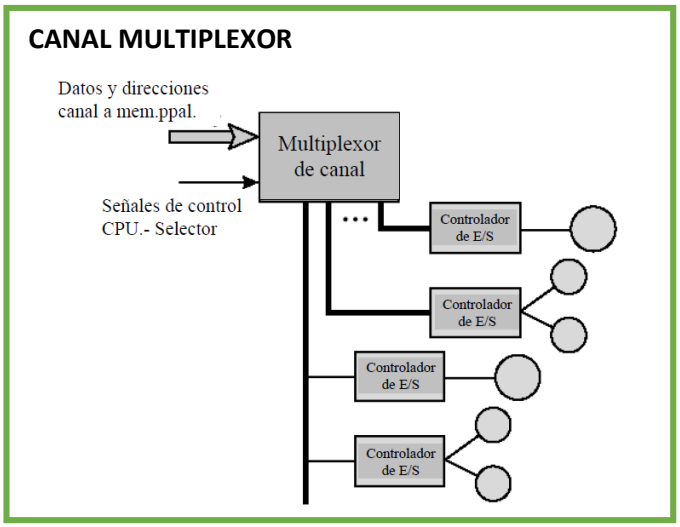
E/S y ordena la ejecución de un programa almacenado en la memoria principal, el cual especifica los dispositivos a utilizar, las áreas de memoria y las prioridades de las transferencias, así como las acciones a tomar en caso de error.

Existen diferentes **tipos de canales de E/S**.

- El canal selector controla varios dispositivos de alta velocidad, pero solo uno a la vez, entonces, se dedica a la transferencia de datos de ese dispositivo. El canal selecciona un dispositivo y realiza la transferencia de datos con este. Este dispositivo, es manejado por un controlador o módulo de E/S, por lo tanto, el canal de E/S ocupa el lugar de la CPU en el control de esos controladores.



- Por otro lado, el canal multiplexor puede manejar E/S con múltiples dispositivos simultáneamente. Dentro de los multiplexores, hay variantes como el multiplexor de bytes, que maneja caracteres, y el multiplexor de bloques, que intercala bloques de datos entre distintos dispositivos.



7. Identificación de la Interrupción

Cuando múltiples dispositivos generan interrupciones, es importante identificar cuál módulo ha causado la interrupción. Algunas técnicas incluyen:

- Líneas dedicadas para cada módulo: Limita el número de dispositivos.
- Consulta de software (polling): La CPU consulta a los módulos para determinar cuál ha generado la interrupción, pero es lento.
- Conexión en cadena (daisy chain): Los módulos están encadenados y la CPU recibe un vector que la identifica, permitiéndole acceder rápidamente a la rutina de servicio.

8. Interrupciones Múltiples y Priorización

Las interrupciones múltiples pueden requerir la asignación de prioridades, donde las interrupciones de mayor prioridad pueden interrumpir a las de menor prioridad. En algunos sistemas, un maestro de interrupciones coordina estas prioridades.

En conclusión, las técnicas de E/S con interrupciones son más eficientes que la programación directa, pero ambas requieren la intervención de la CPU y tienen limitaciones en la velocidad de transferencia de datos. Si el volumen de datos a transferir es grande, pueden surgir cuellos de botella, lo que hace que se utilicen técnicas como el DMA para mejorar la eficiencia en sistemas con grandes demandas de E/S.

Interrupciones

a) Explique el mecanismo de interrupción. [EXAMEN 07/23 – 10/23 – 07/24]

El **mecanismo de interrupción** es un mecanismo mediante el cual se puede interrumpir el procesamiento normal de la CPU. Este mecanismo permite que el flujo secuencial de ejecución de instrucciones de un programa se detenga momentáneamente para atender una situación especial o urgente. Las interrupciones pueden ser de origen interno o externo a la CPU.

Las interrupciones **internas** provienen de eventos que ocurren dentro de la propia CPU o del sistema, como errores de ejecución, excepciones o señales de hardware que requieren atención inmediata. Por otro lado, las interrupciones **externas** son generadas por dispositivos o componentes fuera de la CPU, como periféricos de entrada/salida, temporizadores o incluso solicitudes del sistema operativo para que se realicen determinadas tareas.

¿Cuál es la función de un controlador de instrucciones?

La función de un controlador de interrupciones es gestionar y coordinar las interrupciones. Se encarga de determinar qué dispositivo o evento generó la interrupción, priorizarlas en caso de múltiples interrupciones simultáneas y luego redirigir el flujo de ejecución del procesador hacia un programa de manejo de interrupciones específico.

¿Por qué puede darse una interrupción?

- Por resultado de una ejecución de una instrucción (por ejemplo, desbordamiento aritmético u “overflow”, división por 0).
- Por un temporizador interno del procesador (permite al SO realizar ciertas funciones de manera regular).
- Por una operación de E/S, para indicar la finalización normal de una operación.
- Por un fallo de hardware (por ejemplo, error de paridad en la memoria, perdida de energía).

Tabla: clases de Interrupciones (Stallings)

Programa	Generadas por alguna condición que se produce como resultado de la ejecución de una instrucción, tal como desbordamiento aritmético (*overflow*), división por cero, intento de ejecutar una instrucción máquina inexistente e intento de acceder fuera del espacio de memoria permitido para el usuario.
Temporización	Generadas por un temporizador interno al procesador. Esto permite al sistema operativo realizar ciertas funciones de manera regular.
E/S	Generadas por un controlador de E/S, para indicar la finalización sin problemas de una operación o para avisar de ciertas condiciones de error.
Fallo de hardware	Generadas por un fallo tal como la falta de potencia de alimentación o un error de paridad en la memoria.

Cuando ocurre una interrupción, en casi todos los casos esto implica transferir el control a otro programa (**programa gestor**) que:

- Salve el estado del procesador.
- Corrija o responda a la causa que ocasionó la interrupción.
- Restaure el estado original del procesador.
- Retorne a la ejecución normal del programa interrumpido.

Una vez que el Programa Gestor ha completado su tarea, la CPU puede reanudar la ejecución del programa interrumpido, retomando el punto en el que se dejó. Este proceso permite que el sistema sea más flexible y capaz de responder rápidamente a eventos críticos sin perder información importante.

Jerarquía de Interrupciones: si hay múltiples fuentes que pueden solicitar una interrupción, se establecen cuales son las más importantes. Se consideran:

- **No enmascarables:** las que no pueden ignorarse, que son las que indican eventos peligrosos o de alta prioridad.
- **Enmascarables:** pueden ser ignoradas (con instrucciones podemos inhibir la posible solicitud).

Describa el mecanismo de interrupción.

El mecanismo de interrupción funciona de la siguiente manera:

- Generar la interrupción: Un dispositivo o componente del sistema detecta un evento que requiere atención, por lo que genera una señal de interrupción.
- Interrupción del procesador: Cuando se genera una interrupción, el procesador suspende temporalmente la ejecución del programa actual y cambia su enfoque hacia el controlador de interrupciones.
- Guarda el estado: Antes de manejar la interrupción, el procesador guarda el estado actual del programa en ejecución. Esto incluye la posición del programa en la que se detuvo, los registros de la CPU y otros datos relevantes.
- Rutina de manejo de interrupciones: El controlador de interrupciones determina la naturaleza de la interrupción y consulta una tabla de vectores de interrupción que contiene direcciones de memoria de rutinas específicas de manejo de interrupciones. Estas rutinas son porciones de código que se encargan de gestionar el evento específico asociado con la interrupción.
- Ejecución de la rutina de manejo: El procesador comienza a ejecutar la rutina de manejo de interrupciones correspondiente. Esta rutina se encarga de procesar el evento que generó la interrupción. Una vez que la rutina ha sido ejecutada, el control es devuelto al punto en el programa donde se detuvo inicialmente.
- Restauración del estado: Después de que se maneja la interrupción, el procesador restaura el estado previo guardado, incluyendo los registros y la posición del programa, permitiendo que el programa original se reanude como si la interrupción nunca hubiera ocurrido.
- Continuación de la ejecución: Con el estado restaurado, el programa original continúa su ejecución desde el punto en el que se interrumpió.

El mecanismo de interrupción permite a los sistemas manejar eventos en tiempo real, realizar tareas de gestión de dispositivos y responder eficazmente a situaciones imprevistas sin detener por completo la operación del sistema.

¿Cuáles son las diferencias en la terminación de una subrutina y un gestor de interrupción? [EXAMEN 03/24 – 08/24]

La subrutina regresa al programa principal utilizando la instrucción de retorno RET. En cambio, un gestor de interrupción lo hace después de manejar dicha interrupción, o con la instrucción de retorno IRET. La instrucción IRET es similar a una instrucción RET, por utilizar la pila, pero recupera una copia del registro de estado y la dirección de retorno.

Interrupciones por HARDWARE: estas son las generadas por dispositivos de E/S y se las considera las “verdaderas” interrupciones. En este caso, el sistema de cómputo tiene que manejar estos eventos externos no/ planeados o “asincrónicos”. Los mismos no están relacionados con el proceso en ejecución dado en ese momento y son conocidos como **interrupt request**.

Dentro de las interrupciones por Hardware, están los **Traps** o **Excepciones**, que son interrupciones por Hardware creadas por el procesador en respuesta a ciertos eventos como:

- Condiciones excepcionales (overflow en ALU de punto flotante).
- Falla de programa (al tratar de ejecutar una instrucción no definida).
- Fallas de Hardware (error de paridad de memoria).
- Accesos no alineados o a zonas de memoria protegidos.

b) ¿Cuándo, por qué, para qué y cómo utiliza una de las denominadas interrupciones por Software? [EXAMEN 10/22]

Interrupciones por SOFTWARE: muchos procesadores tienen instrucciones explícitas que afectan al procesador de la misma manera que las interrupciones por hardware. Estas son las interrupciones por Software: son un mecanismo que permite a los procesadores realizar operaciones de manera similar a las interrupciones por Hardware, pero a través de instrucciones explícitas dentro del propio código. Estas interrupciones se utilizan comúnmente para hacer llamadas a funciones del SO, lo que facilita la interacción del software con el sistema y el acceso a servicios que el SO ofrece.

Una de las ventajas de las interrupciones por software es que permiten que las subrutinas del SO se ubiquen en cualquier lugar de la memoria, ya que no es necesario conocer de antemano la dirección de la rutina en tiempo de ejecución. Esto facilita la flexibilidad del sistema, ya que las direcciones de las funciones no están fijas y pueden cambiar sin que el código del programa tenga que modificarse.

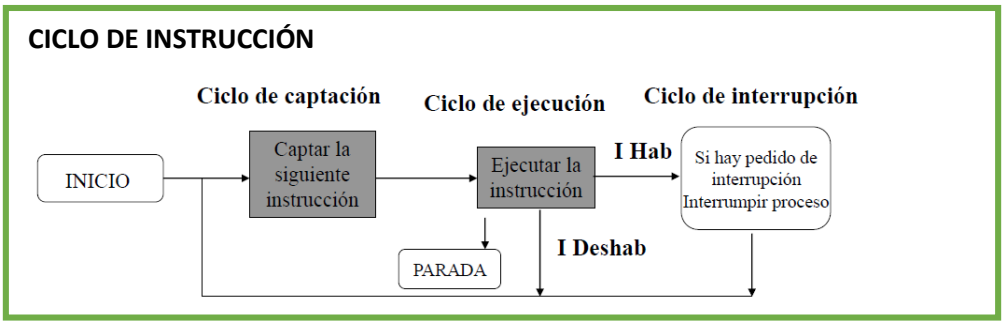
Sin embargo, en algunos sistemas, no se permite realizar una llamada directa a una dirección de una función del SO, puesto que estas direcciones están ubicadas en áreas de memoria reservadas o protegidas. Esto implica que, sin el uso de interrupciones por software, el proceso de invocar funciones del sistema operativo se vuelve más complejo.

¿Qué pasaría si no tuviera las Interrupciones por Software?

En ausencia de interrupciones por software, una de las alternativas sería escribir manualmente todas las funciones que se necesitan dentro del código, lo cual sería muy laborioso y propenso a errores. Otra opción sería, al cargar un programa, "examinar" todas las llamadas a funciones del BIOS y del sistema operativo, y reemplazar las direcciones de estas funciones dentro del código en tiempo de ejecución. Este proceso de sustitución de direcciones sería costoso y menos eficiente que simplemente usar interrupciones por software, que permiten un acceso más directo y flexible a las funciones del sistema operativo sin necesidad de manipular direcciones en el código.

En resumen, las interrupciones por software simplifican la interacción entre el software y el SO, proporcionando una forma eficiente y flexible de hacer llamadas al sistema sin necesidad de conocer las direcciones de las funciones en tiempo de ejecución.

Ciclo de Instrucción: cuenta con 3 pasos: **(→ Paso 1 y 2 en RepasoOC)**



- Ciclo de captación: capta la siguiente instrucción.
- Ciclo de ejecución: ejecuta la instrucción.
- Ciclo de interrupción: si hay pedido de interrupción, se interrumpe el proceso. ¿Cómo funciona este paso?
    - Se comprueba si se ha solicitado alguna interrupción, indicada por alguna señal o flag de pedido de interrupción.
    - Si NO hay señal, se capta la siguiente instrucción.
    - Si hay algún pedido de interrupción pendiente:
        - Se suspende la ejecución del programa en curso.
        - Se guarda su contexto (próxima instrucción a ejecutar y el estado del procesador).
        - Se carga el PC con la dirección de comienzo de una rutina de gestión de interrupción y se inhiben otras interrupciones.
        - Finalizada la rutina de gestión, el procesador retoma la ejecución del programa del usuario en el punto de interrupción.

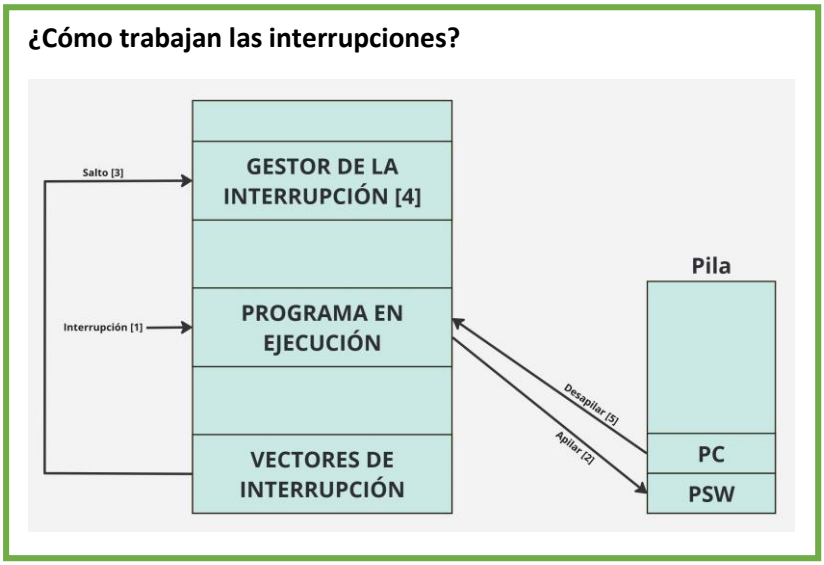
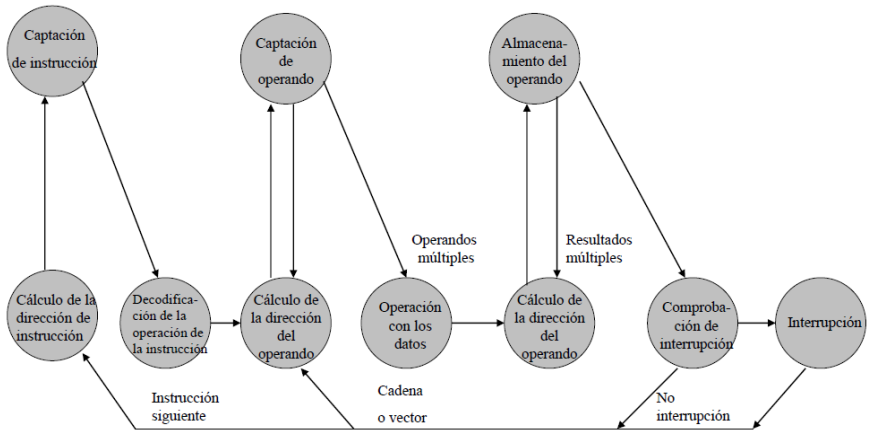


Diagrama de estados de un Ciclo de Instrucción con Interrupciones



b) Describa el tratamiento a realizar cuando hay múltiples fuentes de interrupciones. [EXAMEN 07/23 – 07/24]

Interrupciones múltiples: estas ocurren cuando varios eventos de interrupción se generan de manera simultánea o en rápida sucesión, y el procesador necesita gestionar esas interrupciones de forma ordenada. Para manejar este tipo de situaciones, se utilizan mecanismos que permiten controlar cuándo y cómo el procesador atiende cada interrupción, asegurando que no se pierdan eventos importantes y que el sistema opere de manera eficiente. Se pueden seguir dos alternativas para tratar las interrupciones múltiples:

- La primera es desactivar las interrupciones mientras se está procesando una interrupción. Una interrupción inhabilitada simplemente significa que el procesador puede y debe ignorar la señal de petición de interrupción. Si se produce una interrupción en ese momento, generalmente se mantiene pendiente y será examinada por el procesador una vez este haya activado las interrupciones. El inconveniente del enfoque anterior es que no tiene en cuenta la prioridad relativa ni las solicitudes con un tiempo crítico.
- La otra opción consiste en definir prioridades para las interrupciones y permitir que una interrupción de prioridad más alta pueda interrumpir a un gestor de interrupción de prioridad menor.

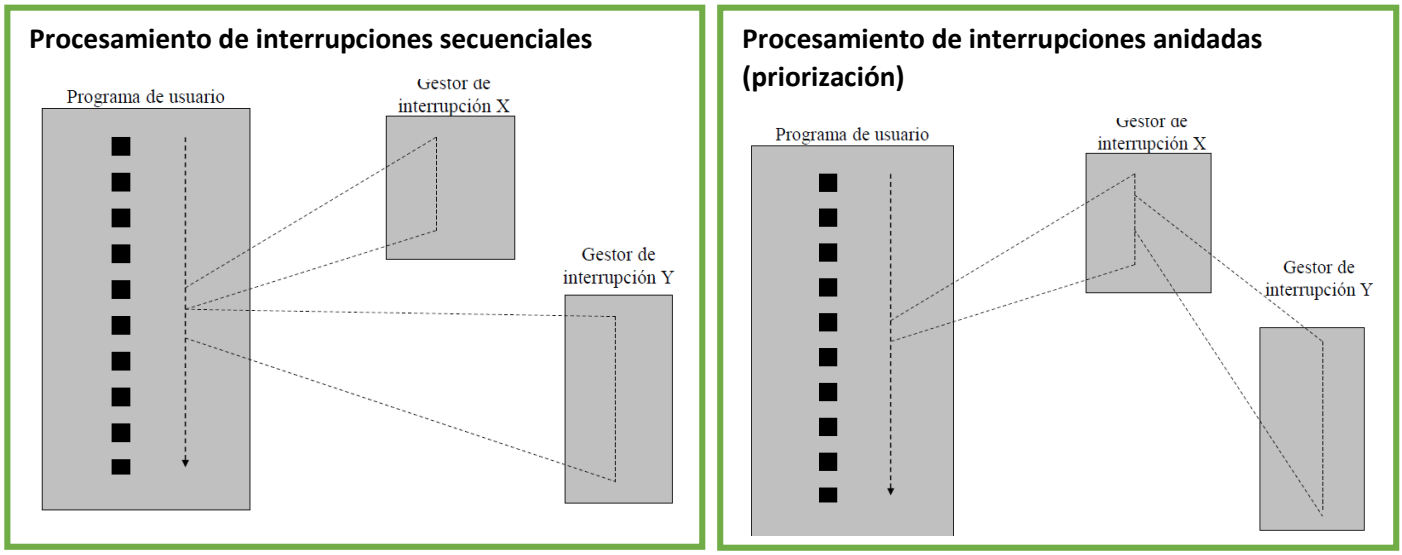
En algunos sistemas, el procesador puede **inhabilitar** las interrupciones momentáneamente. Esto significa que, si se recibe una solicitud de interrupción mientras las interrupciones están inhabilitadas, el procesador ignorará esa señal en ese momento. La interrupción no se pierde, sino que se mantiene pendiente y será revisada una vez que las interrupciones sean habilitadas nuevamente. Este enfoque asegura que las interrupciones críticas no interfieran con tareas importantes que el procesador esté realizando en ese momento.

Cuando ocurre una interrupción, el procesador puede inhabilitar temporalmente las interrupciones, gestionar la interrupción recibida, y luego volver a habilitarlas para seguir atendiendo las solicitudes pendientes. De este modo, las interrupciones se manejan en un **orden secuencial estricto**, donde cada interrupción es atendida en el orden en que fue recibida, sin que ninguna se pierda.

Además, en sistemas con **prioridades de interrupción**, se puede definir qué interrupciones tienen mayor urgencia. Una interrupción con prioridad más alta puede interrumpir el manejo de una interrupción de menor prioridad. Una vez que se gestiona la interrupción de mayor prioridad, el procesador regresa al manejo de las interrupciones previas, de menor prioridad. Esto permite que las interrupciones más críticas se atiendan primero, mientras que las de menor prioridad se gestionan una vez que se ha resuelto la situación urgente.

Finalmente, una vez que todas las rutinas de gestión de interrupciones han sido ejecutadas y todas las solicitudes han sido atendidas, el procesador retoma la ejecución normal del programa del usuario. Este sistema de manejo de interrupciones permite que el procesador responda de manera eficiente a múltiples eventos simultáneos, garantizando que no se pierdan interrupciones importantes y que el flujo de ejecución del programa no se vea interrumpido innecesariamente.

Un módulo de E/S (por ejemplo un controlador de disco) puede intercambiar datos directamente con el procesador. Igual que el procesador puede iniciar una lectura o escritura en memoria, especificando la dirección de una posición concreta de la misma, el procesador también puede leer o escribir datos de o en un módulo de E/S. En este último caso, el procesador identifica un dispositivo específico controlado por un módulo de E/S determinado. Por consiguiente, se puede producir una secuencia de instrucciones, con instrucciones de E/S en lugar de las instrucciones de referencia a memoria. En algunos casos, es deseable permitir que los intercambios de E/S se produzcan directamente con la memoria. En ese caso, el procesador cede a un módulo de E/S la autoridad para leer de o escribir en memoria, para que así la transferencia E/S-memoria pueda producirse sin la intervención del procesador. Durante esas transferencias, el módulo de E/S proporciona a la memoria las órdenes de lectura o escritura, liberando al procesador de cualquier responsabilidad en el intercambio. Esta operación se conoce con el nombre de acceso directo a memoria [DMA].



El **reconocimiento de interrupciones** es un proceso clave para que la CPU pueda gestionar las solicitudes de interrupción generadas por los dispositivos periféricos. Existen varias técnicas para manejar estas interrupciones, que se diferencian en la cantidad de entradas físicas que utilizan y la manera en que se identifican los dispositivos que generan la interrupción.

- **Interrupciones multinivel**: cada dispositivo que puede generar una interrupción tiene su propia entrada física conectada a la CPU. Esta técnica es sencilla de implementar, pero resulta costosa en términos de hardware, ya que requiere múltiples líneas de interrupción físicas para cada dispositivo, lo que puede hacer que el sistema sea más complejo y caro.
- **Línea de interrupción única**: todos los dispositivos están conectados a una única entrada física de pedido de interrupción. Cuando un dispositivo necesita interrumpir, la CPU debe preguntar a cada dispositivo si ha

generado una solicitud de interrupción. Este proceso se conoce como **técnica de Polling** o **encuesta**, y aunque reduce la cantidad de líneas de hardware necesarias, puede ser menos eficiente, ya que la CPU debe comprobar de manera repetitiva cada dispositivo, lo que consume tiempo de procesamiento.

a) ¿A qué método de atención lo conocemos como de "interrupciones vectorizadas"? [EXAMEN 10/22]

- **Interrupciones vectorizadas**: esta es una forma más avanzada de manejar las interrupciones. En este enfoque, el dispositivo que desea interrumpir no solo envía la señal de interrupción a la CPU, sino que también coloca un identificador, conocido como **vector**, en el bus de datos. Este identificador permite a la CPU saber qué dispositivo ha generado la interrupción sin necesidad de preguntar a todos los dispositivos. El vector puede ser colocado directamente por el periférico o, en sistemas más sofisticados, por un **controlador de interrupciones**, que se encarga de gestionar todo el proceso de interrupciones, incluyendo la asignación de los vectores. Las interrupciones vectorizadas permiten una gestión más eficiente de las interrupciones, ya que el procesador puede identificar y manejar rápidamente qué dispositivo requiere atención, reduciendo el tiempo y los recursos necesarios para gestionar múltiples solicitudes de interrupción.

b) Describa como se realiza el reconocimiento de interrupciones vectorizadas mediante el PIC. [EXAMEN 10/23]

En un escenario donde el procesador tiene una **única entrada de pedido de interrupciones**, pero existen varios dispositivos (productores de interrupciones) que pueden generar solicitudes de interrupción, surge la necesidad de gestionar de manera eficiente estas solicitudes para evitar que se pierdan o se gestionen de forma desordenada.

Una forma de solucionar este problema es mediante el uso de un **Dispositivo Controlador Programable de Interrupciones (PIC)**. El PIC actúa como un intermediario entre los dispositivos periféricos que generan interrupciones y la CPU. Su función principal es recibir todas las solicitudes de interrupción de los dispositivos y luego decidir cuál debe ser atendida en función de un algoritmo de priorización. De este modo, el PIC puede gestionar las interrupciones de múltiples dispositivos y enviarlas de manera ordenada a la CPU, a través de una única línea de interrupción.

El conexionado del PIC con el procesador es relativamente sencillo: todos los dispositivos periféricos se conectan al PIC, que a su vez se conecta a la única entrada de interrupción del procesador. El PIC, al recibir una interrupción de cualquiera de los dispositivos conectados, la pone en cola y, dependiendo de la prioridad de las interrupciones y del algoritmo configurado, selecciona cuál interrumpirá al procesador en ese momento.

Cuando el PIC recibe una solicitud de interrupción, puede realizar varias tareas, como:

1. Priorizar las interrupciones: El PIC puede configurar una jerarquía de prioridades, donde las interrupciones de mayor prioridad (por ejemplo, de dispositivos críticos) son atendidas primero.

2. Generar un vector de interrupción: Cuando el PIC decide que debe interrumpir al procesador, puede generar un vector de interrupción que le indica al procesador qué dispositivo ha generado la interrupción y qué rutina debe ejecutar para manejarla.

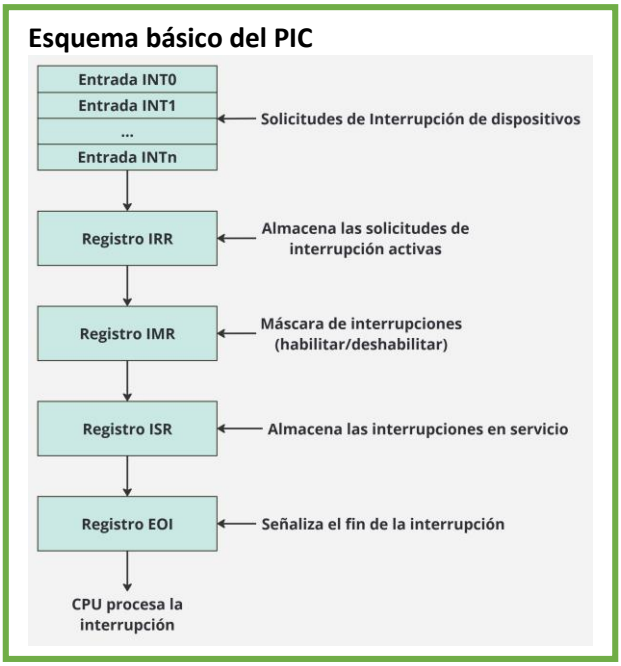
3. Controlar el flujo de interrupciones: El PIC puede manejar el flujo de interrupciones para asegurarse de que las interrupciones más importantes sean atendidas sin que se pierdan o se gestionen de forma incorrecta.

En resumen, el PIC permite que un procesador con una única entrada de interrupción pueda manejar múltiples solicitudes de interrupción de varios dispositivos de forma eficiente, priorizando las interrupciones y enviando la correcta a la CPU en el momento adecuado. Esto mejora el rendimiento y la organización del sistema al evitar la necesidad de que la CPU tenga que manejar directamente cada dispositivo.

a) Esquematice y describa la estructura interna de un controlador programable de interrupciones. [EXAMEN 03/24 - 08/24 – 12/24]

Un **Controlador Programable de Interrupciones (PIC)** es un componente de hardware que gestiona las solicitudes de interrupción de múltiples dispositivos y las transmite de manera organizada a la CPU. Su función es permitir que un sistema con múltiples fuentes de interrupción utilice una sola línea de interrupción para comunicar la solicitud a la CPU, sin perder ninguna petición importante.

Estructura Interna de un PIC:



- **Entradas de Interrupción [INT0 – INTn]**: son las líneas a las que se conectan los dispositivos periféricos. Cada dispositivo genera una solicitud de interrupción, y el PIC decide cuándo y cuál interrumpir, dependiendo de la prioridad y la disponibilidad de la CPU.
- **Registro de Solicitudes de Interrupción [IRR - Interrupt Request Register]**: este registro almacena las solicitudes de interrupción que han sido generadas por los dispositivos. Cada bit de este registro indica si un dispositivo ha solicitado una interrupción.
- **Registro de Servicio de Interrupción [ISR]**: cuando la CPU comienza a atender una interrupción, el PIC transfiere la solicitud al ISR. Este registro indica qué interrupciones están siendo procesadas actualmente.
- **Máscara de Interrupciones [IMR]**: este registro permite habilitar o deshabilitar las interrupciones individuales. (1 = deshabilita / 0 = habilita).
- **Fin de la Interrupción [EOI]**: una vez que la CPU ha procesado una interrupción, el PIC recibe una señal EOI para indicar que la interrupción ha finalizado y puede liberar la línea de interrupción para otras solicitudes.
- **Control de Prioridad**: los PICs suelen permitir establecer una prioridad de interrupciones para decidir qué interrupción se atenderá primero. Esto es útil cuando varias interrupciones se solicitan al mismo tiempo.
- **Dirección de Interrupción**: el PIC asigna una dirección de interrupción o vector de interrupción a cada dispositivo. Este vector le indica a la CPU cuál es el procedimiento o la rutina que debe ejecutar cuando la interrupción se ha procesado.

¿Cuáles son los pasos para el funcionamiento de un PIC?

El funcionamiento de un PIC implica varios pasos:

- Detección de interrupciones: El PIC monitorea constantemente las líneas de interrupción de los dispositivos periféricos y otros eventos en busca de cambios en su estado. Cuando se detecta una interrupción, se genera una solicitud de interrupción hacia el procesador.
- Priorización: Si hay múltiples interrupciones pendientes, el PIC selecciona la de mayor prioridad según su configuración. Esto se basa en las prioridades asignadas a las líneas de interrupción.

- Enmascaramiento: Antes de notificar al procesador sobre una interrupción, verifica si la línea de interrupción está habilitada o enmascarada. Si la línea está enmascarada, la interrupción se ignora.
- Vectorización: Una vez que se determina la interrupción que debe atenderse, el PIC envía un vector de interrupción a la CPU. Este vector contiene información sobre la fuente de la interrupción y apunta a la dirección de memoria de la rutina de manejo de interrupciones correspondiente.
- Procesamiento de interrupción: La CPU recibe el vector de interrupción y ejecuta la rutina de manejo de interrupciones asociada. Esta rutina es la encargada de gestionar la interrupción, guardar el estado necesario y realizar cualquier acción requerida.
- Fin de interrupción (EOI): después de manejar la interrupción, la CPU envía una señal de Fin de Interrupción al PIC para indicar que ha completado el manejo de la interrupción actual. Esto permite que el PIC se prepare para la siguiente interrupción si la hubiera.

Describa las limitaciones existentes al paralelismo a nivel de instrucciones.

El paralelismo a nivel de instrucciones es una técnica que permite que múltiples instrucciones se ejecuten en paralelo dentro de un procesador. Algunas de sus limitaciones son:

- Dependencias de Datos: Las dependencias de datos ocurren cuando una instrucción depende de los resultados de una o más instrucciones anteriores. Esto puede limitar la cantidad de instrucciones que pueden ejecutarse en paralelo, ya que algunas de ellas deben esperar a que se completen las instrucciones precedentes.
- Dependencias de Control: Las dependencias de control se refieren a las instrucciones condicionales, como las instrucciones de salto condicional. El procesador no puede prever de antemano cuál será la próxima instrucción a ejecutar hasta que se resuelva la condición.
- Limitaciones de Ancho de Banda de Memoria: El acceso a memoria principal es una operación que puede restringir el paralelismo a nivel de instrucción. Si el procesador necesita acceder repetidamente a la memoria, es posible que deba esperar a que se complete en las operaciones de lectura/escritura antes de poder avanzar con otras instrucciones.
- Tamaño de Ventana de Ejecución: La cantidad máxima de instrucciones que se pueden emitir y ejecutar en paralelo puede ser limitada por cuestiones de diseño de hardware o por la arquitectura del procesador.
- Saltos y Llamadas a Subrutinas: Las instrucciones de salto y las llamadas a subrutinas cambian el flujo de ejecución y pueden requerir la cancelación de instrucciones en vuelo que ya no son válidas debido al salto. Esto puede llevar a penalizaciones de tiempo significativas.

Interrupciones del MSX88

El MSX88 maneja las interrupciones mediante un conjunto de características tanto en hardware como en software que permiten la gestión eficiente de las solicitudes de interrupción.

Interrupciones en Hardware: el sistema utiliza una única línea de interrupción INT para recibir las solicitudes de interrupción de los dispositivos. Cuando se genera una solicitud, la CPU responde con una señal de reconocimiento INTA. Además, el MSX88 dispone de una línea NMI (Non-Maskable Interrupt), que es de mayor prioridad y no puede ser desactivada, lo que la hace ideal para situaciones críticas, como errores de hardware.

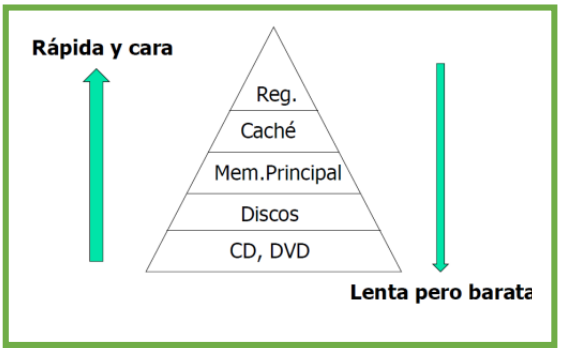
- **Atención de Interrupciones**: el proceso de atención a las interrupciones es por salto indirecto. Esto significa que el control de la CPU se transfiere a una dirección específica para manejar la interrupción cuando esta se produce. En el caso de las interrupciones gestionadas por software, se utiliza la instrucción INT XX, que permite que el software solicite una interrupción. Para retornar desde el gestor de interrupción, se debe usar la instrucción IRET.
- **Tabla de Vectores de Interrupción**: el sistema utiliza una tabla de vectores de interrupción que actúa como un enlace entre el tipo de interrupción (con un número que va del 0 al 255) y el procedimiento encargado de atenderla. Cada entrada de la tabla es una doble palabra (4 bytes) que contiene la dirección del procedimiento que brinda el servicio de interrupción. Dentro de los vectores preasignados, algunos de los más comunes son:
  - Tipo 0: Se utiliza para finalizar la ejecución de un programa.
  - Tipo 3: Actúa como un punto de parada para depuración o seguimiento del programa.
  - Tipo 6: Se utiliza para la lectura de entrada estándar, con la necesidad del registro BX.
  - Tipo 7: Se usa para la escritura de salida estándar, y requiere los registros BX y AL.
- **Controlador de Interrupciones (PIC)**: el PIC (Programmable Interrupt Controller) es un componente crucial para gestionar las interrupciones. Dentro de sus registros internos se encuentran varias estructuras:
  - EOI (End of Interrupt): Comando utilizado para finalizar la interrupción, al que se le asigna el valor 20H.
  - IMR (Interrupt Mask Register): Registro que permite enmascarar interrupciones con un valor de 1.
  - IRR (Interrupt Request Register): Indica si hay una solicitud de interrupción, con bit en 1.
  - ISR (Interrupt Service Register): Muestra qué interrupciones están en servicio, con un bit en 1.
  - INT0...INT7: Cada una de estas líneas de interrupción está asociada a su propio vector, permitiendo la gestión adecuada de hasta 8 interrupciones.
- **Conexionado y Direccionamiento**: los registros internos del PIC se ubican en la dirección de memoria 20H. Estos registros son accesibles mediante operaciones de lectura y escritura en el espacio de E/S utilizando las instrucciones IN y OUT.
- **Interrupciones Hardware Asignadas**: el sistema asigna interrupciones a dispositivos específicos:
  - INT0: Tecla F10.
  - INT1: Timer.
  - INT2: Handshake (intercambio de señales entre dispositivos).
  - INT3: DMA (Acceso Directo a Memoria).
  - INT4 a INT7: No se utilizan en este sistema.

En resumen, el MSX88 organiza y maneja las interrupciones de manera eficiente utilizando una combinación de hardware y software, permitiendo una rápida respuesta ante solicitudes de interrupción sin comprometer el rendimiento general del sistema.

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.

Describa un sistema de memoria basado en jerarquías.



Jerarquía de memoria: es la forma en la que están organizados estos distintos niveles de memoria, que son ubicados en distintos lugares físicos y fabricados con tecnologías diferentes que se gestionan de manera independiente. 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. La jerarquía de memorias cumple lo siguiente:

- La velocidad del sistema deberá ser, aproximadamente, la del nivel más rápido al costo del nivel más barato.
- A medida que nos alejamos de la CPU, cada nivel inferior es más grande, más lento y más barato que el nivel previo (o superior) en la jerarquía.
- Debe haber correspondencia de direcciones en los distintos niveles.

Propiedades a cumplir:

- **Inclusión**: implica que cualquier dato que se almacene en un nivel superior de la jerarquía de memoria (por ejemplo, en la memoria caché de nivel 1 o L1) debe estar también disponible en los niveles inferiores (como las cachés de nivel 2 y 3, o la memoria principal). Esto asegura que no haya datos inconsistentes entre los diferentes niveles de la jerarquía. Si un dato se encuentra en una caché de alto nivel, debe haber una copia idéntica en los niveles inferiores para que el sistema no tenga que recuperar esos datos desde la memoria principal o secundaria en caso de un fallo de caché.
- **Coherencia**: asegura que, si un dato se actualiza en un nivel de la jerarquía (como en la memoria caché), dicha actualización se refleje en los niveles inferiores de manera que todas las copias del dato tengan el mismo valor. Si un sistema tiene varias copias de los mismos datos en diferentes niveles de memoria, es crucial que cuando se modifique un dato en un nivel, los otros niveles se actualicen en consecuencia. De lo contrario, el procesador podría estar trabajando con versiones desactualizadas de los datos, lo que llevaría a errores o comportamientos inesperados en el sistema.

La **inclusión** asegura que los datos en niveles superiores estén también en los inferiores, mientras que la **coherencia** garantiza que esos datos sean consistentes en todos los niveles. Juntas, estas propiedades permiten que el sistema maneje de manera eficiente las cachés y la memoria, mejorando el rendimiento sin comprometer la consistencia de los datos.

a) ¿Por qué funciona un sistema de memoria basado en jerarquía? [EXAMEN 10/23]

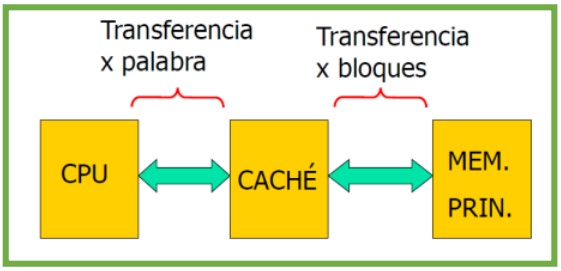
¿Por qué funciona la jerarquía de memorias? → PRINPICIOS DE LOCALIDAD DE REFERENCIAS

(1) 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:

- 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.
- Otra razón en la que se basa dicho principio es que generalmente cuando programamos, declaramos variables relacionadas muy juntas.
- 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).

(2) 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:

- 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.
- 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.
- 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.



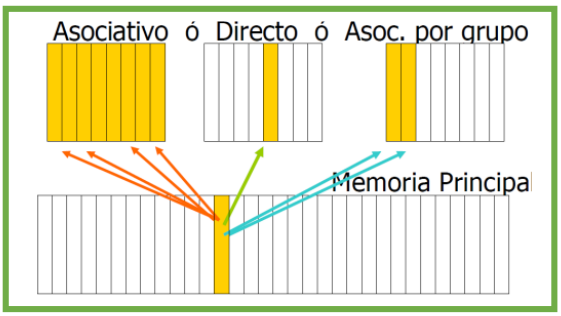
Memoria caché: es una pequeña cantidad de memoria extremadamente rápida que se ubica entre la **memoria principal** y la **CPU**. Su propósito es mejorar la velocidad de acceso a los datos más utilizados, al almacenar temporalmente los datos y las instrucciones que la CPU usa con mayor frecuencia. Esta memoria caché puede estar integrada directamente en el chip de la CPU o en módulos de memoria cercanos a ella.

Relación entre la memoria caché y la memoria principal: la memoria caché se utiliza para aliviar la carga de trabajo de la memoria principal, que es más lenta. Cuando la CPU solicita datos de una dirección de memoria, la caché verifica si esos datos ya están almacenados allí. Si el dato requerido se encuentra en la caché, se puede acceder a él de manera más rápida que si tuviera que recuperarse de la memoria principal.

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 (esto se debe al principio de localidad espacial y temporal).

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 [2n]).



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 copiemos en la Cache para tenerlo disponible para futuros accesos.

Funcionamiento de la memoria caché:

1. Solicitud de datos: La CPU solicita un dato de una dirección específica en la memoria.
2. Verificación en la caché: La caché revisa si el dato solicitado está almacenado en ella.
   - Si el dato está presente: Se dice que **hay un acierto** (hit). En este caso, la caché proporciona rápidamente el dato solicitado a la CPU.
   - Si el dato no está presente: Se produce un **fallo** (miss). En este caso, la memoria caché debe leer el bloque que contiene el dato solicitado desde la memoria principal y copiarlo a la caché. Luego, la caché entrega el dato a la CPU.

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 acierto 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 trabajo la Cache. 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, tendré 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.

La caché utiliza **etiquetas** para identificar qué bloque de la memoria principal está almacenado en cada una de sus líneas. Esto permite hacer un seguimiento eficiente de qué datos están presentes en la caché y facilita su recuperación rápida cuando la CPU los solicita.

El tiempo necesario para manejar un **fallo de caché** depende de la latencia y el ancho de banda de la memoria principal. La **latencia** es el tiempo necesario para acceder a la memoria principal, mientras que el **ancho de banda** se refiere a la

cantidad de información que puede transferirse desde la memoria a la CPU por unidad de tiempo. Cuando ocurre un fallo de caché, el procesador se detiene momentáneamente hasta que el dato solicitado esté disponible en la caché.

En resumen, la memoria caché optimiza el rendimiento del sistema almacenando temporalmente los datos más solicitados, lo que permite que la CPU acceda a ellos más rápidamente que si tuviera que recurrir a la memoria principal. La gestión de aciertos y fallos en la caché es fundamental para mantener la eficiencia del sistema, minimizando los tiempos de espera en el acceso a datos.

Principales características y técnicas relacionadas con el diseño y las prestaciones de la memoria caché:

- **Prestaciones de la jerarquía de memoria**: el tiempo de acceso medio a memoria (T_{acceso}) depende tanto de los aciertos como de los fallos de la caché. El tiempo total de acceso cumple:

$$\mathbf{T_{acceso} = T_{aciertos} + T_{fallos}}$$

Donde el Tiempo de aciertos es el tiempo necesario para acceder a un dato que ya está presente en la caché; y el Tiempo de fallos en memoria es el tiempo necesario cuando ocurre un fallo en la caché. Este se calcula como el producto entre la tasa de fallos (TF) y la penalización por fallo (PF), que es el tiempo adicional que toma obtener el dato desde la memoria principal.

$$\mathbf{T_{fallos} = TF \times PF}$$

Por lo tanto la fórmula para el Tiempo de acceso nos queda:

$$\mathbf{T_{acceso} = T_{aciertos} + TF \times PF}$$

El rendimiento de la jerarquía de memoria puede mejorarse de tres formas:

1. Reduciendo el **tiempo de acierto** ($T_{aciertos}$).
2. Reduciendo la **tasa de fallos** (TF).
3. Reduciendo la **penalización por fallo** (PF).

a) ¿Cuáles son los elementos a tener en cuenta para el diseño de una memoria caché? Analícelos brevemente. [EXAMEN 10/23 – 03/24 – 08/24 – 12/24]

Diseño de la memoria caché: el diseño de la memoria caché implica varios elementos que determinan su efectividad, estos son:

- **Organización (tamaño y cantidad)**: Determinar el tamaño de la caché es fundamental, ya que una mayor capacidad de almacenamiento mejora la probabilidad de acierto, pero también incrementa los costos y el tiempo de acceso. La organización de la caché involucra la relación entre su tamaño y los niveles de la jerarquía de memoria. A mayor tamaño de caché, mayor capacidad para almacenar datos y, por lo tanto, menor tasa de fallos, pero también mayor costo y tiempos de acceso.
- **Política de ubicación**: La forma en que los bloques de la memoria principal se asignan a la caché depende del tipo de mapeo que se utilice. Los principales tipos de mapeo son:
    - Correspondencia directa (Mapeo directo): un bloque sólo puede estar almacenado en un lugar de la caché.

    - Correspondencia totalmente asociativa (Mapeo asociativo): un bloque puede almacenarse en cualquier lugar de la caché.
    - Correspondencia asociativa por conjuntos (Mapeo asociativo por grupo): un bloque puede almacenarse en un conjunto restringido de lugares de la caché.
- **Política de reemplazo**: Define el criterio de qué dato se debe reemplazar cuando la caché está llena y es necesario almacenar un nuevo bloque de datos. Depende del tipo de correspondencia que se use:
    - En correspondencia directa: no hay elección, sólo hay una posible línea para cada bloque. Si se necesita almacenar un nuevo bloque, se reemplaza el bloque existente en esa línea.
    - En correspondencia asociativa: los algoritmos de reemplazo deben ser implementados en hardware para garantizar una operación rápida. Algunos algoritmos comunes son:
        - LRU: Se reemplaza el bloque que no se ha usado en más tiempo.
        - FIFO (First In First Out): Se reemplaza el bloque que ha estado más tiempo en la caché.
        - LFU (Least Frequently Used): Se reemplaza el bloque que se ha usado menos frecuentemente.
        - Aleatoria: Se reemplaza una línea de la caché seleccionada de manera aleatoria.
- **Política de escritura**: determina cómo se manejan las escrituras en la caché (si se actualizan de inmediato en la memoria principal o no). Es crucial para evitar la inconsistencia entre la memoria caché y la memoria principal. Esto es especialmente importante en sistemas con múltiples unidades de procesamiento (CPU), módulos de E/S, y arquitecturas de procesamiento paralelo, donde varias cachés individuales pueden estar involucradas. Además, se debe garantizar que los datos en la caché y en la memoria principal se mantengan sincronizados en todo momento. Cuando la CPU realiza una escritura sobre una línea de caché, el bloque de memoria correspondiente en la memoria principal debe ser actualizado en algún momento, para evitar la inconsistencia de memoria. Las políticas de escritura se dividen en dos categorías principales: aquellas que determinan cómo se gestionan las escrituras en los **aciertos** de la caché (Write-through y Write-back) y aquellas que se aplican cuando ocurre un **fallo de caché** (Write allocate y No-Write allocate).

Analice ventajas y desventajas de poseer varios niveles de caché.

Hay dos aspectos relacionados con este tema, el número de niveles de caché, y el uso de caché unificada frente al de cachés separadas.

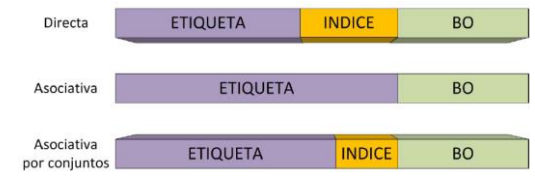
- Cachés multinivel: En general, el uso de un segundo nivel de caché mejora las prestaciones. No obstante, el uso de cachés multinivel complica todos los aspectos de diseño de la caché.
- Caché unificada frente a cachés separadas: Muchos diseños contenían una sola caché para almacenar las referencias tanto a datos como a instrucciones. Más recientemente, se ha hecho normal separar la caché en dos: una dedicada a instrucciones y otra a datos. La ventaja clave del diseño de una caché partida es que elimina la competición por la caché entre el procesador de instrucciones y la unidad de ejecución. Esto es importante en diseños que cuentan con segmentación de cauce de instrucciones.

a) Describa las funciones de correspondencia entre memoria principal y caché. [EXAMEN 10/22 – 07/23 – 07/24]

b) Compare las correspondencias entre la MP y la caché. ¿Cómo se determina, en cada caso, la hilera/fila de ubicación de un bloque de palabras proveniente de la MP? Ejemplifique. [EXAMEN 11/24]

Tipos de Correspondencia o Mapeos

- Correspondencia totalmente asociativa [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. **Ventajas y Desventajas**: cada bloque de la memoria principal puede estar en solo una línea de la caché. Este mapeo es sencillo y económico, pero si dos bloques de memoria deben ir a la misma línea de caché de forma repetida, los fallos serán grandes (pérdidas de caché).



INDICE indicará la línea o el conjunto que le corresponde.
BO representa todas las direcciones que pertenecen al bloque.

- Correspondencia directa [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 → 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 conmutando de bloques. **Ventajas y desventajas**: un bloque de memoria puede ser almacenado en cualquier línea de la caché. Esto da mayor flexibilidad y reduce las colisiones de caché, pero el hardware necesario para realizar las búsquedas es más costoso y lento, ya que se deben revisar todas las líneas de la caché para encontrar el bloque adecuado.

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

- Correspondencia asociativa por conjuntos [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 → grupo 0 → bloques 0 y 1 de la Memoria Cache. Entonces, si la CPU pretende acceder a este bloque, y quiere verificar si esta copiado en Cache, sólo debe verificar el bloque 0 y 1 de Cache, si no está ahí lo tiene que traer de Memoria Principal. **Ventajas y desventajas**: en este caso, la caché se divide en grupos, y cada bloque de memoria corresponde a cualquiera de las líneas dentro de un conjunto específico. Básicamente esta correspondencia combina lo mejor de las otras dos.

b) Analice las políticas de escritura desde el punto de vista de la coherencia de datos. [EXAMEN 10/22 – 07/23 – 07/24]

Políticas de escritura: determina cómo se manejan las escrituras en la caché (si se actualizan de inmediato en la memoria principal o no). Esto es crucial para evitar la inconsistencia de datos entre la memoria caché y la memoria principal.

- **Política de escritura en acierto**
  - Write-through (Escritura inmediata): en esta política, cada vez que la CPU escribe un dato en la caché, la misma escritura se realiza de manera simultánea en la memoria principal.

    Ventajas: garantiza que la memoria principal siempre tenga los mismos datos que la caché, lo que evita la inconsistencia de datos entre ambos niveles.

    Desventajas: al actualizar ambas memorias simultáneamente, se genera un tráfico significativo hacia la memoria principal, lo que puede ralentizar el rendimiento.
  - Write-back (Post-escritura): en esta política, los datos se actualizan únicamente en la caché, y se marca el bloque de memoria con un bit de "sucio" (dirty bit), lo que indica que el bloque de caché contiene datos modificados que aún no han sido escritos en la memoria principal.

    Ventajas: reduce el tráfico hacia la memoria principal, ya que las escrituras solo se realizan cuando un bloque es reemplazado de la caché.

    Desventajas: puede ocasionar que la memoria principal contenga datos erróneos temporalmente hasta que el bloque modificado sea reemplazado en la caché. Esto puede ser problemático en sistemas que requieren que la memoria principal esté actualizada constantemente.
- **Política de escritura en fallo**
  - Write allocate: en esta política, cuando ocurre un fallo de caché, el bloque correspondiente de la memoria principal se carga en la caché y la escritura se realiza en la caché. Dado que la política de escritura es **Write-back**, el bloque de memoria se carga en la caché y la escritura se hace directamente allí, marcando el bit de "sucio" para indicar que la caché tiene una versión modificada del bloque.
  - No-write allocate: en este caso, cuando ocurre un fallo de caché, el bloque no se carga en la caché, sino que se escribe directamente en la memoria principal. Esta política es común con la política de escritura

Write-through, ya que la escritura se realiza directamente en la memoria principal sin necesidad de cargar primero el bloque en la caché.

Conclusión: desde el punto de vista de la coherencia de datos, la política de escritura en caché tiene un impacto significativo en el rendimiento de un sistema, especialmente en términos de la consistencia entre la caché y la memoria principal. El uso de **write-through** puede garantizar que ambas memorias estén sincronizadas, pero aumenta el tráfico hacia la memoria principal, lo que puede ralentizar el sistema. Por otro lado, **write-back** reduce este tráfico, pero puede introducir inconsistencias temporales en la memoria principal. Además, las políticas de **write allocate** y **no-write allocate** determinan cómo se gestionan los fallos de caché y cómo se gestionan las escrituras en esos casos, afectando también la eficiencia del sistema.

a) Si se pretende mejorar el tiempo de acceso medio a la memoria caché ¿Sobre qué parámetros será necesario trabajar y qué propone como medidas para hacerlo? [EXAMEN 11/24]

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 acierto 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.

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, tendré 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.

b) ¿La coherencia de datos de un sistema jerárquico se ve afectada por el uso de DMA? [EXAMEN 03/24 – 08/24 – 12/24]

Sí, la coherencia de datos en un sistema jerárquico de memoria puede verse afectada por el uso de DMA (Acceso Directo a Memoria). ¿Por qué? Porque el DMA permite que los dispositivos de entrada/salida (E/S) transfieran datos directamente a la memoria principal o entre dispositivos sin la intervención de la CPU. Esto es beneficioso para aliviar la carga de la CPU, pero también puede generar problemas de coherencia de datos. En un sistema con caché, los datos en la memoria caché pueden no estar actualizados con respecto a los datos en la memoria principal, especialmente cuando el DMA actualiza la memoria principal sin pasar por la caché.

Por ejemplo, si un dispositivo E/S utiliza DMA para escribir en la memoria principal, y la CPU tiene una copia de esos datos en la caché, puede que no esté al tanto de los cambios realizados por el DMA. Esto puede generar inconsistencias, ya que la caché de la CPU tendría datos obsoletos, mientras que la memoria principal contiene la versión más reciente.

También si la CPU intenta leer los datos de su caché mientras un dispositivo está usando DMA para escribir en la memoria, se corre el riesgo de que la CPU lea datos desactualizados, lo que afecta la coherencia. Para evitar esto, se requieren mecanismos adicionales para garantizar que los datos estén sincronizados entre la memoria caché, la memoria principal y cualquier dispositivo que utilice DMA.

Justifique el uso de dos niveles de caché.

El uso de dos niveles de caché (L1 y L2) es una característica clave en la jerarquía de memoria. Algunos motivos para el uso de dos niveles de caché son:

- Reducción del tiempo de acceso promedio: Permite el acceso rápido a datos de uso común sin la necesidad de acceder a la memoria principal (RAM). El L1 es el más rápido pero el más pequeño y cercano al núcleo del procesador, mientras que el L2 es más grande pero un poco más lento.
- Aprovechamiento de la localidad de referencia: Permite explotar la localidad espacial y temporal, ya que los datos accedidos en el L1 pueden propagarse al L2 si son utilizados repetidamente.
- Optimización de costos y consumo de energía: Permite equilibrar el rendimiento y los recursos disponibles de manera más eficiente. El L1 puede ser diseñado con tecnologías más avanzadas y costosas, mientras que el L2 puede ser accesible con tecnologías más económicas.
- Reducción de la latencia de acceso a la memoria principal: Cuando un dato no está en el L1, pero sí en el L2, aún se puede acceder a él más rápidamente que si tuviera que recuperarse directamente desde la memoria principal. Esto ayuda a reducir la latencia general de acceso a memoria, mejorando el rendimiento del sistema.
- Mayor escalabilidad: Permite una mayor escalabilidad y rendimiento en sistemas de gama alta sin aumentar significativamente los costos y la complejidad.

ANEXO – Buses del Sistema

Estructuras de Interconexión en Computadoras: en los sistemas de computación, todas las unidades (memoria, módulo de entrada/salida (E/S), procesador) deben estar interconectadas de manera eficiente para permitir la correcta comunicación y operación del sistema. Existen distintos tipos de interconexiones que se utilizan para conectar estas unidades entre sí.

- **Interconexión de la Memoria**: la memoria es una de las unidades más críticas en un sistema de computación, y su interconexión cumple varias funciones esenciales.
    - Recibe y entrega datos: almacena los datos que el procesador necesita y los entrega cuando se solicitan.
    - Recibe direcciones: determina la ubicación de trabajo dentro de la memoria, indicándole al sistema dónde debe leer o escribir.
    - Recibe señales de control: incluye señales para realizar operaciones de lectura o escritura y temporización, gestionando cuándo y cómo se accede a los datos almacenados.
- **Interconexión del Módulo de E/S**: el módulo de E/S es funcionalmente similar a la memoria, pero su propósito es facilitar la comunicación entre el procesador y los periféricos del sistema (como teclados, monitores, impresoras, etc.).
    - Recibe y entrega datos del procesador: al igual que la memoria, gestiona la transferencia de datos hacia y desde el procesador.
    - Envía y recibe datos a periféricos: permite la comunicación entre el procesador y los dispositivos de E/S.
    - Recibe direcciones: identifica la ubicación del periférico en el sistema.
    - Gestiona señales de control: el procesador envía señales de control al periférico y recibe señales de control de vuelta. También gestiona las interrupciones, que notifican al procesador que un dispositivo necesita atención.
- **Interconexión del Procesador**: el procesador tiene múltiples funciones en términos de interconexión:
    - Lee instrucciones y datos: el procesador recibe datos y las instrucciones del sistema para su ejecución.
    - Escribe datos procesados: después de realizar las operaciones, el procesador puede escribir los resultados en la memoria o en un módulo de E/S.
    - Envía señales de control: el procesador también envía señales de control a otras unidades del sistema, como la memoria o los módulos de E/S.
    - Recibe señales de interrupción: el procesador recibe señales de interrupción que indican que un evento requiere su atención (por ejemplo, una solicitud de E/S).

¿Qué es un bus?

Buses de Comunicación

Los buses son sistemas de interconexión fundamentales en cualquier arquitectura de computadora. Son medios de transmisión de datos entre diversos dispositivos del sistema. Los buses se agrupan en diferentes categorías según su función, y son responsables de garantizar que las distintas unidades del sistema puedan comunicarse entre sí de manera eficiente.

Un **bus** es un conjunto de caminos de comunicación entre dos o más dispositivos. Normalmente, un bus se presenta como un medio de transmisión de señales eléctricas a través de varias líneas o conductores que permiten el paso de información. Un bus puede tener diferentes tipos de caminos, como líneas para datos, direcciones y control. Por ejemplo, un **dato de 8 bits** puede transmitirse mediante ocho líneas del bus, formando un conjunto de conductores que permiten la transferencia de la información entre los dispositivos interconectados.

Respecto a las características físicas de un bus, este se compone de **conductores eléctricos paralelos** que son típicamente **líneas de metal**. Estos conductores están conectados a **tarjetas de expansión** o dispositivos a través de conectores, facilitando la comunicación entre los distintos componentes del sistema.

- **Tipos de Buses**
    1. **Bus de Datos**: este bus es responsable de transmitir los datos entre las diferentes unidades del sistema, como la CPU, la memoria y los dispositivos de E/S. A nivel de bus, no existe diferencia entre "datos" e "instrucciones", ya que ambas son tratadas como datos. El ancho del bus de datos es clave para determinar el rendimiento del sistema, ya que el bus debe ser lo suficientemente ancho para permitir la transmisión rápida de grandes cantidades de información. Los buses de datos comunes tienen anchos de 8, 16, 32 o 64 bits.
    2. **Bus de Dirección**: el bus de dirección identifica la fuente o destino de los datos que se están transfiriendo. Cuando el procesador quiere acceder a una palabra de memoria, utiliza este bus para enviar la dirección correspondiente. El ancho del bus de dirección determina el tamaño máximo de memoria que puede manejar el sistema. Por ejemplo, un bus de dirección de 16 bits puede direccionar hasta 64K lugares de memoria.
    3. **Bus de Control**: el bus de control transmite señales que gestionan la sincronización de las operaciones del sistema. Entre estas señales se encuentran:
        - Señales de lectura/escritura para indicar si los datos deben leerse de la memoria o escribirse en ella.
        - Petición de interrupción que indica que un dispositivo necesita la atención del procesador.
        - Señales de reloj que sincronizan las operaciones.

- **Problemas de un Único Bus**: en los sistemas informáticos, la interconexión de dispositivos a través de un único bus puede generar varios problemas. Cuando se conectan múltiples dispositivos a un solo bus, surgen retardos de propagación debido a la distancia y el tiempo que tardan las señales en llegar a los dispositivos. Además, si el control del bus pasa de un dispositivo a otro, esta transición puede afectar significativamente las prestaciones del sistema. Para mitigar estos problemas, la mayoría de los sistemas modernos emplean varios buses, lo que permite mejorar el rendimiento y reducir las colisiones y los retardos.

Describa la diferencia entre los buses Dedicados y los Multiplexados.

- **Tipos de Buses**: los buses pueden clasificarse en dos tipos principales según cómo gestionan las señales:
  - Buses Dedicados: acá **se utilizan líneas separadas para direcciones y datos**. Esto significa que se disponen de líneas dedicadas exclusivamente para las direcciones (por ejemplo, 16 líneas de direcciones) y otras para los datos (por ejemplo, 16 líneas de datos). Además, se incluye una línea de control para lectura o escritura (R/W). Este tipo de bus proporciona una **mayor simplicidad y rendimiento** debido a que las señales no comparten el mismo conjunto de líneas, pero requiere más conexiones físicas.
  - Buses Multiplexados: acá, se utilizan las **mismas líneas para transmitir tanto direcciones como datos**, lo que reduce el número de líneas necesarias. En este caso, se suelen emplear 16 líneas que pueden ser utilizadas tanto para direcciones como para datos. Además, se incluye una línea de control para lectura o escritura (R/W) y otra línea para definir si las líneas están transmitiendo direcciones o datos (A/D). Aunque este diseño reduce el número de líneas, **requiere más circuitería** y puede generar cierta pérdida en el rendimiento en comparación con los buses dedicados, ya que se debe alternar entre las funciones de dirección y datos.

Describa los distintos modos de arbitraje y sincronización

- **Arbitraje del Bus**: este es el proceso de controlar el acceso al bus, especialmente cuando varios dispositivos desean acceder a él simultáneamente. En un sistema, solo un módulo puede transmitir a través del bus en un momento dado, y es necesario establecer un mecanismo para decidir cuál tiene prioridad para el acceso al bus. Existen dos tipos principales de arbitraje:
  - Arbitraje Centralizado: en el arbitraje centralizado, **un único dispositivo de hardware**, conocido como el **controlador del bus o árbitro**, es responsable de asignar los tiempos de acceso al bus. Este controlador puede ser un módulo separado del procesador o, en algunos casos, estar integrado dentro del procesador. El árbitro decide cuál de los módulos conectados puede usar el bus en cada momento.
  - Arbitraje Distribuido: en este caso, **cada módulo** conectado al bus tiene su propia lógica para controlar el acceso al bus. En este sistema, no existe un único controlador centralizado; en

cambio, todos los módulos gestionan la solicitud y concesión del acceso al bus de manera distribuida, lo que puede mejorar la eficiencia en sistemas donde la carga de trabajo se distribuye entre múltiples módulos.

- **Temporización**: la temporización o sincronización es un aspecto clave para coordinar las operaciones en un bus y asegurar que los eventos ocurran en el orden correcto. Existen dos tipos principales de temporización:
  - Temporización Síncrona: en este caso, la presencia de un evento está determinada por un **reloj**. El bus incluye una línea de reloj que sincroniza todas las operaciones. El intervalo entre un "uno" y un "cero" en la línea de reloj se conoce como un ciclo de bus, y los dispositivos del bus se sincronizan con el **flanco de subida** del reloj. La mayoría de los eventos en el bus se prolongan durante un solo ciclo de reloj, lo que garantiza una ejecución ordenada y sincrónica de las operaciones.
  - Temporización Asincrónica: la presencia de un evento en el bus es consecuencia y depende de que se produzca un evento previo.

  La temporización sincrónica es más fácil de implementar y comprobar, pero es menos flexible ya que todos los dispositivos deben usar la misma frecuencia de reloj.

Mencione las principales diferencias entre un bus PCI y SCSI.

Bus PCI (Peripheral Component Interconnect Interconexión de Componente Periférico): es un bus de ancho de banda elevado, independiente del procesador, que se puede utilizar como bus de periféricos o bus para una arquitectura de entreplanta.

Bus SCSI (Small Computer System Interface Pequeña interfaz del sistema de cómputo): es una interfaz estándar para la transferencia de datos entre distintos dispositivos del bus de la computadora. Se utiliza para comunicar dispositivos rápidos, como discos CD-ROM, dispositivos de audio y dispositivos de almacenamiento externo de datos. Requiere un controlador de interfaz.

¿Qué elementos característicos definen un bus? [EXAMEN 10/22 – 07/23 – 03/24 – 07/24 – 08/24 – 12/24]

Los elementos claves de un bus son:

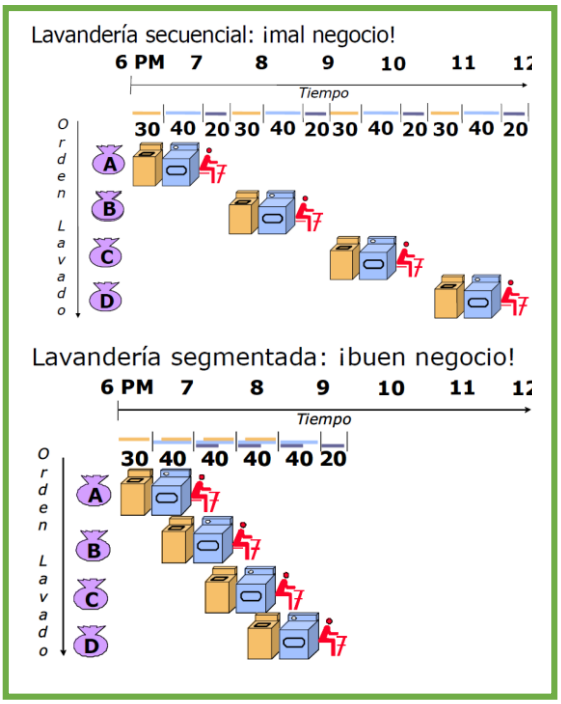
- **Tipos de buses**: las líneas del bus pueden ser dedicadas (está permanentemente asignada a una función o a un subconjunto físico de componentes del computador) o multiplexadas (utiliza las mismas líneas para usos diferentes).
- **Método de arbitraje**: Puede ser centralizado (un único dispositivo hardware es responsable de asignar tiempos en el bus) o distribuido (cada módulo dispone de lógica para controlar el acceso y los módulos actúan conjuntamente para compartir el bus).

- **Temporización**: Es la forma de coordinar los eventos en el bus, puede ser sincrónica (la presencia de un evento en el bus está determinada por un reloj) o asincrónica (la presencia de un evento en el bus es consecuencia y depende de que se produzca un evento previo).
- **Anchura del bus**: Está determinada por la cantidad de líneas que tiene el bus. En un bus de datos, la anchura determina cuántos bits se pueden transferir al mismo tiempo. En un bus de direcciones, la anchura del bus determina la máxima capacidad de memoria posible en el sistema.
- **Tipo de transferencia de datos**: Todos los buses permiten tanto transferencias de escritura (dato de maestro a esclavo) como de lectura (dato de esclavo a maestro). En ciertos buses también son posibles algunas operaciones combinadas, como la lectura modificación escritura (lectura seguida inmediatamente de una escritura en la misma dirección), o la lectura después de escritura (una escritura seguida inmediatamente de una lectura en la misma dirección). Algunos buses permiten también transferencias de bloques de datos.

Segmentación del Cauce y Atascos

a) ¿Qué entiende por segmentación del cauce? [EXAMEN 07/23 – 07/24 – 11/24]

a) ¿Qué es la segmentación del cauce de instrucciones? [EXAMEN 10/23]



La segmentación de cauce, o **pipelining**, es una técnica de diseño hardware que mejora la eficiencia de la CPU permitiendo que varias operaciones se realicen simultáneamente. En lugar de ejecutar una instrucción completa de principio a fin en un solo ciclo de reloj, el proceso de ejecución de las instrucciones se descompone en varias fases o etapas. Cada una de estas fases se realiza de manera simultánea, pero con diferentes instrucciones en cada una de ellas. Esto permite que el procesador aproveche el paralelismo entre las instrucciones de un flujo secuencial, lo que mejora significativamente el rendimiento del sistema.

Un ejemplo clásico de esta estrategia es similar a la línea de ensamblaje en una planta de manufactura [Ejemplo de lavandería al costado]. El producto (en este caso, la instrucción) pasa por varios estados en el proceso de producción. Al mismo tiempo, varios productos pueden ser manipulados simultáneamente, cada uno en una etapa diferente del proceso. Esto permite que el proceso se inicie nuevamente antes de que el primer producto final salga del sistema, aumentando la eficiencia.

Características:

- La segmentación es una mejora a nivel de diseño del hardware. No es visible para el programador.
- Para que sea efectiva, es necesario que las etapas del proceso de ejecución estén bien uniformizadas. Esto significa que el tiempo total del proceso estará determinado por la etapa más lenta, lo que obliga a un diseño cuidadoso de las fases para optimizar la velocidad global.
- El rendimiento de los procesadores segmentados está muy influenciado por el conjunto de instrucciones que el procesador es capaz de manejar.

Tareas a realizar por ciclo: el ciclo de ejecución de una instrucción segmentada consta de varias fases:

- **Búsqueda de la instrucción (Fetch)**, donde se accede a la memoria para obtener la instrucción y se incrementa el contador de programa (PC).
- **Decodificación de la instrucción (Decode)**, lo que implica determinar la operación a realizar y, si es necesario, acceder al banco de registros para obtener los operandos.
- **Fase de ejecución (Execute)**, se realiza la operación en la unidad aritmético-lógica (ALU).
- **Fase de acceso a memoria (Memory Access)**, si la instrucción requiere acceso a memoria, se realiza en esta fase.
- **Fase de almacenamiento (Writeback)**, se guarda el resultado de la operación en el banco de registros, si es necesario.

b) ¿Qué ventajas proporciona su implementación? [EXAMEN 07/23 – 07/24]

La implementación de la segmentación de cauce proporciona varias ventajas, entre ellas:

- Mejora el rendimiento: la ejecución en paralelo acelera la velocidad de ejecución de las instrucciones.
- Aprovechamiento de recursos: mientras una unidad de ejecución realiza una operación, las etapas anteriores y posteriores pueden estar ocupadas con otras instrucciones, lo que permite un uso más completo de las unidades funcionales.
- Mayor paralelismo: al permitir que múltiples instrucciones se ejecuten al mismo tiempo, la segmentación de cauce aumenta el nivel de paralelismo a nivel de instrucciones en un procesador.
- Reducción del ciclo de reloj por instrucción: reduce el tiempo necesario para ejecutar una instrucción completa al dividirla en etapas más pequeñas.
- Mejora de la predicción de saltos: puede combinarse con técnicas de predicción de saltos para reducir las penalizaciones por fallos de predicción de saltos, lo que resulta en una ejecución más eficiente en las instrucciones.

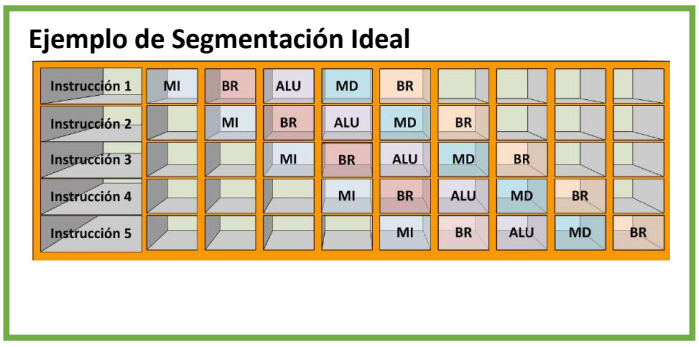
Segmentación del Cauce. b) Rendimiento. [EXAMEN 11/24]

b) ¿Cuánto mejora el rendimiento? [EXAMEN 10/23]

Las prestaciones del cauce segmentado dependen de varios factores. En teoría, el rendimiento máximo que se puede alcanzar es completar una instrucción en cada ciclo de reloj. Si K es el número de etapas del cauce, entonces el rendimiento del procesador segmentado es igual al rendimiento secuencial multiplicado por K. Esto indica que el incremento potencial en el rendimiento con la segmentación es proporcional al número de etapas del cauce, lo que incrementa la productividad o throughput, pero no reduce el tiempo de ejecución de la instrucción en sí misma.

Ejemplo de Segmentación Ideal: en este caso, se cumplen las siguientes suposiciones.

- Todas las tareas duran el mismo tiempo.
- Las instrucciones siempre pasan por todas las etapas.
- Todas las etapas pueden ser manejadas en paralelo.



a) Describa tres diferentes causas que puedan retardar un cauce de instrucciones segmentado. [EXAMEN 07/23 – 07/24]

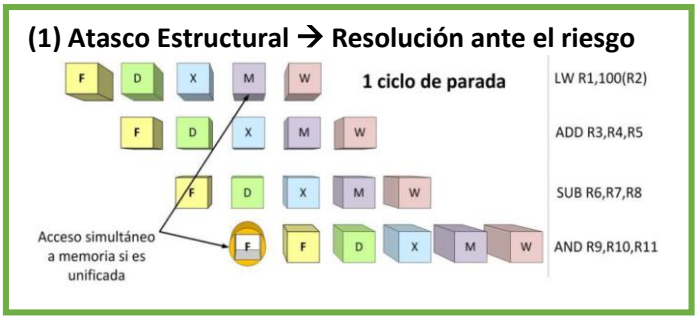
Sin embargo, en la práctica, estas suposiciones no siempre se cumplen. Puesto que, dentro de los problemas principales de la segmentación:

- No todas las instrucciones requieren todas las etapas del cauce. Por ejemplo, la instrucción **MOV AX, mem** en el procesador MSX88, no requieren la etapa de ejecución (X).
- No todas las etapas pueden ser manejadas en paralelo. por ejemplo, **las etapas de búsqueda (F) y acceso a memoria (M) necesitan acceder a la memoria**, lo que puede generar conflictos.
- Deben considerarse los saltos de control, ya que **pueden alterar el flujo de ejecución de las instrucciones** y hacer que se detenga la segmentación.

[Puedo contestar esta pregunta usando los tipos de atascos (que están abajo)].

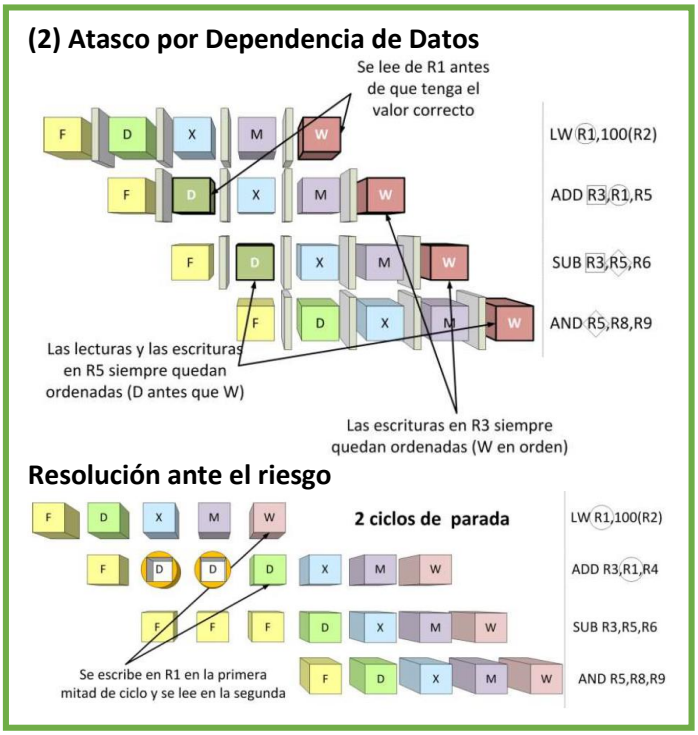
b) ¿Qué retardo produce cada una? [EXAMEN 07/23 – 07/24] → Para hablar de retardos = Ciclos de reloj.

Atascos del cauce [Stalls]: estos atascos, son situaciones que impiden que la siguiente instrucción se ejecute en el ciclo correspondiente. Los atascos pueden ser de varios tipos.



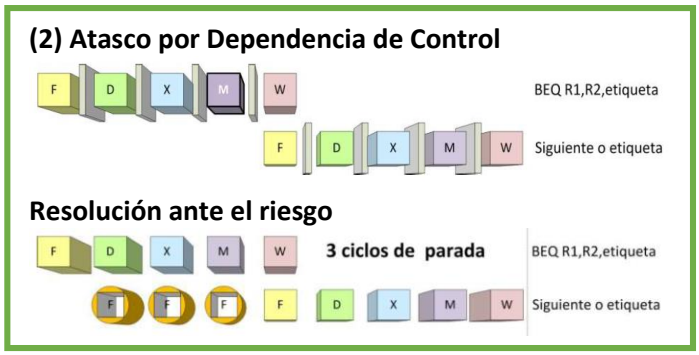
- Estructurales: ocurren cuando dos o más instrucciones requieren utilizar el mismo recurso hardware en el mismo ciclo, generando un conflicto. **Para resolver estos problemas**, se pueden replicar los recursos hardware, segmentar los accesos a unidades funcionales, o realizar turnos en el acceso a los registros. Una técnica común es duplicar los recursos, como agregar sumadores o restadores además de la ALU, o separar las memorias de instrucciones y datos. También se pueden organizar las escrituras en la primera mitad de los ciclos de reloj y las lecturas en la segunda mitad.
- Por dependencia de Datos: suceden cuando una instrucción depende de los datos de otra que se ejecuta previamente. Por ejemplo, si una instrucción produce un dato y otra lo consume, y ese dato no está disponible en el momento necesario, se produce un "stall". El riesgo por dependencia de datos puede **solucionarse** mediante dos enfoques: hardware y software. En hardware, se emplea la técnica de **adelantamiento de operandos** o **forwarding**, que consiste en transmitir directamente el resultado de una instrucción a la siguiente sin esperar a que se complete la escritura. En software, se pueden utilizar instrucciones **NOP** (sin operación) o reordenar el código para evitar que las instrucciones dependan entre sí. En este caso, se distingue entre tres tipos de dependencias de datos:
  - Lectura después de Escritura (**RAW**): Una instrucción genera un dato que una instrucción posterior necesita leer. Esta es una dependencia verdadera y puede causar un retraso si la instrucción que produce el dato no ha terminado antes de que la otra lo necesite.
  - Escritura después de Escritura (**WAW**): Ocurre cuando una instrucción escribe un dato después de que otra instrucción lo haga. Esta dependencia solo ocurre si se permite que las instrucciones se adelanten unas a otras, lo que puede causar inconsistencias en el resultado.

    c) ¿Qué es un riesgo WAR en un cauce segmentado? [EXAMEN 10/23]



  - Escritura después de Lectura (**WAR**): Se da cuando una instrucción modifica un valor antes de que otra instrucción anterior que debía leer ese valor lo haga. En un cauce simple, este tipo de dependencia no es posible, ya que no se permite que las instrucciones se adelanten en el orden de ejecución.

c) Describa problema y posibles soluciones ante riesgos por transferencias de control de programa. [EXAMEN 11/24]



- Por dependencia de Control: ocurre cuando la ejecución de una instrucción en un procesador segmentado se ve afectada negativamente debido a saltos de programa, como instrucciones de salto condicional o incondicional. Estos riesgos pueden dar lugar a retrasos en la ejecución de instrucciones y pueden afectar el rendimiento general del procesador.

Los riesgos por transferencia de control pueden ser: salto condicional tomado, salto condicional no tomado o salto incondicional. Para mitigar estos riesgos, se utilizan técnicas de **predicción de saltos**, que permiten anticipar el resultado del salto y continuar la ejecución de la instrucción sin esperar a resolver la condición. Se pueden usar técnicas **estáticas**, como predecir que nunca se salta o siempre se salta, o **dinámicas**, como un conmutador saltar/no saltar basado en la historia de las instrucciones, que se almacena en una **tabla de historia de saltos** (branch-target buffer). Otra técnica es el **salto retardado**, que utiliza el "hueco de retardo" (delay-slot) para ejecutar instrucciones útiles mientras se resuelve el salto, evitando la parada total del cauce. En algunos casos, se puede precaptar la instrucción de salto y su destino en **flujos múltiples**, donde cada opción de salto se maneja en cauces separados, aunque esto puede aumentar la complejidad y los retrasos en el acceso a los recursos.

En resumen, las posibles soluciones son:

- Adelantar la resolución de los saltos a la etapa de decodificación: en ella se decodifican y se sabe que es un salto. Se puede evaluar la condición de salto (con restador) y se puede calcular la dirección de salto (con sumador).
- Técnica Hardware → Predicción de saltos:

  (a) Técnicas estáticas: se puede predecir que nunca se salta (asume que el salto no se producirá y siempre capta la siguiente instrucción) o predecir que siempre se salta (asume que el salto se producirá y siempre capta la instrucción destino del salto).

  (b) Técnicas dinámicas: conmutador saltar/no saltar basado en la historia de las instrucciones. Es eficaz en bucles. Se hace uso de la tabla de historia de saltos (Branch-target buffer) que es una pequeña cache asociada a la etapa de búsqueda (F).

  (c) Predecir según el código de operación: hay instrucciones con más probabilidades de saltar.

  (d) Flujos múltiples: hay varios cauces (uno por cada opción de salto). Pre-captan cada salto en diferentes cauces. Se debe utilizar el cauce correcto.

  (e) Buffer de bucles: es una memoria muy rápida, gestionada por la etapa de captación de instrucción del cauce. Comprueba el buffer antes de hacer la captación de memoria.
- Técnica Software → Salto retardado o de relleno de ranura de retardo:

  El compilador introduce instrucciones que se ejecutarán en cualquier caso después de la instrucción de salto, y de no ser posible se utilizan instrucciones NOP. Esta técnica requiere reordenar las instrucciones.

Explique los atascos producidos por saltos

Los atascos producidos por saltos son situaciones en las que la ejecución de instrucciones en un cauce segmentado se ve afectado negativamente debido a la presencia de instrucciones de salto condicional o incondicional. Hay tres tipos principales de atascos de control causados por saltos:

- **Salto condicional tomado**: cuando la condición se resuelve como verdadera, lo que significa que se debe tomar el salto. Las instrucciones que siguen al salto condicional y ya han avanzado deben descartarse, lo que conduce a un desperdicio de ciclos de reloj y una penalización en el rendimiento.
- **Salto condicional no tomado**: cuando la condición se resuelve como falsa, lo que significa que no se toma el salto. Las instrucciones que siguen al salto deben continuar su ejecución.
- **Salto incondicional**: ocurre cuando se encuentra una instrucción de salto incondicional. Las instrucciones que siguen ya han avanzado en el cauce y se encuentra en diferentes etapas de ejecución. Cuando se toma el salto, estas instrucciones en etapas posteriores deben ser descartadas, lo que resulta en un desperdicio de ciclos de reloj y una penalización en el rendimiento.

ANEXO – WinMips64

El **procesador WinMIPS** es un procesador basado en la arquitectura MIPS, que es ampliamente conocida por su diseño sencillo y eficiente.

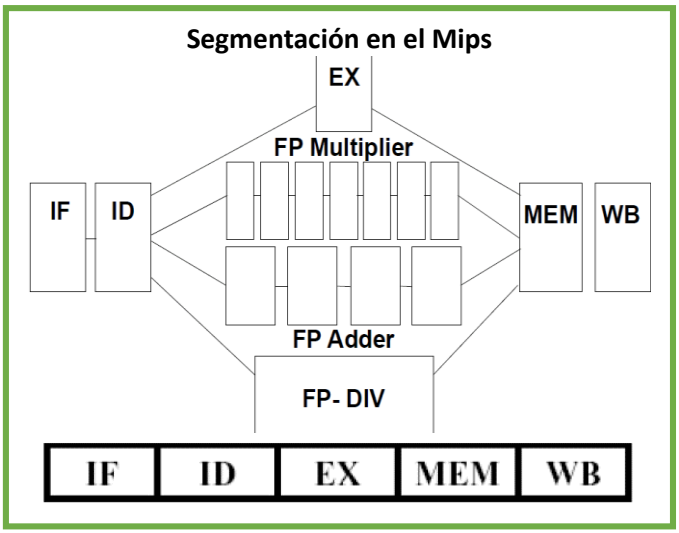
Características del Procesador WinMIPS:

- **Registros de uso general**: el WinMIPS tiene un total de 32 registros de uso general numerados de r0 a r31, cada uno con un tamaño de 64 bits.

  r0 es un registro especial que siempre contiene el valor 0, es decir, no se puede modificar.

  Los otros 31 registros (r1 a r31) se utilizan para operaciones generales, como el almacenamiento temporal de datos o el paso de parámetros entre funciones.
- **Registros de punto flotante**: además de los registros de uso general, el procesador también dispone de 32 registros de punto flotante, numerados de f0 a f31, con un tamaño también de 64 bits. Estos registros son utilizados específicamente para almacenar valores de punto flotante y realizar operaciones de cálculo de precisión decimal (por ejemplo, operaciones matemáticas como suma, multiplicación, etc.).
- **Memoria**: el procesador tiene acceso a una memoria de 230 palabras, donde cada palabra tiene 32 bits de longitud. La memoria es limitada y su tamaño puede ser restringido, lo que implica que se debe gestionar cuidadosamente el uso de memoria en programas que corran en este procesador.
- **Instrucciones**: las instrucciones del WinMIPS tienen una longitud de 1 palabra (32 bits). El conjunto de instrucciones sigue el modelo RISC (Reduced Instruction Set Computing), donde cada instrucción está diseñada para realizar operaciones simples y rápidas.
- **Instrucciones de acceso a memoria**: el acceso a memoria en el procesador WinMIPS está limitado a 2 tipos de instrucciones principales:
  - **LOAD**: Esta instrucción carga un valor desde la memoria hacia un registro. Permite que el procesador recupere datos desde la memoria para su posterior procesamiento.
  - **STORE**: La instrucción STORE almacena el valor contenido en un registro de vuelta en la memoria.
- **Rendimiento y simplicidad**: el procesador WinMIPS es un ejemplo de arquitectura RISC, lo que implica que sus instrucciones son simples y se ejecutan en un solo ciclo de reloj. Esto permite un alto rendimiento con un diseño relativamente sencillo.

Llamadas a procedimientos → EL MIPS no tiene pila de hardware, almacena la dirección de retorno siempre en R31.



Segmentación en el Mips

1. **Búsqueda [IF]**
   - Se accede a memoria por la instrucción.
   - Se incrementa el PC.
2. **Decodificación / Búqueda de operandos [ID]**
   - Se decodifica la instrucción.
   - Se accede al banco de registros por los operandos.
   - Se calcula el valor del operando inmediato con extensión de signo (si hace falta).
   - Si es un salto, se calcula el destino y si se toma o no.
3. **Ejecución / Dirección efectiva [EX]**
   - Si es una instrucción de proceso, se ejecuta en la ALU.
   - Si es un acceso a memoria, se calcula la dirección efectiva.
   - Si es un salto, se almacena el nuevo PC.
4. **Acceso a memoria / terminación del salto [MEM]**
   - Si es un acceso a memoria, se accede
5. **Almacenamiento [WB]**
   - Se almacena el resultado (si lo hay) en el banco de registros

RISC

Historia: la evolución de los sistemas computacionales ha estado marcada por avances significativos a lo largo de las décadas, siendo la introducción de nuevos conceptos fundamentales para la mejora de la arquitectura y el rendimiento de las computadoras. El concepto de "familia" de procesadores fue introducido por IBM en su System/360 en 1964, el cual permitió separar la arquitectura de la implementación. Además, se popularizó la idea de la unidad de control microprogramada, propuesta por Wilkes en 1951 e implementada por IBM en la línea S/360. A lo largo de los años, las computadoras fueron incorporando nuevas tecnologías, como la **memoria caché** en 1968 con el IBM S/360 Modelo 85, y el avance hacia **RAM de estado sólido**.

Un hito importante en la evolución de los procesadores fue la creación de los **microprocesadores**, comenzando con el Intel 4004 en 1971, un avance que dio paso a una variedad de procesadores con distintos propósitos, como los de propósito general, embebidos o empotrados, y procesadores múltiples.

Describa tres características que usted considere las más importantes de las arquitecturas RISC.

Procesadores RISC: es una arquitectura de procesador que se caracteriza por tener un conjunto reducido de instrucciones simples y rápidas, diseñadas para ejecutarse en un solo ciclo de reloj. La idea principal es optimizar la velocidad de ejecución de las instrucciones, utilizando más registros y delegando la complejidad al software (compiladores).

Su principal objetivo es mejorar la eficiencia en la ejecución de programas mediante la simplificación de su repertorio de instrucciones. Los procesadores RISC están caracterizados por un **gran número de registros de uso general** que permiten optimizar el uso de la memoria y las operaciones. Su repertorio de instrucciones es limitado, lo que facilita la segmentación de las instrucciones y mejora el rendimiento al permitir ejecutar una mayor cantidad de instrucciones en cada ciclo de reloj.

Características principales RISC (principios clave):

- Una instrucción por ciclo de reloj: Las máquinas RISC están diseñadas para ejecutar una sola instrucción en cada ciclo de reloj, lo que facilita una ejecución rápida y eficiente.
- Operaciones registro a registro: Las operaciones se realizan principalmente entre registros, lo que reduce la necesidad de acceso a la memoria y mejora la velocidad. Además, fomenta la optimización del uso de registros.
- Modos de direccionamiento sencillos: Los modos de direccionamiento son simples, lo que facilita la decodificación de las instrucciones y mejora el rendimiento.
- Formatos de instrucción sencillos: Las instrucciones en RISC tienen un formato fijo, lo que hace que el procesamiento y la interpretación sean rápidos.
- Diseño cableado (sin microcódigo): Los procesadores RISC suelen tener un diseño más sencillo y directo, sin la necesidad de un microprograma que controle la ejecución de las instrucciones.
- Mayor tiempo/esfuerzo de compilación: Aunque el procesador RISC es simple y rápido, requiere un compilador más sofisticado y que realice un mayor esfuerzo para optimizar las instrucciones.

Procesadores CISC: los procesadores CISC (Complex Instruction Set Computer) estaban diseñados con la finalidad de **facilitar el trabajo del compilador** y mejorar la eficiencia de la ejecución, al permitir secuencias complejas de

operaciones en microcódigo y dar soporte a lenguajes de alto nivel (HLL) más complejos. Estos procesadores tienen un conjunto más amplio y complejo de instrucciones, donde muchas de ellas pueden realizar múltiples operaciones en un solo ciclo. La idea es hacer que el procesador realice tareas más complejas con menos instrucciones, simplificando el trabajo del programador o el compilador. Sin embargo, esto puede hacer que las instrucciones sean más lentas y más difíciles de implementar.

Características principales CISC:

- Facilitar el trabajo del escritor de compiladores.
- Mejorar la eficiencia de la ejecución con secuencias complejas de operaciones en microcódigo.
- Dar soporte a HLL (lenguajes de alto nivel) más complejos.

Sin embargo, los procesadores CISC presentan varios **inconvenientes**, como:

- El alto costo del software en comparación con el hardware.
- La creciente complejidad del lenguaje de máquina.
- Salto semántico: diferencias entre las operaciones del **lenguaje de alto nivel** y las de la **arquitectura**.

Estos factores llevaron a la creación de **repertorios de instrucciones más grandes**, modos de direccionamiento más complejos, y la implementación de varias sentencias de HLL directamente en el hardware, como es el caso del CASE en el procesador VAX.

Características de la ejecución

El estudio de la ejecución de programas escritos en lenguajes de alto nivel (HLL) ha permitido identificar patrones en las operaciones realizadas por los procesadores. Algunos de los **aspectos clave** que se analizan incluyen:

- Operaciones realizadas: funcionamiento del procesador y la interacción con la memoria.
- Operandos utilizados: tipos y frecuencia de uso de esos operandos, organización de la memoria y modos de direccionamiento.
- Secuenciamiento de la ejecución, y la organización del control y del cauce.

Entre las **operaciones más comunes** en los programas de HLL, se encuentran:

- Las **asignaciones**, como el movimiento de datos.
- Las **instrucciones condicionales**, como los IF y los LOOP, que permiten el control secuencial de la ejecución.
- Las **llamadas y retornos de procedimientos**, operaciones que consumen una cantidad significativa de tiempo.

Además, algunas instrucciones de los lenguajes de alto nivel pueden dar lugar a múltiples operaciones en el código máquina, lo que puede afectar la eficiencia de la ejecución.

En cuanto a los **operandos**, estos suelen ser principalmente **variables escalares locales**. Esto implica que la optimización del rendimiento debe centrarse en la gestión eficiente de las variables locales, ya que el acceso a ellas representa una parte importante de la carga computacional de los programas.

Las **llamadas a procedimientos** en los procesadores suelen consumir mucho tiempo y su duración depende principalmente de dos factores: el número de parámetros que se tratan y el nivel de anidamiento de las funciones. Aunque la mayoría de los programas no tienen una secuencia larga de llamadas seguidas por retornos, es importante

señalar que las variables utilizadas en estos procedimientos suelen ser locales, y las referencias a los operandos están localizadas dentro del código, lo que hace que las optimizaciones relacionadas con el manejo de registros sean cruciales. Una de las **consecuencias** principales de esto es que se puede ofrecer un mejor soporte para los lenguajes de alto nivel (HLL) al optimizar las características más utilizadas y que más tiempo consumen, como el manejo de registros y el diseño de los cauces de instrucciones. Para ello, es recomendable un repertorio de instrucciones reducido y simple, que permita un uso eficiente de los registros y evite complejidades innecesarias en la ejecución de las instrucciones.

Amplio banco de registros: este concepto juega un papel crucial en la optimización. Se pueden utilizar dos enfoques: uno por software y otro por hardware.

- Enfoque de software: se basa en el uso del compilador para asignar registros a las variables más utilizadas en un período de tiempo determinado. Requiere el uso de algoritmos sofisticados para analizar el programa.
- Enfoque por hardware: se utilizan más registros, lo que permite mantener más variables en registros durante períodos más largos sin tener que acceder a la memoria constantemente.

Ventanas de registro, variables locales y globales

En cuanto al uso de **registros para variables locales**, el tener un gran número de registros ayuda a reducir el acceso a la memoria, lo que mejora la eficiencia de la ejecución. Sin embargo, este enfoque enfrenta un problema: cada llamada de procedimiento puede cambiar la "localidad" de las variables, ya que los parámetros deben ser pasados y los resultados deben ser devueltos. Además, las variables locales de los programas de llamada deben ser restauradas, lo que implica un manejo adecuado de los registros.

Para manejar estos problemas, se utilizan **ventanas de registro**, que permiten asignar un pequeño conjunto de registros para cada llamada de procedimiento. Cada conjunto de registros contiene tres áreas principales:

- Registros de parámetros.
- Registros de datos locales.
- Registros temporales.

Los registros temporales de un conjunto se solapan con los registros de parámetros del nivel más bajo adyacente, lo que permite pasar parámetros sin necesidad de transferencia de datos. Esta organización ayuda a optimizar el uso de los registros durante las llamadas y retornos de procedimientos.

En cuanto a las **variables globales**, el compilador asigna posiciones de memoria para ellas, pero esto puede ser ineficiente cuando se accede frecuentemente a estas variables. Una posible solución es incorporar registros especiales para variables globales en el procesador, lo que mejora el acceso a ellas sin depender exclusivamente de la memoria.

Amplio banco de registros vs Caché: al comparar un amplio banco de registros con una cache, vemos que el banco de registros almacena variables escalares locales y globales asignadas por el compilador, utilizando mecanismos de salvaguarda y restauración basados en la profundidad de anidamiento. En cambio, la cache se utiliza para almacenar los datos escalares locales que han sido recientemente utilizados, y las variables locales y globales también se almacenan en ella, con el mecanismo de reemplazo estándar para gestionar el acceso.

Amplio banco de registros	Cache
Todos los datos escalares locales.	Datos escalares locales recientemente usados.
Variables individuales.	Bloques de memoria.
Variables globales asignadas por el compilador.	Variables locales y globales usadas.
Salvaguarda/restauración basada en la profundidad del anidamiento.	Salvaguarda/restauración basada en el algoritmo de reemplazo.
Direccionamiento de registro.	**Direccionamiento de memoria.**

Describa las características que diferencian los procesadores RISC respectos a los CISC.

	RISC	CISC
Instrucción por ciclo de reloj	Se ejecuta una instrucción por ciclo de reloj, lo que hace que la ejecución sea más rápida y predecible.	No siempre se ejecuta una instrucción por ciclo de reloj. Las instrucciones complejas pueden requerir varios ciclos de reloj.
Operaciones registro a registro	Las operaciones se realizan principalmente entre registros, lo que reduce el acceso a memoria y mejora la eficiencia.	Las operaciones pueden involucrar directamente memoria, lo que puede hacer que la ejecución sea más lenta.
Modos de direccionamiento sencillos	Utiliza modos de direccionamiento simples, lo que simplifica la decodificación de las instrucciones y mejora el rendimiento.	Tiene más modos de direccionamiento, lo que aumenta la complejidad y puede afectar la velocidad de ejecución.
Formatos de instrucción sencillos	Las instrucciones tienen un formato fijo y sencillo, lo que facilita su procesamiento.	Las instrucciones pueden tener formatos más complejos y variados, lo que hace más difícil la ejecución eficiente.
Diseño cableado (sin microcódigo)	Utiliza un diseño cableado, lo que significa que las instrucciones se ejecutan directamente sin necesidad de microcódigo.	Utiliza microcódigo para interpretar las instrucciones complejas, lo que agrega un nivel de complejidad.
Mayor tiempo/esfuerzo de compilación	Requiere más tiempo de compilación por la simplicidad de instrucciones, (el compilador debe hacer un trabajo más detallado para optimizar el código).	El compilador tiene menos trabajo que hacer, ya que las instrucciones complejas realizan más operaciones en una sola instrucción.
Repertorio de instrucciones	El repertorio de instrucciones es más reducido y simplificado, permitiendo un procesamiento más rápido y eficiente.	El repertorio de instrucciones es mucho más grande y complejo, lo que permite realizar más operaciones en una sola instrucción, pero a costa de una mayor complejidad.
Uso de registros	Se utiliza un gran número de registros de propósito general, lo que permite una mayor eficiencia en el manejo de los operandos y reduce el acceso a memoria.	Suele utilizar menos registros y depende más del acceso a memoria.
Complejidad de la unidad de control	La unidad de control es más sencilla debido a la naturaleza simplificada de las instrucciones y el formato fijo de las mismas.	La unidad de control es más compleja debido a la variedad de instrucciones y modos de direccionamiento.

Estas diferencias reflejan las filosofías de diseño contrastantes entre RISC y CISC, donde RISC prioriza la simplicidad, velocidad y eficiencia a nivel de hardware, mientras que CISC se enfoca en la complejidad y flexibilidad del conjunto de instrucciones.

Optimización del uso de registros basada en el compilador: esta se realiza cuando el número de registros disponibles es pequeño, como 16 o 32. El compilador es responsable de asignar los registros de manera eficiente, ya que los programas de alto nivel no hacen referencia directa a los registros. Este proceso implica asignar un número ilimitado de registros simbólicos a un número fijo de registros reales, y cuando se agotan los registros reales, algunas variables se asignan a posiciones de memoria. Este tipo de optimización se logra utilizando técnicas como el "coloreado de grafos", que ayuda a asignar las variables a los registros de manera eficiente sin causar conflictos.

El debate entre RISC y CISC ha sido largo y polémico, principalmente debido a las diferencias en sus enfoques y objetivos. Una de las razones principales para optar por una arquitectura **CISC** es la simplificación del compilador. La idea es que las instrucciones más complejas pueden facilitar el trabajo de los compiladores, ya que se pueden realizar más operaciones con una sola instrucción. Sin embargo, esta razón no siempre es tan clara en la práctica:

- Dificultad de aprovechamiento de las instrucciones complejas: aunque las instrucciones complejas pueden parecer ventajosas en principio, no siempre son fáciles de optimizar. La optimización en CISC es más difícil, ya que implica tratar con instrucciones de tamaño y velocidad variables.
- Tamaño del programa: si bien se espera que los programas sean más pequeños debido a las instrucciones más complejas, este beneficio es relativo. Hoy en día, la memoria es bastante barata, y el número de bits que ocupa un programa no necesariamente será más pequeño solo porque haya menos instrucciones. De hecho, las instrucciones más complejas pueden requerir códigos de operación más largos, y las referencias a registros requieren más bits.
- Velocidad de ejecución: en teoría, las instrucciones más complejas pueden hacer que el programa sea más rápido. Sin embargo, al optar por usar instrucciones más sencillas, los procesadores pueden tener una unidad de control más compleja y una memoria de control de microprograma más grande, lo que puede aumentar el tiempo de ejecución de las instrucciones simples.

No existe una clara barrera entre RISC y CISC; muchos diseños modernos incluyen características de ambos enfoques. Por ejemplo, arquitecturas como PowerPC y Pentium II combinan elementos de RISC y CISC.

Controversia entre RISC y CISC

- Comparación cuantitativa: implica comparar el tamaño de los programas y su velocidad de ejecución en ambas arquitecturas. Sin embargo, este tipo de comparación no siempre refleja de manera justa las diferencias, ya que las características del hardware y el compilador influyen directamente en los resultados.
- Comparación cualitativa: examina el soporte de lenguajes de alto nivel (HLL) y cómo se optimizan los recursos VLSI (Very Large Scale Integration), es decir, la cantidad de circuitos en un chip.

Problemas con las comparaciones:

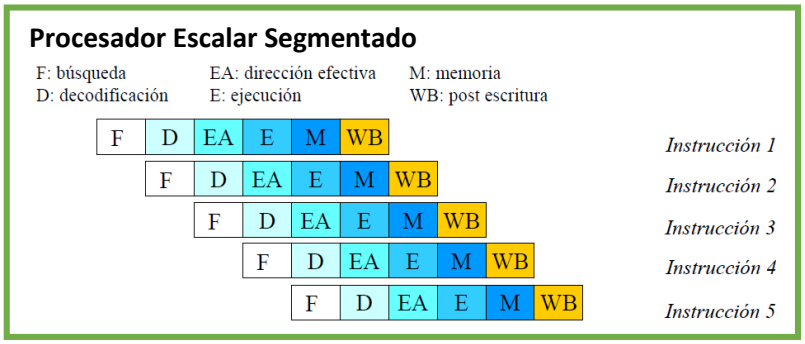
- No existen máquinas RISC y CISC directamente comparables, ya que los diseños evolucionan y se combinan, dificultando la creación de un par de máquinas ideales para comparación.
- No existe un conjunto de programas de prueba definitivo que sea capaz de cubrir todas las diferencias entre ambas arquitecturas.
- Es difícil separar los efectos del hardware de los efectos del compilador, lo que puede influir en el rendimiento general de un sistema.
- La mayoría de las comparaciones se han realizado con máquinas simples, a menudo conocidas como "máquinas de juguete", lo que no refleja el rendimiento real de los productos comerciales.
- Mezcla de RISC y CISC: La mayoría de las máquinas modernas son una mezcla de características de ambos enfoques, lo que hace que la comparación directa sea aún más difícil.

En conclusión, no hay un consenso claro sobre qué enfoque es superior, ya que depende de los requisitos específicos del sistema y las aplicaciones que se desean ejecutar. Ambas arquitecturas tienen sus ventajas y desventajas, y muchos diseños modernos intentan combinar lo mejor de ambos mundos.

Procesadores Superescalares

Procesador Escalar Segmentado

Un procesador escalar segmentado es aquel en el que las instrucciones se ejecutan de manera secuencial, pero con una optimización que permite que nuevas instrucciones puedan iniciarse mientras otras ya están en proceso. Esta técnica de segmentación del cauce mejora el rendimiento al dividir la ejecución en diferentes etapas, permitiendo que varias fases del procesamiento de diferentes instrucciones ocurran de manera simultánea.

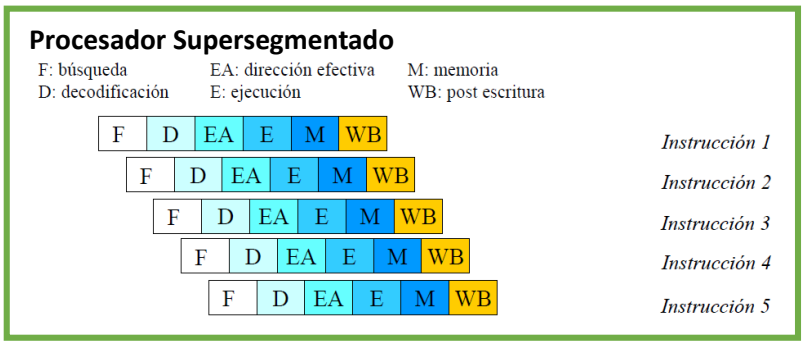


Segmentación del Cauces: una unidad segmentada es una secuencia de etapas donde diferentes fases de ejecución de distintas instrucciones pueden solaparse, permitiendo que las nuevas instrucciones comiencen mientras otras ya están siendo procesadas.

La evolución de los procesadores segmentados dio lugar a dos enfoques de ejecución de mayor rendimiento: **procesadores supersegmentados** y **procesadores superescalares**.

1. Enfoque Supersegmentado

- Optimización del Ciclo de Reloj: Algunas operaciones no necesitan un ciclo de reloj completo, por lo que se subdivide el ciclo de reloj en sub-intervalos, resultando en una mayor frecuencia de reloj.
- División de Etapas: Las etapas "macro" del cauce segmentado se dividen en sub-etapas más pequeñas y rápidas. Esto acelera la transmisión de datos, permitiendo una mayor velocidad en la ejecución de instrucciones individuales.
- Aumento del Paralelismo: Al dividir las etapas de la ejecución, se incrementa el grado de paralelismo y la aceleración percibida, ya que más tareas se ejecutan de manera paralela.



Procesadores Superescalares. a) ¿Cuáles son las características distintivas? [EXAMEN 11/24]

a) ¿Qué características tienen los procesadores Superescalares? [EXAMEN 10/22]

2. Enfoque Superescalar

Un procesador superescalar es aquél que usa múltiples cauces de instrucciones independientes. Cada cauce consta de múltiples etapas, de modo que puede tratar varias instrucciones a la vez. El hecho de que haya varios cauces introduce un nuevo nivel de paralelismo, permitiendo que varios flujos de instrucciones se procesen simultáneamente.

Típicamente, un procesador superescalar capta varias instrucciones a la vez, y a continuación intenta encontrar instrucciones cercanas que sean independientes entre sí y puedan, por consiguiente, ejecutarse en paralelo. Si la entrada de una instrucción depende de la salida de una instrucción precedente, la segunda instrucción no puede

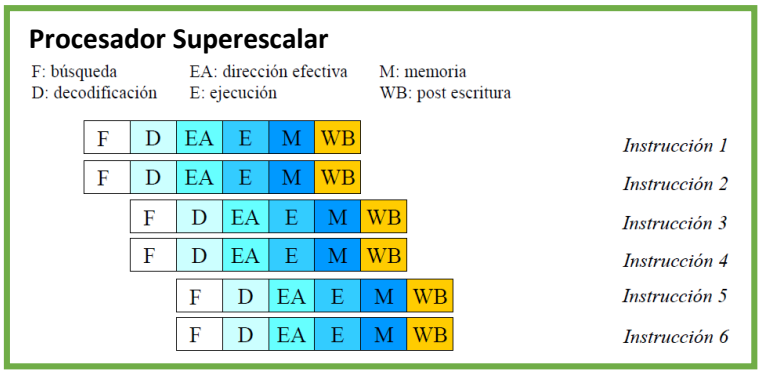
completar su ejecución al mismo tiempo o antes que la primera. Una vez que se han identificado tales dependencias, el procesador puede emitir y completar instrucciones en un orden diferente al del código máquina original.

El procesador puede eliminar algunas dependencias innecesarias mediante el uso de registros adicionales y el renombramiento de las referencias a registros del código original. La mayoría de las máquinas Superescalares emplean métodos tradicionales de predicción de saltos para aumentar su rendimiento [Stallings].

Un procesador superescalar es aquél que usa múltiples cauces de instrucciones independientes. Cada cauce consta de múltiples etapas, de modo que puede tratar varias instrucciones a la vez, por lo que saca provecho del paralelismo en las instrucciones.

Capta varias instrucciones a la vez, y a continuación intenta encontrar instrucciones cercanas que sean independientes entre sí y puedan ejecutarse en paralelo. Las principales características son:

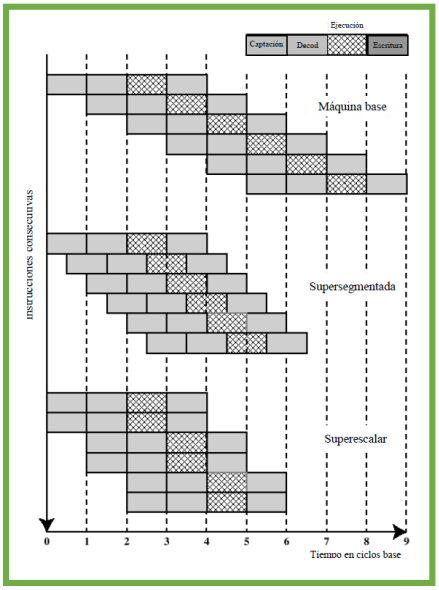
- Ejecución Simultánea: Se pueden ejecutar múltiples instrucciones simultáneamente, lo que conlleva la duplicación de algunas partes de la CPU y ALU.
- Paralelismo de Instrucciones: Permite captar varias instrucciones al mismo tiempo, lo que facilita la ejecución paralela de operaciones, como sumas y multiplicaciones o cargas y almacenes, sin que se interfieran entre sí.



- Incremento de la Aceleración: Al ejecutar más instrucciones en paralelo, se incrementa el grado de paralelismo y, por tanto, la aceleración de la máquina.
- Lógica para determinar dependencias verdaderas entre valores de registros y mecanismos para comunicar esos valores.
- Mecanismos para entregar el estado del procesador en un orden correcto.

Superescalar vs. Supersegmentado

Una solución alternativa para alcanzar mayores prestaciones es la llamada supersegmentación. Esta aprovecha el hecho de que muchas etapas del cauce realizan tareas que requieren menos de medio ciclo de reloj. De este modo, doblando la velocidad de reloj interna se permite la realización de dos tareas en un ciclo de reloj externo.



b) Describa las causas que pueden retardar el funcionamiento de los procesadores superescalares. [EXAMEN 10/22]

Limitaciones: la aproximación superescalar depende de la habilidad para ejecutar múltiples instrucciones en paralelo. La expresión **paralelismo en las instrucciones** se refiere al grado en el que, en promedio, las instrucciones de un programa se pueden ejecutar en paralelo. Para maximizar el paralelismo en las instrucciones, se puede usar una combinación de optimizaciones realizadas por el compilador y de técnicas hardware. Limitaciones fundamentales del paralelismo a las que el sistema tiene que enfrentarse.

- Dependencia de datos verdadera [RAW]. Cuando una instrucción necesita un dato producido por otra instrucción anterior. Ejemplo: la salida de la instrucción (1) es un operando para la instrucción (2) (también puede ser llamada dependencia de flujo o dependencia escritura-lectura).
- Dependencia relativa al procedimiento. Las instrucciones que siguen a una bifurcación (en la que se puede saltar o no) tienen una dependencia relativa al procedimiento en esa bifurcación y no pueden ejecutarse hasta que se ejecute el salto.
- Conflicto en los recursos. Un conflicto en un recurso es una pugna de dos o más instrucciones por el mismo recurso al mismo tiempo. Ejemplos de recursos son las memorias, las cachés, los buses, los puertos del banco de registros y las unidades funcionales (por ejemplo, un sumador de la ALU).
- Dependencia de salida [WAW]. Ocurre cuando dos instrucciones intentan escribir en el mismo registro.
- Antidependencia [WAR]. Se produce cuando una instrucción no puede comenzar hasta que la instrucción anterior haya escrito su resultado.

a) ¿De qué depende el paralelismo de una máquina superescalar? [EXAMEN 07/23 – 03/24 – 07/24 – 08/24 –12/24]

Paralelismo en las Instrucciones y Paralelismo de la Máquina: existe **paralelismo en las instrucciones** cuando las instrucciones de una secuencia son independientes y por tanto pueden ejecutarse en paralelo solapándose. El paralelismo en las instrucciones depende de la frecuencia de dependencias de datos verdaderas y dependencias relativas al procedimiento que haya en el código. Estos factores dependen a su vez de la arquitectura del repertorio de instrucciones y de la aplicación. El paralelismo en las instrucciones depende también de la latencia de una operación: el tiempo que transcurre hasta que el resultado de una instrucción está disponible para ser usado como operando de una instrucción posterior. La latencia determina cuánto retardo causará una dependencia de datos o relativa al procedimiento.

El paralelismo de la máquina es una medida de la capacidad del procesador para sacar partido al paralelismo en las instrucciones. El paralelismo de la máquina depende del número de instrucciones que pueden captarse y ejecutarse al mismo tiempo (número de cauces paralelos) y de la velocidad y sofisticación de los mecanismos que usa el procesador para localizar instrucciones independientes. Tanto el paralelismo en las instrucciones como el paralelismo de la máquina son factores importantes para mejorar las prestaciones. Depende de varios factores:

- Número de instrucciones que se captan por ciclo.
- Número de unidades funcionales disponibles.
- Mecanismos para localizar instrucciones independientes.
  - Identificación de paralelismo y organización de las etapas de ejecución de forma paralela (F, D y E).
  - Renombre de registros (para manejar dependencias de salida o antidependencias).
  - Ventana de instrucciones (emisión desordenada).

b) Compare las políticas de Emision de Instrucciones. [EXAMEN 11/24]

Políticas de Emisión en Procesadores Superescalares: el paralelismo de la máquina no solo es un asunto relacionado con la existencia de múltiples réplicas de cada etapa del cauce. El procesador además tiene que ser capaz de identificar el paralelismo en las instrucciones y organizar la captación, decodificación, y ejecución de las instrucciones en paralelo.

El término **política de emisión de instrucciones** se refiere al protocolo usado para emitir instrucciones. En general, se puede decir que la emisión de una instrucción tiene lugar cuando esta pasa de la etapa de decodificación del cauce a la primera etapa de ejecución.

Esencialmente, el procesador intenta localizar instrucciones más allá del punto de ejecución en curso que puedan introducirse en el cauce y ejecutarse. Con respecto a esto, hay tres ordenaciones importantes:

- El orden en que se captan las instrucciones.
- El orden en que se ejecutan las instrucciones.
- El orden en que las instrucciones actualizan los contenidos de los registros y de las posiciones de memoria.

Las instrucciones deben emitirse y completarse de manera controlada, respetando el orden en el que se captan, se ejecutan y actualizan los registros/memoria. Sin embargo, dependiendo de la política adoptada, la emisión de instrucciones de los procesadores superescalares puede ser:

- Emisión y finalización en orden. La política de emisión de instrucciones más sencilla es emitir instrucciones en el orden exacto en que lo haría una ejecución secuencial (emisión en orden) y escribir los resultados en ese mismo orden (finalización en orden).
- Emisión en orden y finalización desordenada. La finalización desordenada se usa en los procesadores RISC escalares para mejorar la velocidad de las instrucciones que necesitan ciclos. Necesita una lógica de emisión de instrucciones más compleja que la finalización en orden. Además, es más difícil ocuparse de las interrupciones y excepciones.
- Emisión y finalización desordenada. Las instrucciones se emiten a la etapa de ejecución en el orden en que están listas para ser ejecutadas, y se pueden completar en cualquier orden, siempre y cuando no altere el resultado final.

c) Elija una alternativa de emisión/finalización y justifique. [EXAMEN 11/24]

La **emisión y finalización desordenada** es una estrategia poderosa para procesadores superescalares, ya que **aumenta la eficiencia**, maximiza el paralelismo de instrucciones y mejora el rendimiento global del procesador. A través del uso adecuado de **renombrado de registros** y **gestión dinámica de dependencias**, este enfoque permite a los procesadores aprovechar mejor los recursos y manejar excepciones sin comprometer la consistencia del sistema. Además, al mantener la **finalización ordenada**, se garantiza que los resultados se escriban en el orden adecuado, evitando inconsistencias.

b) ¿Cuál es el objetivo de la técnica de renombre de registros? [EXAMEN 03/24 – 08/24 – 12/24]

Renombre de Registros

Para gestionar las dependencias de salida y antidependencias, se utiliza una técnica llamada renombrado de registros. Esto permite evitar conflictos de almacenamiento y asegura que cada instrucción escriba en un registro distinto, asignando registros dinámicamente y modificando las referencias para reflejar esta asignación.

En el renombramiento de registros, las dependencias de salida y antidependencias se eliminan mediante la asignación de nuevos registros a las instrucciones posteriores, evitando que los registros reflejen valores incorrectos en el flujo del programa.

[Stallings] Renombramiento de registros: básicamente, el hardware del procesador asigna dinámicamente los registros, que están asociados con los valores que necesitan las instrucciones en diversos instantes de tiempo. Cuando se crea un nuevo valor de registro (es decir, cuando se ejecuta una instrucción que tiene un registro como operando destino) se asigna un nuevo registro para ese valor. Las instrucciones posteriores que accedan a ese valor como operando fuente en ese registro tienen que sufrir un proceso de renombramiento: las referencias a registros de esas instrucciones han de revisarse para referenciar el registro que contiene el valor que se necesita. De este modo, las referencias a un mismo registro original en diferentes instrucciones pueden referirse a distintos registros reales, suponiendo diferentes valores.

b) ¿Qué características tiene la implementación de un procesador superescalar? [EXAMEN 10/23]

Implementación Superescalar: la implementación de un procesador superescalar requiere varios mecanismos para maximizar la eficiencia:

- Captación Simultánea de Instrucciones: las instrucciones deben ser captadas simultáneamente y decodificadas para identificar dependencias y posibles conflictos. A menudo prediciendo los resultados de las instrucciones de salto condicional y captando más allá de ellas. Estas funciones requieren la utilización de múltiples etapas de captación y decodificación, y lógica de predicción de saltos.
- Detección de Dependencias: se utilizan mecanismos para detectar las dependencias entre los valores de los registros y asegurar que las instrucciones se ejecuten sin interferencias.
- Mecanismos para iniciar o emitir múltiples instrucciones en paralelo.
- Recursos para la ejecución en paralelo de múltiples instrucciones, que incluyan múltiples unidades funcionales segmentadas y jerarquías de memoria capaces de atender múltiples referencias a memoria.
- Mecanismos para entregar el estado del procesador en el orden correcto.
- Excepciones en Procesadores Superescalares: cuando ocurre una excepción, puede haber varias instrucciones en ejecución simultáneamente. Para garantizar que el procesador mantenga un estado consistente, se debe:

En resumen, el procesador superescalar utiliza técnicas avanzadas de segmentación y ejecución paralela para mejorar el rendimiento y la eficiencia de los programas. A través de la ejecución simultánea de múltiples instrucciones y el renombrado dinámico de registros, se optimizan las operaciones en entornos de alta concurrencia. Sin embargo, esto presenta desafíos relacionados con la gestión de excepciones y la dependencia de datos, que deben ser cuidadosamente manejados para mantener la coherencia y precisión en el procesamiento.

Procesamiento Paralelo

a) ¿Qué es y de qué depende el paralelismo de una máquina? [EXAMEN 10/23]

Introducción al Procesamiento Paralelo

El aumento de las demandas de rendimiento ha llevado a la exploración de arquitecturas y técnicas que permiten una mejora significativa en el rendimiento mediante el uso de varios procesadores o recursos paralelos. Aunque es posible mejorar el rendimiento de un solo procesador, la verdadera ganancia de velocidad y capacidad de procesamiento se obtiene mediante el paralelismo, ya sea a nivel de instrucciones o procesos. En este contexto, existen diferentes estrategias de paralelismo y arquitecturas que se pueden aplicar en sistemas computacionales. **Niveles de Paralelismo**:

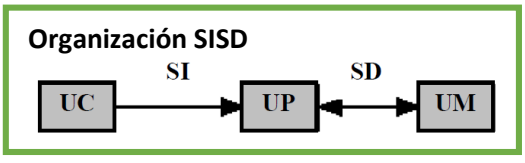
- Paralelismo a Nivel de Instrucción (ILP): se refiere a la capacidad de ejecutar múltiples instrucciones simultáneamente dentro de un solo procesador, aprovechando su capacidad de procesamiento interno.
- Paralelismo a Nivel de Proceso: implica la ejecución concurrente de múltiples procesos, donde cada uno puede estar realizando operaciones distintas en diferentes partes de un sistema.

a) Describa las cuatro variantes de arquitecturas de la taxonomía de Flynn. [EXAMEN 11/24]

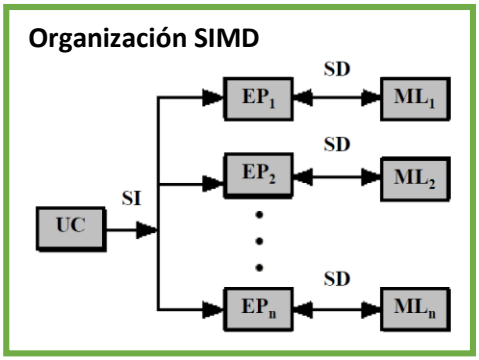
Taxonomía de las Arquitecturas (Flynn): la taxonomía de Flynn es la forma más común de clasificar a los sistemas según sus capacidades de procesamiento paralelo. Flynn propuso las siguientes categorías o clases de computadores:

- **SISD [Single Instruction Single Data]**: una secuencia de instrucciones y una secuencia de datos. Un único procesador interpreta una única secuencia de instrucciones para operar con los datos almacenados en una única memoria. Los computadores monoprocesador caen dentro de esta categoría.
- **SIMD [Single Instruction Multiple Data]**: una secuencia de instrucciones y múltiples secuencias de datos. Una única instrucción máquina controla paso a paso la ejecución simultánea y sincronizada de un cierto número de elementos de proceso. Cada elemento de proceso tiene una memoria asociada, de forma que cada instrucción es ejecutada por cada procesador con un conjunto de datos diferentes. Los procesadores vectoriales y los matriciales pertenecen a esta categoría.
- **MISD [Multiple Instruction streams, Single Data stream]**: múltiples secuencias de instrucciones y una secuencia de datos. Se transmite una secuencia de datos a un conjunto de procesadores, cada uno de los cuales ejecuta una secuencia de instrucciones diferente. Esta estructura nunca ha sido implementada.
- **MIMD [Multiple Instruction streams, Multiple Data stream]**: múltiples secuencias de instrucciones y múltiples secuencias de datos. Un conjunto de procesadores ejecuta diferentes secuencias de instrucciones de manera simultánea, operando sobre diferentes secuencias de datos. Los SMP, los clusters y los sistemas NUMA son ejemplos de esta categoría.

En la organización **SISD** [una secuencia de instrucciones y una secuencia de datos] un solo procesador ejecuta una única secuencia de instrucciones (SI) sobre una secuencia de datos, y los datos son almacenados en una única memoria (UM). Arquitectura típica: Computadoras monoprocesador.



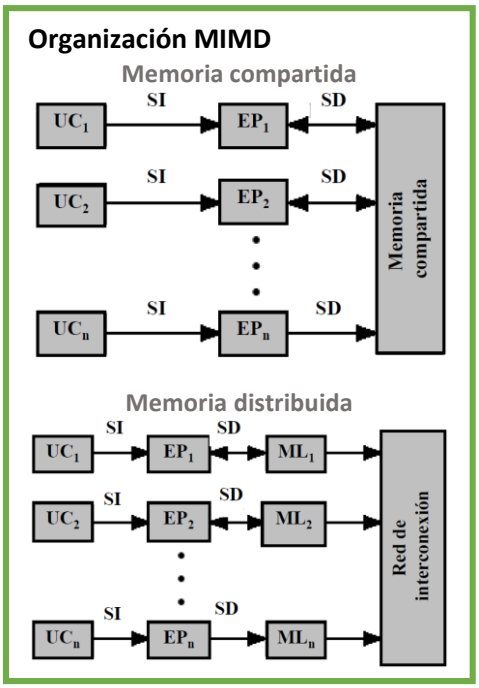
En la organización **SIMD** [una secuencia de instrucciones y múltiples secuencias de datos] una única instrucción controla paso a paso la ejecución de múltiples elementos de proceso (EP) simultáneamente. Cada elemento de proceso tiene su propia memoria dedicada (ML). Cada procesador ejecuta la misma instrucción pero sobre diferentes conjuntos de datos. Algunos ejemplos de esta organización son los procesadores vectoriales y matriciales, utilizados para operaciones de matrices o vectores en paralelo.



En la organización **MISD** [múltiples secuencias de instrucciones y una secuencia de datos] se transmite una secuencia de datos a un conjunto de procesadores, cada uno ejecutando una secuencia diferente de instrucciones. Este tipo de arquitectura nunca ha sido implementada debido a su falta de aplicaciones prácticas eficientes.

a) Que son los MIMD de la taxonomía de Flynn. Describa las variantes que conozca. [EXAMEN 03/24 – 08/24 – 12/24]

En la organización **MIMD** [múltiples secuencias de instrucciones y múltiples secuencias de datos] los procesadores son de uso general; cada uno es capaz de procesar todas las instrucciones necesarias para realizar las transformaciones apropiadas de los datos, es decir, un conjunto de procesadores ejecuta secuencias de instrucciones diferentes en simultáneo. Los computadores MIMD se pueden subdividir además según la forma que tienen los procesadores para comunicarse.



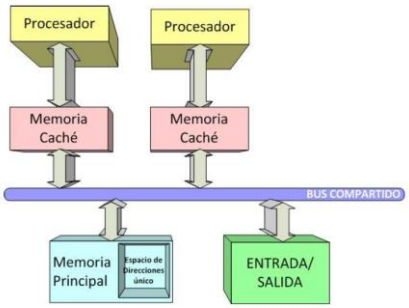
- Memoria compartida: si los procesadores comparten una memoria común, entonces cada procesador accede a los programas y datos almacenados en la memoria compartida, y los procesadores se comunican unos con otros a través de esa memoria.
    - Multiprocesador simétrico (SMP): varios procesadores comparten una única memoria mediante un bus compartido u otro tipo de mecanismo de interconexión. Una característica distintiva de estos sistemas es que el tiempo de acceso a memoria principal es aproximadamente el mismo para cualquier procesador.
    - Organización con acceso no uniforme a memoria (NUMA): el tiempo de acceso a zonas de memoria diferentes puede diferir en un computador NUMA.
- Memoria distribuida: un conjunto de computadores monoprocesador independientes o de SMP pueden interconectarse para formar un cluster. La comunicación entre los computadores se realiza bien mediante conexiones fijas o mediante algún tipo de red.

¿Qué es un multiprocesador Simétrico?

Multiprocesador Simétrico (SMP): puede definirse como un computador con las siguientes características:

1. Hay dos o más procesadores similares de capacidades comparables.

Arquitectura de un SMP



2. Estos procesadores comparten la memoria principal y las E/S y están interconectados mediante un bus u otro tipo de sistema de interconexión, de forma que el tiempo de acceso a memoria es aproximadamente el mismo para todos los procesadores.
3. Todos los procesadores comparten los dispositivos de E/S, bien a través de los mismos canales o mediante canales distintos que proporcionan caminos de acceso al mismo dispositivo.
4. Todos los procesadores pueden desempeñar las mismas funciones.
5. El sistema está controlado por un sistema operativo integrado que proporciona la interacción entre los procesadores y sus programas a los niveles de trabajo, tarea, fichero y datos.
6. Todos los procesadores tienen el mismo tiempo de acceso a la memoria (UMA).

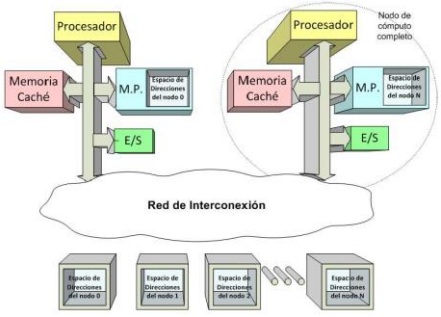
Un SMP tiene las siguientes ventajas potenciales con respecto a una arquitectura monoprocesador:

- Mayores prestaciones: si el trabajo se organiza correctamente en paralelo.
- Alta disponibilidad: un fallo en un procesador no detendrá la computadora.
- Escalado: los fabricantes pueden ofrecer una gama de productos con precios y prestaciones diferentes en función del número de procesadores que configuran el sistema.
- Crecimiento incremental: se pueden añadir más procesadores.
- Bus de tiempo compartido: es el mecanismo más simple para construir un sistema multiprocesador. El Bus de tiempo compartido tiene algunas desventajas, como que la prestación está limitada por el tiempo de ciclo del bus, el hecho de que cada procesador debería estar equipado con una cache para mejorar las prestaciones, y a partir de esto, que se pueden generar problemas de coherencia de cache.

Características de los Clusters

Clusters: Los clusters constituyen la alternativa a los multiprocesadores simétricos (SMP) para disponer de prestaciones y disponibilidad elevadas, y son particularmente atractivos en aplicaciones propias de un servidor. Se puede definir un cluster como un grupo de computadores completos interconectados que trabajan conjuntamente como un único recurso de cómputo, creándose la ilusión de que se trata de una sola máquina (cada computadora: nodo). Los beneficios de los Clusters son:

Arquitectura de un Cluster



- Escalabilidad absoluta: es posible configurar clusters grandes que incluso superan las prestaciones de los computadores independientes más potentes. Un cluster puede tener decenas de máquinas, cada una de las cuales puede ser un multiprocesador.
- Escalabilidad incremental: un cluster se configura de forma que sea posible añadir nuevos sistemas al cluster en ampliaciones sucesivas. Así, un usuario puede comenzar con un sistema modesto y ampliarlo a medida que lo necesite, sin tener que sustituir el sistema de que dispone por uno nuevo que proporcione mayores prestaciones.
- Alta disponibilidad: puesto que cada nodo del cluster es un computador autónomo, el fallo de uno de los nodos no significa la pérdida del servicio.

- Mejor relación precio-prestaciones: al utilizar elementos estandarizados, es posible configurar un cluster con mayor o igual potencia de cómputo que un computador independiente mayor, a mucho menos costo.

b) Compare los sistemas SMP y Clusters. [EXAMEN 11/24]

Comparación Cluster vs SMP

Tanto los clusters como los SPM constituyen configuraciones con varios procesadores que pueden ejecutar aplicaciones con una alta demanda de recursos. Ambas soluciones están disponibles comercialmente, aunque los SMP lo están desde hace más tiempo.

La principal ventaja de un SMP es que resulta más fácil de gestionar y configurar que un cluster. El SMP está mucho más cerca del modelo de computador de un solo procesador para el que están disponibles casi todas las aplicaciones. Otra ventaja de un SMP es que necesita menos espacio físico y consume menos energía que un cluster comparable. Además, los SMP son plataformas estables y bien establecidas.

Con el tiempo, no obstante, las ventajas de los clusters serán las que probablemente harán que sean estos los que dominen en el mercado de servidores de altas prestaciones puesto que los clusters son superiores a los SMP en términos de escalabilidad absoluta e incremental, y además también son superiores en términos de disponibilidad, puesto que todos los componentes del sistema pueden hacerse altamente redundantes.

Acceso no Uniforme a Memoria

Las dos alternativas para ejecutar aplicaciones en un sistema con varios procesadores son los SMP y los clusters. Pero durante algunos años, había otra alternativa conocida como Acceso No Uniforme a Memoria [NUMA]. Algunos términos que se encuentran a menudo en la literatura de computadores NUMA son:

Acceso uniforme a memoria [UMA]:

- Todos los procesadores pueden acceder a toda la memoria principal utilizando instrucciones de carga y almacenamiento.
- El tiempo de acceso de un procesador a cualquier región de la memoria es el mismo.
- El tiempo de acceso a memoria por parte de todos los procesadores es el mismo.

Acceso no uniforme a memoria [NUMA]:

- Todos los procesadores tienen acceso a todas las partes de memoria principal utilizando instrucciones de carga y almacenamiento.
- El tiempo de acceso a memoria de un procesador depende de la región a la que se acceda.
- Diferentes procesadores acceden a diferentes regiones de memoria en diferentes velocidades.

NUMA con coherencia de caché [CC-NUMA]:

- Un computador NUMA en el que la coherencia de caché se mantiene en todas las cachés de los distintos procesadores.
- Organización CC-NUMA típica: cada procesador tiene cache L1 y L2; cada nodo tiene su propia memoria principal; los nodos están conectados por algún tipo de red y cada procesador ve un único espacio de direcciones de memoria. [Orden de acceso a memoria: cache L1, cache L2, Memoria Principal, Memoria Remota].

En síntesis, el objetivo de un computador NUMA es mantener una memoria transparente desde cualquier parte del sistema, al tiempo que se permiten varios nodos de multiprocesador, cada uno con su propio bus u otro sistema de interconexión interna.

Procesamiento Multihebra: su objetivo es aumentar el paralelismo de instrucciones sin añadir la complejidad y el consumo de potencia de la segmentación de cauce y los procesadores superescalares. La secuencia de instrucciones se divide en secuencias más pequeñas llamadas hebras que pueden ejecutarse en paralelo. Hay una amplia variedad de diseños multihebra.

El concepto de hebra utilizado para estudiar los procesadores multihebra puede ser o no ser el mismo que el concepto de hebra en un sistema operativo multiprogramado. Por lo tanto, es útil definir estos términos:

- Proceso: un programa en ejecución que tiene asignados recursos como memoria y espacio de direcciones (programa corriendo en una computadora).
- Conmutación de proceso (process switch) implica el cambio de ejecución entre diferentes procesos.
- Hebra (Thread): es una unidad de trabajo dentro de un proceso que puede ejecutarse independientemente. Cada hebra tiene su propio contexto de procesador (incluido PC y SP) y área de datos para su pila. Se ejecuta secuencialmente y es interrumpible (el procesador cambiaría a otra hebra).
- La conmutación de hebra (thread switch) es más eficiente que la conmutación de proceso.

Tipos de Multihebra

- Multihebra Explícito: ejecución concurrente de instrucciones de diferentes hebras explícitas. Mezcla de instrucciones de diferentes hebras en cauces compartidos o en cauces paralelos. Todos los procesadores comerciales la usan.
- Multihebra Implícito: ejecución concurrente de varias hebras extraídas de un único programa secuencial. Definidas estáticamente por el compilador o dinámicamente por el hardware cómo dividir las instrucciones en hebras.