Unidad 11 - Gestión de Periféricos

1. Buses

Los buses son los sistemas de comunicación entre componentes de la computadora (o entre computadoras). Puede catalogarse según el **tipo de información** que transportan, a los **componentes** que conectan o a la forma en la que transmiten esa información.

Dependiendo de la información que transportan, se clasifican en:

- **Bus de Datos**: proporciona una ruta para mover datos entre módulos del sistema. Es **bidireccional** y su ancho determina el rendimiento general, en combinación con su latencia.
- **Bus de Direcciones**: transporta las rutas que designaran el origen/destino de los datos del bus de datos. Es **unidireccional** y su ancho determina la capacidad de memoria máxima del sistema (cantidad de bits de una dirección de memoria, 8, 16, 32, 64 bits). Además se utiliza para direccionar los puertos de E/S, caso en el que los bits superiores seleccionan un módulo en el bus y los bits inferiores determinan la ubicación de la memoria o el puerto.
- **Bus de Control**: controlan el acceso y uso de los datos y direcciones. Es **bidireccional** y transmite información de *comando* y *sincronización* entre los módulos. Las señales de comando determinan las operaciones, mientras que las de sincronización determinan la validez de los datos y la información de la dirección.

1.1. Generaciones de Buses

Las generaciones en las que se clasifican los buses son:

- **Primera Generación**: había buses separados para memoria y periféricos. Los periféricos eran accedidos a través de instrucciones especiales, con tiempos y protocolos diferentes. La CPU controlaba el sistema, por lo que el sistema no era sencillo de ampliar ni actualizar. Además, una gran parte de la potencia de la CPU se utilizaba en controlar los buses.
- **Segunda Generación**: los buses de unificaron y estandarizaron, la arquitectura se separó en grupos de acuerdo a su velocidad y se incluyó un controlador de bus, que gestiona el flujo de datos de la CPU a los periféricos y entre dispositivos. Los problemas de velocidad se resolvieron aumentando el tamaño del bus y agregando configuración por software (plug & play).
- **Tercera Generación**: se reformula la arquitectura como una red de alta velocidad con topología variable, una configuración con conexiones físicas flexibles, para conectar buses internos como máquinas entre sí. El problema que surge es el de atender solicitudes tan diferente, tarea que recae en gran parte al diseño del software en lugar del hardware. Ejemplos: HyperTransport, InfiniBand, Wishbone.

2. Gestión de Periféricos

La gestión o arbitraje de bus refiere a la coordinación de los accesos de los dispositivos a un bus compartido, para evitar conflictos cuando múltiples dispositivos solicitan acceso simultaneamente.

Los tipos de transferencias que pueden realizarse son:

- Memoria a periférico.
- Periférico a memoria.
- Entre bloques de memoria.

Los mecanismos para gestionar estas transferencias son:

- Gestión **programada**: el procesador controla directamente los accesos.
- Gestión **por interrupciones**: los dispositivos solicitan acceso mediante interrupciones.
- Gestión **por acceso directo** (DMA): un controlador dedicado que gestiona los accesos sin intervención de la CPU.

Y los enfoque para el arbitraje de buses son:

- **Centralizado**: un único árbitro gestiona el bus. Por ejemplo: un controlador dedicado
- **Distribuido**: todos los dispositivos pueden gestionar el bus.

2.1. Métodos de Arbitraje

Como ya dijimos, el arbitraje consiste en coordinar la transferencia de buses entre dispositivos a través de los buses. Existen distintos métodos de arbitraje.

2.1.1. Encadenamiento

Daisy-chain, es un método centralizado y por interrupciones, donde las señales de prioridad pasan de un dispositivo al siguiente en la cadena.

- Ventajas: Simplicidad y escalabilidad.
- Desventajas: la prioridad depende de la cercanía al procesador y si un dispositivo falla, todo el sistema falla.

2.1.2. Rotación

Rotating Priority, un método implementado por acceso directo (DMA). La prioridad cambia en cada ciclo de forma rotativa, distribuyendo el acceso equitativamente.

- Ventajas: equidad en el acceso.
- Desventajas: gestión compleja al aumentar el tamaño del sistema.

2.1.3. Fija

Fix priority, cada dispositivo tiene una prioridad fija y el árbitro siempre selecciona el dispositivo con mayor prioridad entre los que soliciten acceso.
Ventajas: sencillo y de velocidad de respuesta rápida.
Desventajas: dispositivos de baja prioridad pueden sufrir "inanición", por lo que requiere mucho cuidad en la asignación de prioridades.

2.2. Gestión Programada

En este mecanismo las tareas para atender periféricos se organizan en el programa, que atiende cada uno de los periféricos hasta completar la lista de tareas. Para esto, el procesador debe tener el control directo sobre la operación de transferencia:

- Comprobación del estado del dispositivo.
- Envío de comandos de lectura/escritura.
- Transferencia de datos.
- Espera a que el dispositivo termine la transferencia.

Ya que el orden de atención es controlado por el programa, la prioridad de cada dispositivo puede asignarse dinámicamente según el estado del sistema. Una asignación de prioridad se realiza cambiando el orden de consulta.

- **Ventajas**: control total sobre el proceso, robusto a los eventos sincrónicos y tiene alta reutilización en la ejecución.
- **Desventajas**: no puede atender eventos asíncronos, la carga de programación es elevada y genera altos tiempos de frecuencia.

Los **pasos de la gestión programada** son:

- Procesador:
  - Direcciona a un dispositivo y solicita una operación.
  - Comprueba periódicamente los bits de estado.
  - En caso de no haber completado la tarea, vuelve a comprobar más tarde.
- Dispositivo:
  - Cuando recibe una operación, desactiva los bits de estado.
  - Realiza la operación.
  - Activa los bits de estado y espera una nueva operación.

2.3. Gestión por Interrupciones

Unidad 12

2.4. Gestión por Acceso Directo

El mecanismo por **acceso directo a memoria** (Direct Memory Access - DMA) libera al procesador de las tareas de transferencia de datos, permitiendo que dispositivos de diferentes velocidades accedan a la memoria del sistema para leer o escribir, independientemente de la CPU y sin sobrecargarla.
Aunque no se necesite a la CPU para la transferencia de datos, se **necesita el bus del sistema**, por lo que se necesita regular su uso para no quedar acaparado por el controlador DMA. En caso de que este dispositivo haga uso intensivo del bus, el rendimiento del sistema puede verse afectado ya que la CPU no puede leer datos de memoria y debe esperar a que el controlador DMA finalice su tarea **sin ejecutar ninguna instrucción**.

Para resolver, en parte, este problema, una memoria cache dentro de la CPU permite a ésta seguir trabajando. Pero en computadoras sin memoria cache, el DMA debe evitar ocupar el bus mientras la CPU realiza la lectura de una instrucción.

Existen para estos dos tipos de transferencias:

- **Modo Robo de Ciclo**: cuando la CPU concede el bus al DMA, este lo libera al finalizar una tarea. Para proseguir con la siguiente tarea, debe volver a solicitar permiso a la CPU.
  - Ventajas: la CPU puede seguir ejecutando instrucciones, incluso en procesadores sin memoria cache.
  - Desventajas: la transferencia de datos tarda más en realizarse.
- **Modo Transparente**: el DMA solo transfiere datos cuando la CPU no utiliza el bus.
  - Ventajas: nunca se detiene la ejecución.
  - Desventajas: se necesita hardware para determinar cuándo la CPU no utiliza los buses.

2.4.1. Gestión por Acceso Directo con Cache

En casos en los que la CPU dispone de memoria caché, la DMA puede llevar a **problemas de coherencia de caché**. Cuando la CPU accede a la posición de memoria X, el valor de la memoria principal se almacena en caché. Si la CPU realiza operaciones posteriores en X, se actualizará la copia de X en caché, pero no el valor en la memoria principal, por lo que si un dispositivo intenta acceder a X, antes de que la caché se vacía en la memoria principal, el dispositivo

recibirá un valor "caducado" de X.
Del mismo modo, la CPU utilizará un valor "caducado" de X si la copia en caché no se invalida al ser escrito un nuevo valor por un dispositivo.

Las posibles soluciones a este problema son:

- Sistemas de Caché **Coherente**: se implementa un método en el **hardware** externo que escribe una señal en el controlador de caché, invalidando la caché ante una escritura de DMA o descarga la caché ante una lectura de DMA.
- Sistemas de Caché **No-Coherente**: el **sistema operativo** se asegura de que las entradas de caché se vacían antes de una lectura de DMA, o anuladas antes de una escritura de DMA. Debe evitar que subprocesos accedan a esa parte de memoria en ese instante. Hay que tener en cuenta que este proceso sobrecarga la operación de DMA, ya que la mayoría de hardware requiere un bucle para invalidar entradas de caché de forma individual.
- Sistemas de Caché **Híbridos**: mantiene una caché L2 coherente, mientras que la L1 es gestionada por el software.